OPEN SOURCE SOFTWARE FOR TRANSPARENT, REPRODUCIBLE, USABLE BY OTHERS, AND EXTENSIBLE HIGH-THROUGHPUT MOLECULAR SIMULATIONS

By

Justin Byron Gilmer

Dissertation
Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY
in

Materials Science

May 31, 2022

Nashville, Tennessee

Approved:

Clare M^{c}Cabe, Ph.D.

Peter Cummings, Ph.D.

Kane Jennings, Ph.D.

Josh Caldwell, Ph.D.

Shihong Lin, Ph.D.

To my parents and family, supportive beyond measure.
To my friends and colleagues, whose support and guidance made this possible.

ACKNOWLEDGMENTS

I would first like to thank Peter Cummings and Clare M^{c}Cabe, my advisors and mentors, for their advice, candid scientific and professional discussions, and their support in both my academic and professional aspirations. Additionally, they supported and encouraged a multitude of opportunities for me to meet other scientists, travel, present my research, and continually improve myself.

I am grateful for those I have met during my studies who have shaped me into the researcher and developer I am today. I am especially grateful to Christoph Klein, who mentored me in my early years as a graduate student and made me a better Python programmer. Andrew Summers, a colleague, mentor, and friend, was also influential in my scientific projects and further improving my software development skills. Chris Iacovella, who has provided countless hours of assistance, scientific discussions, and has been invaluable during my time as a member of the Cummings and M^{c}Cabe labs. I would also like to acknowledge my colleagues in the M^{c}Cabe and Cummings groups, Matt Thompson, Ray Matsumoto, Alex Yang, Tim Moore, Co Quach, Cal Craven, Alyssa Nelson, Chloe Frame, Duncan Stewart, Parashara Shamaprasad, and Xiaobo Lin for their scientific discussions and general support during my time in the lab. In addition, I would like to thank Dr. Steve Stuart, my research mentor during my undergraduate studies who introduced me to computational chemistry and encouraged me to explore graduate programs.

This work would not have been possible without the financial support of the Vanderbilt Interdiscplinary Materials Science Department, the Vanderbilt School of Engineering which awarded me an IBM Graduate Fellowship, the National Science Foundation (NSF) and the National Institutes of Health (NIH), which funded me through various grants.

Finally, I would not be here today without the love, unwavering support, and guidance from my family and friends. I especially would like to thank my parents, their support, nurturing of my academic interests, and unconditional love has shaped me into the person I am today. I would also like to thank Josh Hardin, my best friend, who has supported me and offered invaluable advice throughout the years.

TABLE OF CONTENTS

*

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

LIST OF NOMENCLATURE

Acronyms

CBMC	Configurational bias Monte Carlo *p. 73*
CG	Coarse Grained *p. 120*
CG	Coarse-grained *p. xii*
CG	Coarse-grained *p. 60*
DFT	Density Functional Theory *p. 161*
DOI	Digital Object Identifier *p. 233*
EMD	Equilibrium Molecular Dynamics *p. 162*
FSA	Fractional Surface Area *p. 139*
GCMC	grand canonical Monte Carlo *p. 72*
GEMC	Gibbs Ensemble Monte Carlo *p. 72*
JC	Forcefield of Joung and Cheatham *p. 75*
LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator *p. 164*
MAE	Mean absolute error *p. 166*
MAPE	Mean absolute percentage error *p. 170*
MC	Monte Carlo *p. 59*
MD	Molecular Dynamics simulations *p. 161*
MEM	Micro-electromechanical system *p. 160*
MIC	Model integrated computing *p. 63*
ML	Machine Learning *p. 161*
MoSDeF	The Molecular Simulation and Design Framework *p. 2*
MSE	Mean squared error *p. 166*
NEM	Nano-electromechanical system *p. 160*
NEMD	Non-Equilibrium Molecular Dynamics *p. 162*
NP	Nanoparticle *p. 144*
NS	N-hydroxysphingosine *p. 77*
NSLD	Neutron scatting length density *p. 78*
OPLS	Optimized Potentials for Liquid Simulations *p. 111*
OPLS-AA	Optimized Potentials for Liquid Simulations: All-Atom *p. 111*
PDB	Protein Data Bank *p. 124*
PPPM	Particle-Particle Particle-Mesh *p. 165*
QSPR	Quantitative Structure-Property Relationship *p. 160*

SASA	Solvent Accessible Surface Area *p. 141*
SMARTS	SMILES Arbitrary Target Specification *p. 114*
SMILES	Simplified Molecular Input Line-Entry System *p. 113*
TraPPE	Transferable Potentials for Phase Equilibria *p. 111*
TRUE	Transparent, Reproducible, Usable by others, Extensible computational studies *p. 2*
VSA	van der Waals Surface Area *p. 256*
XML	Extensible Markup Language *p. 114*

Constants

k_B	Boltzmann's Constant *p. 1*

Other

μ	Coefficient of friction *p. 165*
F_0	Force of adhesion *p. 162*
F_N	Normal force *p. 165*
R^2	Coefficient of determination *p. 170*
R_g	Radius of gyration *p. 146*
`Jupyter`	An interactive python kernel and computing environment, accessible through web applications. *p. 249*
NVT	Canonical ensemble *p. 164*
PMI1	Principal Moments of Inertia *p. 256*

CHAPTER 1

Introduction

Scientific research has been described by some as being in a "reproducibility crisis" [1]. It is becoming increasingly difficult for other researchers to reproduce work described in literature. The broad field of molecular simulation is just as susceptible to these challenges in reproducibility [2]. It is quite common for the necessary input parameters and workflow information to be included in the methods or supplemental information of these publications. However, this is not always the case, key parameters and details of the simulation workflow are commonly omitted, making it difficult for researchers to reproduce, and hopefully, extend the work. This is especially true for soft material simulations (e.g., molecular liquids, polymers, and other biomaterials that are easily deformed at ambient conditions and can only be described by multiple configurations of structures over sufficiently long simulation times to predict their properties), which require complex, usually multistep, *ad hoc* initialization routines and similarly complex simulation workflows based on the application.

As massive computational resources are continually improving and increasing in availability, large scale molecular screening of soft matter is becoming more accessible, in a similar effort to the Materials Genome Initiative (MGI) [3]. MGI [3] leverages computational screening of many hard matter systems to preemptively identify potential candidate materials for experimental synthesis. With the hope that new hard materials for experimental synthesis could be identified and developed in half the time previously required to identify and develop synthesis protocols, the MGI [3] was developed to use computational screening as a cost- and time-effective way to identify systems of interest to direct experimental synthesis efforts. Being able to apply similar approaches to thin films and other soft material designs could have massive impacts on nanotribology and many other complex fields and problems in nanoscience. However, the computational study of soft materials is quite challenging to simulate compared to hard materials (e.g., crystals, alloys, salts, and other materials which do not easily deform at biologically-relevant (ambient) conditions). For example, in soft materials, the intermolecular forces are of the same order of magnitude as the thermal motion (k_BT, where k_B is Boltzmann's constant and T is the absolute temperature), in contrast to hard materials where the interatomic forces are much greater than k_BT (typically resulting in crystalline structures). This means that for soft materials, very long simulation times are required to ensure proper thermodynamic sampling as well as quite large systems to capture the mesoscale order, drastically different to hard materials. Like the MGI [3], specialized tools are necessary to better facilitate the screening and study of candidate soft matter systems. These tools need to be readily available to computational chemists, expose tunable properties of the systems to support screening, be extensible by others, and also promote and support reproducible soft matter simulations. To support these efforts, we have developed the Molecular Simulation and Design Framework (MoSDeF), an open-source set of Python libraries that are designed to address issues of automation/efficiency, accuracy, and reproducibility in molecular simulations, with a major focus on soft material applications [4–8].

The work presented first describes best practices in molecular simulation [9] (Chapter 2). With the goal that by providing more accessible best practices documents which condense many common pitfalls and other issues that impact novice simulators, common sources of errors and hindrances to reproducibility can be caught and fixed much earlier in a simulation workflow. Next, best practices and guiding principles are provided in Chapter 3 to encourage more Transparent, Reproducible, Usable by others, and Extensible (TRUE) [5] science with a focus on computational chemistry. Since MoSDeF has been in development for over a decade at this point [10, 11] development efforts and updates to MoSDeF have been continual during this time and many changes to the tools have occurred. Chapter 4 provides development updates for the three main MoSDeF libraries `mbuild`, `foyer`, and `gmso`. Then, in Chapter 5 [12] and Chapter 6 [6, 13], we explore two science applications of MoSDeF and TRUE to screen over large soft matter parameter spaces for self-assembly and tribological properties. Chapter 5 screens over coverage ratios of patchy, polymer-coated nanoparticles using coarse-grained (CG) parameters to emulate alkane polymer patches and their effect on self-assembled structures. Chapter 6 screens through thin films in shear contact, exploring the effect of terminal group chemistry and composition on tribological properties: coefficient of friction (μ) and force of adhesion (F_0). A decision tree-based random forest (RF) model is developed for both properties above, exploring the impact of thin film composition and terminal group chemistry as a predictive model, leading to orders of magnitude speedup in property prediction. Finally, the results are summarized in Chapter 7.

References

1. Baker, M. 1,500 scientists lift the lid on reproducibility. *Nature* **533,** 452–454. ISSN: 0028-0836, 1476-4687 (May 2016).

2. Schappals, M., Mecklenfeld, A., Kröger, L., Botan, V., Köster, A., Stephan, S., García, E. J., Rutkai, G., Raabe, G. & Klein, P. Round Robin Study: Molecular Simulation of Thermodynamic Properties from Models with Internal Degrees of Freedom. *Journal of Chemical Theory and Computation* **13,** 4270. ISSN: 1549-9618 (Sept. 2017).

3. OSTP. *Materials Genome Initiative for Global Competitiveness* tech. rep. (2011).

4. Contributors, M. *MoSDeF Webpage* https://mosdef.org. Accessed: 2022-03-13.

5. Thompson, M. W., Gilmer, J. B., Matsumoto, R. A., Quach, C. D., Shamaprasad, P., Yang, A. H., Iacovella, C. R., McCabe, C. & Cummings, P. T. Towards Molecular Simulations That Are Transparent, Reproducible, Usable by Others, and Extensible (TRUE). *Molecular Physics* **118,** e1742938. ISSN: 0026-8976, 1362-3028 (June 2020).

6. Summers, A. Z., Gilmer, J. B., Iacovella, C. R., Cummings, P. T. & McCabe, C. MoSDeF, a Python Framework Enabling Large-Scale Computational Screening of Soft Matter: Application to Chemistry-Property Relationships in Lubricating Monolayer Films. *Journal of Chemical Theory and Computation* **0.** PMID: 32004433, null. ISSN: 1549-9618. eprint: https://doi.org/10.1021/acs.jctc.9b01183 (Mar. 0).

7. Klein, C., Summers, A. Z., Thompson, M. W., Gilmer, J. B., McCabe, C., Cummings, P. T., Sallai, J. & Iacovella, C. R. Formalizing Atom-Typing and the Dissemination of Force Fields with Foyer. *Computational Materials Science* **167,** 215–227. ISSN: 0927-0256. http://www.sciencedirect.com/science/article/pii/S0927025619303040 (Sept. 2019).

8. Klein, C., Sallai, J., Jones, T. J., Iacovella, C. R., McCabe, C. & Cummings, P. T. in *Foundations of Molecular Modeling and Simulation: Select Papers from FOMMS 2015* (eds Snurr, R. Q., Adjiman, C. S. & Kofke, D. A.) 79–92 (Springer Singapore, Singapore, 2016). ISBN: 978-981-10-1128-3.

9. Braun, E., Gilmer, J., Mayes, H. B., Mobley, D. L., Monroe, J. I., Prasad, S. & Zuckerman, D. M. Best Practices for Foundations in Molecular Simulations [Article v1.0]. *Living journal of computational molecular science* **1,** 5957. ISSN: 2575-6524 (2019).

10. Sallai, J., Varga, G., Toth, S., Iacovella, C., Klein, C., McCabe, C., Ledeczi, A. & Cummings, P. T. Web- and Cloud-based Software Infrastructure for Materials Design. *Procedia Computer Science* **29,** 2034–2044. ISSN: 1877-0509 (2014).

11. *GitHub - mosdef-hub/metamds: metamds is currently read-only; see signac.io for a recommended data and workflow manager* https://github.com/mosdef-hub/metamds (2019).

12. Craven, N. C., Gilmer, J. B., Spindel, C. J., Summers, A. Z., Iacovella, C. R. & McCabe, C. Examining the Self-Assembly of Patchy Alkane-Grafted Silica Nanoparticles Using Molecular Simulation. *The Journal of Chemical Physics* **154,** 034903. ISSN: 0021-9606, 1089-7690 (Jan. 2021).

13. Quach, C. D., Gilmer, J. B., Pert, D. O., Mason-Hogans, A., Iacovella, C. R., Cummings, P. T. & McCabe, C. High-Throughput Screening of Tribological Properties of Monolayer Films Using Molecular Dynamics and Machine Learning. *The Journal of Chemical Physics,* 5.0080838. ISSN: 0021-9606, 1089-7690 (Feb. 2022).

CHAPTER 2

Best Practices for Molecular Simulation

2.1 Introduction

For the beginner and experienced practitioners of molecular simulation, there are many nuanced details of molecular simulation that might not be completely obvious, but can have drastic effects on the results of the simulation. While not a fully extensive introduction to molecular simulation overall, the work below seeks to inform the simulationist of many ideas and assumptions that are made during a molecular simulation and their subsequent impacts on the results [1][2]. This seeks to provide a useful starting point and source of reference material and general checklists that one should follow when designing and performing molecular simulation. This work serves as a resource to inform users of important details before/during a molecular simulation and best practices to ensure that their simulations are justified and that they are treating the system correctly for the property of interest they want to measure. By providing peer-reviewed best practices for molecular simulation, we hope to further improve the quality of molecular simulations as well as improve their reproducibility.

2.2 Background

Molecular simulation techniques play an important role in our quest to understand and predict the properties, structure, and function of molecular systems, and are a key tool as we seek to enable predictive molecular design. Simulation methods are useful for studying the structure and dynamics of complex systems that are too complicated for pen and paper theory, helping interpret experimental data in terms of molecular motions. Additionally, they are increasingly used for quantitative prediction of properties of use in molecular design and other applications [2–6].

The basic idea of any molecular simulation method is straightforward; a particle-based description of the system under investigation is constructed and then the system is propagated by either deterministic or probabilistic rules to generate a trajectory describing its evolution over the course of the simulation [7, 8]. Relevant properties can be calculated for each "snapshot" (a stored configuration of the system, also called a "frame") and averaged over the entire trajectory to compute estimates of desired properties.

Depending on how the system is propagated, molecular simulation methods can be divided into two main categories: Molecular Dynamics (MD) and Monte Carlo (MC). With MD methods, the equations of

[1]Portions of this chapter reprinted with permission from the following work:

[2]Braun, E., Gilmer, J., Mayes, H. B., Mobley, D. L., Monroe, J. I., Prasad, S. & Zuckerman, D. M. Best Practices for Foundations in Molecular Simulations [Article v1.0]. *Living journal of computational molecular science* **1,** 5957. ISSN: 2575-6524 (2019).

motion are numerically integrated to generate a dynamical trajectory of the system. MD simulations can be used for investigating structural, dynamic, and thermodynamic properties of the system. With MC methods, probabilistic rules are used to generate a new configuration from the present configuration and this process is repeated to generate a sequence of states that can be used to calculate structural and thermodynamic properties but not dynamical properties; indeed, MC simulations lack any concept of time. Thus, the "dynamics" produced by an MC method are not the temporal dynamics of the system, but the ensemble of configurations that reflect those that could be dynamically sampled. This foundational document will focus on the concepts needed to carry out correct MD simulations that utilize good practices. Many, but not all, of the concepts here are also useful for MC simulations and apply there as well. However, there are a number of key differences, which are outside the scope of this current document.

Either method can be carried out with different underlying physical theories to describe the particle-based model of the system under investigation. If a quantum mechanics (QM) description of matter is used, electrons are explicitly represented in the model and interaction energy is calculated by solving the electronic structure of the molecules in the system with no (or few) empirical parameters, but with various approximations to the physics for tractability. In a molecular mechanics (MM) description, molecules are represented by particles representing atoms or groups of atoms. Each atom may be assigned an electric charge and a potential energy function with a large number of empirical parameters (fitted to experiment, QM, or other data) used to calculate non-bonded and bonded interactions. Unless otherwise specified, MD simulations employ MM force fields, which calculate the forces that determine the system dynamics. MM simulations are much faster than quantum simulations, making them the methods of choice for vast majority of molecular simulation studies on biomolecular systems in the condensed phase. However, typically, they are of lower accuracy than QM simulations and cannot simulate bond rearrangements. QM simulations may be too computationally expensive to allow simulations of the time and length scales required to describe the system of interest [6]. The size of the system amenable for to QM simulation also depends on what method is chosen, from high-level ab initio methods to semi-empirical methods; discussion of these methods are outside the scope of this article, and useful references are separately available [9]. Computational resources available are also an important consideration in deciding whether QM simulations are tractable. Roughly, QM simulations might be tractable with hundreds of atoms or fewer, while MD simulations routinely have tens or hundreds of thousands of atoms in the system. Much above that level, coarse-graining methods are used. They reduce resolution and computational cost. Although many of the approaches for atomistic simulations discussed here can apply to coarse-grained simulations, such simulations are not the focus of this paper and we will not discuss how coarse-grained simulations are initially built.

Speed is a particular concern when describing condensed phase systems, as we are often interested in the properties of molecules (even biomacromolecules) in solution, meaning that systems will consist of thousands to hundreds of thousands or millions of atoms. While system size alone does not dictate a classical description, if we are interested in calculations of free energies or transport properties at finite (often laboratory) temperatures, these include entropic contributions (as further discussed below) meaning that fluctuations and correlations of motions within the system affect computed properties, meaning that simulations must not only sample single optimal states but instead must sample the correct distribution of states – requiring simulations of some length. Furthermore, many systems of interest, such as polymers (biological and otherwise) have slow motions that must be captured for accurate calculation of properties. For example, for proteins, relevant timescales span from nanoseconds to seconds or more, and even rearrangements of buried amino acid sidechains can in some cases take microseconds or more, with larger conformational changes and protein folding taking even longer [10, 11]. Recent hardware innovations have made microsecond-length simulations for biological systems of 50-100,000 atoms relatively routine, and herculean efforts have pushed the longest simulations out past the millisecond range. However, the field would like to reach even longer timescales, meaning that switching to a more detailed energy model is only done with some trepidation because slower energy evaluations mean less time available for sampling. Thus the need for speed limits the use of quantum mechanical descriptions.

Thus, for the rest of this document we will restrict ourselves to classical MD.

One other important note is that, within classical molecular simulations, bond breaking and forming is generally not allowed (with notable exceptions such as reactive force fields), meaning that the topology or chemistry of a system will remain constant as a function of time. That is, the particles comprising the system move around, but the chemical identity of each molecule in the system remains a constant over the course of the simulation (with only partial exceptions, such as the case of constant pH simulations [12]). This also means that the notion of pH in molecular simulations primarily refers to the selection of *fixed* protonation states for the components of the system.

Here, we first discuss the scope of this document, then go over some of the fundamental concepts or science topics which provide the underpinnings of molecular simulations, giving references for further reading. Then, we introduce a variety of basic simulation concepts and terminology, with links to further reading. Our goal is not to cover all topics, but to provide some guidance for the critical issues which must be considered. We also provide a checklist (Appendix A) to assist with preparing for and beginning a modeling project, highlighting some key considerations addressed in this work.

2.3 Scope of this document

There are several excellent textbooks on classical simulation methods; some we have found particularly helpful are Allen and Tildesley's "Computer Simulations of Liquids" [13], Leach's "Molecular Modelling" [8], and Frenkel and Smit's "Understanding Molecular Simulations" [7], though there are many other sources. Tuckerman's "Statistical Mechanics: Theory and Molecular Simulation" [14] may be helpful to a more advanced audience.

In principle, anyone with adequate prior knowledge (namely, undergraduate level calculus and physics) should be able to pick up one of these books and learn the required skills to perform molecular simulations, perhaps with help from a good statistical mechanics and thermodynamics book or two. In practice, due to the interdisciplinary and somewhat technical nature of this field, many newcomers may find it difficult and time consuming to understand all the methodological issues involved in a simulation study. The goal of this document is to introduce a new practitioner to some key basic concepts and bare minimum scientific knowledge required for correct execution of these methods. We also provide a basic set of "best practices" that can be used to avoid common errors, missteps and confusion in elementary molecular simulations work. This document is not meant as a full introduction to the area; rather, it is intended to help guide further study, and to provide a foundation for other more specialized best-practices documents focusing on particular simulation areas.

Modern implementations of classical simulations also rely on a large body of knowledge from the fields of computer science, programming, and numerical methods, which will not be covered in detail here.

2.4 Science topics

A new practitioner does not have to be an expert in all of the fields that provide the foundation for our simulation methods and analysis of the data produced by these methods. However, grasping some key concepts from each of these disciplines, described below, is essential. This section serves as a preface for Section 2.5 and suggestions for further reading on these subjects are provided throughout the document. In each subsection, we begin by highlighting some of the critical topics from the corresponding area, then describe what these are and why they are important to molecular simulations.

2.4.1 Classical mechanics

2.4.1.1 Key concepts

Critical concepts from classical mechanics include:

- Newton's equations of motion

- Hamilton's equations

- Point particles and rigid bodies
- Holonomic constraints

Molecular simulation methods work on many-particle systems following the rules of classical mechanics. Basic knowledge of key concepts of classical mechanics is important for understanding simulation methods. Here, we will assume you are already familiar with Newtonian mechanics.

Classical molecular models typically consist of point particles carrying mass and electric charge with bonded interactions (describing bond lengths, angles, and torsions) and non-bonded interactions (describing electrostatic and van der Waals forces). Sometimes it is much more efficient to freeze the internal degrees of freedom and treat the molecule as a rigid body where the particles do not change their relative orientation as the whole body moves; this is commonly done, for example, for rigid models of the water molecule. The timestep for simulation is determined by the fastest frequency motion. Due to the high frequency of the O-H vibrations, accurately treating water classically would require solving the equations of motion with a small timestep (commonly 1 fs). Thus, for computational efficiency water is often instead treated as a rigid body to allow a larger timestep (often double the length). Keeping specified objects rigid in a simulation involves applying holonomic constraints, where the rigidity is defined by imposing a minimal set of fixed bond lengths and angles through iterative procedures during the numerical integration of the equation of motion (see subsection 2.5.6 for more on constraints and integrators).

Classical mechanics has several mathematical formulations, namely the Newtonian, Hamiltonian and Lagrangian formulations. These formulations are physically equivalent, but for certain applications one formulation can be more appropriate than the other. Many simulation methods use the Hamiltonian formulation and therefore basic knowledge of Hamiltonian mechanics is particularly important.

Classical mechanics has several conserved quantities and simulators should be familiar with these, for example, the total energy of a system is a constant of motion. These concepts play an important role in development and proper implementation of simulation methods. For example, a particularly straightforward check of the correctness of an MD code is to test whether energy is conserved.

Most books on molecular simulations have a short discussion or appendices on classical mechanics that can serve the purpose of quick introductions to the basic concepts; Shell's book also has a chapter on simulation methods which covers some of these details [15]. A variety of good books on classical mechanics are also available and give further details on these concepts.

2.4.2 Thermodynamics

2.4.2.1 Key concepts

A variety of thermodynamic concepts are important for molecular simulations:

- Temperature and pressure
- Internal energy and enthalpy
- Gibbs and Helmholtz free energy
- Entropy

One of the main objectives of molecular simulations is to estimate/predict thermodynamic behavior of real systems as observed in the laboratory. Typically this means we are interested in macroscopic systems, consisting of 10^{23} particles or more (i.e. at least a mole of particles). But properties of interest include not only macroscopic, bulk thermodynamic properties, such as density or heat capacity, but also microscopic properties like specific free energy differences associated with, say, changes in the conformation of a molecule. For this reason, it is important to understand key concepts in thermodynamics, such as temperature, pressure, entropy, internal energy, various forms of free energy, and the relationships between them. Paramount, however, is an understanding of the connection between thermodynamics and statistical mechanics, which allows us to relate macroscopic, experimental measurements to the behavior of the much smaller system that is simulated. This topic involves a variety of subtleties and thus can be a confusing and difficult, so we refer the reader to a more extensive discussion in one of several books [15, 16].

As an example, consider temperature. In a macroscopic sense, we understand this quantity intuitively as how hot or cold something is. The laws of thermodynamics provide us with a further abstraction, telling us that this is in fact the derivative of the internal energy with respect to the entropy. This mathematical definition itself is not particularly helpful, but provides a starting point for other derivations. If we want to understand temperature from the point of view of understanding molecular behavior, we finally must turn to statistical mechanics. Since molecular dynamics is mostly used to simulate behavior at the molecular or atomistic level, it is necessary to utilize statistical-mechanical expressions in computing what would be observed as the macroscopic, thermodynamic temperature.

This discussion should not provide the impression that statistical mechanics is more important than thermodynamics. The two are intimately connected and we must rely on both to successfully conduct and obtain information from MD simulations. In particular, thermodynamics provides rigid rules that must be satisfied if we are to faithfully reproduce reality. For instance, if energy is not conserved, the first law is not satisfied and we are for sure simulating a system out of equilibrium (i.e. we are somehow adding or removing energy). In this sense, the laws of thermodynamics provide us rigorous sanity checks in addition to many useful mathematical relations for computing properties. Basic thermodynamic principles thus also dictate proper simulation protocols and associated best practices.

The concept of the thermodynamic limit is important here. Specifically, as the size of a finite system is increased, keeping the particle number density roughly constant, at some point it is said to reach the thermodynamic limit where its behavior is bulk-like and no longer depends on the extent of the system. Thus, small systems will exhibit unique behaviors that reflect their microscopic size, but sufficiently large systems are said to have reached the thermodynamic limit and macroscopic thermodynamics applied. This is due to the fact that the effect of interfaces or boundaries have largely been removed, and, more importantly, that averages of system properties are now over a sufficiently large number of molecules that any instantaneous snapshot of the system roughly corresponds to average behavior (i.e. fluctuations in properties become negligible with increasing system size).

Although we usually think of thermodynamics applying macroscopically and statistical mechanics applying on the microsopic level, it is important to remember that the laws of thermodynamics still hold *on average* regardless of the length scale. That is, a molecule in contact with a thermal bath will exchange energy with the bath, but its average energy is a well-defined constant. This allows us to define thermodynamic quantities associated with microscopic events, such as the binding of a ligand to a protein. This is useful because it allows us to assign molecular meaning to well-defined thermodynamic processes that can only be indirectly probed by experiment. Importantly, as long as we have carefully defined our ensemble and thermodynamic path, we can apply the powerful relationships of thermodynamics to more easily calculate many properties of interest. For instance, one may use molecular dynamics to efficiently numerically integrate the Clapeyron equation and construct equations of state along phase coexistence curves [17, 18].

2.4.2.2 Books

Equilibrium thermodynamics is taught in most undergraduate programs in physics, chemistry, biochemistry and various engineering disciplines. Depending on the background, the practitioner can choose one or more of the following books to either learn or refresh their basic knowledge of thermodynamics. Here are some works we find particularly helpful:

- Atkins and De Paula's "Physical Chemistry" [19], chapters 1 to 4.
- McQuarrie and Simon's extensive work, "Physical Chemistry: A Molecular Approach" [20]
- Dill's "Molecular Driving Forces" [16]

- Kittel and Kroemer's "Thermal Physics" [21]

- Shell [15]: Chapters 1-15.

2.4.3 Classical statistical mechanics

2.4.3.1 Key concepts

Key concepts from statistical mechanics are particularly important and prevalent in molecular simulations:

- Fluctuations

- Definitions of various ensembles

- Time averages and ensemble averages

- Equilibrium versus non-equilibrium

Traditional discussions of classical statistical mechanics, especially concise ones, tend to focus first or primarily on macroscopic thermodynamics and microscopic *equilibrium* behavior based on the Boltzmann factor, which tells us that configurations $\mathbf{r}^N$ occur with (relative) probability $\exp\left[-U(\mathbf{r}^N)/(k_B T)\right]$, based on potential energy function U and temperature T in absolute units. Dynamical phenomena and their connection to equilibrium tend to be treated later in discussion, if at all. However, as the laws of statistical mechanics arise naturally from dynamical equations, we will discuss dynamics first.

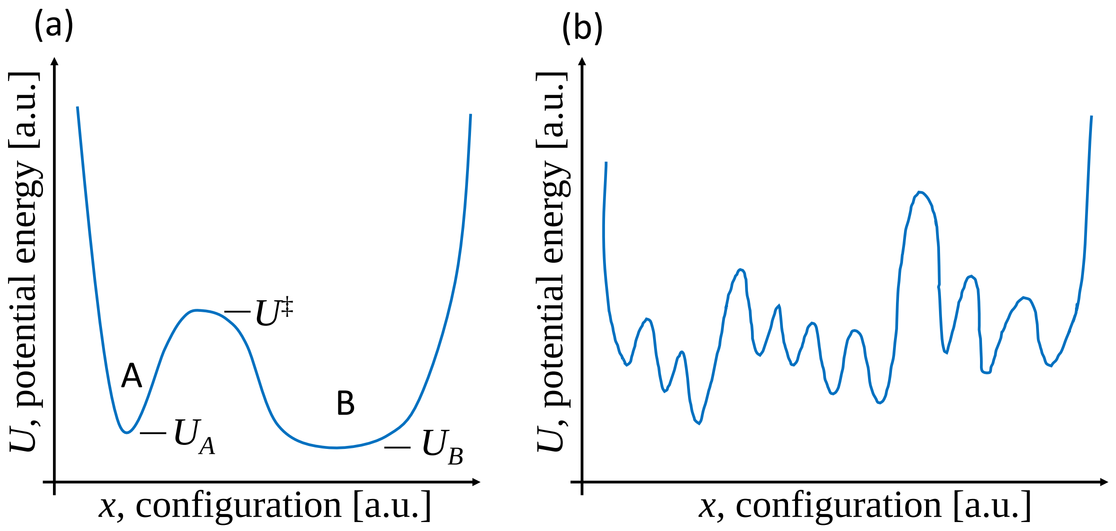

(a)
U, potential energy [a.u.]
$U^\ddagger$
A
U_A
B
U_B
x, configuration [a.u.]
(b)
U, potential energy [a.u.]
x, configuration [a.u.]

Figure 2.1: Energy landscapes. (a) A highly simplified landscape used to illustrate rate concepts and (b) a schematic of a more complex landscape with numerous minima and ambiguous state boundaries.

The key dynamical concept to understand is embodied in the twin characteristics of timescales and rates. The two are literally reciprocals of one another. In Figure 2.1(a), assume you have started an MD simulation in basin A. The trajectory is likely to remain in that basin for a period of time – the "dwell" timescale – which increases exponentially with the barrier height, $(U^{\ddagger} - U_A)$. Barriers many times the thermal energy $k_B T$ imply long dwell timescales, approximated as the reciprocal of $\exp\left[(U^{\ddagger} - U_A)/(k_B T)\right]$. The rate coefficient k_{AB} relates to the transition probability per unit time per amount of reactant(s). All transitions occur in a random, *stochastic* fashion and are predictable only in terms of average behavior. More detailed discussions of rates and rate coefficients can be found in numerous textbooks (e.g., [16, 22]).

Once you have understood that MD behavior reflects system timescales, you must set this behavior in the context of an *extremely* complex energy landscape consisting of almost innumerable minima and barriers, as schematized in Figure 2.1(b). Each small basin represents something like a different rotameric state of a protein side chain or perhaps a tiny part of the Ramachandran spaces (backbone phi-psi angles) for one or a few residues. Observing the large-scale motion of a protein then would require an MD simulation longer than the sum of all the timescales for the necessary hops, bearing in mind that numerous stochastic reversals are likely during the simulation. Because functional biomolecular timescales tend to be on μs - ms scales and beyond, it is challenging if not impossible to observe them in traditional MD simulations. There are numerous enhanced sampling approaches [23, 24] but these are beyond the scope of this discussion and they have their own challenges which often are much harder to diagnose (see [25] and https://github.com/dmzuckerman/Sampling-Uncertainty).

What is the connection between MD simulation and equilibrium? The most precise statement we can make is that an MD trajectory is a single sample of a process that is relaxing to equilibrium from the starting configuration [22, 26]. *If* the trajectory is long enough, it should sample the equilibrium distribution – where each configuration occurs with frequency proportional to its Boltzmann factor. In such a long trajectory (only), a time average thus will give the same result as a Boltzmann-factor-weighted, or ensemble, average. We refer to such a system, where the time and ensemble averages are equivialent, as "ergodic." Note that the Boltzmann-factor distribution implies that every configuration has some probability, and so it is unlikely that a single conformation or even a single basin dominates an ensemble. Beware that in a typical MD trajectory it is likely that only a small subset of basins will be sampled well – those most quickly accessible to the initial configuration. It is sometimes suggested that multiple MD trajectories starting structures can aid sampling, but unless the equilibrium distribution is known in advance, the bias from the set of starting structures is simply unknown and harder to diagnose.

A fundamental equilibrium concept that can only be sketched here is the representation of systems of enormous complexity (many thousands, even millions of atoms) in terms of just a small number of coordinates or states. The conformational free energy of a state, e.g., F_A or F_B is a way of expressing the average or summed behavior of all the Boltzmann factors contained in a state: the definition requires that the probability (or population) p^{eq} of a state in equilibrium be proportional to the Boltzmann factor of its conformational free energy: $p_A^{\text{eq}} \sim \exp(-F_A/k_BT)$. Because equilibrium behavior is caused by dynamics, there is a fundamental connection between rates and equilibrium, namely that $p_A^{\text{eq}}k_{AB} = p_B^{\text{eq}}k_{BA}$, which is a consequence of "detailed balance". There is a closely related connection for on- and off-rates with the binding equilibrium constant. For a *continuous* coordinate (e.g., the distance between two residues in a protein), the probability-determining free energy is called the "potential of mean force" (PMF); the Boltzmann factor of a PMF gives the relative probability of a given coordinate. Any kind of free energy implicitly includes *entropic* effects; in terms of an energy landscape (Figure 2.1), the entropy describes the *width* of a basin or the number of arrangements a system can have within a particular state. One way to think of this it is that entropy of a state relates to the *volume* of 6N-dimensional phase space that the state occupies, which in the one-dimensional case is just the *width*. These points are discussed in textbooks, as are the differences between free energies for different thermodynamic ensembles – e.g., A, the Helmholtz free energy, when T is constant, and G, the Gibbs free energy, when both T and pressure are constant – which are not essential to our introduction [16, 22].[3]

A final essential topic is the difference between equilibrium and non-equilibrium systems. We noted above that an MD trajectory is not likely to represent the equilibrium ensemble because the trajectory is probably too short. However, in a living cell where there is no shortage of time, biomolecules may exhibit non-equilibrium behavior for a quite different reason – because they are *driven* by the continual addition and removal of (possibly energy-carrying) substrate and product molecules. In this type of non-equilibrium situation, the distribution of configurations will not follow a Boltzmann distribution. Specialized simulation approaches are available to study such systems [24, 27] but they are not beginner-friendly. Non-equilibrium molecular concepts pertinent to cell biology have been discussed at an introductory level (e.g. http://www.physicallensonthecell.org/). Notably, many experiments are conducted at non-equilibrium conditions; for example, membrane diffusion coefficients are commonly measured by setting up a concentration gradient across the membrane and measuring the flux. It can be tempting to the beginner to setup an MD simulation in the same manner as such an experiment. However, maintaining non-equilibrium conditions is typically more complicated in an MD simulation than in an experiment as large reservoirs are commonly required. Frequently, equilibrium methods can provide the same or similar information as a non-equilibrium experiment; users should seek to obtain familiarity with such methods before choosing to conduct a non-equilibrium MD simulation.

2.4.3.2 Books

Books which we recommend as particularly helpful in this area include:

[3]Occasionally F is used to refer to either appropriate free energy, A or G, but this is not standard.

- Reif's "Fundamentals of Statistical and Thermal Physics" [28]
- McQuarrie's "Statistical Mechanics" [29]
- Dill and Bromberg's "Molecular Driving Forces" [16]
- Hill's "Statistical Mechanics: Principles and Selected Applications" [30]
- Shell's "Thermodynamics and Statistical Mechanics" [15]
- Zuckerman's "Statistical Physics of Biomolecules" [22]
- Chandler's "Introduction to Modern Statistical Mechanics" [31]

2.4.3.3 Online resources

Several online resources have been particularly helpful to people learning this area, including:

- David Kofke's notes: http://www.eng.buffalo.edu/~kofke/ce530/Lectures/lectures.html
- Scott Shell's notes: https://engineering.ucsb.edu/~shell/che210d/assignments.html

2.4.4 Classical electrostatics

2.4.4.1 Key concepts

Key concepts from classical electrostatics include

- The Coulomb interaction and its long-range nature
- Polarizability, dielectric constants, and electrostatic screening
- When and why we need lattice-sum electrostatics and similar approaches

Electrostatic interactions are both some of the longest-range interactions in molecular systems and the strongest, with the interaction (often called "Coulombic" after Coulomb's law) between charged particles falling off as $1/r$ where r is the distance separating the particles. Atom-atom interactions are thus necessarily long range compared to other interactions in these systems (which fall off as $1/r^3$ or faster). This means atoms or molecules separated by considerable distances can still have quite strong electrostatic interactions, though this also depends on the degree of shielding of the intervening medium (or its relative permittivity or dielectric constant).

The static dielectric constant of a medium, or relative permittivity ε_r (relative to that of vacuum), affects the prefactor for the decay of these long range interactions, with interactions reduced by $\frac{1}{\varepsilon_r}$. Water has a high relative permittivity or dielectric constant close to 80, whereas non-polar compounds such as n-hexane may have relative permittivities near 2 or even lower. This means that interactions in non-polar media such as non-polar solvents, or potentially even within the relatively non-polar core of a larger molecule such as a protein, are effectively much longer-range even than those in water. The dielectric constant of a medium also relates to the degree of its electrostatic response to the presence of a charge; larger dielectric constants correspond to larger responses to the presence of a nearby charge.

It turns out that atoms and molecules also have their own levels of electrostatic response; particularly, their electron distributions polarize in response to their environment, effectively giving them an internal dielectric constant. This polarization can be modeled in a variety of ways, such as (in fixed charge force fields) building in a fixed amount of polarization which is thought to be appropriate for simulations in a generic "condensed phase" or by explicitly including polarizability via QM or by building it into a simpler, classical model which includes polarizability such as via explicit atomic polarizabilities [32, 33] or via Drude oscillator-type approaches [34], where inclusion of extra particles attached to atoms allows for a type of effective polarization.

Because so many interactions in physical systems involve polarity, and thus significant long-range interactions that decay only slowly with distance, it is important to regard electrostatic interactions as fundamentally long-range interactions. Indeed, contributions to the total energy of a system from distant objects may be even more important in some cases than those from nearby objects. Specifically, since interactions between charges fall off as $1/r$, but the volume of space at a given separation distance increases as r^3, distant interactions can contribute a great deal to the energies and forces in molecular systems. In practice, this means that severe errors often result from neglecting electrostatic interactions beyond some cutoff distance [8, 35–38]. Thus, we prefer to include *all* electrostatic interactions, even out to very long ranges. Once this is decided, it leaves simulators with two main options, only one of which is really viable. First, we can simulate the actual finite (but large) system which is being studied in the lab, including its boundaries. But this is impractical, since macroscopic systems usually include far too many atoms (on the order of at least a mole or more). The remaining option, then, is to apply periodic boundary conditions (see subsection 2.5.2) to tile all of space with repeating copies of the system. Once periodic boundary conditions are set up, defining a periodic lattice, it becomes possible to include all long-range electrostatic interactions via a variety of different types of sums which can be described as "lattice sum electrostatics" or Ewald-type electrostatics [38, 39] where the periodicity is used to make possible an evaluation of all long range electrostatic interactions, including those of particles with their own periodic images.

In practice, lattice sum electrostatics introduce far fewer and less severe artifacts than do cutoff schemes, so these are used for most classical all-atom simulation algorithms at present. A variety of different efficient lattice-sum schemes are available [39]. In general these should be used whenever long range electrostatic interactions are expected to be significant; they may not be necessary in especially nonpolar systems and/or with extremely high dielectric constant solvents where electrostatic interactions are exclusively short range, but in general they should be regarded as standard (see also subsection 2.5.7, below).

2.4.4.2 Books

On classical electrostatics, we have found the undergraduate-level work by David J. Griffiths, "Introduction to Electrodynamics" [40], to be quite helpful. The graduate-level work of Jackson, "Classical Electrodynamics" [41], is also considered a classic/standard work, but may prove challenging for those without a background relatively heavy in mathematics.

2.4.5 Molecular interactions

2.4.5.1 Key concepts

Molecular simulations are, to a large extent, about molecular interactions, so these are particularly key, including:

- Bonded and nonbonded interactions
- The different types of nonbonded interactions and why they are separated in classical descriptions
- The dividing line between bonded and nonbonded interactions

Key interactions between atoms and within or between molecules are typically thought of as consisting of two main types – bonded and non-bonded interactions. While these arise from similar or related physical effects (ultimately all tracing back to QM and the basic laws of physics) they are typically treated in rather distinct manners in molecular simulations so it is important to consider the two categories separately.

Bonded interactions are those between atoms which are connected, or nearly so, and relating to the bonds connecting these atoms. In typical molecular simulations these consist of bond stretching terms, angle bending terms, and terms describing the rotation of torsional angles, as shown in Figure 2.2. Torsions typically involve four atoms and are often of two types – "proper" torsions, around bonds connecting groups of atoms, and "improper" torsions which involve neighbors of a central atom; these are often used to ensure the appropriate degree of planarity or non-planarity around a particular group (such as planarity of an aromatic ring). It is important to note that the presence of bonded interactions between atoms does not preclude their also having non-bonded interactions with one another (see discussion of exclusions and 1-4 interactions, below).

Nonbonded interactions between atoms are all interactions which are included in the potential energy of the system *aside* from bonded interactions. Commonly these include at least point-charge Coulomb electrostatic interactions and "non-polar" interactions modeled by the Lennard-Jones potential or another similar

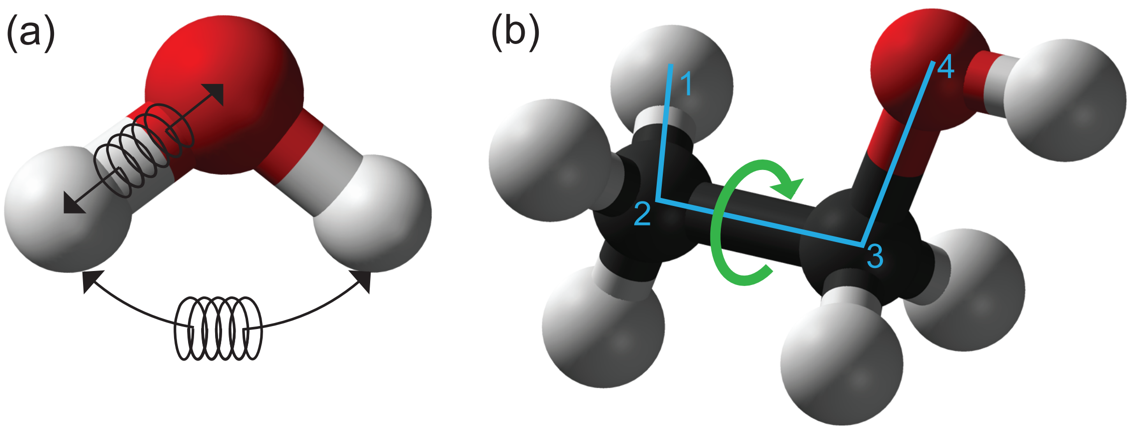


Figure 2.2: Standard MM force fields include terms that represent (a) bond and angle stretching around equilibrium values, using harmonic potentials with spring constants fit to the molecules and atoms to which they are applied; and (b) rotation around dihedral angles (green arrow) defined using four atoms, typically using a cosine expansion.

potential which describes short range repulsion and weak long-range interaction even between non-polar atoms. Additional terms may also be included, such as interactions between fixed multipoles, interactions between polarizable sites, or occasionally explicit potentials for hydrogen bonding or other specialized terms. These are particularly common in polarizable force fields such as the AMOEBA model.

Often, the energy functions used by molecular simulations explicitly neglect nonbonded interactions between atoms which are immediately bonded to one another, and atoms which are separated by only one intervening atom, partly to make it easier to ensure that these atoms have preferred geometries dictated by their defined equilibrium lengths/angles regardless of the nonbonded interactions which would otherwise be present. This neglect of especially short range nonbonded interactions between near neighbors is called "exclusion", and energy functions typically specify which interactions are excluded.

The transition to torsions, especially proper torsions, is where exclusions typically end. However, many all-atom energy functions commonly used in biomolecular simulations retain only *partial* nonbonded interactions between terminal atoms involved in a torsion. The atoms involved in a torsion, if numbered beginning with 1, would be 1, 2, 3, and 4, so the terminal atoms could be called atoms 1 and 4, and nonbonded interactions between such atoms are called "1-4 interactions". These interactions are often present but reduced, though the exact amount of reduction differs by the energy function or force field family. For example, the AMBER family force fields usually reduce 1-4 electrostatics to $\frac{1}{1.2}$ of their original value, and 1-4 Lennard-Jones interactions to $\frac{1}{2}$ of their original value. 1-4 interactions are essentially considered the borderline between the bonded and non-bonded regions. These short-range interactions can be quite strong and there is potentially a risk of them overwhelming longer-range interactions, hence their typical reduction.

2.4.5.2 Books

For a discussion of molecular interactions, we recommend "Intermolecular and surface forces" by Jacob N. Israelachvili. A variety of other books discuss these from a simulation perspective, e.g. Leach [8] and Allen and Tildesley [13].

2.5 Basic simulation concepts and terminology

Above, we covered a variety of fundamental concepts needed for understanding molecular simulations and the types of interactions and forces we seek to model; here, we shift our attention to understanding basics of how molecular simulations actually work.

2.5.1 Force fields

The term "force field" simply refers to the included terms, particular form, and specific implementation details, including parameter values, of the chosen potential energy function.[4]

Most of the terms included in potential energy functions have already been detailed in subsection 2.4.5, with the most common being Coulombic, Lennard-Jones, bond, angle, and torsional (dihedral) terms (Figure 2.2). Here, we very briefly describe the mathematical forms used to represent such interactions.

Non-bonded interactions of the Lennard-Jones form are well-described throughout the literature (for instance see Ch. 4 of Leach [8]); these model a short-range repulsion that scales as $1/r^{12}$ and a long-range attraction that scales as $1/r^6$. Coulombic interactions, including both short and long-range components, are described in detail elsewhere in this document. To represent bonded interactions, harmonic potentials are often employed. The same is true for angles between three bonded atoms, but the harmonic potential is applied with respect to the angle formed and not the distance between atoms. Torsional terms are also commonly employed, usually consisting as sums of cosines, i.e. a cosine expansion.

While the above are perhaps the most common potentials used, there are a variety of common variations as well. More exotic potentials based on three-body intermolecular orientations, or terms directly coupling bond lengths and bending angles are also possible. Some historic force fields also added an explicit (non-Coulombic) hydrogen bonding term, though these are less frequently used in many cases today. Additionally, other choices of potential function are of course acceptable, including Buckingham or Morse potentials, or the use of "improper" dihedral terms to enforce planarity of cyclic portions of molecules. This may even

[4]It is worth noting there is a occasionally a bit of ambiguity when the term "force field" is used. In some cases it is used to refer to a library of parameters that could be applied to assign an energy function to a specific molecular system via a parameterization process after applying some specific chemical perception like atom typing to that system [42]. For example, one might speak of the AMBER ff15FB [43] protein force field, which essentially provides a recipe for assigning parameters to a protein once atom types are assigned. In other cases, "force field" is used to refer to the specifics of the potential energy function after application to a specific system — what could also be called a "parameterized system". For our purposes here, the distinction between a force field library and a parameterized system is not particularly important, but it is worth noting the potential ambiguity.

include empirical corrections based on discrete binning along a particular set of degrees of freedom [44, 45], as well as applied external fields (i.e. electric fields) and force field terms describing the effect of degrees of freedom, such as solvent, that have been removed from the system via "coarse-graining." [46] For a more in-depth discussion of common (as well as less common) force field terms, see Ch. 4 of Leach [8], or for an in-depth review of those specific to simulating biomolecules, see Ponder & Case [32].

Functional forms used to describe specific terms in a potential energy function may be vastly different in mathematical character even though they seek to describe the same physics. For instance, the Lennard-Jones potential implements an r^{-12} term to represent repulsions, while an exponential form is used in the Buckingham potential. This results in very different mathematical behavior at very short distances and as a result differences in numerical implementation as well as evaluation efficiencies via a computer. For this reason, one functional form may be preferred above another due to enhanced numerical stability or simplicity of implementation, even though it is not as faithful to the underlying physics. In this regard, force field selection is a form of selecting a model – one should carefully weigh the virtues of accuracy and convenience or speed, and be ever-conscious of the limitations introduced by this decision (for instance, see Becker, Tavazza, Trautt & Buarque De Macedo [47]). It is also important to know that most MD simulation engines only support a subset of functional forms. For those forms that are supported, the user manuals of these software packages are often excellent resources for learning more about the rationale and limitations of different potential energy functions and terms (e.g. see Part II of Amber reference manuals[48] and Ch. 4 of the reference manual for GROMACS [49]).

For practical purposes, most beginning users will not be fitting a force field or choosing a functional form, but will instead be using an existing force field that already relies on a particular functional form and is available in their simulation package of choice, so for such users it is more important to know how the functional form represents the different interactions involved than to necessarily be able to justify why that particular functional form was chosen.

Many examples of force fields abound in the literature — in fact, too many to provide even a representative sample or list of citations, as most force fields are specifically developed for particular systems or categories of systems under study. However, reviews are available describing and comparing force fields for biomolecular simulations [32, 50], solid, covalently-bonded materials [51], polarizable potentials [52], and models of water [53, 54], to name just a few. Many force fields are open-source and parameter file libraries may be found through the citations in the resources above or are often distributed with molecular simulation packages. Limited databases of force fields also exist, most notably for simulations of solid materials where interatomic potentials display a much wider array of mathematical forms [55, 56].

Specification of a force field involves not just a choice of functional form, but the details of the specific parameters for all of the interacting particles which will be considered — that is, the specific parameters governing the interactions as specified by the functional form. Parameters are usually specific to certain types of atoms, bonds, molecules, etc., and include point charges on atoms if electrostatic terms are in use.

Some choices which are often considered auxiliary actually comprise part of the choice of the force field or interaction model. Specifically, settings such as the use of constraints, the treatment of cut-offs and other simulation settings affect the final energies and forces which are applied to the system. Thus, to replicate a particular force field as described previously, such settings should be matched to prior work such as the work which parameterized the force field. The choice of how to apply a cutoff, such as through direct truncation, shifting of the potential energy function, or through the use of switching functions, should be maintained if identical matches to prior work computing the properties of interest are desired. This is especially important for the purposes of free energy calculations, where the potential energy itself is recorded. However, force fields are in some cases slow to adapt to changes in protocol, so current best practices seem to suggest that lattice-sum electrostatics should be used for Coulomb electrostatics in condensed phase systems, even if the chosen force field was fitted with cutoff electrostatics, and in many cases long-range dispersion corrections should be applied to the energy and pressure to account for truncated Lennard-Jones interactions [57, 58].

For almost all force fields, many versions, variants, and modifications exist, so if you are using a literature force field or one distributed with your simulation package of choice, it is important to pay particular attention (and make note of) exactly what version you are using and how you obtained it so you will be able to accurately detail this in any subsequent publications.

As clearly described in Becker, Tavazza, Trautt & Buarque De Macedo [47], it is of paramount importance to understand the capabilities and limitations of various force field models that may seem appropriate for one's work. Depending on the physics being simulated and the computational resources at hand, no force field in the literature may provide results that accurately reproduce experiment. But with so many force fields to pick from, how is this possible? The issue lies in what is termed "transferability." Simply put, a classical description of dynamics, as implemented in MD, cannot universally describe all of chemistry and physics. At some level of finer detail, all of the potential functions described above are simply approximations. Due to this, force field developers must often make the difficult decision of sacrificing accuracy or generality. For instance, a force field may have been developed to very accurately describe a single state point, in which case it is obvious that extensive testing should be performed to ensure that it is also applicable at other conditions. Even with force fields developed to be general and transferable, it is essential to ensure that the desired level of realism is achieved, especially if applying such a model to a new system (even more caution is advised when mixing force fields!). Either way, it is always a good idea to check results against previous literature when possible. This helps ensure that the force field is being implemented properly and, though it may seem laborious on a short-time horizon, can pay substantial dividends in the long-run.

Because this balance of accuracy versus generality and transferability can be challenging, some efforts eschew transferability entirely and instead build “bespoke” force fields, where each molecule is considered as a unique entity and assigned parameters independently of any other molecule or representation of chemical space (e.g. [59]). Such approaches offer the opportunity to assign all molecules with parameters assigned in a consistent way; however, they are unsuitable for applications where speed needs to exceed that of the parameter assignment process – so, for example, for docking of a large library of potential ligands to a target receptor, if compounds must be screened at seconds or less per molecule, such approaches may not be suitable.

2.5.2 Periodic boundary conditions

Periodic boundary conditions allow more accurate estimation of bulk properties from simulations of finite, essentially nanoscale systems. More precisely, simulations of comparatively small systems with periodic boundary conditions can be a good approximation to the behavior of a small subsystem in a larger bulk phase (or at least are a much better approximation than simply simulating a nanodroplet or a finite system surrounded by vacuum). Periodic boundary conditions can alleviate many of the issues with finite size effects because each particle interacts with periodic images of particles in the same system. Clearly, though, it is undesirable for a single particle to interact with the same particle multiple times. To prevent this, a cut-off of many non-bonded interactions should be chosen that is less than half the length of the simulation box in any dimension. (However, as noted in subsection 2.4.4 these cut-offs are not normally applied to electrostatic interactions because truncating these interactions induces worse artifacts than does including interactions with multiple copies of the same particle. Instead, what are often termed “cut-offs” that are applied to electrostatics are instead a shift from short-range to long-range treatments.) Such cut-offs impose a natural lower limit to the size of a periodic simulation box, as the box must be large enough to capture all of the most significant non-bonded interactions. Further information on periodic boundary conditions and discussion of appropriate cut-offs may be found in Leach [8], sections 6.5 and 6.7 and Shell [60]’s lecture on Simulations of Bulk Phases.

It is very important to note that periodic boundary conditions are simply an approximation to bulk behavior. They DO NOT effectively simulate an infinitely sized simulation box, though they do reduce many otherwise egregious finite-size effects. This is most easily seen by imagining the placement of a solute in a periodic simulation box. The solute will be replicated in all of the surrounding periodic images. The concentration of solute is thus exactly one per the volume of the box. Although proper selection of non-bonded cutoffs will guarantee that these solutes do not directly interact (hence the common claim that such systems are at infinite dilution), they may indirectly interact through their perturbation of nearby solvent. If the solvent does not reach a bulk-like state between solutes, the simulation will still suffer from obvious finite-size effects.

Macroscopic, lab-scale systems, or bulk systems, typically consist of multiple moles of atoms/molecules and thus from a simulation perspective are effectively infinite systems. We attempt to simulate these by simulating finite and fairly small systems, and, in a sense, the very idea that the simulation cell is not infinite, but simply periodic, immediately gives rise to finite-size effects. Thus, our typical goal is not to remove these completely but to reduce these to levels that do not adversely impact the results of our simulations. Finite-size effects are particularly apparent in the electrostatic components of simulations, as these forces are inherently longer ranged than dispersion forces, as discussed in subsection 2.4.4. One should always check that unexpected long-range correlations (i.e. on the length-scale of the simulation box) do not exist in molecular structure, spatial position, or orientation. It should also be recognized that periodic boundary conditions innately change the definition of the system and the properties calculated from it. Many derivations, especially those involving transport properties, such as diffusivity [61], assume infinite and not periodic boundary conditions. The resulting differences in seemingly well-known expressions for computing properties of interest are often subtle, yet may have a large impact on results. Such considerations should be kept in mind when comparing results between simulations and with experiment.

2.5.3 Main steps of a molecular dynamics simulation

While every system studied will present unique challenges and considerations, the process of performing a molecular dynamics simulation generally follows these steps:

1. System preparation
2. Minimization/Relaxation
3. Equilibration
4. Production

Additional explanations of these steps along with procedural details specific to a given simulation package and application may be found in a variety of tutorials [62, 63]. It should be noted that these steps may be difficult to unambiguously differentiate and define in some cases. Additionally, it is assumed that prior to performing any of these steps, an appropriate amount of deliberation has been devoted to clearly defining the system and determining the appropriate simulation techniques.

2.5.3.1 System preparation

System preparation focuses on preparing the starting state of the desired system for input to an appropriate simulation package, including building a starting structure, solvating (if necessary), applying a force field, etc. Because this step differs so much depending on the composition of the system and what information is available about the starting structure, it is a step which varies a great deal depending on the nature of the system at hand and as a result may require unique tools.

Given the variable nature of system preparation, it is highly recommended that best practices documents specific to this issue and to the type of system of interest be consulted. If such documents do not exist, considerable care should be exercised to determine best practices from the literature.

Loosely speaking, system preparation can be thought of as consisting of two *logical* components which are not necessarily consecutive or separate. One comprises building the configuration of the system in the desired chemical state and the other applying force field parameters.

For building systems, freely available tools for constructing systems are available and can be a reasonable option (though their mention here should not be taken as an endorsement that they necessarily encapsulate best practices). Examples include tools for constructing specific crystal structures, proteins, and lipid membranes, such as Moltemplate [64], Packmol [65], and Atomsk [66].

A key consideration when building a system is that the starting structure ideally ought to resemble the equilibrium structure of the system at the thermodynamic state point of interest. For instance, highly energetically unfavorable configurations of the system, such as blatant atomic overlaps, should be avoided. In some sense, having a good starting structure is only a convenience to reduce equilibration times (if the force field is adequate); however, for some systems, equilibration times might otherwise be prohibitively long.

System preparation is arguably the most critical stage of a simulation and in many cases receives the least attention; if your system preparation is flawed, such flaws may prove fatal. Potentially the worst possible outcome is if the prepared system is not what you intended (e.g. it contains incorrect molecules or protonation states) but is chemically valid and well described by your force field and thus proceeds without error through the remaining steps — and in fact this is a frequent outcome of problems in system preparation. It should not be assumed that a system has been prepared correctly if it is well-behaved in subsequent equilibration steps; considerable care should be taken here.

Assignment or development of force field parameters is also critical, but is outside the scope of this work. For our purposes, we will assume you have already obtained or developed force field parameters suitable for your system of interest.

2.5.3.2 Minimization

The purpose of minimization, or relaxation, is to find a local energy minimum of the starting structure so that the molecular dynamics simulation does not immediately "blow up" (i.e. the forces on any one atom are not so large that the atoms move an unreasonable distance in a single timestep). This involves standard minimization algorithms such as steepest descent. For a more involved discussion of minimization algorithms utilized in molecular simulation, see Leach [8], sections 5.1-5.7.

2.5.3.3 Assignment of velocities

Minimization ideally takes us to a state from which we can begin numerical integration of the equations of motion without overly large displacements (see Leach [8], section 7.3.4); however, to begin a simulation, we need not just positions but also velocities. Minimization, however, provides only a final set of positions. Thus, starting velocities must be assigned; usually this is done by assigning random initial velocities to atoms in a way such that the correct Maxwell-Boltzmann distribution at the desired temperature is achieved as a starting point. The actual assignment process is typically unimportant, as the Maxwell-Boltzmann distribution will quickly arise naturally from the equations of motion. Since the momentum of the center-of-mass of the simulation box is conserved by Newtonian dynamics, this quantity is typically set to zero by removing the center-of-mass velocity from all particles after random assignment, preventing the simulation box from drifting.

In some cases, we seek to obtain multiple separate and independent simulations of different instances or realizations of a particular system to assess error, collect better statistics, or help gauge dependence of results on the starting structure. It is worth noting that even very small differences in initial configuration, such as small changes in the coordinates of a single atom, lead to exponential divergence of the time evolution of the system [13], meaning that simply running different simulations starting with different initial velocities will lead to dramatically different time evolution over long enough times. An even better way to generate independent realizations is to begin with different starting configurations, such as different conformations of the molecule(s) being simulated, as this leads to behavior which is immediately different. When rigid molecules are present, care must be taken to prevent components of velocities assigned along constraints from being forced to zero [67]. With a thermostat present, this only delays system equilibration, but for NVE simulations, such as might be used in hybrid MC/MD simulations, it can result in violations of equipartition and large subsequent errors [68].

2.5.3.4 Equilibration

Ultimately, we usually seek to run a simulation in a particular thermodynamic ensemble (e.g. the NVE or NVT ensemble) at a particular state point (e.g. target energy, temperature, and pressure) and collect data for analysis which is appropriate for those conditions and not biased depending on our starting conditions/configuration. This means that usually we need to invest simulation time in bringing the system to the appropriate state point as well as relaxing away from any artificially induced metastable starting states. In other words, we are usually interested in sampling the most relevant (or most probable) configurations in the equilibrium ensemble of interest. However, if we start in a less-stable configuration a large part of our equilibration may be the relaxation time (this may be very long for biomolecules or systems at phase equilibrium) necessary to reach the more relevant configuration space.

The most straightforward portion of equilibrium is bringing the system to the target state point. Usually, even though velocities are assigned according to the correct distribution, a thermostat will still need to add or remove heat from the system as it approaches the correct partitioning of kinetic and potential energies. For this reason, it is advised that a thermostatted simulation is performed prior to a desired production simulation, even if the production simulation will ultimately be done in the NVE ensemble. This phase of equilibration can be monitored by assessing the temperature and pressure of the system, as well as the kinetic and potential energy, to ensure these reach a steady state on average. For example, an NPT simulation is said to have equilibrated to a specific volume when the dimensions of the simulation box fluctuate around constant values with minimal drift. This definition, though not perfectly rigorous, is usually suitable for assessing the equilibration of energies, temperature, pressure, and box dimensions during equilibration simulations.

A more difficult portion of equilibration is to ensure that other properties of the system which are likely to be important are also no longer changing systematically with simulation time. At equilibrium, a system may still undergo slow fluctuations with time, especially if it has slow internal degrees of freedom – but key properties should no longer show systematic trends away from their starting structure. Thus, for example, for biomolecular simulations it is common to examine the root mean squared deviation (RMSD) of the molecules involved as a function of time, and potentially other properties like the number of hydrogen bonds between the biomolecules present and water, as these may be slower to equilibrate than system-wide properties like the temperature and pressure.

Once the kinetic and potential energies fluctuate around constant values and other key properties are no longer changing with time, the equilibration period has reached its end. In general, if any observed properties still exhibit a systematic trend with respect to simulation time (e.g. Figure 2.4) this should be taken as a sign that equilibration is not yet complete.

Depending on the target ensemble for production, the procedure for the end of equilibration is somewhat different. If an NVE simulation is desired, the thermostat may be removed and a snapshot selected that is simultaneously as close to the average kinetic and potential energies as possible. This snapshot, containing both positions and velocities may be used to then start an NVE simulation that will correspond to a temperature close to that which is desired. This is necessary due to the fact that only the average temperature is obtained through coupling to a thermostat (see subsection 2.5.4), and the temperature fluctuates with the kinetic energy at each timestep.

If the target is a simulation in the NVT ensemble at a particular density, equilibration should be done in the NPT ensemble. In this case, the system may be scaled to the desired average volume before starting a production simulation (and if rescaling is done, additional equilibration might be needed).

The schematic below (Figure 2.5) demonstrates what is generally an appropriate equilibration work-flow for common production ensembles. Clearly, this schematic cannot cover every case of interest, but should provide some idea of the general approach. For more information on equilibration procedures, see Leach [8], section 7.4 and Shell [60], lectures on Molecular dynamics and Computing properties.

2.5.3.5 Production

Once equilibration is complete, we may begin collecting data for analysis. Typically this phase is called "production". The main difference between equilibration and production is simply that in the production simulation, we plan to retain and analyze the collected data. Production must always be preceded by equilibration appropriate for the target production ensemble, and production data should never be collected immediately after a change in conditions (such as rescaling a box size, energy minimizing, or suddenly changing the temperature or pressure) except in very specific applications where this is the goal.

For bookkeeping purposes, sometimes practitioners choose to discard some initial production data as additional equilibration; usually this is simply to allow additional equilibration time after a change in protocol (such as a switch from NPT to NVT), and the usual considerations for equilibration apply in such cases (see Shell [60], lecture on Computing Properties).

Analysis of production is largely outside the scope of this work, but requires considerable care in computing observables and assessing the uncertainty in any computed properties. Usually, analysis involves computing expectation values of particular observables, and a key consideration is to obtain *converged* estimates of these properties — that is, estimates that are based on adequate simulation data so that they no longer depend substantially on the length of the simulation which was run or on its initial conditions. This is closely related to the above discussion of equilibration. Depending on the relaxation timescales involved, one may realize only after analysis of a "production" trajectory that the system was still equilibrating in some sense.

A separate Best Practices document addresses the critical issues of convergence and error analysis; we refer the reader there for more details [69] (https://github.com/dmzuckerman/Sampling-Uncertainty). For more specific details on procedures and parameters used in production simulations, see the appropriate best practices document for the system of interest.

One other key consideration in production is what data to store, and how often. Storing data especially frequently can be tempting, but utilizes a great deal of storage space and does not actually provide significant value in most situations. Particularly, observations made in MD simulations are correlated in time (e.g. see https://github.com/dmzuckerman/Sampling-Uncertainty [69]) so storing data more frequently than the autocorrelation time results in storage of essentially redundant data. Thus, storing data more frequently than intervals of the autocorrelation time is generally unnecessary. Of course, the autocorrelation time is not known *a priori* which can make it necessary to store *some* redundant data. Disk space may also be a limiting factor that dictates the frequency of storing data, and should at least be considered. Trajectory snapshots can be particularly large. However, if there are no disk space limitations it may be best to avoid discarding uncorrelated data so sampling *at* intervals of the autocorrelation time may be appropriate.

If disk space proves limiting, various strategies can be used to reduce storage use, such as storing full-precision trajectory snapshots only less frequently and storing reduced-precision ones, or snapshots for only a portion of the system, more often. However, these choices will depend on the desired analysis.

For many applications, it will likely be desirable to store energies and trajectory snapshots at the same time points in case structural analysis is needed along with analysis of energies. Since energies typically use far less space, however, these can be stored more often if desired.

2.5.4 Thermostats

Here, we discuss why thermostats, which seek to control the temperature of a simulation, are (often) needed for molecular simulations. We review background information about thermostats and how they work, introduce some popular thermostats, and highlight common issues to understand and avoid when using thermostats in MD simulations.

2.5.4.1 Thermostats seek to maintain a target temperature

As mentioned above, molecular dynamics simulations are used to observe and glean properties of interest from some system of study. In many cases, to emulate experiments done in laboratory conditions (exposed to the surroundings), sampling from the canonical (constant temperature) ensemble is desired [70]. Generally, if the temperature of the system must be maintained during the simulation, some thermostat algorithm will be employed.

2.5.4.2 Background and How They Work

The temperature of a molecular dynamics simulation is typically measured using kinetic energies as defined using the equipartition theorem: $\frac{3}{2}Nk_{\mathrm{B}}T = \left\langle \sum_{i=1}^{N} \frac{1}{2}m_i v_i^2 \right\rangle$. The angled brackets indicate that the temperature is defined as a time-averaged quantity. If we use the equipartition theorem to calculate the temperature for a single snapshot in time of a molecular dynamics simulation [8, 22] instead of time-averaging, this quantity is referred to as the instantaneous temperature. The instantaneous temperature will not always be equal to the target temperature; in fact, in the canonical ensemble, the instantaneous temperature should undergo fluctuations around the target temperature.

Thermostat algorithms work by altering the Newtonian equations of motion that are inherently microcanonical (constant energy). Thus, it is preferable that a thermostat not be used if it is desired to calculate dynamical properties such as diffusion coefficients; instead, the thermostat should be turned off after equilibrating the system to the desired temperature. However, while all thermostats give non-physical dynamics, some have been found to have little effect on the calculation of particular dynamical properties, and they are commonly used during the production simulation as well [71].

There are several ways to categorize the many thermostatting algorithms that have been developed. For example, thermostats can be either deterministic or stochastic depending on whether they use random numbers to guide the dynamics, and they can be either global or local depending on whether they are coupled to the dynamics of the full system or of a small subset. Many of the global thermostats can be made into local "massive" variants by coupling separate thermostats to each particle in the system rather than having a single thermostat for the whole system. There are also several methods employed by thermostat algorithms to control the temperature. Some thermostats operate by rescaling velocities outside of the molecular dynamics' equations of motion, e.g., velocity rescaling is conducted after particles' positions and momenta have been updated by the integrator. Others include stochastic collisions between the system and an implicit bath of particles, or they explicitly include additional degrees of freedom in the equations of motion that have the effect of an external heat bath.

2.5.4.3 Popular Thermostats

Within this section, various thermostats will be briefly explored, with a small description of their uses and possible issues that are associated with each. This is not an exhaustive study of available thermostats, but is instead a survey of just some of the more popular and historic thermostats used in MD.

1. **Gaussian**

The goal of the Gaussian thermostat is to ensure that the instantaneous temperature is exactly equal to the target temperature. This is accomplished by modifying the force calculation with the form $F = F_{interaction} + F_{constraint}$, where $F_{interaction}$ is the standard interactions calculated during the simulation and $F_{constraint}$ is a Lagrange multiplier that keeps the kinetic energy constant. The reasoning for the naming of this thermostat is due to its use of the Gaussian principle of least constraint to determine the smallest perturbative forces needed to maintain the instantaneous temperature [70]. Clearly, this thermostat does not sample the canonical distribution; it instead samples the isokinetic (constant kinetic energy) ensemble. However, the isokinetic ensemble samples the same configurational phase space as the canonical ensemble, so position-dependent (structural) equilibrium properties can be obtained equivalently with either ensemble [72]. However, velocity-dependent (dynamical) properties will not be equivalent between the ensembles. This thermostat is used only in certain advanced applications [72].

2. **Simple Velocity Rescaling**

   The simple velocity rescaling thermostat is one of the easiest thermostats to implement; however, this thermostat is also one of the most non-physical thermostats. This thermostat relies on rescaling the momenta of the particles such that the simulation's instantaneous temperature exactly matches the target temperature [70]. Similarly to the Gaussian thermosat, simple velocity rescaling aims to sample the isokinetic ensemble rather than the canonical ensemble. However, it has been shown that the simple velocity rescaling fails to properly sample the isokinetic ensemble except in the limit of extremely small timesteps [73]. Its usage can lead to simulation artifacts, so it is not recommended [73, 74].

3. **Berendsen**

   The Berendsen [75] thermostat (also known as the weak coupling thermostat) is similar to the simple velocity rescaling thermostat, but instead of rescaling velocities completely and abruptly to the target kinetic energy, it includes a relaxation term to allow the system to more slowly approach the target. Although the Berendsen thermostat allows for temperature fluctuations, it samples neither the canonical distribution nor the isokinetic distribution. Its usage can lead to simulation artifacts, so it is not recommended [73, 74].

4. **Bussi-Donadio-Parrinello (Canonical Sampling through Velocity Rescaling)**

   The Bussi [76] thermostat is similar to the simple velocity rescaling and Berendsen thermostats, but instead of rescaling to a single kinetic energy that corresponds to the target temperature, the rescaling is done to a kinetic energy that is stochastically chosen from the kinetic energy distribution dictated by the canonical ensemble. Thus, this thermostat properly samples the canonical ensemble. Similarly to the Berendsen thermostat, a user-specified time coupling parameter can be chosen to vary how abruptly the velocity rescaling takes place The choice of time coupling constant does not affect structural properties, and most dynamical properties are fairly independent of the coupling constant within a broad range [76].

5. **Andersen**

   The Andersen [77] thermostat works by selecting particles at random and having them “collide” with a heat bath by giving the particle a new velocity sampled from the Maxwell-Boltzmann distribution. The number of particles affected, the time between “collisions”, and how often it is applied to the system are possible variations of this thermostat. The Andersen thermostat does reproduce the canonical ensemble. However, it should only be used to sample structural properties, as dynamical properties can be greatly affected by the abrupt collisions.

6. **Langevin**

   The Langevin [78] thermostat supplements the microcanonical equations of motion with Brownian dynamics, thus including the viscosity and random collision effects of an implicit solvent. It uses a general equation of the form $F = F_{interaction} + F_{friction} + F_{random}$, where $F_{interaction}$ is the standard interactions calculated during the simulation, $F_{friction}$ is the damping used to tune the “viscosity” of the implicit bath, and F_{random} effectively gives random collisions with solvent molecules. The frictional and random forces are coupled through a user-specified friction damping parameter. Careful consideration must be taken when choosing this parameter; in the limit of a zero damping parameter, both frictional and random forces go to zero and the dynamics become microcanonical, and in the limit of an infinite damping parameter, the dynamics are purely Brownian.

7. **Nosé-Hoover**

   The Nosé-Hoover thermostat [70] abstracts away the thermal bath from the previous thermostats and condenses it into a single additional degree of freedom. This fictitious degree of freedom has a “mass” that can be changed to interact with the particles in the system in a predictable and reproducible way while maintaining the canonical ensemble. The choice of “mass” of the fictitious particle (which in many simulation packages is instead expressed as a time damping parameter) can be important as it affects the fluctuations that will be observed. For many reasonable choices of the mass, dynamics are well-preserved [71]. This is one of the most widely implemented and used thermostats. However, it should be noted that with small systems, ergodicity can be an issue [70, 79]. This can become

Table 2.1: Basic summary of popular thermostats. ✗ indicates that the thermostat does not fulfill the statement, ✓ indicates that the thermostat does fulfill the statement, and (✓) indicates that the thermostat fulfills the statement under certain circumstances.

Thermostat	Ensemble	Deterministic/ Stochastic	Global/ Local	Physical?	Correct Structural Properties?	Correct Dynamical Properties?
None	Microcanonical	Deterministic		✓	✓	✓
Gaussian	Isokinetic	Deterministic	Global	✗	✓	✗
Simple Velocity Rescaling	Undefined	Deterministic	Global	✗	✗	✗
Berendsen	Undefined	Deterministic	Global	✗	✗	✗
Bussi	Canonical	Stochastic	Global	✗	✓	(✓)
Andersen	Canonical	Stochastic	Local	✗	✓	✗
Langevin	Canonical	Stochastic	Local	✗	✓	✗
Nosé-Hoover	Canonical	Deterministic	Global	✗	✓	(✓)

important even in systems with larger numbers of particles if a portion of the system does not interact strongly with the remainder of the system, such as in alchemical free energy calculations when a solute or ligand is non-interacting. Martyna et al. [79] discovered that by chaining thermostats, ergodicity can be enhanced, and most implementations of this thermostat use Nosé-Hoover chains.

2.5.4.4 Summary

Table 2.1 serves as a general summary and guide for exploring the usage of various thermostats. Knowing the system you are simulating and the benefits and weaknesses to each thermostat is crucial to successfully and efficiently collect meaningful, physical data. If you are only interested in sampling structural properties such as radial distribution functions, many of the given thermostats can be used, including the Gaussian, Bussi, Andersen, Langevin, and Nosé-Hoover thermostats. If dynamical properties will be sampled, it is preferable to turn off the thermostat before beginning production cycles, but the Bussi and Nosé-Hoover thermostats (and in cases with implicit solvent, the Langevin thermostat), can often be used without overly affecting the calculation of dynamical properties. Since dynamical properties are unimportant during equilibration, faster algorithms like the Andersen or Bussi thermostats can be used, with a switch to the Nosé-Hoover thermostat for production. Overall, the Bussi thermostat has been shown to work well for most purposes, and its use is recommended as a general-purpose thermostat.

2.5.5 Barostats

Here, we discuss why barostats are used, give their background, discuss roughly how they work, describe some popular options, and summarize with some recommendations.

2.5.5.1 Motivation

Typically, thermodynamic properties of interest are measured under open-air conditions in a laboratory, which (for short timescales) means at they are measured at essentially constant temperature and pressure. To obtain a non-atmospheric pressure, some device, like a piston, inert gas, etc., would be needed to control the pressure and volume of the system [14, 60]. Such conditions correspond to what is called the isothermal-isobaric ensemble, probably one of the most popular ensembles for MD simulations. As is the case with thermostats, if the pressure must be maintained in a simulation, a barostat algorithm will be needed to sample this ensemble.

2.5.5.2 Background and How They Work

Barostat algorithms control pressure alone, not temperature, so if the target ensemble is isothermal-isobaric, they must be applied with a thermostat. If a barostat is applied without a thermostat, only the number of particles (N), the pressure (P), and the enthalpy (H) of the system are held constant. This is known as the isoenthalpic-isobaric ensemble (NPH). To sample from the isothermal-isobaric ensemble (NPT), a thermostating algorithm like the ones discussed earlier must also be applied.

Much of the background information on barostats is analogous to thermostats. The pressure of a molecular dynamics simulation is commonly measured using the virial theorem (an expectation value relating to positions and forces) [8, 60]. When pairwise interactions and periodic boundary conditions are considered, different approaches are often utilized [13, 14, 60]. Regardless, these formulas give pressure as a time-averaged quantity, similar to the temperature. If we use these formulas to calculate the pressure for a single snapshot, this quantity is referred to as the instantaneous pressure. The instantaneous pressure will not always be equal to the target pressure; in fact, in the NPH and NPT ensembles, the instantaneous pressure should undergo fluctuations around the target pressure.

For the purpose of molecular modeling, consider a hypothetical system that is being compressed and/or expanded by a fictitious piston that has some mass which acts in all directions uniformly. Since the piston is acting on the system from all directions, it can be considered as applying a uniform compression or expansion. The mass of the piston can be tuned to change the compression of the system, which will change how often the particles in the system will interact with the system enclosure. These impacts from the particles on the "enclosure" will impart a stress on the system box from the surroundings and serve as a type of barostat.

The next section will describe the main differences between the many barostats that are available, and give some recommendations for proper use. Some barostats work based on scaling or rescaling the coordinates in the system (the volume and the center-of-mass coordinates of the molecules involved), whereas others work by modifying the equations of motion to ensure constant pressure.

2.5.5.3 Popular Barostats

Here, we introduce a few notable barostats and give a high-level summary of each, noting some key issues. This is not an exhaustive list of barostats and barostat algorithms, just a sampling of popular and historic ones used in MD.

1. **Simple volume rescaling**

   Every time this barostat is executed, the volume of the system is modified such that the instantaneous pressure is exactly equal to the target pressure. This does **not** sample the proper ensemble and thus cannot be used for production sampling [60]. This also does not smoothly approach the target pressure either, which might cause very unphysical issues with the system during integration.

2. **Berendsen**

   The Berendesen [75] weak coupling barostat is very similar to the Berendsen thermostat discussed earlier. It seeks to improve upon the simple volume rescaling method mentioned above. This is achieved by coupling the system to a weakly interacting pressure bath [75]. This bath scales the volume periodically by a scaling factor, which produces more realisitc fluctuations in the pressure as it slowly approaches the target pressure. In contrast to volume rescaling, Berendsen will approach the target pressure more realistically, but the ensemble it is sampling from is not well defined and cannot be guaranteed to be NPT or NPH. Berendsen can be useful for the beginning stages of equilibration, but should **not** be used for production sampling.

3. **Andersen**

   First described by Andersen [77] in 1980, the system is coupled to a fictitious pressure bath, by adding an additional degree of freedom to the equations of motion. This behaves as if the system is being acted upon by an isotropic piston. This is similar to the Nosé-Hoover thermostat, which is also an extended system algorithm. This barostat does sample the correct ensemble. However, it is isotropic in nature and applying anisotropic pressures to parts of the system is not possible.

4. **Parrinello-Rahman**

   The Parrinello-Rahman [80] barostat is an extension to the Andersen barostat. Unlike the Andersen barostat, Parrinello-Rahman supports the anisotropic scaling of the size and shape of the simulation box [80]. This can be quite useful in solid simulations, where phase changes can be shape changes in a crystal lattice, compared to a liquid or gas, which has no well defined shape. This barostat has essentially the same properties as the Andersen one, with the additional support anisotropy.

5. **Martyna-Tuckerman-Tobias-Klein (MTTK)**

   The MTTK barostat has substantial similarity to the Parrinello-Rahman and Andersen barostats. When Parrinello-Rahman's equations of motion were discovered to hold true only in the limit of large systems, the MTTK barostat introduced alternate equations of motion to correctly sample the ensemble for smaller systems as well [81, 82]. Thus, MTTK [81, 82] is usually seen as an improvement over Parrinello-Rahman [80] for such systems.

6. **Monte Carlo**

   Constant pressure may also be achieved by periodically performing Monte Carlo moves that adjust the system volume. For an explanation of how such moves are accepted or rejected, see "Monte Carlo simulations in other ensembles" in Shell [60]. These MC barostats are computationally advantageous in that the virial need not be computed, and they may be easily extended to accommodate anisotropic systems. They rigorously explore the correct distribution of volumes in the NPT ensemble. However, they do not preserve dynamic fluctuations. Unlike for extended system barostats, there is no sense of relaxation time over which the volume of the system responds. Instead, the rate at which the volume may respond is limited by the frequency with which MC moves are performed and the maximum allowed change in volume. Thus, long-time dynamics are not accurately reproduced in any sense for MC barostats.

2.5.5.4 Summary

The simple volume rescaling and Berendsen barostats are not recommended for collection of production data, as they do not sample from any correct ensemble, nor do they utilize any "realistic" approach to achieve the target pressure. They can, however, be used for approaching the target pressure. The Berendsen barostat acts in a more realistic fashion in this regard compared to the volume rescaling barostat, which itself is primarily useful only as a very stable thermostat for very early simulation stages if other algorithms have trouble beginning from particularly strained starting structures. (Alternatively, such issues can be avoided by running NVT equilibration before using a barostat, Figure 2.5.) Extended ensemble barostats are suitable for the production runs of most systems. It is usually not recommended to use these for the equilibration process, as these barostats do not behave as well when not near the target pressure. These can be affected by the starting configuration and pressure values much more than the Berendsen or simple volume rescaling barostats. MTTK and Parinello-Rahman allow for more flexibility in terms of the shape modulation of the simulation box. However, not all extended-ensemble barostats have been implemented all simulation engines, limiting user choice. It is recommended to begin with the Berendsen barostat to quickly bring the system to the target pressure, and then switch to an extended ensemble barostat for final equilibration and production.

2.5.6 Integrators

For systems consisting of more than three interacting bodies with no constrained degrees of freedom, there is no analytical solution to the equations of motion. Instead, we must approximate the dynamics in a discrete manner. This is usually termed numerical integration of the equations of motion. Algorithms to perform this integration take many forms and are usually called integrators. Here, we explain the need for integrators, discuss key criteria like energy conservation, and highlight a number of commonly used integrators.

2.5.6.1 Desirable integrator properties

So-called "good" integrators contain certain features that are appealing for molecular simulations. We start with the most obvious feature, which is that the integrator induces little error in the dynamics. Since integration is fundamentally about taking discrete steps to approximate continuous dynamics, this discretization process introduces errors (as can be observed by comparison to analytically soluble problems, like the harmonic oscillator). These errors are termed discretization errors, whereas additional errors called truncation errors are also accumulated through loss of precision during computer calculations. As will be discussed shortly, there are many strategies for avoiding discretization errors. For truncation errors, the only solution is to utilize a higher precision data type during calculations (i.e. use doubles instead of floats).

Integrators that minimize discretization error should preserve phase-space volume and conserve energy. If phase space volume is not preserved, then the sampled ensemble at a later timestep will not be the same as that in which the system was initialized. This means that the collected data will not in fact reflect the ensemble of interest. Luckily, this issue may be avoided simply by guaranteeing that the integrator is reversible [7]. More details may be found in Tuckerman, Berne & Martyna [83], but basically if the mathematical operator representing the integrator preserves phase space volume, it also satisfies the definition of reversibility: if the operator is applied to propagate forward by Δt, the starting condition may be recovered by in turn applying the operator to the result using $-\Delta t$ as the timestep.

Energy conservation is also a desirable integrator property and is imperative in simulating the microcanonical (NVE) ensemble. This is a much trickier property to examine, and varies with different integrators. For instance, some classes of integrators better-preserve energy over short times, while others better-preserve energy at long times. The latter is generally preferred, though it may necessitate other sacrifices such as greater energy fluctuations away from the desired, exact system energy. When the energy does change over the course of a simulation, it is said to "drift." The most common reason for energy drift is due to a timestep that is overly long. If the timestep is much too long, the system can become unstable and blow up (energies become very large) due to overlap of atoms. Even when the timestep is long enough that the system is still stable over long times, it may be too long for the chosen integrator to conserve energy. Other simulation parameters may also impact energy drift, such as the method of truncating forces and energies, as well as the choice of numerical precision. The latter effect, due to truncation errors, will become obvious if two simulations with different timesteps are compared. Shorter timesteps, and hence more steps to achieve a simulation of the same length, will result in *more* drift, since errors get larger with the number of calculations performed by the computer. This is exactly opposite to the behavior that is expected for poor energy conservation associated with discretization error, where a shorter timestep will reduce energy drift.

Overall, then, integrators do exhibit energy fluctuations that are timestep-dependent. All Verlet-equivalent integrators exhibit energy fluctuations which decrease with the square of the timestep [13], which is often an important check when assessing the correctness of an implementation. Thus, both energy drift and energy fluctuations are important criteria to understand when assessing integrators, and can be useful measures of simulation quality in the NVE ensemble.

Additionally, it is also desirable that an integrator be computationally efficient. Integrator cost mostly appears in the length of the timestep that may be taken while still avoiding discretization error. As discussed further below, the timestep must be at least an order of magnitude less than the smallest timescale of motion present in the system. However, depending on the accuracy of the integrator with respect to reproducing the true dynamics, a smaller timestep might be necessary. If the integrator requires a very small timestep to avoid discretization error, then the computational cost greatly increases. Hence, a truly "good" integrator allows for long timesteps while still achieving low discretization error. This has the added benefit of also reducing truncation error, which is proportional to the number of timesteps taken. It is worth noting that the issue of integrator choice versus timestep is not always simple; in some cases, a "better" integrator might allow longer timesteps but also carry an additional computational cost that outweighs the benefits of an increased timestep.

2.5.6.2 Deterministic integrators

The most commonly used integrators are variants of the Verlet algorithm (e.g. Velocity Verlet or Leapfrog). Such integrators include terms for updating particle positions up to the order of the square of the timestep (i.e. they include forces). Inclusion of higher-order terms is favored in other families of algorithms, but generally leads to greater complexity and reduced computational efficiency at only marginal improvement in accuracy. Detailed discussion and derivation of many common integrators may be found in section 7.3 of Leach [8] and 4.3 of Frenkel & Smit [7]. Such integrators are not applicable, however, for simulations involving stochastic dynamics, as discussed below.

2.5.6.3 Stochastic integrators

Stochastic dynamics simulations include application of a random force to each particle, and represent discretizations of either Langevin or Brownian dynamics. A detailed description of such stochastic dynamics may be found in McQuarrie [29], Chapter 20. As detailed in subsection 2.5.4, it is common to apply temperature control through the use of Langevin dynamics. As a brief aside, this highlights the fact that the choice of integrator is often tightly coupled to the choice of thermostat and/or barostat. Different combinations may demonstrate better performance and for expanded ensemble methods it is necessary to utilize an integrator specific to the selected temperature- or pressure-control algorithm.

With Langevin or other stochastic dynamics, the random forces usually prevent the integrator from preserving phase-space volume, which ends up dictating the choice of timestep. Specifically, despite issues with phase-space volume, some stochastic integration schemes achieve preservation of *part* of the full phase-space (i.e. configurations *or* velocities are preserved) [84] via cancellation of error. In practice these issues are easily remedied through an appropriate choice of timestep depending on the integration scheme.

Stochastic dynamics necessarily perturbs dynamics. Specifically, with Langevin or Brownian dynamics, calculations of any dynamic properties with longer timescales than the application of the random forces will be very different than those from deterministic trajectories. If one is interested in only configurational or thermodynamic properties of the system, this is of no consequence. If dynamics are of interest, the dependence of these properties on the integrator parameters (e.g. friction factor) should be assessed [71].

2.5.6.4 Choosing an appropriate timestep

The maximum timestep for a molecular dynamics simulation is dependent on the choice of integrator and the assumptions used in the integrator's derivation. For the commonly used second order integrators, such as the Verlet and Leapfrog algorithms, the velocities and accelerations should be approximately constant over the timestep. Thus, the timestep is limited by the highest frequency motion present in the system, which for all-atom simulations is usually bond vibrations. It is commonly found that using a timestep that is one tenth of this vibration's characteristic period is sufficient to conserve energy in the microcanonical ensemble. For example, if hydrogen molecules are present in the simulation box and the H-H bond vibration is the highest-frequency motion in the system with its force field harmonic force constant set to 500 N/m, the oscillation period can be calculated using the equation for simple harmonic motion ($T = 2\pi\sqrt{\frac{\mu}{k}}$, where μ is the reduced mass and k is the force constant) to be 8 fs; thus, a 0.5 fs timestep can be used. As another example, if an ab initio MD simulation is being conducted in which C-H bond vibrations are known to be the highest-frequency motion, infrared spectra can be consulted to find that this bond vibration frequency will be approximately 3000 cm^{-1}, which is 11 fs; thus, either a 0.5 or 1.0 fs timestep would be recommended. For all-atom simulations with constraints on the high-frequency bonds, timesteps can be commonly increased to 2 fs; coarse-grained simulations with particles of higher mass and smaller force constants can have much larger timesteps. After choosing a timestep, a test simulation should be run in the microcanonical ensemble to ensure that the choice of timestep yields dynamics that conserve energy. The timestep should also be short enough that properties calculated from the simulation, regardless of ensemble, are independent of the chosen timestep. This is because an inappropriately large timetep can lead to subtle changes to the ensemble being simulated [8, 13] and alter computed thermodynamic and transport properties, especially in stochastic simulations or those coupled to thermostats or barostats [84]. Methods also exist to increase the timestep beyond the limit imposed by the system's highest-frequency motion. Some examples of these enhanced timestepping algorithms include multiple-timestep methods which separately integrate high-frequency motion from low-frequency motion and schemes which repartition atomic masses to decrease the highest-frequency motion seen in the system[85, 86].

2.5.7 Long range electrostatics

In view of the long-range nature of Coulombic interactions (subsection 2.4.4), handling of electrostatics is particularly important in many systems. Here we describe the motivation for the different treatments of these terms, and give an overview of the core idea of the basic algorithms typically employed.

2.5.7.1 Motivation

The calculation of non-bonded interactions is generally the most time-consuming step of classical energy calculation. While the number of type of bonded interactions remain unchanged during an MD simulation, the strength and importance of non-bonded interactions varies substantially as a simulation proceeds.

Additionally, Coulombic interactions fall off only very slowly with distance, as r^{-1}, further complicating handling of non-bonded interactions in two different ways. First, calculating all Coulomb interactions over a periodic system results in needing to compute a sum which is conditionally convergent — that is, the value of the sum depends on the order *in which it is evaluated* [8], meaning we must exercise extreme care or the result will be ambiguous. Second, long-range interactions may be relevant, but determining pairwise distances is an expensive computation that grows with the square of the number of atoms involved.

As discussed in subsection 2.5.2, simulations designed to represent bulk systems are generally performed under periodic boundary conditions, so that the electrostatic potential at any point is due to all the other charges in the system including all of their periodic copies. Given that this is the goal, a set of different methods have been developed to efficiently compute the electrostatic potential due to this infinite, periodic system.

In the early days of simulations, electrostatic interactions were often simply truncated at a particular cutoff radius (r_c). This, however, creates artificial boundary effects and other problems [13], as well as neglecting important long-range interactions.

2.5.7.2 Ewald Summation

The Ewald summation technique [87] provides one way to efficiently handle long-range electrostatics in periodic systems. To understand this technique, consider the relationship between the charge distribution and the Coulombic potential written in the differential form (the Poisson equation):

$$\nabla^2 \phi(x) = -\frac{1}{\varepsilon}\rho(x)$$

where $\phi(x)$ is the potential at point x, $\rho(x)$ is the charge density at point x and ε is the permittivity of the medium. The standard way to determine the potential from this equation is to first discretize the equation and then solve, but this requires the functions ρ and ϕ to be smooth. However, here, because we use point charge electrostatics, ρ is a set of delta functions.

The Ewald method is based on (temporarily) replacing the point charge distributions by smooth charge distributions in order to apply existing numerical techniques to solve this partial differential equation (PDE). The most common smooth function used in the Ewald method is the Gaussian distribution, although other distributions have been used as well. Thus the overall charge distribution is divided into a short-range or "direct space" component (ρ^{sr}) involving the original point charges screened by the Gaussian-distributed charge of the same magnitude (Figure 2.6) but opposite sign, and a long-range component involving Gaussian-distributed charges of the original sign (ρ^{lr}). The screening distribution is of opposite sign to allow the screened interactions to fall off rapidly with distance, as we will see below. The sum of the short-range ρ^{sr} and the long-range ρ^{lr} charge distributions is still the same as the original charge distribution.

Unlike the original, full potential, the direct space screened interaction (Figure 2.6, top) decays rapidly. In fact, it decays even faster than Van der Waals interactions ($1/r^6$) and hence relative short cutoffs, comparable to those used for Van der Waals interactions, can be used for handling direct-space Coulomb interactions (Figure 2.7).

The potential due to long-range charge interactions does not decay rapidly, and thus requires consideration of all periodic copies. This would pose severe problems if calculated via direct summation, but the smoothness of the charge ρ^{lr} (and hence potential (ϕ^{lr}) allows the use of fast PDE solvers. Specifically, while the sum is long-ranged in real space, taking the Fourier transform converts it into a sum in reciprocal space which is short-ranged in reciprocal space, damped by a factor $\exp\left(-k^2\sigma^2/2\right)$ where k is the reciprocal space vector and σ is the width of the Gaussian.

The final term in Ewald summation is a so-called self term which gets subtracted out of the overall sum; it is calculated only once at the beginning of the simulation as it depends only on the charge magnitudes and not their positions. It also does not contribute to the force.

2.5.7.3 Grid based Ewald summation

Ewald summation as described in the previous section takes $O(n^{3/2})$ time, where n is the number of charge sites. Switching to a discrete Fourier transform can reduce the cost to to $O(nlog(n))$. Discretization involves spreading the charge over a grid. Several common grid-based implementations are available which tackle this problem, including Particle-Particle Particle Mesh (P3M), Particle Mesh Ewald (PME) and Smooth Particle Mesh Ewald (SPME). Specifics are chosen in each case to combine accuracy, speed and ease of implementation. In this subsection, we give an overview of the grid-based approach.

Grid-based Ewald summation approaches involve five general steps:

1. Charge assignment: In this step, charges are interpolated onto the grid. While the original PME method uses Lagrangian interpolation for charge assignment, the SPME method uses the smoother cardinal B-splines (hence the name Smooth-PME) to distribute charge onto the grid.

2. Transformation of the grid to reciprocal space: A Fast Fourier Transform (FFT) is used to convert the charges on the grid to their equivalent Fourier space structure factors.

3. Energy calculation: The reciprocal space potential is calculated by solving the Poisson equation in Fourier space, and the reciprocal space potential is then stored on the grid.

4. Transformation of the grid back to real space: An Inverse FFT is used to convert the reciprocal space potential back to the real space.

5. Force calculation: The force is given by the gradient of the potential. PME [36], SPME [88] and P3M [89] use different methods for calculating the force given the resulting potential.

Optimizing the performance of grid based methods can be somewhat challenging; many key choices are made in method implementation and only relative few settings are exposed to the user. Some typical options include:

- Grid dimensions or spacing: A fine grid would be slower, requiring interpolation and calculations for more grid points, but in principle accuracy should be higher.

- Screening function: The width of the Gaussian screening function can often be tuned, but the ideal width is coupled with the direct space cutoff giving limited room for tuning. In some cases alternate, non-Gaussian screening functions are available.

- Direct-space cutoff: This is typically kept at or near the value used for the van der Waals cutoff. Decreasing the cutoff improves the direct space performance but increases the complexity of the reciprocal space calculations.

In principle, it is possible to optimize settings for handling of long-range electrostatics in order to achieve considerable efficiency gains while maintaining accuracy, though this can involve considerable care [90]. For novice users, we suggest typically using well-validated or default settings for the particular method employed, and only deviating from these with careful consideration and testing.

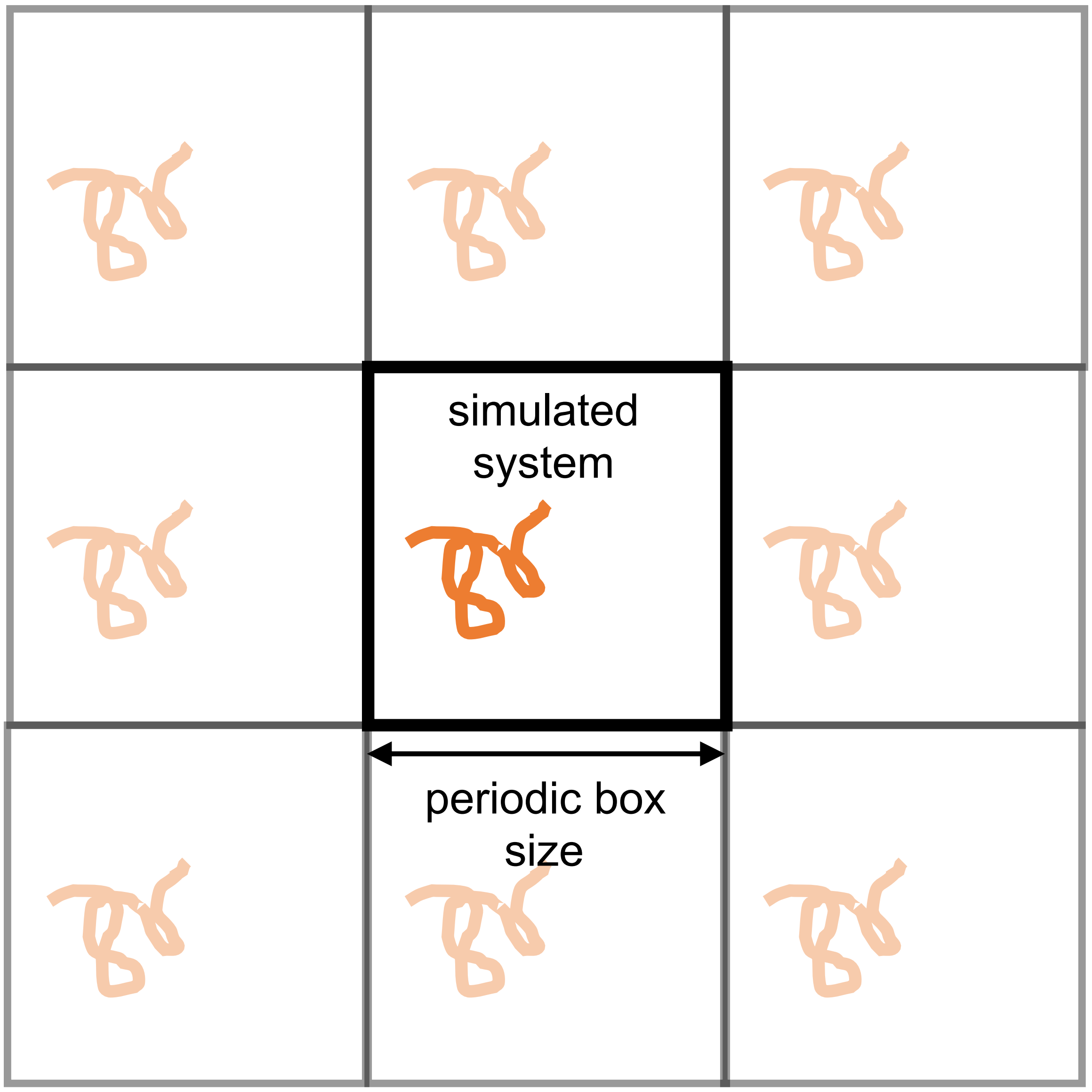


Figure 2.3: Periodic boundary conditions are shown for a simple 2D system. Note that the simulated system is a sub-ensemble within an infinite system of identical, small ensembles.

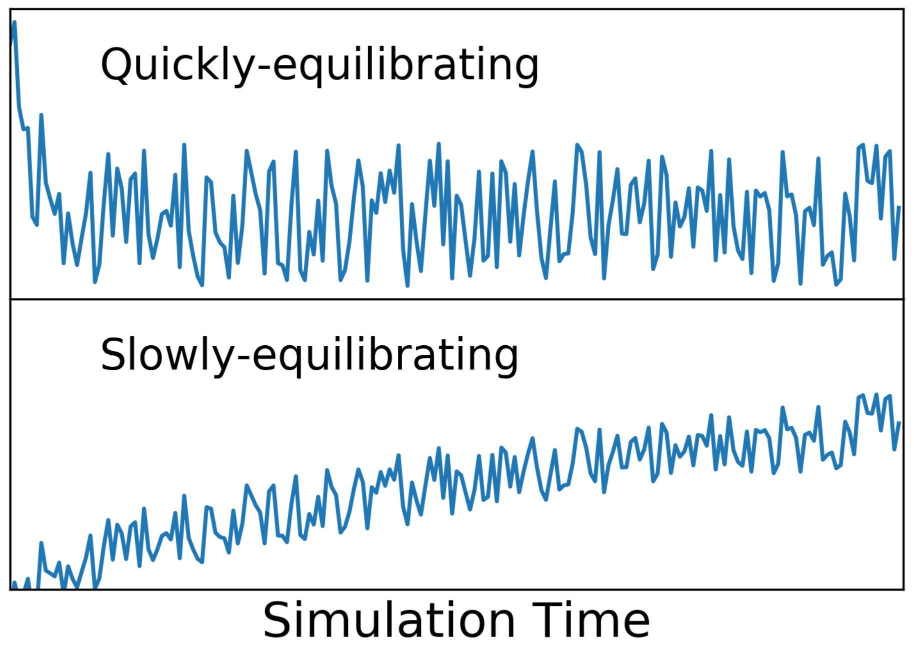


Figure 2.4: Shown are graphs of a hypothetical computed property (vertical axis) versus simulation time (horizontal axis). For some system properties, equilibration may be relatively rapid (top panel), while for others it may be much slower (bottom panel). If it there is ambiguity as to whether or not a key property is still systematically changing, as in the bottom panel, equilibration should be extended.

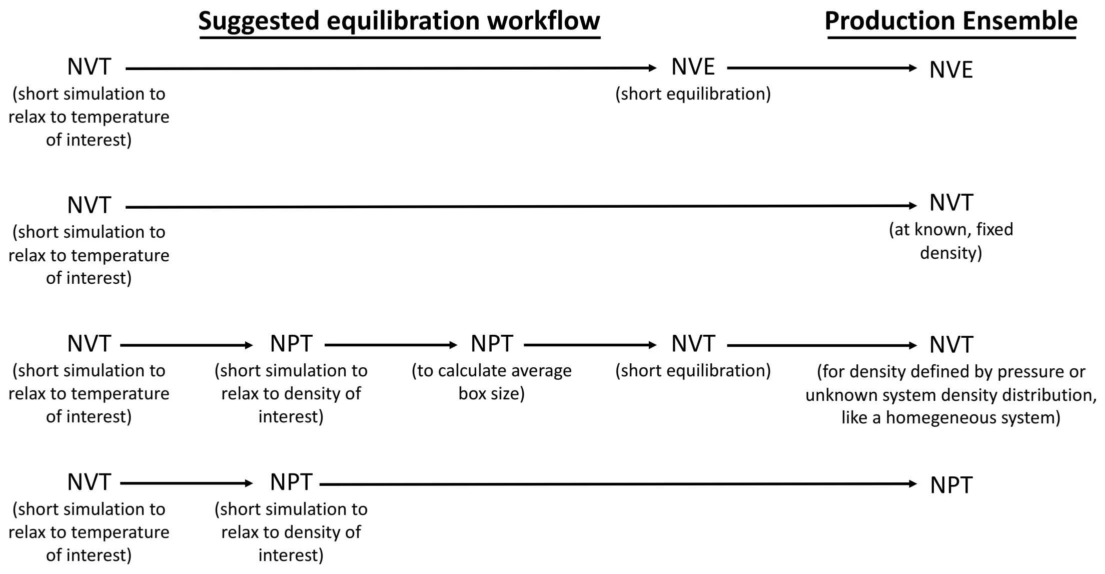

Suggested equilibration workflow
Production Ensemble
NVT
(short simulation to relax to temperature of interest)
NVE
(short equilibration)
NVE
NVT
(short simulation to relax to temperature of interest)
NVT
(at known, fixed density)
NVT
(short simulation to relax to temperature of interest)
NPT
(short simulation to relax to density of interest)
NPT
(to calculate average box size)
NVT
(short equilibration)
NVT
(for density defined by pressure or unknown system density distribution, like a homegeneous system)
NVT
(short simulation to relax to temperature of interest)
NPT
(short simulation to relax to density of interest)
NPT

Figure 2.5: Common equilibration work-flows are shown; these vary depending on the target ensemble for production simulations (right). Typically, an initial phase of equilibration at constant volume and temperature is needed to bring the system to the desired target temperature or energy. For stability reasons, this initial phase is usually needed even if the goal is to also bring the system to a target pressure. If the production ensemble is an NVE ensemble, an initial NVT simulation is usually followed by a short additional NVE equilibration before collection of production data. If the production ensemble is NVT, protocols may differ depending on whether it is necessary to allow the system to equilibrate to a particular density/volume or whether the volume is selected *a priori* (second and third rows). And if production is to be NPT, it is usually equilibrated first at NVT before equilibrating to the target pressure (final row).

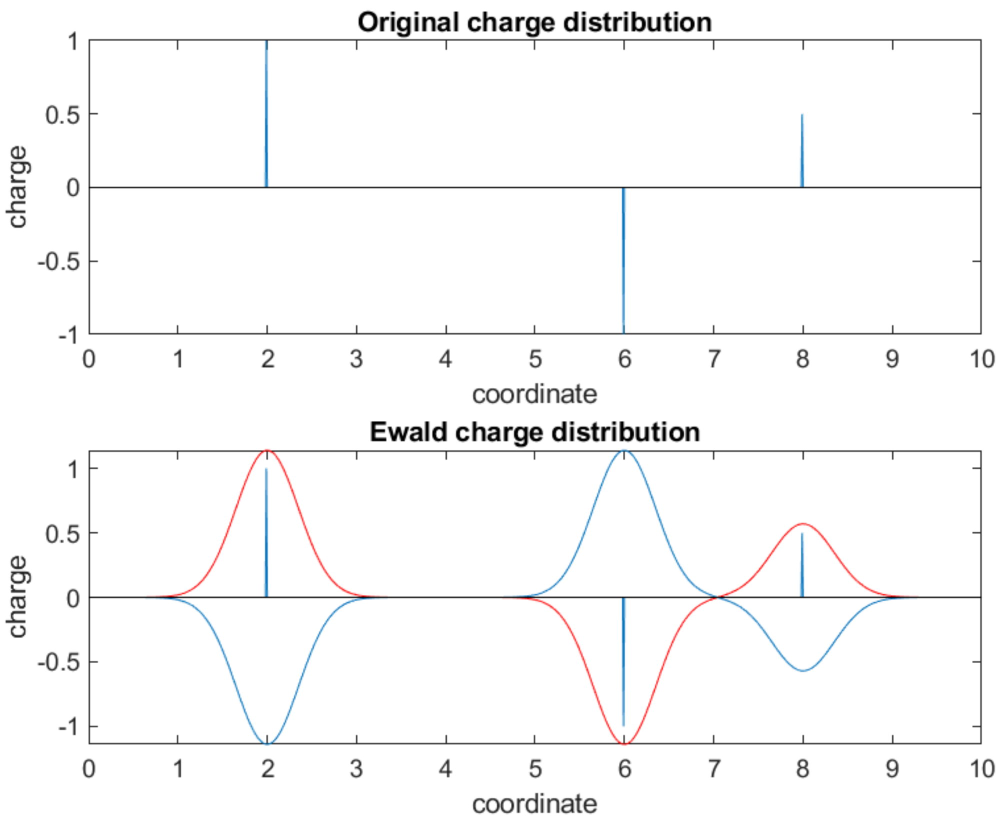


Figure 2.6: Screening charge distribution. (Top) The original charge distribution. (Bottom) Point charges can be split into Direct space (blue) and Reciprocal space charges (red). The direct space charge consists of the original charges and Gaussian-distributed screening charges of opposite sign. The reciprocal space charge is only the Gaussian-distributed charge of the original sign. Together these sum to the original charge distribution, but computation of the electrostatic potential due to each component becomes much easier.

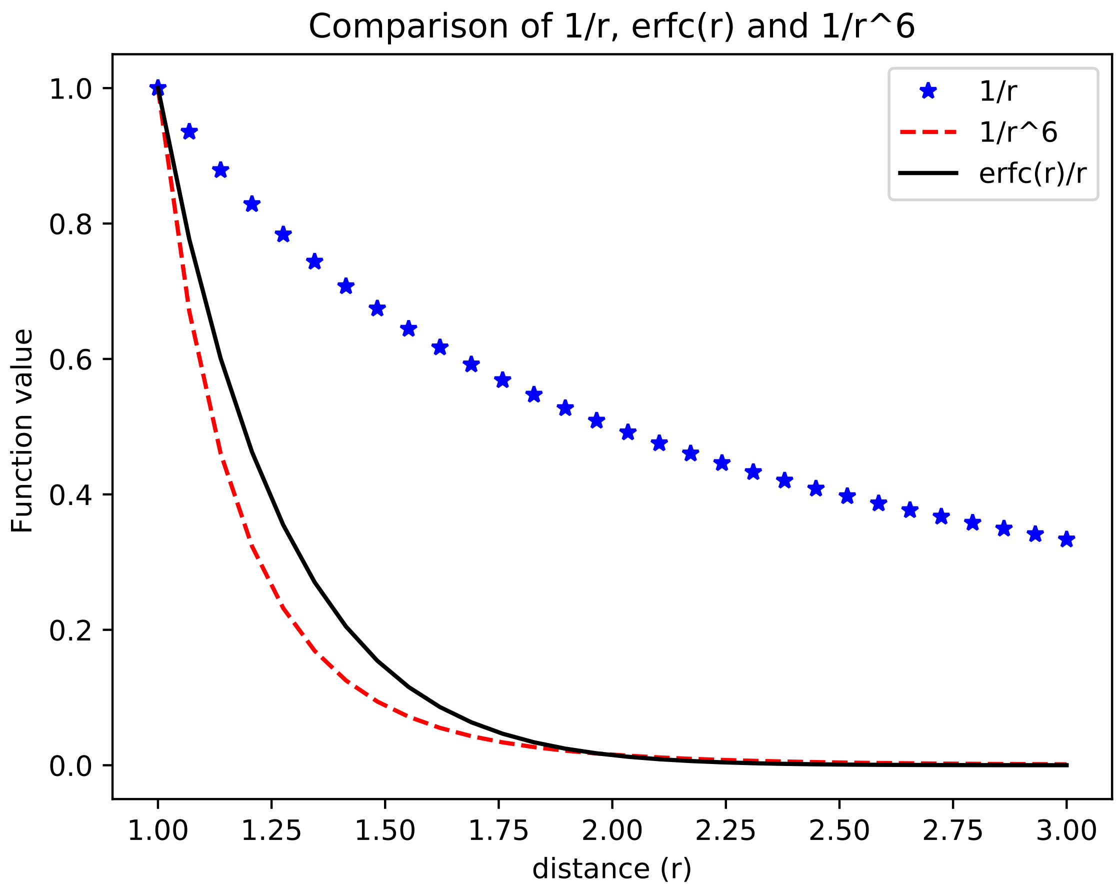


Figure 2.7: Comparison of decay of the original r^{-1} term for Coulomb interactions (blue,*), the resulting direct-space term after Gaussian screening (black,-) and the r^{-6} in van der Waals term (red, -.). Note: The value on the vertical axis has been scaled to allow easy visualization of the relative decay of each term.

2.6 Should you run MD?

A critical question *before* preparing an MD simulation of your system is whether you even *should* use MD for your system in view of the resources you have and what information you hope to obtain. MD is a tool, but it may not be the right tool for your problem. Before beginning any study, it is critical to sort out what questions you want to answer, what resources (computational and otherwise) you have at your disposal, and whether you have any information about your system(s) of interest that indicate you can realistically expect to answer those questions given a set of MD simulations. Try to understand basic concepts of statistical uncertainty ([91] and https://github.com/dmzuckerman/Sampling-Uncertainty [69]) and use these to make an educated guess regarding your chances of extracting pertinent and reliable information from your simulation.

As noted above, the frequency of the fastest vibrational motions in a system of interest sets a fundamental limit on the timestep which, given fixed computational resources, sets a limit on how much simulation time can be covered with any reasonable amount of computer time. Thus, as noted in Section 2.2, the longest all-atom MD simulations are on the microsecond to millisecond timescale. This means that if your system is complex and answering your questions will require sampling critical events that have a timescale of seconds or longer, MD will not be the right tool for the job. You could invest a huge amount of effort running MD simulations and find that they did not address your questions.

Ideally, you should have some information before beginning that the relevant timescales for your system might be accessible given the amount of MD you can afford to run, or you could plan a set of exploratory MD simulations to assess feasibility. But do not simply plunge in and attempt to run simulations until you find the answers to your questions, as the required timescales could end up being orders of magnitude longer than what you can afford to spend. Time is only one consideration out of many; disk storage and computer time availability can also prove limiting factors, and opportunity cost, as well, is not to be overlooked.

Ultimately, we ought to be assessing whether MD is the best tool for the job. For our problem of interest, will it really be faster to answer your questions using an MD simulation, or are there experiments which could be done which would answer the question more quickly? Maslow famously wrote, “I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail.” We do not want to end up in a position where we are running MD simulations not because they are the best tool for the job, but because they are the only tool available to us. If an experiment could answer our key questions with far less cost and time, and the questions are indeed compelling, perhaps our time might be better spent finding a suitable experimental collaborator rather than running a set of simulations. To give a concrete example, one could imagine using molecular dynamics simulations to screen a library of commercially available compounds for binding to a potential protein target, but if the compounds are commercially available at an inexpensive rate and a suitable assay is available, it might be far faster and cheaper to simply screen the compounds.

So, count the cost of your potential simulations, assess whether they realistically have a chance of answering the questions you seek to answer, and then carefully decide whether the likelihood of success is worth the cost. If not, don't tackle that problem with MD.

2.7 Use your MD simulations and interpret the results with care and caution

Analysis of molecular simulations is largely outside the scope of this work; however, some words of caution are worthwhile. It is tempting, especially for those new to or outside of the area, to view simulations as providing "the answer", giving full mechanistic insight in atomistic detail into what happens in a particular situation and why it happens. Instead, MD results are better thought of as the results of a computational experiment that results from a particular model (including force field), system composition, and protocol. The resulting simulation(s) might or might not be statistically meaningful, relevant to experiment, or useful, but results will be obtained regardless.

This, then, leads us to one of the real dangers of molecular simulations: A simulation produces results, which tempt users to interpret or overinterpret them, whether the results are meaningful or not. For example, even a very short, unequilibrated MD simulation can produce movies which appear interesting and, by virtue of the fact that they result from MD, reveal the positions of all the atoms in a system as a function of time. It's easy to run several short MD simulations where (for example) the composition of the system is varied, and conclude that any observed differences are a result of variations in composition. But as noted in subsubsection 2.5.3.3, even simulations started from the *same* structure but slightly different initial positions or velocities will diverge over time yielding different results, so perhaps any differences are simply a result of this divergence rather than due to the change in conditions. Thus, analysis will require great care and caution to avoid overinterpreting data, and error analysis becomes particularly critical (as discussed in https://github.com/dmzuckerman/Sampling-Uncertainty [69]).

In summary, then, do not use MD simulations simply to make movies and inspect these. Considerable care must be exercised to avoid overinterpeting the full atomistic detail they provide. While movies in some cases can be useful, proper error analysis is always essential.

2.8 Conclusions

Molecular simulations are particularly exciting, because they provide a detailed view into the structure and function of systems at a molecular or atomistic level. Additionally, they allow us to precisely compute thermodynamic and statistical properties and connect these to underlying motions, structure, and function. Thus MD has played a significant role in our field in suggesting new experiments, generating ideas, and helping to provide mechanistic understanding. Advances in hardware, software, methods and force fields also make MD-based calculations particularly appealing for predictive molecular design, where simulations could be used to help guide experiments to develop materials or molecules with desired properties.

Still, MD simulations require considerable care, as conducting them requires choosing a variety of settings, and the optimal choice of settings typically depends on the problem being considered. Thus, it is our hope that this document provides a helpful overview of some of the fundamental considerations for preparing and conducting MD simulations and paves the way for more specialized documents which will focus on calculations of specific properties or for specific classes of systems, since the approach employed will often need to vary depending on such choices.

This document also provides a checklist covering some of the key points raised in this work and highlighting particularly common sources of failure; we encourage readers to follow this when considering a simulation study.

Our focus here has been on the basics — focusing on things you need to understand before beginning to prepare simulations for yourself. Additionally, we have primarily focused on issues relating to how simulations are conducted, and leave data analysis for a separate treatment. As a starting point relating to data analysis, readers should probably review the Best Practices document on sampling and uncertainty estimation (https://github.com/dmzuckerman/Sampling-Uncertainty [69]).

Please remember that this is an updatable work, so we welcome contributions and suggestions via our GitHub issue tracker at https://github.com/MobleyLab/basic_simulation_training.

References

1. Braun, E., Gilmer, J., Mayes, H. B., Mobley, D. L., Monroe, J. I., Prasad, S. & Zuckerman, D. M. Best Practices for Foundations in Molecular Simulations [Article v1.0]. *Living journal of computational molecular science* **1,** 5957. ISSN: 2575-6524 (2019).

2. Nussinov, R. The Significance of the 2013 Nobel Prize in Chemistry and the Challenges Ahead. *PLoS Comput. Biol.* **10,** 2013–2014. http://journals.plos.org/ploscompbiol/article?id=%2010.1371/journal.pcbi.1003423 (2014).

3. Towns, J., Cockerill, T., Dahan, M., Foster, I., Gaither, K., Grimshaw, A., Hazlewood, V., Lathrop, S., Lifka, D., Peterson, G. D., Roskies, R., Scott, J. R. & Wilkens-Diehr, N. XSEDE: Accelerating Scientific Discovery. *Comput. Sci. Eng.* **16,** 62–74. ISSN: 1521-9615. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6866038 (2014).

4. Kirchmair, J., Göller, A. H., Lang, D., Kunze, J., Testa, B., Wilson, I. D., Glen, R. C. & Schneider, G. Predicting drug metabolism: experiment and/or computation? *Nat. Rev. Drug Discov.* **14,** 387–404. ISSN: 1474-1776 (2015).

5. Sresht, V., Lewandowski, E. P., Blankschtein, D. & Jusuf, A. Combined Molecular Dynamics Simulation Molecular-Thermodynamic Theory Framework for Predicting Surface Tensions. *Langmuir* **33,** 8319–8329 (2017).

6. Bottaro, S. & Lindorff-Larsen, K. Biophysical experiments and biomolecular simulations: A perfect match? *Science* **360,** 355–360. http://science.sciencemag.org/content/361/6400/355/tab-pdf (2018).

7. Frenkel, D. & Smit, B. *Understanding Molecular Simulation: From Algorithms to Applications* ISBN: 978-0-08-051998-2 (Elsevier Science, 2001).

8. Leach, A. R. *Molecular Modelling: Principles and Applications* 2001.

9. Jensen, F. *Introduction to Computational Chemistry* Second. ISBN: 9780470011867 (John Wiley & Sons, West Sussex, England, 2007).

10. Schlick, T. *Molecular Modeling and Simulation: An Interdisciplinary Guide* 2nd ed. (Springer, New York, 2010).

11. Mobley, D. L. Let's Get Honest about Sampling. *J Comput Aided Mol Des* **26,** 93–95 (Jan. 2012).

12. Chen, W., Morrow, B. H., Shi, C. & Shen, J. K. Recent Development and Application of Constant pH Molecular Dynamics. *Molecular Simulation* **40,** 830–838. ISSN: 0892-7022 (Aug. 2014).

13. Allen, M. P. & Tildesley, D. J. *Computer Simulation of Liquids* 2nd ed. (Oxford University Press, New York, NY, Aug. 2017).

14. Tuckerman, M. *Statistical mechanics: theory and molecular simulation* (Oxford university press, 2010).

15. Shell, M. S. *Thermodynamics and Statistical Mechanics* ISBN: 978-1-139-02887-5 (Cambridge University Press, Cambridge, 2015).

16. Dill, K. A. & Bromberg, S. *Molecular Driving Forces: Statistical Thermodynamics in Biology, Chemistry, Physics, and Nanoscience* Second (Garland Science, 2010).

17. Kofke, D. A. Direct evaluation of phase coexistence by molecular simulation via integration along the saturation line. *J. Chem. Phys.* **98,** 4149–4162 (1993).

18. Gonzalez Salgado, D. & Vega, C. Melting point and phase diagram of methanol as obtained from computer simulations of the OPLS model. *J. Chem. Phys.* **132,** 094505 (2010).

19. Atkins, P. & Paula, J. d. *Atkins' Physical Chemistry* Tenth Revised Edition (Oxford University Press, 2014).

20. McQuarrie, D. A. & Simon, J. D. *Physical Chemistry: A Molecular Approach* (University Science Books, July 1997).

21. Kittel, C. & Kroemer, H. *Thermal Physics* Second, 473. ISBN: 0-7167-1088-9 (1980).

22. Zuckerman, D. M. *Statistical Physics of Biomolecules: An Introduction* en (CRC Press, June 2010).

23. Zuckerman, D. M. Equilibrium Sampling in Biomolecular Simulations. *Annual Review of Biophysics* **40,** 41–62 (2011).

24. Chong, L. T., Saglam, A. S. & Zuckerman, D. M. Path-Sampling Strategies for Simulating Rare Events in Biomolecular Systems. *Current Opinion in Structural Biology. Theory and simulation • Macromolecular assemblies* **43,** 88–94. ISSN: 0959-440X (Apr. 2017).

25. Grossfield, A. & Zuckerman, D. M. Quantifying Uncertainty and Sampling Quality in Biomolecular Simulations. *Annu Rep Comput Chem* **5,** 23–48. ISSN: 1574-1400 (Jan. 2009).

26. Zuckerman, D. M. *FAQ on Trajectory Ensembles | Statistical Biophysics Blog* en-US. Nov. 2015.

27. Zuckerman, D. M. & Chong, L. T. Weighted Ensemble Simulation: Review of Methodology, Applications, and Software. *Annual Review of Biophysics* **46,** 43–57 (2017).

28. Reif, F. *Fundamentals of Statistical and Thermal Physics* 651. ISBN: 1-57766-612-7 (Waveland Press, Inc., Long Grove, IL, 2009).

29. McQuarrie, D. A. *Statistical Mechanics* (University Science Books, 2000).

30. Hill, T. L. *Statistical Mechanics: Principles and Selected Applications* (Dover Publications, 1987).

31. Chandler, D. *Introduction to Modern Statistical Mechanics* (Oxford University Press, Sept. 1987).

32. Ponder, J. W. & Case, D. A. Force fields for protein simulations. *Advances in Protein Chemistry* **66,** 27–85. ISSN: 0065-3233 (2003).

33. Ponder, J. W., Wu, C., Ren, P., Pande, V. S., Chodera, J. D., Schnieders, M. J., Haque, I., Mobley, D. L., Lambrecht, D. S., DiStasio, R. A., Head-Gordon, M., Clark, G. N. I., Johnson, M. E. & Head-Gordon, T. Current Status of the AMOEBA Polarizable Force Field. *J. Phys. Chem. B* **114,** 2549–2564. ISSN: 1520-6106 (Mar. 2010).

34. Lemkul, J. A., Huang, J., Roux, B. & Mackerell Jr., A. D. An Empirical Polarizable Force Field Based on the Classical Drude Oscillator Model: Development History and Recent Applications. *Chem. Rev.* **116,** 4983–5013. ISSN: 0009-2665 (May 2016).

35. York, D. M., Darden, T. A. & Pedersen, L. G. The Effect of Long-range Electrostatic Interactions in Simulations of Macromolecular Crystals: A Comparison of the Ewald and Truncated List Methods. en. *J. Chem. Phys.* **99,** 8345–8348. ISSN: 0021-9606, 1089-7690 (Nov. 1993).

36. Darden, T., York, D. & Pedersen, L. Particle Mesh Ewald: An $N \cdot \log(N)$ Method for Ewald Sums in Large Systems. *The Journal of Chemical Physics* **98,** 10089–10092. ISSN: 0021-9606, 1089-7690. eprint: https://doi.org/10.1063/1.464397 (June 1993).

37. Piana, S., Lindorff-Larsen, K., Dirks, R. M., Salmon, J. K., Dror, R. O. & Shaw, D. E. Evaluating the Effects of Cutoffs and Treatment of Long-Range Electrostatics in Protein Folding Simulations. en. *PLoS ONE* **7,** e39918. ISSN: 1932-6203 (June 2012).

38. Sagui, C. & Darden, T. A. MOLECULAR DYNAMICS SIMULATIONS OF BIOMOLECULES: Long-Range Electrostatic Effects. *Annual Review of Biophysics and Biomolecular Structure* **28,** 155–179 (1999).

39. Cisneros, G. A., Karttunen, M., Ren, P. & Sagui, C. Classical Electrostatics for Biomolecular Simulations. en. *Chemical Reviews* **114,** 779–814. ISSN: 0009-2665, 1520-6890 (Jan. 2014).

40. Griffiths, D. J. *Introduction to Electrodynamics* 4th (Cambridge University Press, July 2017).

41. Jackson, J. D. *Classical Electrodynamics* 3rd (Wiley, 1998).

42. Mobley, D. L., Bannan, C. C., Rizzi, A., Bayly, C. I., Chodera, J. D., Lim, V. T., Lim, N. M., Beauchamp, K. A., Shirts, M. R., Gilson, M. K. & Eastman, P. K. *Open Force Field Consortium: Escaping Atom Types Using Direct Chemical Perception with SMIRNOFF v0.1* Preprint (Biophysics, Mar. 2018).

43. Wang, L.-P., McKiernan, K. A., Gomes, J., Beauchamp, K. A., Head-Gordon, T., Rice, J. E., Swope, W. C., Martínez, T. J. & Pande, V. S. Building a More Predictive Protein Force Field: A Systematic and Reproducible Route to AMBER-FB15. *J. Phys. Chem. B* **121.** PMID: 28306259, 4023–4039. eprint: https://doi.org/10.1021/acs.jpcb.7b02320. https://doi.org/10.1021/acs.jpcb.7b02320 (2017).

44. Mackerell Jr., A. D., Feig, M. & Brooks III, C. L. Extending the treatment of backbone energetics in protein force fields: Limitations of gas-phase quantum mechanics in reproducing protein conformational distributions in molecular dynamics simulations. *J. Comput. Chem.* **25,** 1400–1415 (2004).

45. Perez, A., MacCallum, J. L., Brini, E., Simmerling, C. & Dill, K. A. Grid-Based Backbone Correction to the ff12SB Protein Force Field for Implicit-Solvent Simulations. *J. Chem. Theory Comput.* **11,** 4770–4779. ISSN: 1549-9618 (Oct. 2015).

46. Sanyal, T. & Shell, M. S. Coarse-grained models using local-density potentials optimized with the relative entropy: Application to implicit solvation. *J. Chem. Phys.* **145,** 034109. ISSN: 0021-9606 (2016).

47. Becker, C. A., Tavazza, F., Trautt, Z. T. & Buarque De Macedo, R. A. Considerations for choosing and using force fields and interatomic potentials in materials science and engineering. *Current Opinion in Solid State and Materials Science* **17,** 277–283. ISSN: 1359-0286 (2013).

48. Case, D., Cerutti, D., Cheatham, III, T., Darden, T., Duke, R., Giese, T., Gohlke, H., Goetz, A., Greene, D., Homeyer, N., Izadi, S., Kovalenko, A., Lee, T., LeGrand, S., Li, P., Lin, C., Liu, J., Luchko, T., Luo, R., Mermelstein, D., Merz, K., Monard, G., Nguyen, H., Omelyan, I., Onufriev, A., Pan, F., Qi, R., Roe, D., Roitberg, A., Sagui, C., Simmerling, C., Botello-Smith, W., Swails, J., Walker, R., Wang, J., Wolf, R., Wu, X., Xiao, L. & Kollman, P. *Amber Reference Manuals* http://ambermd.org/Manuals.php.

49. Apol, E., Apostolov, R., Berendsen, H. J. C., van Buuren, A., Bjelkmar, P., van Drunen, R., Feenstra, A., Fritsch, S., Groenhof, G., Junghans, C., Hub, J., Kasson, P., Kutzner, C., Lambeth, B., Larsson, P., Lemkul, J. A., Lindahl, V., Lundborg, M., Marklund, E., Meulenhoff, P., Murtola, T., Páll, S., Pronk, S., Schulz, R., Shirts, M., Sijbers, A., Tieleman, P., Wennberg, C., Wolf, M., Abraham, M., Hess, B., van der Spoel, D. & Lindahl, E. *GROMACS Documentation Reference Manual* http://manual.gromacs.org/documentation/.

50. Riniker, S. Fixed-Charge Atomistic Force Fields for Molecular Dynamics Simulations in the Condensed Phase: An Overview. *Journal of Chemical Information and Modeling* **58,** 565–578. ISSN: 1520-5142 (2018).

51. Mishra, R. K., Mohamed, A. K., Geissbühler, D., Manzano, H., Jamil, T., Shahsavari, R., Kalinichev, A. G., Galmarini, S., Tao, L., Heinz, H., Pellenq, R., van Duin, A. C., Parker, S. C., Flatt, R. J. & Bowen, P. cemff: A force field database for cementitious materials including validations, applications and opportunities. *Cement and Concrete Research* **102,** 68–89. ISSN: 0008-8846 (2017).

52. Lopes, P. E., Roux, B. & Mackerell Jr., A. D. Molecular modeling and dynamics studies with explicit inclusion of electronic polarizability: Theory and applications. *Theoretical Chemistry Accounts* **124,** 11–28. ISSN: 1432-881X. arXiv: NIHMS150003 (2009).

53. Onufriev, A. V. & Izadi, S. Water models for biomolecular simulations. *Wiley Interdisciplinary Reviews: Computational Molecular Science* **8.** ISSN: 1759-0884 (2018).

54. Vega, C. & Abascal, J. L. Simulating water with rigid non-polarizable models: A general perspective. *Physical Chemistry Chemical Physics* **13,** 19663–19688. ISSN: 1463-9076 (2011).

55. Tadmor, E., Elliott, R.S., Sethna, J., Miller, R. & Becker, C. *Knowledgebase of Interatomic Models (KIM)* https://openkim.org.

56. Hale, L., Trautt, Z. & Becker, C. *Interatomic Potentials Repository Project* https://www.ctcms.nist.gov/potentials/.

57. Shirts, M. R., Mobley, D. L., Chodera, J. D. & Pande, V. S. Accurate and Efficient Corrections for Missing Dispersion Interactions in Molecular Simulations. *J Phys Chem B* **111,** 13052–13063 (Nov. 2007).

58. Isele-Holder, R. E., Mitchell, W. & Ismail, A. E. Development and Application of a Particle-Particle Particle-Mesh Ewald Method for Dispersion Interactions. *J. Chem. Phys.* **137,** 174107. ISSN: 0021-9606 (Nov. 2012).

59. Dupradeau, F.-Y., Pigache, A., Zaffran, T., Savineau, C., Lelong, R., Grivel, N., Lelong, D., Rosanski, W. & Cieplak, P. The R.E.D. Tools: Advances in RESP and ESP Charge Derivation and Force Field Library Building. en. *Phys. Chem. Chem. Phys.* **12,** 7821–7839. ISSN: 1463-9076 (July 2010).

60. Shell, M. S. *Principles of modern molecular simulation methods: Lecture Notes* https://engineering.ucsb.edu/~shell/che210d/assignments.html.

61. Yeh, I.-C. & Hummer, G. System-Size Dependence of Diffusion Coefficients and Viscosities from Molecular Dynamics Simulations with Periodic Boundary Conditions. *J. Phys. Chem. B* **108,** 15873–15879 (2004).

62. Lemkul, J. *GROMACS Tutorials* http://www.bevanlab.biochem.vt.edu/Pages/Personal/justin/gmx-tutorials.

63. Madej, B. & Walker, R. *AMBER Tutorial B0: An Introduction to Molecular Dynamics Simulations Using AMBER* http://ambermd.org/tutorials/basic/tutorial0/index.html.

64. Jewett, A. *Moltemplate* https://www.moltemplate.org/. Nov. 2018.

65. Martínez, L., Andrade, R., Birgin, E. G. & Martínez, J. M. PACKMOL: A Package for Building Initial Configurations for Molecular Dynamics Simulations. *Journal of Computational Chemistry* **30,** 2157–2164. ISSN: 0192-8651 (Oct. 2009).

66. Hirel, P. Atomsk: A Tool for Manipulating and Converting Atomic Data Files. *Computer Physics Communications* **197,** 212–219. ISSN: 0010-4655 (Dec. 2015).

67. Joswiak, M. N., Duff, N., Doherty, M. F. & Peters, B. Size-Dependent Surface Free Energy and Tolman-Corrected Droplet Nucleation of TIP4P/2005 Water. *J. Phys. Chem. Letters* **4.** PMID: 26296177, 4267–4272 (2013).

68. Palmer, J. C., Haji-Akbari, A., Singh, R. S., Martelli, F., Car, R., Panagiotopoulos, A. Z. & Debenedetti, P. G. Comment on "The putative liquid-liquid transition is a liquid-solid transition in atomistic models of water" [I and II: J. Chem. Phys. 135 , 134503 (2011); J. Chem. Phys. 138, 214504 (2013)]. *J. Chem. Phys.* **148,** 137101. ISSN: 0021-9606 (2018).

69. Grossfield, A., Patrone, P. N., Roe, D. R., Schultz, A. J., Siderius, D. & Zuckerman, D. M. Best Practices for Quantification of Uncertainty and Sampling Quality in Molecular Simulations [Article v1.0]. *Living Journal of Computational Molecular Science* **1.** ISSN: 2575-6524 (2019).

70. Hünenberger, P. H. Thermostat algorithms for molecular dynamics simulations. *Advanced Computer Simulation,* 105–149. ISSN: 0065-3195 (2005).

71. Basconi, J. E. & Shirts, M. R. Effects of Temperature Control Algorithms on Transport Properties and Kinetics in Molecular Dynamics Simulations. *J. Chem. Theory Comput.* **9,** 2887–2899. ISSN: 1549-9618 (July 2013).

72. Minary, P., Martyna, G. J. & Tuckerman, M. E. Algorithms and novel applications based on the isokinetic ensemble. I. Biophysical and path integral molecular dynamics. *J. Chem. Phys.* **118,** 2510–2526. ISSN: 0021-9606 (2003).

73. Braun, E., Moosavi, S. M. & Smit, B. Anomalous effects of velocity rescaling algorithms: the flying ice cube effect revisited. *J. Chem. Theory Comput.* **14,** 5262–5272 (2018).

74. Harvey, S. C., Tan, R. K.-Z. & Cheatham, T. E. The Flying Ice Cube: Velocity Rescaling in Molecular Dynamics Leads to Violation of Energy Equipartition. en. *J. Comp. Chem.* **19,** 726–740. ISSN: 1096-987X (1998).

75. Berendsen, H. J. C., Postma, J. P. M., van Gunsteren, W. F., DiNola, A. & Haak, J. R. Molecular Dynamics with Coupling to an External Bath. *The Journal of Chemical Physics* **81,** 3684–3690. ISSN: 0021-9606, 1089-7690 (Oct. 1984).

76. Bussi, G., Donadio, D. & Parrinello, M. Canonical sampling through velocity rescaling. *J. Chem. Phys.* **126,** 014101. ISSN: 0021-9606 (2007).

77. Andersen, H. C. Molecular Dynamics Simulations at Constant Pressure and/or Temperature. *The Journal of Chemical Physics* **72,** 2384–2393. ISSN: 0021-9606, 1089-7690 (Feb. 1980).

78. Schneider, T. & Stoll, E. Molecular-dynamics study of a three-dimensional one-component model for distortive phase transitions. *Physical Review B* **17,** 1302–1322. ISSN: 0163-1829 (1978).

79. Martyna, G. J., Klein, M. L. & Tuckerman, M. Nosé-Hoover chains: the canonical ensemble via continuous dynamics. *J. Chem. Phys.* **97,** 2635–2643. ISSN: 0021-9606 (1992).

80. Parrinello, M. & Rahman, A. Polymorphic Transitions in Single Crystals: A New Molecular Dynamics Method. *Journal of Applied Physics* **52,** 7182–7190. ISSN: 0021-8979, 1089-7550 (Dec. 1981).

81. Martyna, G. J., Tobias, D. J. & Klein, M. L. Constant pressure molecular dynamics algorithms. *J. Chem. Phys.* **101,** 4177–4189. ISSN: 0021-9606 (1994).

82. Martyna, G. J., Tuckerman, M. E., Tobias, D. J. & Klein, M. L. Explicit reversible integrators for extended systems dynamics. *Molecular Physics* **87,** 1117–1157. ISSN: 0026-8976 (1996).

83. Tuckerman, M., Berne, B. J. & Martyna, G. J. Reversible multiple time scale molecular dynamics. *The Journal of Chemical Physics* **97,** 1990–2001. ISSN: 0021-9606. eprint: https://doi.org/10.1063/1.463137 (Aug. 1992).

84. Fass, J., Sivak, D., Crooks, G. E., Beauchamp, K. A., Leimkuhler, B. & Chodera, J. Quantifying configuration-sampling error in Langevin simulations of complex molecular systems. *bioRxiv,* 266619. eprint: https://www.biorxiv.org/content/early/2018/02/16/266619.full.pdf. https://www.biorxiv.org/content/early/2018/02/16/266619 (2018).

85. Berne, B. J. *Molecular Dynamics in Systems with Multiple Time Scales: Reference System Propagator Algorithms* in *Computational Molecular Dynamics: Challenges, Methods, Ideas* (eds Deuflhard, P., Hermans, J., Leimkuhler, B., Mark, A. E., Reich, S. & Skeel, R. D.) (Springer, Berlin, 1999), 297–317. ISBN: 978-3-642-58360-5.

86. Hopkins, C. W., Le Grand, S., Walker, R. C. & Roitberg, A. E. Long-time-step molecular dynamics through hydrogen mass repartitioning. *J. Chem. Theory Comput.* **11,** 1864–1874 (2015).

87. Ewald, P. P. Die Berechnung optischer und elektrostatischer Gitterpotentiale. *Annalen der Physik* **369,** 253–287. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/andp.19213690304 (1921).

88. Essmann, U., Perera, L., Berkowitz, M. L., Darden, T., Lee, H. & Pedersen, L. G. A Smooth Particle Mesh Ewald Method. *Journal of Chemical Physics* **103,** 8577 (1995).

89. Eastwood, J., Hockney, R. & Lawrence, D. P3M3DP-The three-dimensional periodic particle-particle/particle-mesh program. *Computer Physics Communications* **19,** 215–261. ISSN: 0010-4655. http://www.sciencedirect.com/science/article/pii/0010465580900521 (1980).

90. Paliwal, H. & Shirts, M. R. Using Multistate Reweighting to Rapidly and Efficiently Explore Molecular Simulation Parameters Space for Nonbonded Interactions. *J. Chem. Theory Comput.* **9,** 4700–4717. ISSN: 1549-9618 (Nov. 2013).

91. Grossfield, A., Patrone, P. N., Roe, D. R., Schultz, A. J., Siderius, D. W. & Zuckerman, D. M. Best Practices for Quantification of Uncertainty and Sampling Quality in Molecular Simulations [Article v1.0]. en. *Living Journal of Computational Molecular Science* **1,** 5067 (1 2019).

CHAPTER 3

Towards Molecular Simulations that are Transparent, Reproducible, Usable By Others, and Extensible (TRUE)

3.1 Introduction

Systems composed of soft matter (e.g., liquids, polymers, foams, gels, colloids, and most biological materials) are ubiquitous in science and engineering, but molecular simulations of such systems pose particular computational challenges, requiring time and/or ensemble-averaged data to be collected over long simulation trajectories for property evaluation. Performing a molecular simulation of a soft matter system involves multiple steps, which have traditionally been performed by researchers in a "bespoke" fashion, resulting in many published soft matter simulations not being reproducible based on the information provided in the publications. To address the issue of reproducibility and to provide tools for computational screening, we have been developing the open-source Molecular Simulation and Design Framework (MoSDeF) software suite.

In this chapter [1][2], we describe a set of principles to create Transparent, Reproducible, Usable by others, and Extensible (TRUE) molecular simulations. MoSDeF facilitates the publication and dissemination of TRUE simulations by automating many of the critical steps in molecular simulation, thus enhancing their reproducibility. We provide several examples of TRUE molecular simulations: All of the steps involved in creating, running and extracting properties from the simulations are distributed on open-source platforms (within MoSDeF and on GitHub), thus meeting the definition of TRUE simulations.

3.2 Background

Reproducibility in scientific research has become a prominent issue, to the extent that some have opined that science has a "reproducibility crisis" [2]. Along with the rest of the scientific community, computational scientists are grappling with the central question: How can a computational study be performed and published in such a way that it can be replicated by others? This has become increasingly important as researchers seek to harness the ever expanding computational power to perform large-scale computational screening of materials [3–9] inspired by the materials genome initiative (MGI)[10], where reproducibility issues commonly faced in small-scale studies will only be compounded as the number of simulations grow by orders of magnitude.

Addressing the issues of reproducibility in soft matter simulation is particularly challenging, given the

[1]Portions of this chapter reprinted with permission from the following work:

[2]Thompson, M. W., Gilmer, J. B., Matsumoto, R. A., Quach, C. D., Shamaprasad, P., Yang, A. H., Iacovella, C. R., McCabe, C. & Cummings, P. T. Towards Molecular Simulations That Are Transparent, Reproducible, Usable by Others, and Extensible (TRUE). *Molecular Physics* **118,** e1742938. ISSN: 0026-8976, 1362-3028 (June 2020).

complexity of the simulation inputs and workflows. Here we define soft matter as anything easily deformed at room temperature, e.g., liquids, polymers, foams, gels, colloids, and most biological materials. Figure 3.1 shows a schematic of the general multi-step workflow for performing atomistic simulations of soft matter systems, proceeding from system “chemistry” (chemical composition and state conditions such as phases(s), temperature, pressure, and composition) to “properties” (e.g., structural, thermodynamic and transport properties, phase equilibria, and dielectric properties) In such systems, the differences in potential energy between distant configurations are on the same order as the thermal motion, requiring time and/or ensemble-averaged data to be collected over long simulation trajectories for property evaluation. The equilibration procedures and system sizes considered may strongly influence the resulting measured properties, since one must consider both the local conformations of the underlying components, along with any mesoscopic structuring present in the system. To capture sufficiently large length and time scales, soft matter simulations are typically performed using methods such as molecular dynamics (MD) or Monte Carlo (MC) that employ empirical force fields to model the interactions between atoms and molecules; the appropriate force field parameters must be identified before the simulation can be performed.

Some commonly available force fields, such as the Optimized Potential for Liquid System (OPLS)[11] and the General Amber Force Field (GAFF)[12] contain thousands of possible parameters that are differentiated by their chemical context (e.g., the element a given interaction site represents, the number and identity of bonded neighbors, the local environment of bonded neighbors, the type of system, etc.). Selecting the appropriate force field parameters for a particular use case is often non-trivial. Workflows may also involve the optimization of specific parameters, such as partial charges, or require separate procedures to develop novel force fields, such as coarse-grained (CG) models, before a simulation can be performed. Furthermore, due to the complex nature of the underlying constituents (e.g., highly branched polymers), setup of an initial system configuration may be challenging and require additional relaxation procedures to ensure system stability[13].

As such, soft matter simulations typically require multi-step workflows with many inputs. The various steps are often accomplished by separate pieces of syntactically and/or semantically incompatible software tools, that may require translators or *ad hoc* modifications to facilitate interoperability. These tools, and especially the “glue” code that facilitates interoperability, are typically neither publicly available nor version-controlled. The latter is particularly important for long-term reproducibility, since to repeat a particular calculation may require using versions of the relevant codes used when the work was originally published, which could be a number of years ago.

The above complexities often make it difficult for researchers themselves to fully capture and preserve the procedures used to perform a simulation, let alone clearly disseminate these to the rest of the community. A typical soft matter simulation publication provides an overview of the methods and procedures used but

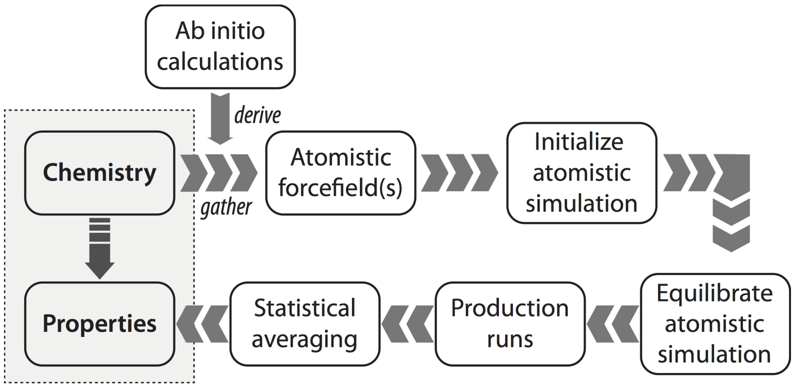


Figure 3.1: Schematic of the typical process required to compute properties of soft matter systems from system "chemistry," which refers to chemical composition and state (including temperature, pressure and composition), starting from the need to either gather or derive force field parameters to model the system. For coarse-grained (CG) simulations, the CG force fields are often derived from atomistic simulations.

falls significantly short of including the necessary information to unambiguously reproduce the published work. This information includes, but is not limited to, citations to the sources of force field(s) used, the numeric parameters of the force field(s) used, how the force field parameters were assigned to the system, constants and options provided in the underlying algorithms, and the exact choices used in constructing the initial configuration of the system. It is important to recognize that the results from a simulation can depend on the minute details[14]. These details include, but are not limited to, the random seed used to generate a distribution, the specific force field parameters and how they were used, the exact procedures employed to equilibrate a system, etc. For example, small variations in force fields (e.g., changes in distances at which interactions are truncated, different partial charges, the specific method for handling long-ranged interactions, etc.) can change some predicted properties quite significantly[14–16]. The minute details may also be inherent to the software used to perform the simulations, and thus the use of "in-house" or commercial (i.e., closed-source) software stymies reproducibility. If the source code cannot be viewed, the underlying algorithms and inputs cannot be examined, the quality of the code and whether it has undergone proper validation is unknown, and errors cannot be identified. As an example, a long standing disagreement related to phase transitions in supercooled water was only recently settled after the source code of the in-house software used to perform the calculations was shared. The differences in observed transitions were attributed to a subtle error in how velocities were assigned when initializing the many short MD simulations in the hybrid MD/MC workflows.[17, 18]. The use of open-source simulation engines therefore clearly enhances reproducibility, as the underlying source code can be examined (note, the use of open-source simulation engines is now routine for MD studies, but often these engines and other open-source codes are modified to implement new force field parameters or functional forms, and MC studies still commonly use in-house software). However, it is atypical for input scripts and data files for open-source simulation engines to be included as part of a publication and thus reproducibility still largely depends on the thoroughness of the description of the methods and model in the text. Furthermore, the algorithms and specific choices used to generate a data file, which may influence the results and their validity (e.g., how a force field was applied), are lost if the software and/or procedures used to generate the data file are not made available. Even when using open-source simulation engines, researchers still routinely use in-house software for other steps in the process, i.e. generation of initial configurations, selection of force field parameters, and analysis. Furthermore, if a workflow relies upon manipulation or modification of individual pieces of software by a user (e.g., initializing a system using software with a graphical user interface, GUI[19]), or human-modification of files, it is often difficult to capture and convey the exact procedures in such a manner that they can be reproduced by another researcher.

Fortuitously, several researchers have proposed general guidelines for increasing reproducibility in computational research, which can be used to infer best practices for soft matter simulation. Donoho *et al.*[20] propose that all details of computations – code and data – should be made "conveniently available" to other researchers; they also provide arguments in favor of the creation and use of community developed software libraries and the use of scripting. Others[21, 22] have proposed succinct "rules" as keys to reproducible computational research, including version control, replacement of manual input with scripts, and public access to these scripts, input files, and resulting data. It was also noted that computational frameworks that integrate different tools within a common environment naturally satisfy many of these rules. One of the most vocal proponents of reproducibility in computational science[23, 24], has gone as far as asserting that GUIs are the "enemy of reproducibility". GUIs hide details and require human interaction and manipulation in contrast to scripts, which fully reveal the way in which calculations are performed. A classic example is Excel spreadsheets, where the relationship between calculation cells and data is normally hidden, and the order of calculation is not obvious, nor necessarily controllable. In 2010, Harvard University economists Reinhart and Rogoff published a highly cited and influential paper on the role of debt in limiting growth in national economies[25]. The study, based on data manipulated within an Excel spreadsheet, was often cited by politicians favoring austerity policies in the wake of the 2008 financial crisis while public economic policy was being formed. Subsequently, Herndon *et al.*[26] found that the spreadsheet contained errors in formulae that dramatically changed the conclusions.

Determining how these guidelines for reproducibility should be — and/or can be — implemented in soft matter simulation is in itself a challenge. For example, simply providing code is not effective if that code is poorly written or not well documented and has subtle issues, such as dependencies within a code (e.g., use of external libraries, especially if they are proprietary/non-free or difficult to obtain/install) These issues may create barriers to proper compilation/installation and hence hamper reproducibility. Similarly, providing a raw data file without defining the structure of it, and/or without appropriate metadata, does little to aid in reproducibility. Since journals largely do not provide mechanisms for sharing code, scripts, and/or data (aside from supplemental material), it is also not clear how such information should best be shared.

As such, in order to implement best practices, we assert that the development of new tools and standards will be required, in order to facilitate necessary changes to the way in which simulators perform and publish their research. However, development of new tools does little to improve reproducibility if those tools are not used; to be widely adopted by the community, they must provide additional value to researchers, e.g., minimizing errors, reducing development time, preventing knowledge loss, providing novel functionality, etc.

For almost a decade now, we have been developing a robust Python-based, open-source integrated software framework for performing simulations of soft matter systems with the goal of implementing best practices and enabling reproducibility. This framework, known as the Molecular Simulation and Design Framework (MoSDeF) [27], was developed initially at Vanderbilt University, in collaboration with computer scientists in the Institute for Software Integrated Systems [28], to facilitate screening studies of monolayer lubrication using MD methods. MoSDeF provides a core foundation and includes tools for programmatic system construction (`mBuild`)[29, 30], tools for encoding force field usage rules and their application (`Foyer`)[31–33], and has recently integrated the `signac` framework[34, 35], developed at the University of Michigan as a means of improved data and workflow management. The MoSDeF toolkit has been used in various published results[8, 9, 29, 33, 36, 37] and ongoing research projects, with the primary MoSDeF tools having each been downloaded over 18,000 times from Anaconda Cloud[38] since February 2017. Despite being initially developed for monolayer lubrication, the underlying tools can be and have been applied to soft matter systems in general, and the modular, object-oriented design naturally allows for intuitive extension. Current MoSDeF activities are expanding the capabilities related to:

- Initializing system configurations by providing a plugin architecture for community contributions
- Providing initialization routines for a wide variety of common systems
- Developing an improved backend that will support an increased number of force field types and simulation engines, including open-source MC software
- Developing modules that implement methods for partial charge assignment
- Including improved support and libraries for coarse-grained models
- Developing modules that allow for reproducible derivation of coarse-grained and atomistic force fields

- Developing workflows for free energy methods and phase equilibria
- Specifically identifying and implementing best practices within the various modules/workflows that improve reproducibility.

Through the MoSDeF integrated framework, the exact procedures used to set up and perform simulation workflows and associated metadata (i.e., the provenance) can be scripted, encapsulated, version-controlled, preserved, and later reproduced by other researchers. This allows molecular simulation studies to be conducted and published in a manner that is TRUE: Transparent, Reproducible, Usable by others, and Extensible.

The remainder of this paper is organized as follows. In Section 3.3, we briefly review MoSDeF an its capabilities. In Section 3.4, we consider four examples of TRUE molecular simulations in diverse application areas. Finally, in Section 3.5, we summarize our conclusions and prospects for future development of MoSDeF.

3.3 Overview of MoSDeF

3.3.1 MoSDeF tools and capabilities

MoSDeF is a set of an open-source Python libraries, designed to facilitate the construction and parameterization of systems for molecular simulation. MoSDeF includes routines to output syntactically correct configuration files in formats used by common simulation engines, currently supporting GROMACS[39–41], LAMMPS[42], HOOMD-blue[43–45], and Cassandra [46], as well as other common file formats (e.g., `MOL2`, `PDB`) through integration with the open-source ParmEd[47] parameter editing package. Each library (i.e., Python module) in MoSDeF is designed such that it can be used as a standalone package, in combination with other libraries within MoSDeF, or with other libraries developed and used by the community. This composability/modularity is an essential design feature in terms of the robust development of MoSDeF, allowing the framework to be more modifiable, testable, extensible, and have fewer bugs than monolithic approaches[14]. MoSDeF is implemented using concepts from the computer science/software engineering field of model integrated computing (MIC)[48, 49], a systems engineering approach, pioneered at the Institute for Software Integrated Systems (ISIS) at Vanderbilt, that emphasizes the creation of domain-specific modeling languages that capture the features of the individual components of a given process, at the appropriate level of abstraction. By using concepts from MIC, MoSDeF can easily be abstracted and is able to capture the relationships that exist between data and processes regardless of the level of abstraction, essential for ensuring that system initialization scripts are transparent and usable by others. MoSDeF follows a modern open-source development model with special emphasis on effective code sharing, accepting external feedback, and bug reporting.

- All modules and workflows developed for MoSDeF build upon the scientific Python stack, thus enabling transparency, promoting code reuse, lowering barriers to entry for new users, and promoting further community driven, open-source development.

- GitHub is used for hosting MoSDeF's version-controlled software development, deployment, and documentation/tutorials, using a pull request (i.e., fork-pull) model that allows for code review and automated testing, helping ensure proper standards have been followed and allows for automated testing of software and software artifacts.

- Automated builds and testing of the software are hosted on Travis CI[50] and also on Microsoft's Azure Pipelines[51] to ensure that proposed modifications to the code do not break the current performance and the `CodeCov`[52] tool is used to ensure that modifications to the code are covered by unit tests.

- All software developed as part of the MoSDeF project are open-source, with the standard MIT license[53] that allows free use, reuse, modifications, as well as commercialization.

- Slack[54] is used to facilitate effective collaborative communication and software development across a wide geographic area[55].

By developing software in a modular, extensible, open-source manner, using freely available tools designed for collaborative code development (e.g., git, GitHub, and Slack), we are creating a long-term community-developed effort, similar to the success seen by other tools in the community (e.g., GROMACS[39–41], VMD[56], LAMMPS[42], HOOMD-Blue[44, 57], etc.). This has become especially important as the group of MoSDeF developers has expanded beyond Vanderbilt University. A recent U.S. National Science Foundation grant[58] has provided support for leading molecular simulation research groups from Vanderbilt, the universities of Michigan, Notre Dame, Delaware, Houston and Minnesota, along with Boise State University and Wayne State University to further improve and increase support of MoSDeF as described below. This group spans a broad range of expertise, and an equally broad range of scientific applications, open-source simulation codes (HOOMD-Blue[44, 57], Cassandra[59], GOMC[60, 61] and CP2K[62]), workflow and data management software[34, 35, 63] and algorithms and analysis tools; computer scientists from ISIS are also involved in the collaboration, helping to ensure the use of best practices and provide novel insight into algorithmic and software development. In combination, this collaboration is working to dramatically expand the capabilities of MoSDeF and thus facilitate researchers in the area of molecular simulation to be able to publish TRUE simulations.

Here, we briefly describe the two key tools used in the current version of MoSDeF, focusing on the specific aspects of the tools that help to enable TRUE simulations.

3.3.1.1 mBuild

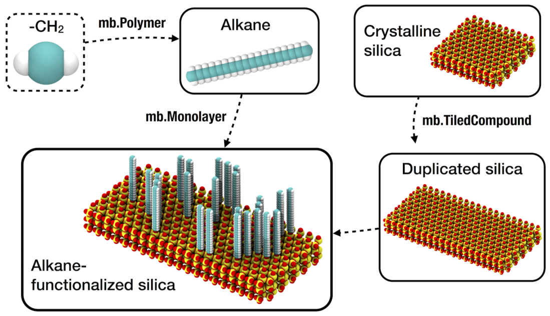
The mBuild Python library[29, 30] is a general purpose tool for constructing system configurations in a programmatic (i.e., scriptable) fashion. While tools exist in the community for system construction[64–66], they tend to be system specific (e.g., bilayer construction), often employ GUIs which may hamper reproducibility[23] and may be limiting for workflows that require automation, and most are designed around the concept that components of the system can be described by self-contained templates (e.g., where a system can be constructed by simply duplicating a template that describes a molecule). Such existing tools have typically not been designed to work for systems where bonds are added between different components (e.g., polymer grafted surfaces) or for systems where one component is semi-infinite (e.g., a silica substrate that is periodic in-plane) and most do not allow programmatic variation of specific structural/chemical aspects (e.g., the length of a polymer, the polymer repeat unit, size of a substrate, etc.); mBuild was designed specifically to provide this missing functionality.

Rather than providing a tool to perform initialization that only applies to a specific family of systems (e.g., monolayers), mBuild provides a library of functions that users can combine, extend, and add to, in order to construct specific systems of interest. mBuild allows users to hierarchically construct complex systems from smaller, interchangeable pieces that can be connected, through the use of the concept of generative, or procedural, modeling[29]. This is achieved through mBuild's underlying Compound data structure, which is a general purpose "container" that can describe effectively anything within the system: an atom, a coarse-grained bead, a collection of atoms, a molecule, a collection of Compounds, or operations (e.g., a Compound that includes a routine to perform polymerization). To join Compounds (e.g., attachment of a Compound that defines a polymer to a Compound that defines a surface), Compounds can include Ports that define both the location and orientation of a possible connection. In mBuild, a user (or algorithm) defines which Ports on two Compounds should be connected and the underlying routines in the software automatically performs the appropriate translations and orientations of the Compounds (see Klein *et al.*[29] for more details). This creates a new (composite) Compound that contains both of the original Compounds, now appropriately oriented and positioned in space, with an explicit bond between them; since Compounds are general data structures, the same operations (rotation, translation, connecting of Ports, etc.) that were performed on the underlying Compounds can be performed on this new composite Compound. The mBuild library can be used to create systems from "scratch" whereby a user implements all the relevant code to define the building blocks and how they should be connected, or by using and/or extending the various"recipes" included in mBuild. mBuild includes (but is not limited to) "recipes" for initializing polymers, tilings (e.g., duplicating a unit cell, including bonding information), patterning (disk, sphere, random, etc.), lattices either from a Crystallographic Information File (CIF), their Bravais lattice parameters, or the vectors describing the prism, box filling (via integration with PACKMOL[67]), monolayers and brushes on flat, curved, and spherical surfaces, and bilayers and lamellar structures.

As an example, Fig. 3.2 shows a schematic and associated code for the construction of an alkane grafted silica surface. In this code, a custom `Compound` class is defined for a CH_2 moiety alongside a `Compound` from the `mBuild` library that defines a crystalline silica surface; a"recipe" included in `mBuild` that performs polymerization (`Polymer`) is used to connect copies of the CH_2 moieties; the `Monolayer` "recipe", also included in `mBuild`, is used to perform the functionalization of the silica surface with the polymer, returning a single composite `Compound` of the functionalized surface (for readability, the terminal groups are ignored in this example). This example also highlights how system construction can be programmatically varied, e.g., the `Polymer` class takes as input the number of repeat units (in this case, set to 18). Similarly, the size of the substrate can be toggled in the `Monolayer` class, where `tile_x` and `tile_y` define the number of times the substrate is duplicated in the respective dimension. The number of chains attached to the surface can also be modified via the number passed to the `Random2DPattern` class (here set to 25). Because `Compound`s are general containers, changes to, e.g., the length of the polymer, do not require changes to the rest of the script, namely the `Monolayer` class. Similarly, characteristics such as the repeat unit passed to the `Polymer` class can be readily changed without need to change other aspects of the script. As a result, by using the `mBuild` library, complex system initialization and variation/extension can often be accomplished without the need to write significant amounts of code. As this example shows, by using concepts from MIC, construction of systems in `mBuild` can be trivially abstracted (i.e., the level of complexity reduced) to the level most appropriate to describe (i.e., model) the system, without making system construction a "black box". Since `mBuild` is an open-source, freely available Python library, scripts that unambiguously define all the steps needed to initialize a system can be easily shared and disseminated with publications, with all code easily interrogated, allowing system construction to be reproduced and improving transparency; `mBuild` has additionally been architected so that users can contribute custom "recipes" for system initialization via a plug-in environment, further allowing such routines to be easily shared, utilized and extended by others.

3.3.1.2 Foyer

The `Foyer` library[31] is a tool for applying force fields to molecular systems (i.e., atom-typing). `Foyer` provides a standardized approach to defining chemical context (i.e., atom-typing rules)[32, 68] along with the associated force field parameters. While there are freely available tools to aid in atom-typing[69–73], these are typically specific to a particular force field or simulator, and/or capture the atom-typing and parametrization in a hierarchy (either through specific placement in a parameter file read by the code or as nested if/else



```
import mbuild as mb

class CH2(mb.Compound):
    """An mBuild Compound representing a methylene bridge. """
    def __init__(self):
        super(CH2, self).__init__()

        # Load the CH2 coordinates and bonds from a structure file
        mb.load(filename='ch2.pdb', compound=self)

        # Locate the carbon atom
        carbon = list(self.particles_by_name('C'))[0]

        # Add Ports for the two dangling bonds
        self.add(mb.Port(anchor=self[0], orientation=[0, 1, 0],
            separation=0.07), label='up')
        self.add(mb.Port(anchor=self[0], orientation=[0, -1, 0],
            separation=0.07), label='down')

# Create a polymer of length 18
polymer = mb.Polymer(monomers=CH2(), n=18)

# Attach copies of the polymer to a surface
from mbuild.lib.surfaces import Betacristobalite
functionalized_surface = mb.Monolayer(surface=Betacristobalite(),
    chains=polymer, pattern=mb.Random2DPattern(25), tile_x=2, tile_y=1)
```

Figure 3.2: Python script that uses `mBuild` to define a class for a –CH_3– group, create a polymer composed of multiple –CH_2– groups, and connects copies of this polymer to a surface. Note for simplicity, the terminal CH_3 group is not shown.

statements within the source code). `Foyer` does not encode usage rules into the source code, instead defining usage rules and parameters in an `XML` file that is an extension of the `OpenMM`[74] force field file format. The `Foyer` software itself is used to interpret and apply the rules and thus the software is not limited to use with only a single force field type. By separating the usage rules from the source code, changes or extensions to force field parameters/rules does not require changes to the code itself. Force field usage rules are encoded using a combination of a SMARTS-based[75] annotation scheme, which defines the molecular environment (i.e., chemical context) associated with a given parameter, and `overrides` that define rule precedence (i.e., which atom type to choose when multiple rules can apply to an interaction site). The use of `overrides` avoids the need to define rule precedence via the order of the rules within a file (See [68] for more details). As an example, Listing 1 shows the contents of an `XML` file that contains parameters and usage rules from the OPLS force field for linear alkanes. We note that `Foyer` allows user-defined input (by pre-pending with an underscore), allowing SMARTS to be used for non-elemental interaction sites (e.g., an interaction site that represents a coarse-grained bead or an united atom interaction site). As such, the exact parameters and their usage can be readily captured and disseminated along with a simulation and/or publication. This provides an improved way to disseminate custom force field parameter sets and/or novel force field parameters (e.g., see Ref. [33]) that reduces ambiguity, as the format used by `Foyer` to encode the usage rules and parameters is both human and machine readable; thus parameterization rules provided in a publication can be automatically tested for accuracy and completeness. To further enhance reproducibility, the `XML` force field files additionally include a `doi` tag for the source of the parameters (see Listing 3.1); upon successful atom-typing, `Foyer` outputs a BibTeX file of references with the relevant DOIs, significantly improving the transparency as to the origin of parameters used in a simulation and therefore reproducibility.

3.3.2 Other Community Tools

Here we briefly highlight other simulation tools and efforts with a considerable focus on reproducibility and transparency, several with similar and/or complementary functionality to MoSDeF. We do not include discussion of commercial tools, as the need to purchase software places a fundamental roadblock in terms of reproducibility.

The Atomic Simulation Environment (ASE)[76] is a Python toolkit that provides wrappers to various programs/libraries allowing atomistic simulations to be setup, run and analyzed within a single script. Support is provided for numerous electronic structure codes and several MD simulation engines; however, as ASE is primarily focused on hard matter systems it does not currently support robust tools for initialization of complex soft matter systems or atom-typing.

`Pysimm`[77, 78], is an open-source Python toolkit for soft matter systems providing routines for system

Listing 3.1: Listing 1. `OpenMM` formatted `XML` file for linear alkanes using the OPLS force field[11].

```
<ForceField>
  <AtomTypes>
    <Type name="opls_135" class="CT" element="C" mass="12.01100" def="[C;X4](C)(H)(H)H"
    desc="alkane CH3" doi="10.1021/ja9621760"/>
    <Type name="opls_136" class="CT" element="C" mass="12.01100" def="[C;X4](C)(C)(H)H"
    desc="alkane CH2" doi="10.1021/ja9621760"/>
    <Type name="opls_140" class="HC" element="H" mass="1.00800" def="H[C;X4]"
    desc="alkane H" doi="10.1021/ja9621760"/>
  </AtomTypes>
  <HarmonicBondForce>
    <Bond class1="CT" class2="CT" length="0.1529" k="224262.4"/>
    <Bond class1="CT" class2="HC" length="0.1090" k="284512.0"/>
  </HarmonicBondForce>
  <HarmonicAngleForce>
    <Angle class1="CT" class2="CT" class3="CT" angle="1.966986067" k="488.273"/>
    <Angle class1="CT" class2="CT" class3="HC" angle="1.932079482" k="313.800"/>
    <Angle class1="HC" class2="CT" class3="HC" angle="1.881464934" k="276.144"/>
  </HarmonicAngleForce>
  <RBTorsionForce>
    <Proper class1="CT" class2="CT" class3="CT" class4="CT" c0="2.9288" c1="-1.4644"
    c2="0.2092" c3="-1.6736" c4="0.0" c5="0.0"/>
    <Proper class1="CT" class2="CT" class3="CT" class4="HC" c0="0.6276" c1="1.8828"
    c2="0.0" c3="-2.5104" c4="0.0" c5="0.0"/>
    <Proper class1="HC" class2="CT" class3="CT" class4="HC" c0="0.6276" c1="1.8828"
    c2="0.0" c3="-2.5104" c4="0.0" c5="0.0"/>
  </RBTorsionForce>
  <NonbondedForce coulomb14scale="0.5" lj14scale="0.5">
    <Atom type="opls_135" charge="-0.18" sigma="0.35" epsilon="0.276144"/>
    <Atom type="opls_136" charge="-0.12" sigma="0.35" epsilon="0.276144"/>
    <Atom type="opls_140" charge="0.06" sigma="0.25" epsilon="0.12552"/>
  </NonbondedForce>
</ForceField>
```

setup and wrappers that support LAMMPS MD[65] and Cassandra MC[59] engines, allowing a simulation workflow to be encoded in a Python script. Of particular note, `pysimm` includes routines that simplify the process for performing complex workflows such as simulated growth/crosslinking of polymers[79]. We note that since both ASE and `pysimm` are also developed as Python libraries, there is a natural level of interoperability between these tools and MoSDeF. `Hoobas` is another open-source molecular building package that facilitates the construction of polymers for molecular dynamics simulation [80, 81]. `indigox` is an open-source package that utilizes the CherryPicker algorithm to help parametrize molecules based on fragments of previously-parametrized molecules [82]. `Open Babel` is a library of cheminformatics functions that support constructing molecular models, SMARTS-matching, and basic molecular dynamics functions with basic molecular mechanics force fields [83, 84]. `OpenKIM` is a multifaceted toolkit providing a portal for storage of interatomic models and their associated data, and an application programming interface (API) created such that models can work seamlessly (and correctly) between different simulation engines; we note this API is designed to ensure parameters are defined correctly, not to perform atom-typing or to encode usage rules and does not provide tools for system initialization or workflow management. To date, `OpenKIM` has largely focused on atomic systems (i.e., a system is defined solely by its atoms, and "bonds" are an outcome of atomic positions), whereas most soft-matter force fields include both non-bonded and bonded parameters and assign different parameters to atoms based on the bonds. The Open Force Field consortium[85–87] has developed a variety of open-source tools that utilize chemical perception via SMIRKS[88] patterns to identify atom types and other force field parameters pertinent to each atom in a chemical system, similar to `Foyer`'s underlying methodology. `WebFF` is an ongoing NIST-project aimed at developing an infrastructure for modeling soft materials and curating force field data for traceable data provenance [89]. `BioSimSpace` provides an API that allows users to mix-and-match various molecular modeling tools, facilitating the use of complex workflows involving molecular dynamics, metadynamics, various water models, various force fields, free energy methods, and various simulation engines[90]. `signac` is a Python library that provides basic components required to create a well-defined and collectively accessible data space and enables data access and modification through a homogeneous data interface that is agnostic to the data source. `signac-flow` is an extension of the `signac` framework[35], which aids in the management of highly complex data spaces. `signac-flow` allows submission to high performance computing (HPC) scheduling systems, including both PBS and SLURM. Since `signac-flow` captures the entire workflow definition and execution, it can be used to facilitate reproducible workflows. `mBuild` and `Foyer` have been used in combination with `signac-flow` in several past and on-going research projects by the authors[8, 9]. `FireWorks` is another workflow manager that supports dynamic workflows using `MongoDB`[91, 92].

3.4 TRUE Molecular Simulations

We have defined TRUE molecular simulations as ones that are transparent, reproducible, usable by others and extensible. In this section, we provide some examples of TRUE simulations utilizing the capabilities of MoSDeF. But first we define what we mean by these terms in the context of molecular simulation.

A simulation is *transparent* when all the information needed to exactly follow the steps undertaken by the original author(s) (such as all scripts used to set up the system, details of force field implementation, all input files to the simulation engines, any other needed input files) are visible to anyone in the community. This requires the sharing of this information in a version-controlled persistent open-source repository, such as GitHub. This information, in and of itself, may only be useful to a true expert; however, few simulations published today meet even this standard. A transparent simulation is *reproducible* when sufficient information (in supplementary information and/or documentation) is provided so that future researchers interested in duplicating the work can construct and run the reported simulation. From this point of view, a self-contained workflow - such as a Jupyter notebook, or a virtual machine - is highly desirable. As defined here, reproducibility does not require a high level of expertise - for example, the calculation could be reproduced by a student in a class, a newcomer to simulation, etc. In particular, Jupyter notebooks provide a convenient, high-level representation of a script that integrates with other common Python tools and can be converted directly into a Python script using `nbconvert`. Two caveats about reproducibility must be borne in mind. First, we note that in molecular simulation, reproducibility is unlikely to be exact, in the following sense: Two MD simulations, when run on different architecture machines, will not generate the same trajectory due to differences in the handling of floating point operations. As in any nonlinear dynamical system, small differences between trajectories (due to different rounding errors) grow exponentially large over time. Even when run on the same computer, two simulations may not give the exact same trajectory. This is because of parallelized computing, in which parts of the calculation are done by separate processors and then gathered (added together) in an order that is not predictable due to fluctuations in message passing times. The problem is exacerbated even more in MC simulations, where a difference in random number seed will generate a different sequence of random numbers on the same machine with the same random number generator. On different machines, trying to achieve reproducibility in MC simulations at the level of configurations on different machines requires using the same random number generator with reproducible arithmetic (IEEE standard-compliant) with the same seed; additionally, the same issue with parallelization noted for MD simulations also applies.[93] However, we do not expect reproducibility at the level of individual simulation trajectories; what we expect is statistical reproducibility in the averages of the properties calculated over the course of the simulation. Second, many simulations that are reported in the literature require prodigious amounts of computational resources, such as millions of hours on a leadership-class supercomputer. In this case, having available all of the codes means that reproducibility is limited to those having available to them similar levels of computing resources. In this case, we propose that researchers may also elect to make available a simplified version of the reported calculation accessible to those that have limited computational resources (for example, using a much smaller system size and shorter simulation times or a single physiochemical statepoint instead of many). Such scaled-down versions could also have considerable pedagogical value.

We define a transferable, reproducible simulation to be *usable by others* when a future researcher can utilize the available files and documentation to reproduce the calculation and make use of the data generated - for example, to analyze the trajectory/trajectories for different properties. This requires a level of documentation that includes information about where output files are located in the data space created by reproducing the simulation, and how these files might be analyzed in different ways. Finally, a transferable, reproducible, usable-by-others simulation is *extensible* if the documentation is sufficiently detailed that a future researcher could change characteristics of the simulation - such as changing molecular species, state conditions, simulation engine, properties calculated, etc.

By adhering to the principles of TRUE simulations, researchers will enable their work to be utilized in ways that have not been hitherto possible. In particular, it will create resources that lower the barrier to entry into the field of molecular simulation, as well as allow researchers to distribute their research in a more useful fashion. Using MoSDeF is not necessary to create TRUE simulations, but as the examples below illustrate, MoSDeF makes it considerably easier to distribute TRUE simulations by automating and standardizing many of the steps, thus minimizing the documentation needed to create a TRUE simulation. Also, the open-source nature of MoSDeF offers the ability for researchers to make contributions to the code base in the form of methods, recipes, force fields, etc.

3.4.1 Ethane VLE using TraPPE

One popular application of molecular simulation involves the use of Monte Carlo (MC) methods, often employing extended ensembles in techniques such as Gibbs Ensemble Monte Carlo (GEMC) or grand canonical Monte Carlo (GCMC), to simulate vapor-liquid equilibria (VLE) properties. Briefly, GEMC involves simulating two distinct simulation boxes (which generally have different densities and compositions) and performing MC moves to perturb both systems to balance the chemical potentials and pressures between the two simulation boxes[94, 95], thus reaching phase equilibrium. This involves particle displacements within boxes, particle exchanges across boxes, and box volume changes[94, 95]. GCMC methods, on the other hand, involve simulating a single simulation box, but performing MC moves to insert or delete particles from a reservoir[96]. Additionally, more complex MC moves have been proposed to accelerate equilibration for systems containing complex molecules, including configurational bias Monte Carlo (CBMC) methods[97]. The transferable potentials for phase equilibria (TraPPE) force field has been designed for conducting simulations for phase equilibria[98, 99]. Here, we present a TRUE workflow that examines ethane vapor-liquid coexistence at a single thermodynamic statepoint. This workflow utilizes `mBuild`[29, 30] to initialize two simulation boxes of ethane (vapor and liquid phases), `Foyer`[31, 68] to apply the TraPPE-United Atom (TraPPE-UA) force field[98], and GOMC[60, 61, 100] to perform a GEMC simulation. `mBuild` is used to pack ethane into two different simulation boxes according to the vapor and liquid densities at 236 K (Figure 3.3).

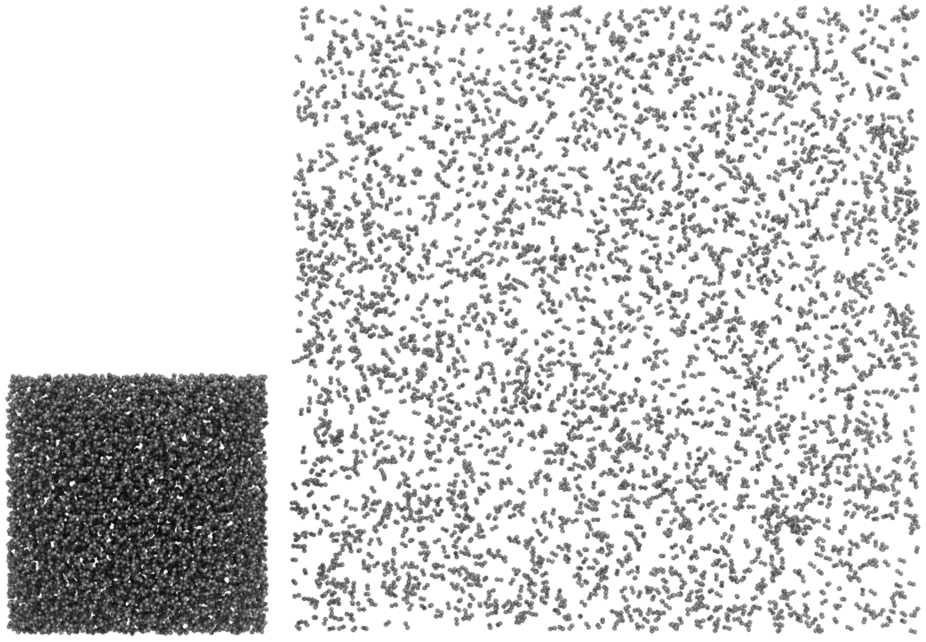


Figure 3.3: Two boxes of ethane constructed in `mBuild`. Liquid phase (left) and vapor phase (right) are simulated simultaneously in GEMC.

`Foyer` is used to systematically apply TraPPE force field parameters. GOMC (version 2.40) is used to perform the GEMC simulation at 236 K, with simulation parameters consistent with the TraPPE implementation[98, 99]: Lorentz-Berthelot combining rules, fixed bonds, 1.4nm Lennard-Jones cutoffs, analytical tail corrections, and Ewald summations for electrostatic interactions. The resultant analysis validates the vapor pressure, vapor density, and liquid density at 236K against published reference data (Figure 3.4)[98, 99]. A link to this GitHub repository can be found in the supporting information. All Python dependencies related to building, simulating, and analyzing are openly available and well-documented, and routines are built on top of these dependencies that expose chemistry and statepoint variables.

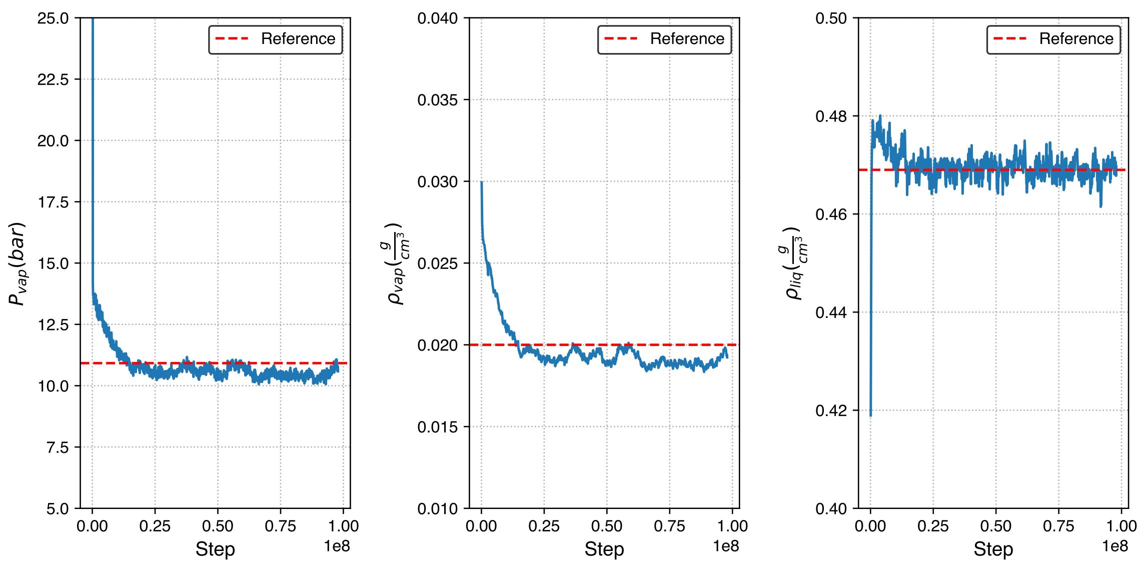


Figure 3.4: Vapor pressure (left), vapor density (middle), and liquid density (right) plots for ethane at 236 K, using GEMC in GOMC with the TraPPE force field.

This workflow is transparent and reproducible, as this workflow and relevant software packages are open-source and available on GitHub[30, 31, 61, 101]. Furthermore, the workflow is usable by others, as the logged quantities can be analyzed for other properties beyond vapor pressure and densities. Lastly, this workflow is extensible, as there is a pattern and clear room to implement other state points, molecules, force fields, or simulation engines in addition to implementing workflow managers to facilitate large-scale screening studies.

3.4.2 Graphene Slit Pore

Graphene has been extensively researched as an electrode material for energy storage applications[102–104] in recent years mainly due to its high surface area[102, 103, 105]. Furthermore, the interactions between graphene pores and fluid molecules were studied with MD simulations through the use of slit pore models[106, 107]. Here we demonstrate a TRUE simulation workflow for a graphene slit pore solvated with aqueous NaCl. This TRUE graphene simulation was performed with the use of `mBuild`[29, 30], `Foyer`[31, 68], GROMACS[39–41, 108–110], and `MDTraj`[111]. `Pore-Builder`[112], an `mBuild` recipe, was also used to initialize the graphene sheets contained in the system.

This specific system, a graphene slit pore filled with aqueous NaCl, was initialized with `mBuild`. `mBuild` compounds of the specific molecules were initialized with the `mbuild.load()` function using `MOL2` files. Once the molecule compounds were initialized, the `GraphenePoreSolvent` class within `Pore-Builder` was utilized. This specific class makes use of the `mbuild.Lattice` class and the `mbuild.solvate` function to build a graphene slit pore system solvated with fluid specified by the user. In this system, the

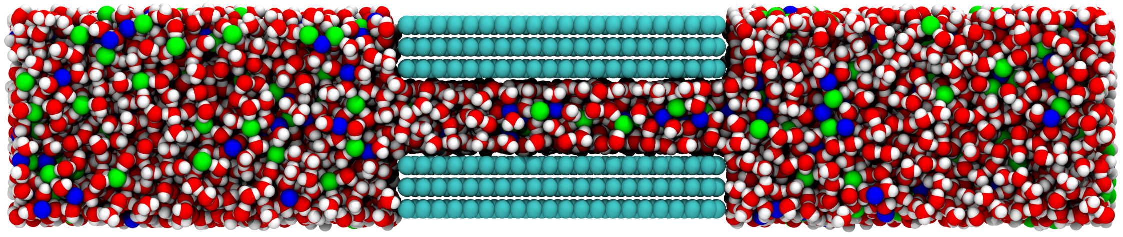


Figure 3.5: A snapshot of the graphene slit pore system containing graphene carbon (cyan), water (red for oxygen and white for hydrogen), sodium ions (blue) and chlorine ions (green).

graphene slit pore was built with three sheets on each side, and a pore width of 1.5nm. Additionally, the length of the graphene sheet in the x direction was set to 5 nm and the length of the graphene sheet in the z direction was set to 4 nm. The bulk region of fluid was set to 6nm on each side of the slit pore. 5200 waters and 400 Na and Cl ions were solvated into the system. A snapshot of the system is shown in Figure 3.5.

Once the graphene slit pore system was initialized as an `mBuild` compound, it was atom-typed and parametrized with `Foyer`. Three force fields were used in this system, with their information stored in three separate `XML` files: GAFF[12], SPC/E[113], and the force field of Joung and Cheatham (JC)[114]. GAFF describes the interactions between the graphene atoms, SPC/E describes the water interactions, and JC describes the Na and Cl interactions. Each force field uses 12-6 Lennard-Jones interactions, point charges, and harmonic bonds and angles.

The simulation was run with GROMACS 2018.5. Steepest descent energy minimization was first performed for 1000 steps to remove any energetic clashes from the initial configuration. Afterwards, a series of two MD simulations were performed with the following parameters: cutoffs of 1.4nm for Coulombic and van der Waals interactions, a temperature of 300K controlled with the v-rescale thermostat with a time constant of 0.1ps, particle mesh Ewald to handle long-range electrostatics, and a timestep of 1fs. Additionally, the graphene atoms were frozen in place. A GROMACS `NDX` file was created with a `Water_and_ions` group so that the thermostat could be applied to the fluids; no thermostat is applied to the graphene, as the graphene is kept rigid. First NVT equilibration was performed to further relax the system of any unfavorable configurations for 100,000 steps. Afterwards, NVT sampling was performed for 2,500,000 steps. In the sampling run, all bonds were constrained using the LINCS[115] algorithm.

Once the sampling simulation was performed, the number density profile of each fluid type is calculated with the use of `MDTraj` and plotted with `Matplotlib`. The results are shown in Figure 3.6. From these results, we observe that the water molecules are mainly structured near the pore walls at 1.2nm and 2.0nm. Additionally, there are two smaller peaks around 1.4 and 1.8nm indicating structuring of water in the middle

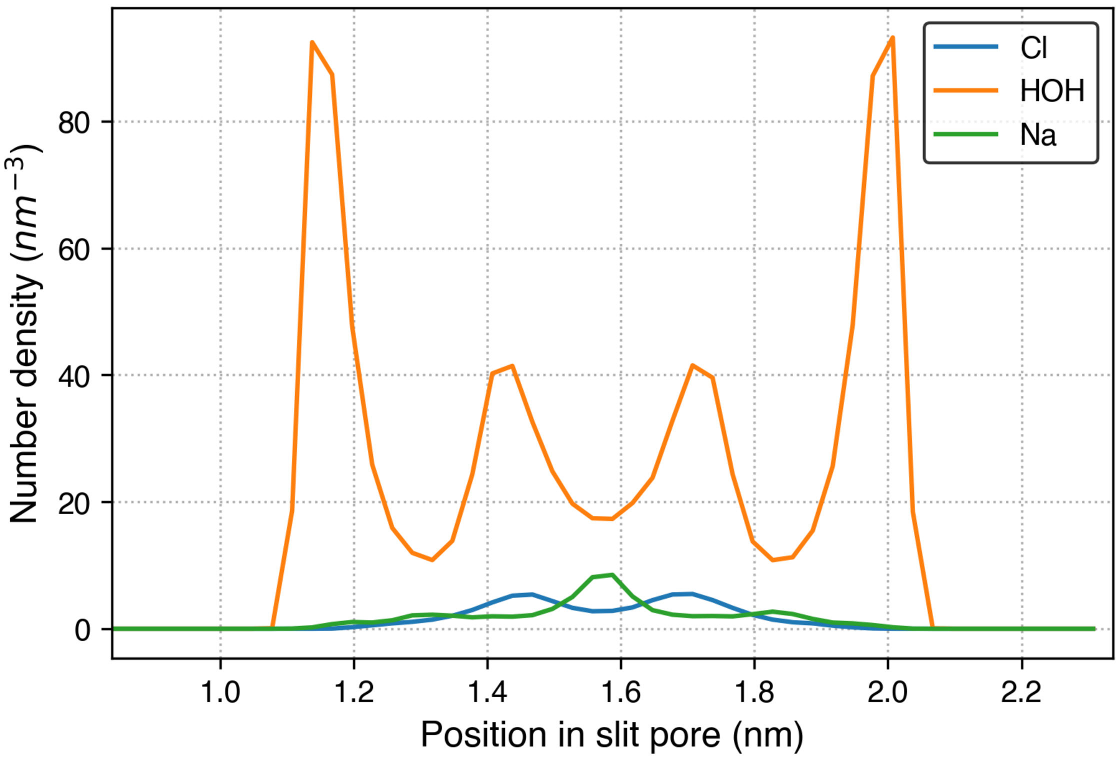


Figure 3.6: Number Density Profiles across the width of the pore for water, Na, and Cl.

of the pore. The Na ions are structured in the middle of the pore around 1.6nm and the the Cl ions are structured to each side of the Na ions, at around 1.5 and 1.7 nm. If the graphene was positively or negatively charged, we would expect different structure behavior of the ions. This simulation can be extended to further understand the effect of various parameters on the fluid structure within the pore. For example, the user can easily specify a different pore width to study how this impacts the structure of water and ions. This workflow is encapsulated in a Jupyter notebook, providing the user access to modify any of these high-level parameters.

The workflow for simulating a graphene slit pore satisfies the conditions to be a TRUE simulation. First, this workflow is transparent as all scripts, input files, and force field information are available for anyone to view[116]. Next, this workflow is reproducible as the exact steps to set up and run the simulation are contained within a Jupyter notebook. Barring differences in computer architectures and parallelization schemes, a user running this Jupyter notebook should be able to reproduce the number density profiles from the reference simulation. Additionally, the trajectories are kept within the workflow directory, allowing users to analyze properties other than number density. Finally, the functions and classes used to initialize the graphene slit pore system are sufficiently documented so that a user may change characteristics of the simulation if they wish. For example, a user can extend this workflow to study additional aqueous solutions.

3.4.3 Lipid Bilayers

MD is a common technique used to perform simulation of biological systems. An example TRUE biological simulation workflow is demonstrated in the `true_lipids` repository on GitHub[117]. This workflow focuses on simulating lipids found in the outermost layer of the skin, the stratum corneum (SC). The SC, which is primarily composed of ceramides (CER), cholesterol (CHOL), and free fatty acids (FFA) [118], essentially controls the barrier function of the skin [119]. In this workflow a hydrated pre-assembled bilayer of skin lipids configuration was initialized, simulated, and analyzed in a well-documented and reproducible fashion.

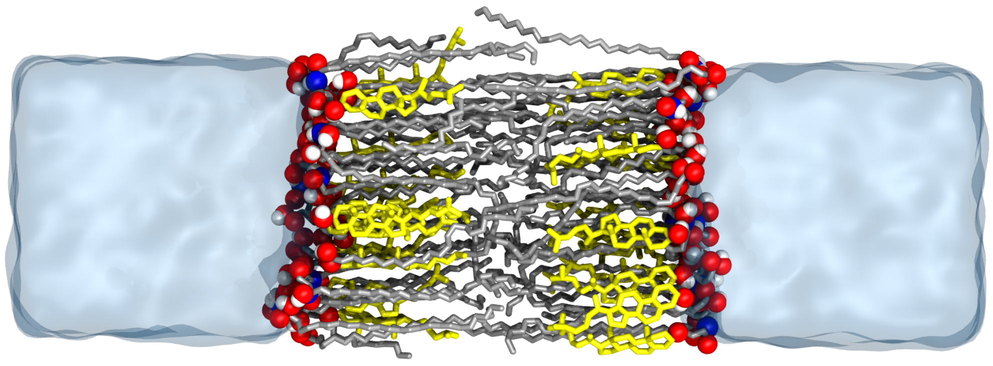


Figure 3.7: Simulation snapshot of lipid bilayer containing CER N-hydroxysphingosine C24:0 (CER NS), cholesterol and lignoceric acid. The CER NS and FFA tails are shown in silver, cholesterol in yellow, and the headgroup oxygen, nitrogen and hydrogen atoms in red, blue and white respectively.

`mBuild` was used to initialize the system configuration, specifically utilizing the `Bilayer`[120] recipe. A simplified model system containing only CER N-hydroxysphingosine (NS) C24:0, CHOL, and FFA C24:0 was chosen for this example; however a more complex mixture could be easily built by the `Bilayer` recipe. For each leaflet of the bilayer, 36 lipids were randomly placed on a 6x6 lattice and rotated about the bilayer normal axis. The lattice was set up and spaced such that the lateral area occupied by each lipid was equal to the target and as designated by the `area_per_lipid` parameter. In addition, the lipids were rotated about a randomly chosen axis parallel to the bilayer by the tunable `tilt_angle` parameter. Finally, 20 waters per lipid were added to each of the two ends of the simulation box at a density of $1\,\mathrm{g\,cm^{-3}}$. The full system contains 72 lipids and 2880 water molecules. While many of the steps involved in setting up the initial configuration involve random number generators, exact reproducibility on the same machine was enforced by the initialization of a random seed.

Simulations were conducted using the GROMACS 2018.5 [39–41, 108–110] MD engine, using a modified CHARMM36 force field [121, 122] with a 1fs timestep. The system was first energy-minimized using the steepest descent algorithm for 20000 steps in order to remove high energy atomic contacts. Temperature fluctuations were stabilized by running a 500 ps NVT simulation using the Berendsen thermostat [123] at 305 K with a time constant of 1ps. Next, the volume fluctuations were stabilized with a 10ns NPT simulation at 305K and 1atm. This step and all others hereinafter in this section were in the NPT ensemble and use the Nosé-Hoover thermostat [124] with a time constant of 1 ps and the Parinello-Rahman barostat [125] with a time constant of 10 ps and a compressibility of $4.5 \times 10^{-5}\text{bar}^{-1}$. Still at 1 atm, the system was linearly annealed to 340 K over 5 ns, held at 340 K for 15 ns, linearly cooled to 305 K over 5 ns, and held at 305 K for 25 ns in order to accelerate equilibration of the rotational orientation of the lipids. Finally, the system was run for 20 ns at 305 K and 1 atm, saving coordinates to a trajectory file every 10 ps. The final snapshot of the system is shown in Figure 3.7. More details on the simulation parameters can be found in Appendix B.

The trajectory from the final step was analyzed using `MDTraj` [111]. Neutron scattering is a popular tool to experimentally obtain structural information of lipid lamella. A neutron scattering length density (NSLD) profile was calculated for this simulated system along the bilayer normal axis in Figure 3.8.

It is apparent from these profiles that the 24-carbon fatty acid tail of the CER and the 24-carbon FFA tail interdigitate, as indicated by the high density peak in the center of the profile. One can also observe that the 16-carbon sphingosine tail of the CER and CHOL do not interdigitate, and are not present in the middle of the bilayer as there is a low-density trough in their deuteration profiles. The scattering length densities at the outer edges of the bilayer suggest that the CHOL headgroup is located closer to the center of the bilayer compared to that of other lipids. In addition to the NSLD profiles, an area per lipid of 32.90 Å^2, a tail tilt angle 10.8°, a nematic order parameter of 0.9414 and a bilayer height of 48.13 Å were calculated in the workflow.

All of these values and plots can be reproduced by executing the workflow. Furthermore, by extending the workflow to screen over the parameter space, one could identify trends in the calculated values. The `Bilayer` recipe is highly modular allowing the user to easily create reproducible bilayer structures containing different lipid types, system sizes, compositions, or water content using an intuitive Python script.

3.4.4 Friction Reduction Via Thin Film Coatings

Thin film coatings can be used to modify the surface properties of different systems[126]. One potential application of these coatings is to improve tribological properties of surfaces at the micro and nanoscale[37, 126, 127]. In this example, we present a TRUE simulation of a thin film coated system, which is based on an in-depth study by Summers *et al.*[37]. Specifically, we built a system of two 50×50 rectangular silica

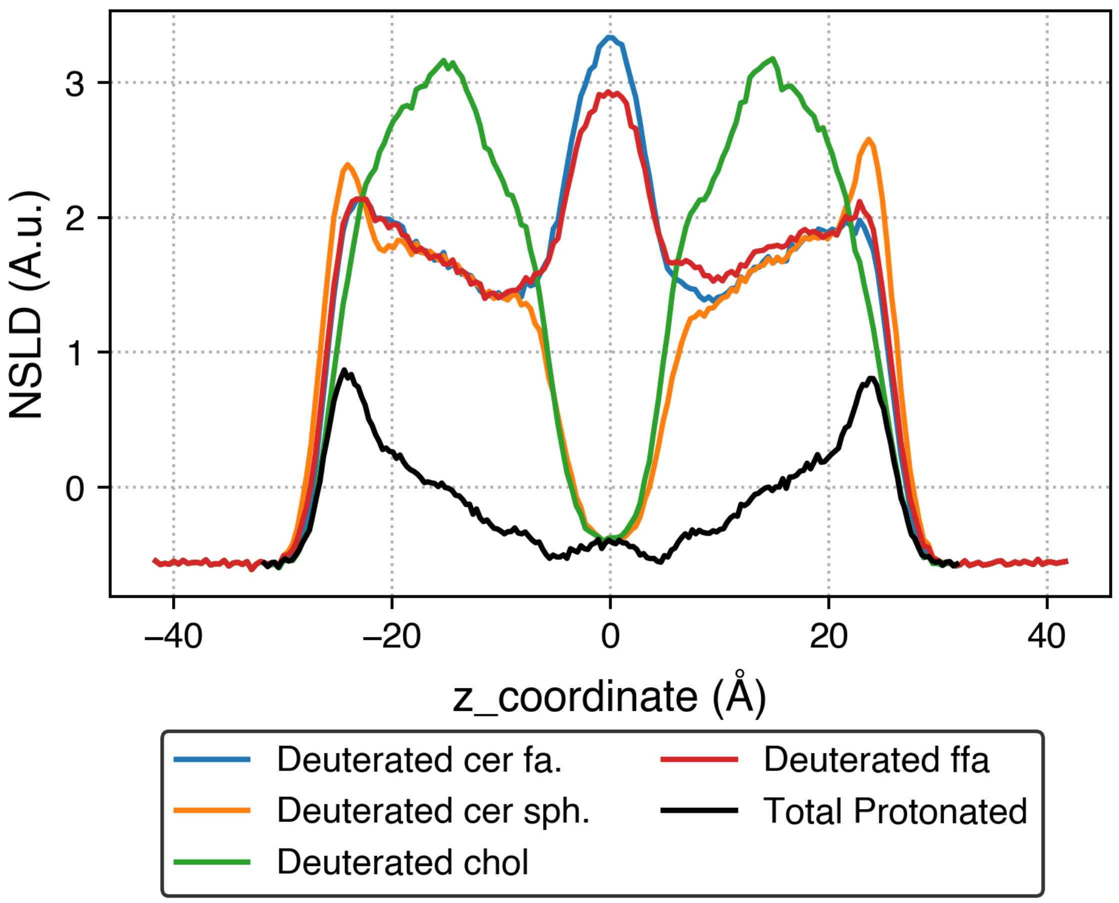


Figure 3.8: Simulated NSLD profiles for specifically deuterated lipid tails.

surfaces, parallel to one another. Each surface was coated with 100, 17-carbon long, alkylsilane chains, all of which were terminated with a methyl group. Surface oxygens not functionalized with chains were backfilled with hydrogen caps to form protonated hydrolysis. These systems were created with `mBuild`[29, 30], and atom-typed, parametrized with `Foyer`[31, 68] using OPLS-aa[11] force field parameters. A visualization of the described system is presented in Figure 3.9.

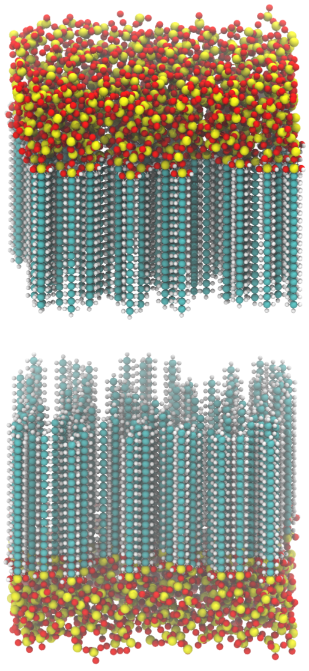


Figure 3.9: Thin film coated surfaces model

The system was simulated with LAMMPS[42] and GROMACS[39–41, 108–110]. MD simulations were run under the canonical ensemble (NVT) and a Nosé-Hoover thermostat maintaining the temperature at 298K[124]. Long-range electrostatics were calculated using the particle-particle particle-mesh (PPPM) algorithm [128]. The `rRESPA` time step algorithm was utilized with 0.25 fs, 0.5 fs, 0.5 fs, and 1.0 fs timesteps for bonds, angles, dihedrals, and non-bonded interactions, respectively[129]. The simulation started with energy minimization with LAMMPS for 10,000 steps, followed by another 50,000 steps with GROMACS to bring the monolayers to a more relaxed state. This process continued with NVT equilibration by GROMACS to bring the system to a desired stable state for 1,000,000 steps. The workflow proceeded to use GROMACS to compress the system by pulling the top surfaces along the z axis to come into contact with the bottom surface. In the next step, the top surface was sheared against the bottom surface by imposing a force to pull it along the z axis at the rate of 0.01 $\frac{nm}{ps}$. The production run was designed to simulate for 5,000,000 steps, which would be equivalent to 10 ns of shearing. From the GROMACS output file, the properties of the system can be calculated. The steady-state production period used for final data analysis was determined using the automatic equilibration detection method provided by `pymbar`[130, 131]. By using a defined method to determine the steady-state cutoff, the consistency of the data analysis process can be ensured, holding to the first two criteria of TRUE, transparent and reproducible. The calculated nematic order of three example runs were determined and are presented in Figure 3.10.

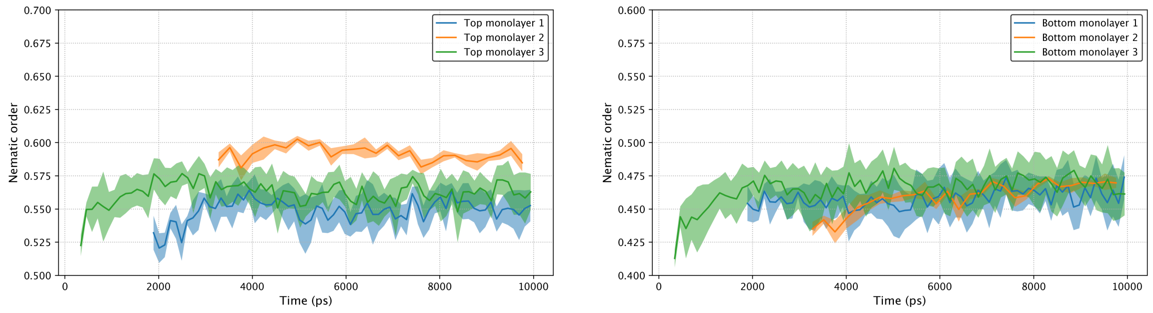


Figure 3.10: Steady state nematic order of the thin film coated on top and bottom surfaces

This example focuses on showcasing the extensibility of TRUE, emphasizing the ability to modify and expand the project beyond the original study and parameters of interest. This goal can be achieved by applying Object-Oriented Programming (OOP) design principles, in combination with open-source libraries. Encapsulating frequently used code into classes and functions helps improve the reusability of codes and make it easier to create novel systems, just by changing and adding new variables. By pairing MoSDeF suite libraries, `mBuild`[29, 30] and `Foyer`[31, 68], with other open-source libraries, such as `signac` and `signac-flow`, part of the `signac` framework[34, 132], the extensibility could be made even more manageable and achievable. Most importantly, all building blocks of the project have to be properly documented, either as comments in the code or in a separate manual. These practices can help projects expand effectively. For instance, in this example, although the arguments and variables were defined such that the surfaces were filled with 100, 17-carbon long, alkylsilane chains, each then capped with a methyl group, many unique systems can be created by altering the chain density, backbone chain length, backbone chemistries, terminal groups, and others as need arises. The latter part of the example shows how we can expand the project from the original system by varying backbone chain lengths. For the sake of demonstration, we only show the first few steps of the workflow, starting with setting up the workspace using `signac`[34, 132], building corresponding systems with `mBuild`[29, 30], and atom-typing, parametrizing with `Foyer`[31, 68]. Other steps of the simulation can be added analogously. We implement `signac-flow`[34, 132] as the workflow manager. These tools will become vital when needing to run a complete workflow and managing thousands of systems. All scripts and files needed to run the above example are located in a GitHub repository[133].

Users can interface with this example through the Jupyter notebook located within the repository. By providing properly documented codes and scripts used to set up the system, using open-source libraries to perform simulation and data analysis, the first three criteria of TRUE are also satisfied. This example workflow is an instance of a Transparent, Reproducible, Usable by others, and Extensible, or concisely, TRUE simulation.

3.5 Conclusions

In this paper, we have outlined some of the key issues related to reproducibility in molecular simulations of soft matter. We have also discussed many practices that computational scientists could implement in efforts to result in more reproducible science, such as using scripts instead of manual input, using open-source software tools, and using version control and modern software development practices when developing software. In this paper, we assert three central claims:

- The goal in computational molecular science should be simulations that are TRUE: Transparent, Reproducible, Usable by others, and Extensible.
- Scientific results reported in the literature that depend on molecular simulations should adhere to the above characteristics.
- Use of the Molecular Simulation and Design Framework (MoSDeF) is one way to enable TRUE simulations.

To demonstrate the second claim, we revisit some "ten rules" papers[21–23] that provide succinct instructions for practicing reproducible science and demonstrate how the above example workflows utilize MoSDeF to this end. A common recommendation in these discussions is that every step in a workflow should be automated and free of manual input, i.e. scriptable. MoSDeF, in its current state, is a set of Python libraries designed specifically to address this. In a single Python script (or Jupyter notebook), each step of a molecular simulation workflow (generation of particle coordinates, application of a force field, running of a molecular simulation, and analysis of the results) can be specified and run. The objective of measuring physical properties from some chemical input can be achieved with one call to an executable (although the simulation may take some hours or days to run). In order for these scripts, which include many imports to other Python libraries, to produce identical (or sufficiently identical) results some years in the future, the underlying libraries must be version-controlled. The core MoSDeF packages (`mBuild` and `Foyer`) undergo regular releases, tagged with semantic version numbers, every few weeks or months as they are developed. Other packages, such as simulation engines, the packages in the signac framework, and underlying scientific Python packages, are also version-controlled and undergo regular releases. Specifying the version of each software package used in a simulation workflow is not necessarily sufficient to ensure reproducible science, but it is a significant improvement over the use of *ad hoc* or in-house scripts that often lack version control, proper testing, or releases. Similarly, it has been argued that the use of community-developed software libraries, and the extension of such libraries, further promotes reproducibility as compared to closed-source, in-house development[20]. MoSDeF is a set of open-source that interface with other open-source, community-developed libraries and software tools.

Additionally, MoSDeF makes use of virtually no GUIs - or, more specifically, no GUIs that hide the details of a simulations protocol from the user. Some molecular visualization tools (`NGLview`, `py3DMol`, `VMD`, `ovito`, `fresnel`) can be used in conjunction with MoSDeF, but these are only tools to visualize systems and do not hide workflow details or replace steps in a workflow.

Finally, we would like to discuss an additional benefit of shifting toward more reproducible computational studies: the facilitation of large-scale screening of physiochemical space. Continuous improvements in computer hardware and recent advancements in machine learning methodologies have driven interest in studying large data sets, typically many orders of magnitude larger than typically seen in the literature. Provided that each step in a workflow can be automated - in other words, scriptable with no manual input - a single simulation can be repeated with different physical inputs (e.g. at different thermodynamic statepoints or with different chemistries) by only modifying the input parameters. For example, consider some system at temperature and pressure (T,P) for which we care about some physical property A. One can run a simulation at (T_1,P_1) and get property A_1 but later decide we want to look at some other temperature and/or pressure. One could manually move some files around and get property A_2 from statepoint (T_2,P_2) without prohibitive trouble, but doing this once is a plausible source of human error and repeating this process many times is not feasible. Screening over N statepoints in a reproducible manner necessitates that running a workflow at a single statepoint is reproducible. We hope the practices outlined in this paper and the use of MoSDeF can enable reproducible computational science at each scale.

3.6 Acknowledgments

The development of MoSDef, as reported in this paper, was primarily supported by the National Science Foundation (NSF) through grant OAC-1835874 “Software for Building a Community-Based Molecular Modeling Capability Around the Molecular Simulation Design Framework (MoSDeF).” Earlier developments leading to MoSDeF were supported by previous NSF grants CBET-1028374 "Cyber-Enabled Design of Functional Nanomaterials", OAC-1047828 “Development of an Integrated Molecular Design Environment for Lubrication Systems (iMoDELS)” and OAC-1535150 “Development of a Software Framework for Formalizing Forcefield Atom-Typing for Molecular Simulation.” The development of code within MoSDeF specifically relevant to energy storage systems and the example described in subsection 3.4.2 were supported as part of the Fluid Interface Reactions, Structures and Transport (FIRST) Center, an Energy Frontier Research Center funded by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences. The development of code within MoSDeF specifically relevant to self-assembling lipid systems and the example described in subsection 3.4.3 were supported by National Institute of Arthritis and Musculoskeletal and Skin Diseases grant R01AR072679 "Insights into Skin Barrier Function: In Silico and Experimental Studies of Healthy and Diseased Stratum Corneum Lipid Models." We also acknowledge key contributions to MoSDeF by former group members Christoph Klein (myriad.com) and Andrew Z. Summers (enthought.com) and by ISIS staff Peter Volgesi, Umesh Timalsina, and Janos Sallai.

References

1. Thompson, M. W., Gilmer, J. B., Matsumoto, R. A., Quach, C. D., Shamaprasad, P., Yang, A. H., Iacovella, C. R., McCabe, C. & Cummings, P. T. Towards Molecular Simulations That Are Transparent, Reproducible, Usable by Others, and Extensible (TRUE). *Molecular Physics* **118,** e1742938. ISSN: 0026-8976, 1362-3028 (June 2020).

2. Baker, M. 1,500 scientists lift the lid on reproducibility. *Nature* **533,** 452–454. ISSN: 0028-0836, 1476-4687 (May 2016).

3. Tadmor, E. B., Elliott, R. S., Phillpot, S. R. & Sinnott, S. B. NSF cyberinfrastructures: A new paradigm for advancing materials simulation. *Current Opinion in Solid State and Materials Science* **17,** 298–304. ISSN: 1359-0286. http://linkinghub.elsevier.com/retrieve/pii/S1359028613000971 (Dec. 2013).

4. Jain, A., Errington, J. R. & Truskett, T. M. Communication: Phase Behavior of Materials with Isotropic Interactions Designed by Inverse Strategies to Favor Diamond and Simple Cubic Lattice Ground States. *The Journal of Chemical Physics* **139,** 141102. ISSN: 0021-9606, 1089-7690 (Oct. 2013).

5. Wilmer, C. E., Leaf, M., Lee, C. Y., Farha, O. K., Hauser, B. G., Hupp, J. T. & Snurr, R. Q. Large-Scale Screening of Hypothetical Metal-Organic Frameworks. *Nature Chemistry* **4,** 83. ISSN: 1755-4349. http://www.ncbi.nlm.nih.gov/pubmed/22270624 (Feb. 2012).

6. Hachmann, J., Olivares-Amaya, R., Atahan-Evrenk, S., Amador-Bedolla, C., Sanchez-Carrera, R. S., Gold-Parker, A., Vogt, L., Brockway, A. M. & Aspuru-Guzik, A. The Harvard Clean Energy Project: Large-Scale Computational Screening and Design of Organic Photovoltaics on the World Community Grid. *The Journal of Physical Chemistry Letters* **2,** 2241–2251. ISSN: 1948-7185 (Sept. 2011).

7. Afzal, M. A. F., Haghighatlari, M., Ganesh, S. P., Cheng, C. & Hachmann, J. Accelerated Discovery of High-Refractive-Index Polyimides via First-Principles Molecular Modeling, Virtual High-Throughput Screening , and Data Mining. *The Journal of Physical Chemistry C* **123,** 14610–14618. ISSN: 1932-7447 (June 2019).

8. Thompson, M. W., Matsumoto, R., Sacci, R. L., Sanders, N. C. & Cummings, P. T. Scalable Screening of Soft Matter: A Case Study of Mixtures of Ionic Liquids and Organic Solvents. *The Journal of Physical Chemistry B* **123,** 1340–1347. ISSN: 1520-6106 (Feb. 2019).

9. Matsumoto, R. A., Thompson, M. W. & Cummings, P. T. Ion Pairing Controls Physical Properties of Ionic Liquid-Solvent Mixtures. *The Journal of Physical Chemistry B* **123,** acs.jpcb.9b08509. ISSN: 1520-6106 (2019).

10. OSTP. *Materials Genome Initiative for Global Competitiveness* tech. rep. (2011).

11. Jorgensen, W. L., Maxwell, D. S. & Tirado-Rives, J. Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids. *J. Am. Chem. Soc.* **118,** 11225–11236. ISSN: 0002-7863 (Nov. 1996).

12. Wang, J., Wolf, R. M., Caldwell, J. W., Kollman, P. A. & Case, D. A. Development and testing of a general Amber force field. *Journal of Computational Chemistry* **25,** 1157–1174. ISSN: 0192-8651 (2004).

13. Jo, S., Kim, T. & Im, W. Automated Builder and Database of Protein/Membrane Complexes for Molecular Dynamics Simulations. *PLoS ONE* **2** (ed Yuan, A.) e880. ISSN: 1932-6203 (Sept. 2007).

14. Schappals, M., Mecklenfeld, A., Kröger, L., Botan, V., Köster, A., Stephan, S., García, E. J., Rutkai, G., Raabe, G. & Klein, P. Round Robin Study: Molecular Simulation of Thermodynamic Properties from Models with Internal Degrees of Freedom. *Journal of Chemical Theory and Computation* **13,** 4270. ISSN: 1549-9618 (Sept. 2017).

15. Chen, B., Siepmann, J. I., Karaborni, S. & Klein, M. L. Vapor-Liquid and Vapor-Solid Phase Equilibria of Fullerenes: The Role of the Potential Shape on the Triple Point. *The Journal of Physical Chemistry B* **107,** 12320–12323. ISSN: 1520-6106 (Nov. 2003).

16. Silva Fernandes, F. M. S., Freitas, F. F. M. & Fartaria, R. P. S. Phase Diagram and Sublimation Enthalpies of Model C 60 Revisited. *The Journal of Physical Chemistry B* **108,** 9251–9255. ISSN: 1520-6106 (July 2004).

17. Palmer, J. C., Haji-Akbari, A., Singh, R. S., Martelli, F., Car, R., Panagiotopoulos, A. Z. & Debenedetti, P. G. Comment on "The putative liquid-liquid transition is a liquid-solid transition in atomistic models of water" [I and II: J. Chem. Phys. 135 , 134503 (2011); J. Chem. Phys. 138, 214504 (2013)]. *J. Chem. Phys.* **148,** 137101. ISSN: 0021-9606 (2018).

18. Smart, A. G. *The war over supercooled water* Aug. 2018. (2019).

19. Jo, S., Kim, T., Iyer, V. G. & Im, W. CHARMM-GUI: A web-based graphical user interface for CHARMM. *Journal of Computational Chemistry* **29,** 1859–1865. ISSN: 0192-8651 (Aug. 2008).

20. Donoho, D. L., Maleki, A., Rahman, I. U., Shahram, M. & Stodden, V. Reproducible Research in Computational Harmonic Analysis. *Computing in Science Engineering* **11,** 8–18. ISSN: 1521-9615 (2009).

21. Sandve, G. K., Nekrutenko, A., Taylor, J. & Hovig, E. Ten simple rules for reproducible computational research. *PLoS Comput. Biol.* **9** (ed Bourne, P. E.) e1003285. ISSN: 1553-734X (Oct. 2013).

22. Elofsson, A., Hess, B., Lindahl, E., Onufriev, A., van der Spoel, D. & Wallqvist, A. Ten simple rules on how to create open access and reproducible molecular simulations of biological systems. *PLoS Computational Biology* **15,** 2–5. ISSN: 1553-7358 (2019).

23. Barba, L. A. The hard road to reproducibility. *Science* **354,** 142–142. ISSN: 0036-8075 (Oct. 2016).

24. Barba, L. A. *Lorena A. Barba blog* http://lorenabarba.com/category/blog/.

25. Reinhart, C. M. & Rogoff, K. S. Growth in a Time of Debt. *Am. Econ. Rev.* **100,** 573–578. ISSN: 0002-8282 (Aug. 2010).

26. Herndon, T., Ash, M. & Pollin, R. Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. *Cambridge J. Econ.* **38,** 257–279. ISSN: 0309-166X (Aug. 2014).

27. Contributors, M. *MoSDeF Webpage* https://mosdef.org. Accessed: 2022-03-13.

28. *Institute for Software Integrated Systems* https://www.isis.vanderbilt.edu/ (2019).

29. Klein, C., Sallai, J., Jones, T. J., Iacovella, C. R., McCabe, C. & Cummings, P. T. in *Foundations of Molecular Modeling and Simulation. Molecular Modeling and Simulation (Applications and Perspectives)* (eds Snurr, R. Q., Adjiman, C. S. & Kofke, D. A.) 79–92 (Springer, Singapore, Singapore, 2016). http://link.springer.com/10.1007/978-981-10-1128-3%7B%5C_%7D5%20http://dx.doi.org/10.1007/978-981-10-1128-3%7B%5C_%7D5.

30. *mBuild Github repository* https://github.com/mosdef-hub/mbuild (2018).

31. *Foyer Github repository* https://github.com/mosdef-hub/foyer (2018).

32. Iacovella, C. R., Sallai, J., Klein, C. & Ma, T. *Idea Paper : Development of a Software Framework for Formalizing* 2016.

33. Black, J. E., Silva, G. M. C., Klein, C., Iacovella, C. R., Morgado, P., Martins, L. F. G., Filipe, E. J. M. & McCabe, C. Perfluoropolyethers: Development of an All-Atom Force Field for Molecular Simulations and Validation with New Experimental Vapor Pressures and Liquid Densities. *The Journal of Physical Chemistry B* **121,** 6588–6600. ISSN: 1520-6106 (July 2017).

34. Adorf, C. S., Dodd, P. M., Ramasubramani, V. & Glotzer, S. C. Simple data and workflow management with the signac framework. *Computational Materials Science* **146,** 220–229. ISSN: 0927-0256. https://authors.elsevier.com/a/1WbPL3In-uhIuD%20http://linkinghub.elsevier.com/retrieve/pii/S0927025618300429%20https://www.mendeley.com/research-papers/simple-data-workflow-management-signac-framework/ (Feb. 2018).

35. *Signac documentation* https://signac.readthedocs.io/en/latest/ (2018).

36. Summers, A. Z., Iacovella, C. R., Cummings, P. T. & McCabe, C. Investigating Alkylsilane Monolayer Tribology at a Single-Asperity Contact with Molecular Dynamics Simulation. *Langmuir* **33,** 11270–11280. ISSN: 0743-7463 (Oct. 2017).

37. Summers, A. Z., Gilmer, J. B., Iacovella, C. R., Cummings, P. T. & McCabe, C. MoSDeF, a Python Framework Enabling Large-Scale Computational Screening of Soft Matter: Application to Chemistry-Property Relationships in Lubricating Monolayer Films. *Journal of Chemical Theory and Computation* **0.** PMID: 32004433, null. ISSN: 1549-9618. eprint: https://doi.org/10.1021/acs.jctc.9b01183 (Mar. 0).

38. *MoSDeF-Anaconda Cloud* https://anaconda.org/mosdef/ (2018).

39. Hess, B., Kutzner, C., van der Spoel, D. & Lindahl, E. GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *J. Chem. Theory Comput.* **4,** 435–447. ISSN: 1549-9618 (2008).

40. Berendsen, H. J. C., van der Spoel, D. & van Drunen, R. GROMACS: A message-passing parallel molecular dynamics implementation. *Comput. Phys. Commun.* **91,** 43–56. ISSN: 0010-4655 (Sept. 1995).

41. Abraham, M. J., Murtola, T., Schulz, R., Pá ll, S., Smith, J. C., Hess, B. & Lindahl, E. GROMACS: High Performance Molecular Simulations through Multi-Level Parallelism from Laptops to Supercomputers. *SoftwareX* **1-2,** 19–25. ISSN: 2352-7110 (Sept. 2015).

42. Plimpton, S. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics* **117,** 1–19. ISSN: 0021-9991. http://www.sciencedirect.com/science/article/pii/S002199918571039X (Mar. 1995).

43. Glaser, J., Nguyen, T. D., Anderson, J. A., Lui, P., Spiga, F., Millan, J. A., Morse, D. C. & Glotzer, S. C. Strong Scaling of General-Purpose Molecular Dynamics Simulations on GPUs. *Computer Physics Communications* **192,** 97–107. ISSN: 0010-4655 (July 2015).

44. *HOOMD-blue* http://glotzerlab.engin.umich.edu/hoomd-blue. 2017. http://glotzerlab.engin.umich.edu/hoomd-blue (2018).

45. Anderson, J. A., Glaser, J. & Glotzer, S. C. HOOMD-blue: A Python Package for High-Performance Molecular Dynamics and Hard Particle Monte Carlo Simulations. *Computational Materials Science* **173,** 109363. ISSN: 0927-0256 (Feb. 2020).

46. Shah, J. K., Marin-Rimoldi, E., Mullen, R. G., Keene, B. P., Khan, S., Paluch, A. S., Rai, N., Romanielo, L. L., Rosch, T. W., Yoo, B. & Maginn, E. J. Cassandra: An Open Source Monte Carlo Package for Molecular Simulation. *Journal of Computational Chemistry* **38,** 1727–1739. ISSN: 0192-8651, 1096-987X (July 2017).

47. Contributors, P. *ParmEd: Parameter/topology editor and molecular simulator* https://github.com/ParmEd/ParmEd. Accessed: 2022-03-15.

48. Sztipanovits, J. & Karsai, G. Model-integrated computing. *Computer* **30,** 110–111 (1997).

49. Iacovella, C. R., Varga, G., Sallai, J., Mukherjee, S., Ledeczi, A. & Cummings, P. T. A model-integrated computing approach to nanomaterials simulation. *Theoretical Chemistry Accounts* **132,** 1315. ISSN: 1432-881X (Jan. 2013).

50. *Travis CI: A hosted continuous integration service* http://travis-ci.org (2018).

51. *Implement continuous integration and continuous delivery (CI/CD) for the app and platform of your choice.* https://azure.microsoft.com/en-us/services/devops/pipelines/ (2019).

52. *CodeCov landing page* https://codecov.io/.

53. (SPDX), T. S. P. D. E. *MIT License* https://spdx.org/licenses/MIT.html.

54. *Slack virtual shared workplace tool* https://slack.com (2018).

55. Davenport, M. How Slack-ing helps chemists manage their labs. *Chemical & Engineering News* **94,** 23–24. https://cen.acs.org/articles/94/i29/Slack-ing-helps-chemists-manage.html (2016).

56. Humphrey, W., Dalke, A. & Schulten, K. VMD: Visual molecular dynamics. *Journal of Molecular Graphics* **14,** 33–38. ISSN: 0263-7855. http://www.ncbi.nlm.nih.gov/pubmed/8744570%20http://linkinghub.elsevier.com/retrieve/pii/0263785596000185 (Feb. 1996).

57. Anderson, J. A. & Travesset, A. Molecular dynamics on graphic processing units: HOOMD to the rescue. *Computing in Science & Engineering* **10** (2008).

58. *Collaborative Research: NSCI Framework: Software for Building a Community-Based Molecular Modeling Capability Around the Molecular Simulation Design Framework (MoSDeF)* https://www.nsf.gov/awardsearch/showAward?AWD%7B%5C_%7DID=1835874.

59. Shah, J. K., Marin-Rimoldi, E., Mullen, R. G., Keene, B. P., Khan, S., Paluch, A. S., Rai, N., Romanielo, L. L., Rosch, T. W., Yoo, B. & Maginn, E. J. Cassandra: An open source Monte Carlo package for molecular simulation. *Journal of Computational Chemistry* **38,** 1727–1739. ISSN: 0192-8651 (July 2017).

60. *GPU-Optimized Monte Carlo (GOMC) Home Page* Apr. 2018. http://gomc.eng.wayne.edu/ (2018).

61. GOMC. *GPU-Optimized Monte Carlo (GOMC) github repository* Apr. 2018. https://github.com/GOMC-WSU/GOMC.

62. *CP2K Open Source Molecular Dynamics* Apr. 2018. https://www.cp2k.org/ (2018).

63. *Signac-flow webpage* http://signac-flow.readthedocs.io (2018).

64. Salomon-Ferrer, R., Case, D. A. & Walker, R. C. An overview of the Amber biomolecular simulation package. *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **3,** 198–210. ISSN: 1759-0876. http://onlinelibrary.wiley.com/doi/10.1002/wcms.1121/full%20http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.600.8472%7B%5C&%7Drep%20=rep1%7B%5C%5C&%7Dtype=pdf%20LB%20-%20tF2J (2013).

65. Plimpton, S., Jones, M. & Crozier, P. *Pizza.py Toolkit* Feb. 2006. http://pizza.sandia.gov/.

66. Jewett, A. I., Zhuang, Z. & Shea, J.-E. Moltemplate a Coarse-Grained Model Assembly Tool. *Biophysical Journal* **104,** 169a. ISSN: 0006-3495 (Jan. 2013).

67. Martínez, L., Andrade, R., Birgin, E. G. & Martínez, J. M. PACKMOL: A Package for Building Initial Configurations for Molecular Dynamics Simulations. *Journal of Computational Chemistry* **30,** 2157–2164. ISSN: 0192-8651 (Oct. 2009).

68. Klein, C., Summers, A. Z., Thompson, M. W., Gilmer, J. B., McCabe, C., Cummings, P. T., Sallai, J. & Iacovella, C. R. Formalizing Atom-Typing and the Dissemination of Force Fields with Foyer. *Computational Materials Science* **167,** 215–227. ISSN: 0927-0256. http://www.sciencedirect.com/science/article/pii/S0927025619303040 (Sept. 2019).

69. Wang, J., Wang, W., Kollman, P. A. & Case, D. A. Automatic atom type and bond type perception in molecular mechanical calculations. *J. Mol. Graph. Model.* **25,** 247–260. ISSN: 1093-3263 (2006).

70. Vanommeslaeghe, K. & MacKerell Jr, A. D. Automation of the CHARMM General Force Field (CGenFF) I: bond perception and atom typing. *J. Chem. Inf. Model.* **52,** 3144–3154. ISSN: 1549-9596 (Dec. 2012).

71. Malde, A. K., Zuo, L., Breeze, M., Stroet, M., Poger, D., Nair, P. C., Oostenbrink, C. & Mark, A. E. An Automated Force Field Topology Builder (ATB) and Repository: Version 1.0. *J. Chem. Theory Comput.* **7,** 4026–4037. ISSN: 1549-9618 (Dec. 2011).

72. Ribeiro, A. A. S. T., Horta, B. A. C. & de Alencastro, R. B. MKTOP: a program for automatic construction of molecular topologies. *J. Braz. Chem. Soc.* **19,** 1433–1435. ISSN: 0103-5053 (Aug. 2008).

73. Eggimann, B. L., Sunnarborg, A. J., Stern, H. D., Bliss, A. P. & Siepmann, J. I. An online parameter and property database for the TraPPE force field. *Mol. Simul.* **40,** 101–105. ISSN: 0892-7022 (2014).

74. *SimTK: OpenMM: Project Home* Feb. 2018. https://simtk.org/projects/openmm (2018).

75. Daylight Chemical Information Systems, I. *Daylight Theory: SMARTS - A Language for Describing Molecular Patterns* http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html. Accessed: 2022-03-14.

76. Hjorth Larsen, A., Jørgen Mortensen, J., Blomqvist, J., Castelli, I. E., Christensen, R., Dułak, M., Friis, J., Groves, M. N., Hammer, B., Hargus, C., Hermes, E. D., Jennings, P. C., Bjerre Jensen, P., Kermode, J., Kitchin, J. R., Leonhard Kolsbjerg, E., Kubal, J., Kaasbjerg, K., Lysgaard, S., Bergmann Maronsson, J., Maxson, T., Olsen, T., Pastewka, L., Peterson, A., Rostgaard, C., Schiøtz, J., Schütt, O., Strange, M., Thygesen, K. S., Vegge, T., Vilhelmsen, L., Walter, M., Zeng, Z. & Jacobsen, K. W. The atomic simulation environment-a Python library for working with atoms. *Journal of Physics: Condensed Matter* **29,** 273002. ISSN: 0953-8984. http://stacks.iop.org/0953-8984/29/i=27/a=273002?key=%20crossref.20f9751653d872507bf6c0cc5737032c (July 2017).

77. Fortunato, M. E. & Colina, C. M. pysimm: A python package for simulation of molecular systems. *SoftwareX* **6,** 7–12. ISSN: 2352-7110 (2017).

78. Fortunato, M. E. & Colina, C. M. *pysimm* https://github.com/polysimtools/pysimm.

79. Abbott, L. J., Hart, K. E. & Colina, C. M. Polymatic: a generalized simulated polymerization algorithm for amorphous polymers. *Theoretical Chemistry Accounts* **132,** 1334. ISSN: 1432-881X, 1432-2234 (Mar. 2013).

80. Girard, M., Ehlen, A., Shakya, A., Bereau, T. & de la Cruz, M. O. Hoobas: A highly object-oriented builder for molecular dynamics. *Computational Materials Science* **167,** 25–33. ISSN: 0927-0256 (2019).

81. *Hoobas Github repository* https://bitbucket.org/NUaztec/hoobas/src/master/ (2019).

82. *indigox Github repository* https://github.com/allison-group/indigox (2019).

83. O'Boyle, N. M., Banck, M., James, C. A., Morley, C., Vandermeersch, T. & Hutchison, G. R. Open Babel: An Open chemical toolbox. *Journal of Cheminformatics* **3,** 1–14. ISSN: 1758-2946 (2011).

84. *openbabel Github repository* https://github.com/openbabel (2019).

85. Mobley, D. L., Bannan, C. C., Rizzi, A., Bayly, C. I., Chodera, J. D., Lim, V. T., Lim, N. M., Beauchamp, K. A., Shirts, M. R., Gilson, M. K. & Eastman, P. K. *Open Force Field Consortium: Escaping Atom Types Using Direct Chemical Perception with SMIRNOFF v0.1* Preprint (Biophysics, Mar. 2018).

86. Zanette, C., Bannan, C. C., Bayly, C. I., Fass, J., Michael, K., Shirts, M. R., Chodera, J. D. & Mobley, D. L. Toward learned chemical perception of force fi eld typing rules. *J Chem Theory Comput* **15,** 402–423. ISSN: 1549-9618 (2019).

87. Mobley, D. L. *open-forcefield-group/smirff99Frosst: Version 1.0.1* version v1.0.1. Sept. 2016.

88. *https://www.daylight.com/dayhtml/doc/theory/theory.smirks.html*

89. *WebFF Github repository* https://github.com/usnistgov/WebFF-Documentation (2019).

90. Hedges, L. O., Mey, A. S. J. S., Laughton, C. A., Gervasio, F. L., Mulholland, A. J., Woods, C. J. & Michel, J. BioSimSpace: An Interoperable Python Framework for Biomolecular Simulation. *J. Open Source Softw.* **4,** 1831 (2019).

91. Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G., Rignanese, G.-M. & Hautier, G. FireWorks: A Dynamic Workflow System Designed for High-Throughput Applications. *Concurr. Comput. Pract. Exp.* **27.** CPE-14-0307.R2, 5037. ISSN: 1532-0634 (2015).

92. *FireWorks Github repository* https://github.com/materialsproject/fireworks (2019).

93. Phillips, C. L., Anderson, J. A. & Glotzer, S. C. Pseudo-random number generation for Brownian Dynamics and Dissipative Particle Dynamics simulations on GPU devices. *Journal of Computational Physics* **230,** 7191–7201. ISSN: 1090-2716 (2011).

94. Panagiotopoulos, A. Z., Quirke, N., Stapleton, M. & Tildesley, D. J. Phase equilibria by simulation in the gibbs ensemble alternative derivation, generalization and application to mixture and membrane equilibria. *Molecular Physics* **63,** 527–545. ISSN: 1362-3028 (1988).

95. Panagiotopoulos, A. Z. Direct determination of phase coexistence properties of fluids by monte carlo simulation in a new ensemble. *Molecular Physics* **61,** 813–826. ISSN: 1362-3028 (1987).

96. Adams, D. J. Grand canonical ensemble monte carlo for a lennard-jones fluid. *Molecular Physics* **29,** 307–311. ISSN: 1362-3028 (1975).

97. Ilja Siepmann, J. & Frenkel, D. Conflgurational bias monte carlo: A new sampling scheme for flexible chains. *Molecular Physics* **75,** 59–70. ISSN: 1362-3028 (1992).

98. Martin, M. G. & Siepmann, J. I. Transferable Potentials for Phase Equilibria. 1. United-Atom Description of *n* -Alkanes. *The Journal of Physical Chemistry B* **102,** 2569–2577. ISSN: 1520-6106, 1520-5207 (Apr. 1998).

99. Shah, M. S., Siepmann, J. I. & Tsapatsis, M. Transferable potentials for phase equilibria. Improved united-atom description of ethane and ethylene. *AIChE J.* **63,** 5098–5110. ISSN: 0001-1541. http://doi.wiley.com/10.1002/aic.15816%20http://onlinelibrary.wiley.com/wol1/doi/10.1002/aic.15816/fullpdf%20https://api.wiley.com/onlinelibrary/tdm/v1/articles/10.1002%7B%5C%%7D%202Faic.15816%20http://dx.doi.org/10.1002/aic.15816%20LB%20-%20ld7g (2017).

100. Nejahi, Y., Soroush Barhaghi, M., Mick, J., Jackman, B., Rushaidat, K., Li, Y., Schwiebert, L. & Potoff, J. GOMC: GPU Optimized Monte Carlo for the simulation of phase equilibria and physical properties of complex fluids. *SoftwareX* **9,** 20–27. ISSN: 2352-7110 (2019).

101. *MoSDeF TraPPE Github repository* https://github.com/ahy3nz/mosdef%7B%5C_%7Dtrappe.

102. Fu, C., Kuang, Y., Huang, Z., Wang, X., Yin, Y., Chen, J. & Zhou, H. Supercapacitor based on graphene and ionic liquid electrolyte. *Journal of Solid State Electrochemistry* **15,** 2581–2585. ISSN: 1432-8488 (2011).

103. Zhan, C., Lian, C., Zhang, Y., Thompson, M. W., Xie, Y., Wu, J., Kent, P. R., Cummings, P. T., en Jiang, D. & Wesolowski, D. J. Computational Insights into Materials and Interfaces for Capacitive Energy Storage. *Advanced Science* **1700059.** ISSN: 2198-3844 (2017).

104. Zhang, Y., Dyatkin, B. & Cummings, P. T. Molecular Investigation of Oxidized Graphene: Anatomy of the Double-Layer Structure and Ion Dynamics. *Journal of Physical Chemistry C* **123,** 12583–12591. ISSN: 1932-7455 (2019).

105. Meyer, J. C., Geim, A. K., Katsnelson, M. I., Novoselov, K. S., Booth, T. J. & Roth, S. The structure of suspended graphene sheets. *Nature* **446,** 60–63. ISSN: 0028-0836 (2007).

106. Mahurin, S. M., Mamontov, E., Thompson, M. W., Zhang, P., Turner, C. H., Cummings, P. T. & Dai, S. Relationship between pore size and reversible and irreversible immobilization of ionic liquid electrolytes in porous carbon under applied electric potential. *Applied Physics Letters* **109,** 143111. ISSN: 0003-6951 (2016).

107. Feng, G. & Cummings, P. T. Supercapacitor capacitance exhibits oscillatory behavior as a function of nanopore size. *Journal of Physical Chemistry Letters* **2,** 2859–2864. ISSN: 1948-7185 (2011).

108. Lindahl, E., Hess, B. & van der Spoel, D. GROMACS 3.0: a package for molecular simulation and trajectory analysis. *Journal of Molecular Modeling* **7,** 306–317. ISSN: 1610-2940 (Aug. 2001).

109. Van Der Spoel, D., Lindahl, E., Hess, B., Groenhof, G., Mark, A. E. & Berendsen, H. J. GROMACS: Fast, flexible, and free. *Journal of Computational Chemistry* **26,** 1701–1718. ISSN: 0192-8651 (Dec. 2005).

110. Pronk, S., Páll, S., Schulz, R., Larsson, P., Bjelkmar, P., Apostolov, R., Shirts, M. R., Smith, J. C., Kasson, P. M., van der Spoel, D., Hess, B. & Lindahl, E. GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics* **29,** 845–854. ISSN: 1460-2059. arXiv: 1011.1669 (Apr. 2013).

111. McGibbon, R. T., Beauchamp, K. A., Harrigan, M. P., Klein, C., Swails, J. M., Hernández, C. X., Schwantes, C. R., Wang, L.-P., Lane, T. J. & Pande, V. S. MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. *Biophysical Journal* **109,** 1528–1532. ISSN: 0006-3495 (Oct. 2015).

112. *Graphene-Pore Github repository* https://github.com/rmatsum836/Pore-Builder (2019).

113. Berendsen, H. J. C., Grigera, J. R. & Straatsma, T. P. The Missing Term In Effective Pair Potentials. *Journal of Physical Chemistry* **91,** 6269–6271 (1987).

114. Joung, I. S. & Cheatham, T. E. Determination of alkali and halide monovalent ion parameters for use in explicitly solvated biomolecular simulations. *Journal of Physical Chemistry B* **112,** 9020–9041. ISSN: 1520-6106 (2008).

115. Hess, B., Bekker, H., Berendsen, H. J. C. & Fraaije, J. G. E. M. {LINCS}: {A} {L}inear {C} onstraint {S}olver for {M }olecular {S}imulations. *Journal of Computational Chemistry* **18,** 1463 (1997).

116. *true_graphene Github repository* https://github.com/rmatsum836/true%5C_graphene (2019).

117. *true_lipids Github repository* https://github.com/uppittu11/true%5C_lipids (2019).

118. Weerheim, A. & Ponec, M. Determination of stratum corneum lipid profile by tape stripping in combination with high-performance thin-layer chromatography. *Archives of Dermatological Research* **293,** 191–199. ISSN: 0340-3696 (Apr. 2001).

119. Madison, K. C. Barrier Function of the Skin: "La Raison d'Être" of the Epidermis. *Journal of Investigative Dermatology* **121,** 231–241. ISSN: 0022-202X. http://www.sciencedirect.com/science/article/pii/S0022202X15303560%20https://linkinghub.elsevier.com/retrieve/pii/S0022202X15303560 (Aug. 2003).

120. *Bilayer Builder Github repository* https://github.com/uppittu11/mbuild%5C_bilayer (2019).

121. Guo, S., Moore, T. C., Iacovella, C. R., Strickland, L. A. & Mccabe, C. Simulation study of the structure and phase behavior of ceramide bilayers and the role of lipid headgroup chemistry. *Journal of Chemical Theory and Computation* **9,** 5116–5126. ISSN: 1549-9618 (2013).

122. Klauda, J. B., Venable, R. M., Freites, J. A., O'Connor, J. W., Tobias, D. J., Mondragon-Ramirez, C., Vorobyov, I., MacKerell, A. D. & Pastor, R. W. Update of the CHARMM All-Atom Additive Force Field for Lipids: Validation on Six Lipid Types. *The Journal of Physical Chemistry B* **114,** 7830–7843 (2010).

123. Berendsen, H. J., Postma, J. v., van Gunsteren, W. F., DiNola, A. & Haak, J. Molecular dynamics with coupling to an external bath. *J. Chem. Phys.* **81,** 3684–3690. ISSN: 0021-9606 (1984).

124. Nosé, S. A Unified Formulation of the Constant Temperature Molecular Dynamics Methods. *Journal of Chemical Physics* **81,** 511 (1984).

125. Parrinello, M. & Rahman, A. Polymorphic Transitions in Single Crystals: A New Molecular Dynamics Method. *Journal of Applied Physics* **52,** 7182–7190. ISSN: 0021-8979, 1089-7550 (Dec. 1981).

126. Vilt, S. G., Leng, Z., Booth, B. D., McCabe, C. & Jennings, G. K. Surface and Frictional Properties of Two-Component Alkylsilane Monolayers and Hydroxyl-Terminated Monolayers on Silicon. *The Journal of Physical Chemistry C* **113,** 14972–14977. ISSN: 1932-7447. eprint: https://doi.org/10.1021/jp904809h (Aug. 2009).

127. Summers, A. Z., Iacovella, C. R., Billingsley, M. R., Arnold, S. T., Cummings, P. T. & McCabe, C. Influence of Surface Morphology on the Shear-Induced Wear of Alkylsilane Monolayers: Molecular Dynamics Study. *Langmuir* **32.** PMID: 26885941, 2348–2359. ISSN: 1520-5827. eprint: https://doi.org/10.1021/acs.langmuir.5b03862 (2016).

128. Darden, T., York, D. & Pedersen, L. Particle Mesh Ewald: An $N \cdot \log(N)$ Method for Ewald Sums in Large Systems. *The Journal of Chemical Physics* **98,** 10089–10092. ISSN: 0021-9606, 1089-7690. eprint: https://doi.org/10.1063/1.464397 (June 1993).

129. Tuckerman, M., Berne, B. J. & Martyna, G. J. Reversible multiple time scale molecular dynamics. *The Journal of Chemical Physics* **97,** 1990–2001. ISSN: 0021-9606. eprint: https://doi.org/10.1063/1.463137 (Aug. 1992).

130. Chodera, J. D. A Simple Method for Automated Equilibration Detection in Molecular Simulations. *Journal of Chemical Theory and Computation* **12.** PMID: 26771390, 1799–1805. ISSN: 1549-9618, 1549-9626. eprint: https://doi.org/10.1021/acs.jctc.5b00784 (Apr. 2016).

131. Shirts, M. R. & Chodera, J. D. Statistically optimal analysis of samples from multiple equilibrium states. *The Journal of Chemical Physics* **129,** 124105. eprint: https://doi.org/10.1063/1.2978177 (2008).

132. Adorf, C. S., Ramasubramani, V., Dice, B. D., Henry, M. M., Dodd, P. M. & Glotzer, S. C. *glotzerlab/signac* version 1.0.0. Development and deployment supported by MICCoM, as part of the Computational Materials Sciences Pro- gram funded by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, Materials Sciences and Engineering Division, under Subcontract No. 6F-30844. Project conceptualization and implementation supported by the National Science Foundation, Award DMR 1409620. Feb. 2019.

133. *Tribology Example Github repository* https://github.com/daico007/TRUE-nanotribology.

CHAPTER 4

Recent MoSDeF Development Efforts

4.1 Introduction

During the "initialization" phase of a soft matter simulation workflow, multiple steps occur which are quite common sources of error or other deficiencies that may reduce the reproducibility of the study. These steps are: creation of the soft matter system (particle placement, etc.), application of force field interaction parameters to the newly created system, and translation of the system and parameters to a format that a simulation engine can read. MoSDeF has three standalone libraries that are developed with these steps in mind. For system initialization, `mBuild` [1–3], short for "Molecule Builder", is a python library designed to programmatically create various molecular input structures, from crystalline systems, polymer networks, nanoparticles, and polymer-coated substrates in a reproducible fashion, promoting TRUE [4] principles to assist with more reproducible molecular simulations. `mBuild` was deigned to make the initialization of soft matter systems much easier, but first class support is also available for hard materials. Simulation of these systems are commonly performed using force fields that describe the interactions between various particles, where a single periodic element can have 10's to 100's of interaction parameter combinations based on the chemical context of that particle. This makes the process of assigning the correct interactions (atom types) to the system quite challenging and highly error prone. This is further complicated by the fact that these parameters and interaction terms are usually applied or discovered during the simulation engine input step, which makes transferability of these simulations much more difficult. `Foyer` [5] was designed as a simulation engine agnostic way to store force field information, defining the chemical context for each parameter using SMARTS [6] strings with additional custom grammar to support coarse-grained systems as well. By treating the system as a large collection of various graphs and subgraphs, using subgraph isomorphism [7] and the SMARTS grammar above to define the subgraphs for the various atom types, a much more reproducible and automated way to apply atom types to the system is possible. Developments and a in-depth description of `foyer` is provided in Section 4.4. Finally, once the system has been built and the parameters assigned, the system needs to be written out to many simulation engines based on the type of system and the force field chosen. This requires a generic data structure to store the system topology as well as the force field parameters and functional forms of the force field interactions, while also having knowledge of the supported simulation engines and how to convert this data into a format usable by the engine. The General Molecular Simulation Object (GMSO) is the third key library of MoSDeF, providing this functionality in a simulation engine agnostic manner. By storing the force field functional forms using symbolic mathematics, tracking units, and supporting a wide range of simulation engines in an extensible manner, GMSO is becoming the de-facto data structure to store parametrized system information in the MoSDeF ecosystem. However, all three MoSDeF libraries are designed such that they can operate independently of one another, but many convenience methods are exposed when using all the libraries in tandem.

This chapter [12] highlights the MoSDeF libraries and their recent development efforts to improve reproducibility in molecular simulation, and supporting TRUE principles [4].

4.2 `mBuild` developments

4.2.1 Introduction

`mBuild` [1, 3, 8, 9], while still using the same data structures, design patterns, and overall design philosophy, has changed quite drastically since Klein, Sallai, Jones, Iacovella, McCabe & Cummings [1]. Initially, `mbuild` supported both the design of various soft matter systems as well as a parametrization scheme that leveraged `GROMACS` [10, 11] force field parameter files to handle parameter assignment and eventual coercion into file formats that could be parsed by GROMACS. However, as `foyer` [5] began to take shape, with its SMARTS [6] based atom typing and parameter assignment, `mBuild` and `foyer` were finally separated into two standalone packages [9, 12]. Instead of a monolithic design scheme, composable, independent libraries were developed to provide researchers with more granularity in terms of the specific libraries they might need. For example, one who uses interaction terms that are only based on elemental information and position, would not need to leverage the more complex SMARTS-based atom typing of `foyer`, `mbuild` provides all the information necessary in that case. `mBuild` and the MoSDeF libraries in general, have underwent massive overhauls to be more composable, and standalone libraries based on what a user might require, while also providing excellent interoperability between the libraries if desired. The section below highlights major changes to `mBuild` since the 2016 paper. These changes add improved support for hard material crystalline systems, enhancements to the polymer building routines, a robust plugin system for the community to create and distribute custom, complex system building workflows, and improved support for triclinic (non-orthogonal) simulation boxes.

4.2.2 `mBuild` Updates

4.2.2.1 Plugin System and Development

There may be cases where your `mbuild.Compounds` and/or building scripts can be generalized to support a broad range of systems. Such objects would be a valuable resource for many researchers, and might justify

[1]Portions of this chapter reprinted with permission from the following work:

[2]Klein, C., Summers, A. Z., Thompson, M. W., Gilmer, J. B., McCabe, C., Cummings, P. T., Sallai, J. & Iacovella, C. R. Formalizing Atom-Typing and the Dissemination of Force Fields with Foyer. *Computational Materials Science* **167,** 215–227. ISSN: 0927-0256. http://www.sciencedirect.com/science/article/pii/S0927025619303040 (Sept. 2019).

development of a Python package that could be distributed to the community.

`mBuild` has been developed with this in mind, in the form of a plug-in system. Detailed below are the specifications of this system, how to convert an existing Python project into an mBuild-discoverable plug-in, and an example.

4.2.2.1.1 Entry Points

The basis of the plug-in system in mBuild is the https://packaging.python.org/guides/creating-and-discovering-plugins/ #using-package-metadata (`setuptools.entry_points package`). This allows other packages to register themselves with the `entry_point` group we defined in mBuild, so they are accessible through the `mbuild.recipes` location. Imagine you have a class named `my_foo` that inherits from `mb.Compound`. It is currently inside of a project `my_project` and is accessed via a direct import, i.e. `from my_project import my_foo`. You can register this class as an entry point associated with `mbuild.recipes`. It will then be accessible from inside mBuild as a plug-in via `mbuild.recipes.my_foo` and a direct import will be unnecessary. The call `import mbuild` discovers all plug-ins that fit the `entry_point` group specification and makes them available under `mbuild.recipes`.

4.2.2.1.2 Registering a Recipe

Here we consider the case that a user already has a Python project set up with a structure similar to the layout below. This project (Listing 4.1) can be found here: https://github.com/justinGilmer/mbuild-fcc. The two important files for the user to convert their `mBuild` plug-in to a discoverable plug-in are `setup.py` and `mbuild_fcc.py`. To begin, lets first inspect the `mbuild_fcc.py` file, a shortened snippet is below in Listing 4.2.

Listing 4.1: Sample Python project that extends `mBuild` functionality. In this case, an extension of the `mbuild.Lattice` object for FCC crystal lattices.

```
mbuild_fcc
├── LICENSE
├── README.md
├── mbuild_fcc
│   ├── mbuild_fcc.py
│   └── tests
│       ├── __init__.py
│       └── test_fcc.py
└── setup.py
```

Listing 4.2: Shortened code snippet that represents an `mBuild Compound` based on a repeat unit of the FCC lattice.

```python
import mbuild

class FCC(mbuild.Compound):
    """Create a mBuild Compound with a repeating unit of the FCC unit cell.

    ... (shortened for viewability)

    """

    def __init__(self, lattice_spacing=None, compound_to_add=None, x=1, y=1, z=1):
        super(FCC, self).__init__()

        # ... (shortened for viewability)

if __name__ == "__main__":
    au_fcc_lattice = FCC(lattice_spacing=0.40782,
                         compound_to_add=mbuild.Compound(name="Au"),
                         x=5, y=5, z=1)
    print(au_fcc_lattice)
```

There are two notable lines in this file that we need to focus on when developing this as a plug-in for mBuild. The first is the import statement `import mbuild`. We must make sure that mbuild is installed since we are inheriting from `mbuild.Compound`. When you decide to distribute your plug-in, the dependencies must be listed. The second is to select the name of the plug-in itself. It is considered good practice to name it the name of your `class`. In this case, we will name the plug-in `FCC`. The last step is to edit the `setup.py` file such that the plug-in can be registered under the `entry_point` group `mbuild.plugins`, see (Listing 4.3).

Listing 4.3: Snippet of `setup.py` registering the `FCC` object as a discoverable `mBuild` plug-in.

```python
1 from setuptools import setup
2
3 setup(
4     ...
5     entry_points={ "mbuild.plugins":[ "FCC = mbuild_fcc.mbuild_fcc:FCC"]},
6     ...
7 )
```

The important section is the `entry_points` argument on line 5. Here we define the entry_point group we want to register with: `"mbuild.plugins"`. Finally, we tell Python what name to use when accessing this plug-in. Earlier, we decided to call it `FCC`. This is denoted here by the name before the assignment operator `FCC =`. Next, we pass the location of the file with our plug-in: `mbuild_fcc.mbuild_fcc` as if we were located at the `setup.py` file. Then, we provide the name of the class within that Python file we want to make discoverable `:FCC`.

Since the `setup.py` file is located in the top folder of the python project, the first `mbuild_fcc` is the name of the folder, and the second is the name of the python file. The colon (`:`) is used when accessing the class that is in the python file itself.

4.2.2.1.3 Putting it all together

Finally, we have `FCC = mbuild_fcc.mbuild_fcc:FCC`. To test this feature, you should clone the `mbuild-fcc` project listed above.

```
git clone https://github.com/justinGilmer/mbuild-fcc
```

Make sure you have mBuild installed, then run the command below after changing into the `mbuild-fcc` directory.

```
cd mbuild-fcc
pip install -e .
```

Note that this command will install this example from source in an editable format.

4.2.2.1.4 Trying it Out

To test that you set up your plug-in correctly, try importing mBuild: If you do not receive error messages, your plug-in should be discoverable!

```python
import mbuild
help(mbuild.recipes.FCC)
```

4.2.2.2 Hard Material Support

While `mbuild` has supported soft materials and their construction since its inception, support of crystalline hard materials and molecular crystals was limited. Support for crystalline systems was added with the addition of the `mbuild.Lattice` data structure in `mbuild` version 0.6.0. Using a standard bravais angle notation , or lattice vectors to represent the unit cell of a crystal lattice, the user can now define hard materials and other repeating materials using crystallographic notation. Crystallographic notation has been standardized by the IUCR, and this is also how crystalline materials are commonly defined when serialized to a form that is both human and machine readable, the crystallographic index file (CIF) is the most common file format for experimentalists and computational researchers to represent a crystal structure. This also allowed first class support in `mbuild` to create crystalline hard materials and a convenient path to now parse CIF files, thanks to the CIF reader provided by the `garnett` package , previously not possible.

Below is a summary and walkthrough of interfacing with the `mbuild.Lattice` structure, with common crystalline systems provided as examples.

- **Variable-dimension crystal structures**
  - `Lattice` supports the dimensionality of `mBuild`, which means that the systems can be in 1D, 2D, or 3D. Replace the necessary vector components with 0 to emulate the dimensionality of interest.
- **Multicomponent crystals**
  - `Lattice` can support an indefinite amount of lattice points in its data structure.
  - The repeat cell can be as large as the user defines useful.
  - The components that occupy the lattice points are `mbuild.Compound` objects.
    - This allows the user to build any system since compounds are only a representation of matter in this design.
    - Molecular crystals, coarse grained, atomic, even alloy crystal structures are all supported
- **Triclinic Lattices**
  - With full support for triclinic lattices, any crystal structure can technically be developed.

  - Either the user can provide the lattice parameters, or if they know the vectors that span the unit cell, that can also be provided.

- **Generation of lattice structure from crystallographic index file** (CIF) **formats**

4.2.2.2.1 Lattice Data Structure Introduction

Below we will explore the relevant data structures that are attributes of the `Lattice` class. This information will be essential to build desired crystal structures.

- **Lattice.lattice_spacing**

  This data structure is a (3,) array that details the lengths of the repeat cell for the crystal. You can either use a `numpy` array object, or simply pass in a list and `Lattice` will handle the rest. Remember that `mBuild`'s units of length are in nm. You must pass in all three lengths, even if they are all equivalent. These are the lattice parameters a, b, c when viewing crystallographic information.

  For Example:

  ```
  lattice_spacing = [0.5, 0.5, 0.5]
  ```

- **Lattice.lattice_vectors**

  `lattice_vectors` is a 3x3 array that defines the vectors that encapsulate the repeat cell. This is an optional value that the user can pass in to define the cell. Either this must be passed in, or the 3 Bravais angles of the cell from the lattice parameters must be provided. If neither is passed in, the default value are the vectors that encase a cubic lattice. Most users will **not** have to use these to build their lattice structure of interest. It will usually be easier for the users to provide the 3 Bravais angles instead. If the user then wants the vectors, the `Lattice` object will calculate them for the user.

  For example: Cubic Cell

  ```
  lattice_vectors = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
  ```

- **Lattice.angles**

  `angles` is a (3,) array that defines the three Bravais angles of the lattice. Commonly referred to as α, β, γ in the definition of the lattice parameters.

  For example: Cubic Cell

  ```
  angles = [90, 90, 90]
  ```

- **Lattice.lattice_points**

  `lattice_points` can be the most common source of confusion when creating a crystal structure. In crystallographic terms, this is the minimum basis set of points in space that define where the points in the lattice exist. This requires that the user does not over define the system. MIT's OpenCourseWare has an excellent PDF for more information https://ocw.mit.edu/courses/earth-atmospheric-and-planetary-sciences/12-108-structure-of-earth-materials-fall-2004/lecture-notes/lec7.pdf.

  The other tricky issue that can come up is the data structure itself. `lattice_points` is a dictionary where the `dict.key` items are the `string` id's for each basis point. The `dict.values` items are a nested list of fractional coordinates of the unique lattice points in the cell. If you have the same `Compound` at multiple lattice_points, it is easier to put all those coordinates in a nested list under the same `key` value. Two examples will be given below, both FCC unit cells, one with all the same id, and one with unique ids for each lattice_point.

  For Example: FCC All Unique

  ```
  lattice_points = {'A' : [[0, 0, 0]],
                    'B' : [[0.5, 0.5, 0]],
                    'C' : [[0.5, 0, 0.5]],
                    'D' : [[0, 0.5, 0.5]]
                    }
  ```

  For Example: FCC All Same

  ```
  lattice_points = {'A' : [[0, 0, 0], [0.5, 0.5, 0], [0.5, 0, 0.5], [0, 0.5, 0.5]]
  ```

4.2.2.2.2 Lattice Public Methods

The `Lattice` class also contains methods that are responsible for applying `Compounds` to the lattice points, with user defined cell replications in the x, y, and z directions.

- **Lattice.populate(compound_dict=None, x=1, y=1, z=1)**

  This method uses the `Lattice` object to place `Compounds` at the specified `lattice_points`. There are 4 optional inputs for this class.

  - `compound_dict` inputs another dictionary that defines a relationship between the `lattice_points` and the `Compounds` that the user wants to populate the lattice with. The `dict.keys` of this dictionary must be the same as the `keys` in the `lattice_points` dictionary. However, for the `dict.items` in this case, the `Compound` that the user wants to place at that lattice point(s) will be used. An example will use the FCC examples from above. They have been copied below:

Listing 4.4: FCC lattice, with unique mappings for each lattice point

```
lattice_points = {'A' : [[0, 0, 0]],
                  'B' : [[0.5, 0.5, 0]],
                  'C' : [[0.5, 0, 0.5]],
                  'D' : [[0, 0.5, 0.5]]
                  }

# compound dictionary
a = mbuild.Compound(name='A')
b = mbuild.Compound(name='B')
c = mbuild.Compound(name='C')
d = mbuild.Compound(name='D')
compound_dict = {'A' : a, 'B' : b, 'C' : c, 'D' : d}
```

Listing 4.5: FCC lattice, with a single mapping to all lattice points

```
lattice_points = {'A' : [[0, 0, 0], [0.5, 0.5, 0], [0.5, 0, 0.5], [0, 0.5, 0.5]] }

# compound dictionary
a = mbuild.Compound(name='A')
compound_dict = {'A' : a}
```

4.2.2.2.3 Example Lattice Systems

Below are examples of homogeneous and heterogeneous 2D and 3D lattice structures using the `Lattice` class. Beginning with Listing 4.6, the creation of a simple-cubic lattice (where the repeat points are only at the vertices of the cubic cell). Next, Listing 4.7 showcases a body-centered cubic (BCC) structures with two repeat units, one at the eight vertices and the final in the center of the cell, of CsCl. Then, Listing 4.8 is a face-centered cubic (FCC) structure and Listing 4.9 is a diamond lattice, and finally, a 2D hexagonal sheet of graphene (Listing 4.10) is also provided.

Listing 4.6: Creation of a $2 \cdot 2 \cdot 2$ repeat cell of polonium, which traditionally forms a simple-cubic lattice.

```python
import mbuild as mb
import numpy as np

# define all necessary lattice parameters
spacings = [0.3359, 0.3359, 0.3359]
angles = [90, 90, 90]
points = [[0, 0, 0]]

# define lattice object
sc_lattice = mb.Lattice(lattice_spacing=spacings, angles=angles, lattice_points={'Po' :
 ↪  points})

# define Polonium Compound
po = mb.Compound(name='Po')

# populate lattice with compounds
po_lattice = sc_lattice.populate(compound_dict={'Po' : po}, x=2, y=2, z=2)

# visualize
po_lattice.visualize()
```

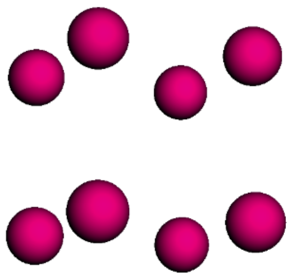


Figure 4.1: Polonium simple cubic (SC) structure

Listing 4.7: Creation of a cesium chloride CsCl lattice, which is effectivelyA a body-centered cubic (BCC) cell.

```python
import mbuild as mb
import numpy as np

# define all necessary lattice parameters
spacings = [0.4123, 0.4123, 0.4123]
angles = [90, 90, 90]
point1 = [[0, 0, 0]]
point2 = [[0.5, 0.5, 0.5]]

# define lattice object
bcc_lattice = mb.Lattice(lattice_spacing=spacings, angles=angles, lattice_points={'A' :
 ↪  point1, 'B' : point2})

# define Compounds
cl = mb.Compound(name='Cl')
cs = mb.Compound(name='Cs')

# populate lattice with compounds
cscl_lattice = bcc_lattice.populate(compound_dict={'A' : cl, 'B' : cs}, x=2, y=2, z=2)

# visualize
cscl_lattice.visualize()
```

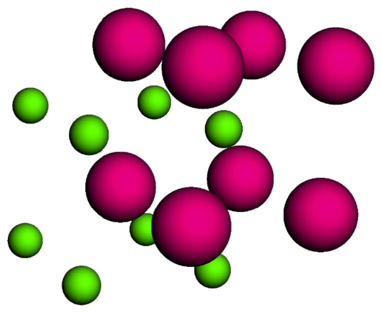


Figure 4.2: CsCl body-centered cubic (BCC) structure

Listing 4.8: Creation of a Cu face-centered cubic (FCC) lattice.

```python
import mbuild as mb
import numpy as np

# define all necessary lattice parameters
spacings = [0.36149, 0.36149, 0.36149]
angles = [90, 90, 90]
points = [[0, 0, 0], [0.5, 0.5, 0], [0.5, 0, 0.5], [0, 0.5, 0.5]]

# define lattice object
fcc_lattice = mb.Lattice(lattice_spacing=spacings, angles=angles, lattice_points={'A' :
    ↪  points})

# define Compound
cu = mb.Compound(name='Cu')

# populate lattice with compounds
cu_lattice = fcc_lattice.populate(compound_dict={'A' : cu}, x=2, y=2, z=2)

# visualize
cu_lattice.visualize()
```

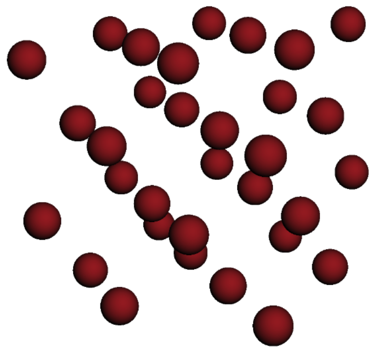


Figure 4.3: Cu face-centered cubic (FCC) structure

Listing 4.9: Si cubic diamond lattice.

```python
import mbuild as mb
import numpy as np

# define all necessary lattice parameters
spacings = [0.54309, 0.54309, 0.54309]
angles = [90, 90, 90]
points = [[0, 0, 0], [0.5, 0.5, 0], [0.5, 0, 0.5], [0, 0.5, 0.5],
          [0.25, 0.25, 0.75], [0.25, 0.75, 0.25], [0.75, 0.25, 0.25], [0.75, 0.75, 0.75]]

# define lattice object
diamond_lattice = mb.Lattice(lattice_spacing=spacings, angles=angles, lattice_points={'A'
↪  : points})

# define Compound
si = mb.Compound(name='Si')

# populate lattice with compounds
si_lattice = diamond_lattice.populate(compound_dict={'A' : si}, x=2, y=2, z=2)

# visualize
si_lattice.visualize()
```

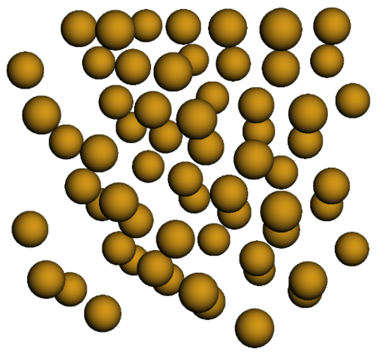


Figure 4.4: Si diamond (Cubic) structure

Listing 4.10: Graphene 2D hexagonal lattice

```python
import mbuild as mb
import numpy as np

# define all necessary lattice parameters
spacings = [0.246, 0.246, 0.335]
angles = [90, 90, 120]
points = [[0, 0, 0], [1/3, 2/3, 0]]

# define lattice object
graphene_lattice = mb.Lattice(lattice_spacing=spacings, angles=angles, lattice_points={'A'
 ↪  : points})

# define Compound
c = mb.Compound(name='C')

# populate lattice with compounds
graphene = graphene_lattice.populate(compound_dict={'A' : c}, x=5, y=5, z=1)

# visualize
graphene.visualize()
```

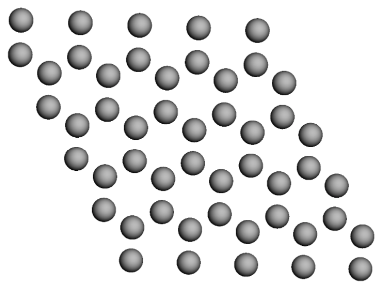


Figure 4.5: Graphene (2D) structure

4.3 GMSO developments

4.3.1 Introduction

GMSO is designed to enable the flexible, general representation of chemical topologies for molecular simulation. Efforts are made to enable lossless, bias-free storage of data, without assuming particular chemistries, models, or using any particular engine's ecosystem as a starting point. The scope is generally restrained to the preparation, manipulation, and conversion of input files for molecular simulation, i.e. before engines are called to execute the simulations themselves. In the scope of molecular simulation, we loosely define a chemical topology as everything needed to reproducibly prepare a chemical system for simulation. This includes particle coordinates and connectivity, box information, force field data (functional forms, parameters tagged with units, partial charges, etc.) and some optional information that may not apply to all systems (i.e. specification of elements with each particle). There are many tools in the ecosystem that attempt to provide similar utilities, although there are limitations we will highlight below that made extending those tools more difficult, spurring the development of GMSO. A similar utility for biomolecular-related simulations is the Parameter Editor (ParmEd) [13]. ParmEd was designed to represent the interaction terms and topological information of biomolecular systems and simulation engine formats commonly used for this purpose. For MoSDeF's original goal, to better support the automatable and reproducible creation and parametrization of soft matter systems, the engines and potential energy functional forms supported by ParmEd was more than sufficient, ParmEd is still a key backend data structure in the MoSDeF tools for storing the parametrized information and outputting to its supported simulation engines. However, as MoSDeF has developed and its users have begun to explore more exotic interaction terms and corresponding potential energy functional forms, the focus for biomolecular simulations limits parmed for our uses. There is also the OpenForceField (OpenFF) Initiative's to support storing, manipulating, and converting molecular mechanics data, OpenFF Interchange [14]. While still heavily focused on biomolecular systems and simulation engines, MoSDeF and OpenFF have developed interoperability between the two ecosystems, with further development underway. Although parmed[13] and OpenFF Interchange [14] are similar and provide for some of the needs that MoSDeF requires for its parametrized system and topological information, support for arbitrary functional forms and a force field file format that supports foyer's SMARTS-based atomtyping engine [5] (Section 4.4) were not tractable outcomes. This led to the development of GMSO, a data structure for parameterized systems and their topological information, using symbolic mathematics to represent the potential energy functional forms, and the unyt [15] library for unit conversion based on the simulation engine the system is converted to. Features that are enabled based on the design of gmso are defined in the following section.

4.3.2 Features enabled by GMSO

- Supporting a variety of models in the molecular simulation/computational chemistry community: No assumptions are made about an interaction site representing an atom or bead, instead atomistic, united-atom/coarse-grained, polarizable, and other models can be extended and supported within GMSO.

- Greater flexibility for exotic potentials: The AtomType (and analogue classes for intramolecular interactions) uses sympy to store any potential that can be represented by a mathematical expression.

- Easier development for glue to new engines: by not being designed for compatibility with any particular molecular simulation engine or ecosystem, it becomes more tractable for developers in the community to add glue for engines that are not currently supported.
- Compatibility with existing community tools: No single molecular simulation tool will be a silver bullet, so `GMSO` includes functions to convert objects. These can be used in their own right to convert between objects in-memory and also to support conversion to file formats not natively supported at any given time. Currently supported conversions include `ParmEd`, `OpenMM`, `mBuild`, `MDTraj`, with others being added as need arises.
- Native support for reading and writing many common file formats (`XYZ`, `GRO`, `TOP`, `LAMMPSDATA`, `MOL2`) and indirect support, through other libraries, for additional formats.

4.4 Formalizing atom-typing and the dissemination of force fields with `foyer`

4.4.1 Background

Considerable efforts have been undertaken by many research groups to develop accurate classical force fields for a wide range of systems [16–22]. Force fields are often expressed as a set of analytical functions with adjustable fitting parameters that describe the interactions between constituents of a system (often discrete atoms but, more generally, interaction sites). Classical force fields are typically able to achieve high accuracy by creating sets of highly specific fitting parameters (i.e., atom types), in which each atom type describes an interaction site within a different chemical context. The chemical context is typically defined by the bonded environment of an interaction site (e.g., the number of bonds and the identity of the bonded neighbors) and may also consider, among other factors, the bonded environment of the neighbors, and/or the specific molecule/structure within which the interaction site is included. Consequently, a force field may include tens or even hundreds of different atom types for a given element. For example, there are 347 atom types that apply to carbon in the OPLS force field parameter set distributed with GROMACS[23] where each atom type corresponds to a carbon atom within a different chemical context. Thus, while force field development efforts have reduced — or in some cases completely eliminated — the need for researchers to generate their own fitting parameters, determining which parameters (i.e., atom types) to use can still be a tedious and error prone task. Failure to properly identify the chemical context and atom type of an interaction site will inevitably lead to the unfaithful implementation of the force field and thus inconsistent results.

Part of the difficulty in performing atom-typing (i.e., determining which atom type applies to an interaction site) stems from the fact that there is not yet a standardized way of unambiguously expressing chemical context and parameter usage. As such, journal articles that report novel force field parameters may vary significantly in terms of their clarity. In many cases, parameters are reported in a tabular format with minimal annotations and few (if any) examples of how to appropriately assign the atom types. Since this approach does not allow for automated evaluation, different users of the force field may apply the atom types differently based on their own interpretation of the information provided. Journal articles that utilize existing force fields often do not report the specific fitting parameters and typically do not specify which atom types were chosen for the interaction sites, instead providing citation(s) to the source of the force field parameters. Even if the source of the parameters is clearly and fully specified, usage may again depend on the clarity of the original source(s) and the interpretation by the end user, hampering reproducibility. Force field parameter files that aggregate a large number of atom types (often thousands) into a single source suffer from some of the same issues. Often, they include only brief, unstructured — and sometimes ambiguous — annotations as to parameter usage, and may, or may not, provide clear citations of the original source of the parameters.

To apply force fields, users can perform atom-typing manually (e.g., creation of an atom-typed template of a molecule or unit cell), although manual assignment of parameters becomes tedious and error prone for large molecules and/or complex systems, and manual manipulation of files is not considered a good practice in terms of reproducibility [24]. Furthermore, manual assignment of parameters does not lend itself well to workflows such as screening [25], where thousands of unique systems with different chemical constituents and structures may need to be atom-typed in an automated fashion. To avoid manual assignment, end- users often develop in-house software to apply force fields in an automated fashion; however, such software is not typically made freely available to the community and may be very limited in scope and applicability. Without access to the same software, the exact atom-typing cannot be reproduced by others and if the source code is not made freely available, the logic used to interpret and apply the force field is unknown and if there are errors in the software/logic, these cannot be identified. There exist a number of freely available atom-typing tools that read in a force field parameter file and execute a set of rules to apply the force field to a chemical topology [26–32], enabling the exact atom-typing process to be reproduced. However, many of these atom-typing tools are either closed-source [27, 30], simulation engine-specific [29, 32], and/or force field-specific [28, 30, 31], which limits their utility. Furthermore, these tools almost universally rely on a rigid hierarchy of rules[26], where rules must be called in a precise order such that more general atom types are only chosen when more specialized matches do not exist (i e. ., the order of rules defines the precedence). Maintaining, let alone constructing, these hierarchies is challenging, especially for a large number of atom types. In order to add a new atom type or correct an error in hierarchical schemes, a developer must have a complete picture of the hierarchy and know exactly where the relevant rule should be placed such that it does not inadvertently override other rules. This may impose practical limits on functionality, where, for example, a user is not able to easily extend the rules to include new atom types, or that such attempts to extend the rules result in incorrect atom-typing for other systems. For many tools, this approach is further complicated by the encoding of the hierarchy as a set of heavily nested if/else statements within the source code of the software. These heavily nested if/else hierarchies may be difficult to validate and debug, and any changes or extensions to the rules, no matter how trivial, require modification of the source code itself. Reproducibility issues may therefore arise if users make modifications or extensions to a piece of software and these changes are not made freely available to the larger community and/or incorporated into the main software distribution. This also creates a situation where there are effectively two sets of rules since there is no guarantee that the logic statements in the source code (i.e., the machine readable rules) agree with the textual annotations in the force field parameter file (i.e., the human readable rules).

Several atom-typing tools have been developed that remove the need to encode atom type usage rules within the source code itself. A unifying feature of these tools is the use of the simplified molecular input line-entry system (SMILES)[33] language, or variants thereof, for describing chemical structures associated with an atom type. For example, Yesselman, et al.[32] developed an atom-typing toolset for the CHARMM simulation engine, termed MATCH, that relies on assigning parameters by representing a molecule of interest as a graph and performing subgraph matching against a library of fragments with known parameters. These fragments are represented as “super smiles”, an extension of the SMILES language. By using super smiles and storing these fragments in text files separate from the software, chemical context is expressed without the need to define a rigid if/else hierarchy within the software and thus new atom types and rules (i.e., fragments encoded as super smiles) can be added without modifying the code used to evaluate them. In recent work, Mobley and coworkers[34] have developed an approach to defining force fields, termed SMIRNOFF, that effectively eliminates explicit atom types altogether, instead using SMIRKS (another language related to SMILES[33]) to identify chemical fragments that are associated with a set of force field parameters. Similar to Yesselman, et al.[32], application of the force field relies on re- presenting the system as a graph and rules as subgraphs. In other work, the Enhanced Monte Carlo (EMC) software developed by in’t Veld[35] encodes chemical context of an atom type using SMILES. In all cases, the use of the SMILES-based approaches not only removes the need to encode usage within the source code, but associates parameters with a human and machine readable definition of their chemical context, although, these approaches all still require rules be specified in a particular order to enable correct atom-typing.

In this work, we present `Foyer`, a Python library for performing atom-typing based upon first-order logic over graph structures, designed to address many of the aforementioned issues, with a particular emphasis on reproducibility and the dissemination of force fields to the community. `Foyer` relies upon a force-field-agnostic formalism to express atom-typing and parameterization rules in a way that is expressive enough for human consumption while simultaneously being machine readable, allowing a single, unambiguous format to be constructed for both dissemination and use by software. This logic is implemented via SMARTS[6] to encode chemical context and "overrides" statements to define rule precedence. SMARTS extends the SMILES language to support substructure definitions and allows expression of greater chemical detail and logic operations within the chemical patterns. In `Foyer`, SMARTS has been extended such that it allows user-defined "elements" (not in the periodic table) to be leveraged within the chemical context definitions, thus enabling both atomistic and non-atomistic force fields to be used. By using SMARTS to define chemical context, atom type definitions do not appear in the source code, and thus force fields can be created and evolved without modification to the code used to evaluate them. Rule precedence is explicitly defined by the aforementioned overrides statements, thus atom-typing rules can appear in any order in the file and include recursive definitions to other atom types, eliminating physical placement in the file as a source of error and providing increased flexibility. Since this iterative approach used by `Foyer` evaluates all rules, automated evaluation can be used to help ensure that `Foyer` force field definitions (1) encompass all atom types in the force field and (2) are sufficiently descriptive without conflicting rules, both necessary conditions for publishing force fields in a way that is unambiguous and reproducible. The `Foyer` software provides routines that create syntactically correct input files for a variety of common simulation engines and is designed to take, as input, chemical topologies from several of other community developed tools (e.g., `ParmEd` [13], `OpenMM`[36–38], and mBuild[1, 2, 39]). The manuscript is organized as follows. subsection 4.4.2 introduces the use of SMARTS and overrides for encoding force field usage. This section also presents the XML (Extensible Markup Language) file format used by `Foyer`, which builds upon the `OpenMM` force field file format[36–38] extended to support the definition of the associated SMARTS, overrides statements, textual descriptions of the parameters, and digital objective identifiers (DOIs) for the source of each atom type. subsection 4.4.3 provides an overview of the `Foyer` software used to evaluate the SMARTS and overrides statements encoded in the XML file format, including the iterative process —and its optimizations —used for determining atom types. This section also discusses force field validation and verification within `Foyer`. subsection 4.4.4 provides examples of the use of the `Foyer` software to perform atom-typing of several different chemical systems. subsection 4.4.5 discusses best practices for the use of `Foyer` and force field annotation scheme in terms of reproducibility, focusing on the use of version control and related open-source software development tools to enable the creation and evolution of force fields in a transparent, testable manner. subsection 4.4.6 provides concluding remarks.

4.4.2 Defining chemical context and rule precedence

4.4.2.1 XML file format

`Foyer` utilizes the `OpenMM` force field XML format [38] to encode parameters, where this format is extended to allow for the definitions of chemical context and rule precedence (discussed below). To briefly summarize the `OpenMM` file format, atom types and forces are encoded as XML tags with various attributes defining the types of elements that they apply to (by name only), as well as the associated parameters for that interaction (e.g., the equilibrium bond length and spring constant for a harmonic bond). Listing 4.11 provides an example of encoding the OPLS force field parameters for linear alkanes in the `OpenMM` XML format (note, this Listing does not include our extensions). As shown in Listing 4.11, the XML format provides clear descriptions of each of the parameters/properties defined in the file (e.g., element="C" indicates the entry is defining a carbon atom), along with additional tags that provide unambiguous descriptions of the types of interactions being used (e.g., the`< HarmonicBondForce >` tag is used to define the use of a harmonic force to define bonds). As such, this file format includes a wealth of metadata that is both human and machine readable. For more detailed information, we refer the reader to the `OpenMM` manual where this force field file format is extensively documented [38].

Listing 4.11: OpenMM formatted XML file for linear alkanes using the OPLS force field.

```xml
<ForceField>
<AtomTypes>
    <Type name="opls_135" class="CT" element="C" mass="12.01100"/>
    <Type name="opls_136" class="CT" element="C" mass="12.01100"/>
    <Type name="opls_140" class="HC" element="H" mass="1.00800"/>
</AtomTypes>
<HarmonicBondForce>
    <Bond class1="CT" class2="CT" length="0.1529" k="224262.4"/>
    <Bond class1="CT" class2="HC" length="0.1090" k="284512.0"/>
</HarmonicBondForce>
<HarmonicAngleForce>
    <Angle class1="CT" class2="CT" class3="CT" angle="1.966986067" k="488.273"/>
    <Angle class1="CT" class2="CT" class3="HC" angle="1.932079482" k="313.800"/>
    <Angle class1="HC" class2="CT" class3="HC" angle="1.881464934" k="276.144"/>
</HarmonicAngleForce>
<RBTorsionForce>
    <Proper class1="CT" class2="CT" class3="CT" class4="CT" c0="2.9288" c1="-1.4644"
     ↪  c2="0.2092" c3="-1.6736" c4="0.0" c5="0.0"/>
    <Proper class1="CT" class2="CT" class3="CT" class4="HC" c0="0.6276" c1="1.8828"
     ↪  c2="0.0" c3="-2.5104" c4="0.0" c5="0.0"/>
    <Proper class1="HC" class2="CT" class3="CT" class4="HC" c0="0.6276" c1="1.8828"
     ↪  c2="0.0" c3="-2.5104" c4="0.0" c5="0.0"/>
</RBTorsionForce>
<NonbondedForce coulomb14scale="0.5" lj14scale="0.5">
    <Atom type="opls_135" charge="-0.18" sigma="0.35" epsilon="0.276144"/>
    <Atom type="opls_136" charge="-0.12" sigma="0.35" epsilon="0.276144"/>
    <Atom type="opls_140" charge="0.06" sigma="0.25" epsilon="0.12552"/>
</NonbondedForce>
</ForceField>
```

The flexible nature of XML allows it to be readily extended via the addition of new tags/attributes without fundamentally changing the original format, as new tags/attributes can simply be ignored by software that does not require them. As shown in Table 4.1, and discussed in detail later, four new attributes have been added to the atom type entries in the existing `OpenMM` XML file format to enable the functionality needed to encode usage rules in `Foyer`: def, desc, doi, and overrides. The use of XML additionally allows sanity checks to be performed by using XML schemas to ensure the expected attributes have been provided in the file.

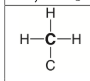
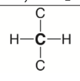
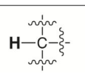
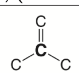
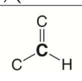
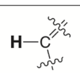
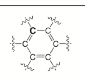
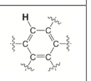
Table 4.1: Extensions to the atom type definitions in the `OpenMM` XML format.

Attribute	Description	Example
`def`	Chemical context of an atom type via SMARTS	`[C;X4](H)(H)(H) C`
`desc`	Textual description of the atom type	`Alkane CH3`
`doi`	Digital object identifier to the atom type source	`10.1021/ja9621760`
`overrides`	Atom type(s) the current rule is given precedence over	`opls_136`

4.4.2.2 Using SMARTS to define chemical context

The chemical context of an interaction site is typically defined by its bonded environment, notably the number of bonds and the identities of bonded neighboring interaction sites, but may also include longer range information, such as the bonded environment of neighbors. To encode this information, `Foyer` utilizes SMARTS[6], a language for defining chemical patterns. SMARTS is an extension of the more commonly used SMILES[33] notation, providing additional tokens that enable users to express greater chemical detail and logic operations. SMARTS notation is expressed as strings that simultaneously include arbitrary chemical complexity but are concise and clear enough for human consumption, in addition to being machine readable. As an example, consider defining the chemical context of OPLS-AA atom types for carbon and hydrogen atoms in a linear alkane, as shown in Listing 4.12 (note, only the < `AtomTypes` > section of the file is shown, as this is the only section that differs from Listing 4.11). The reader is referred to Table 4.2 for a visual depiction of these atom types.

Table 4.2: 2D depictions of molecular fragments referred to in the text.

Alkane			Alkene			Benzene	
C, CH$_3$	**C, CH$_2$**	**H**	**C, (R2-C=)**	**C, (RH-C=)**	**H**	**C**	**H**
							
opls_135	opls_136	opls_140	opls_141	opls_142	opls_144	opls_145	opls_146

To encode the chemical context, the def attribute is added to the `OpenMM` XML format to encode the corresponding SMARTS string. Here, the atom type that specifies the terminal methyl group, "opls135" ("-CH$_3$") can be expressed as [C;X4](C)(H)(H)H in the SMARTS notation. In this SMARTS notation, [C;X4]

Listing 4.12: Atom type definitions for carbon and hydrogen atoms in a linear alkane using the OPLS force field. Note, only the section that applies to atom types is shown for clarity.

```
<ForceField>
    <AtomTypes>
        <Type name="opls_135" class="CT" element="C" mass="12.01100"
         ↪  def="[C;X4](C)(H)(H)H" desc="alkane CH3"/>
        <Type name="opls_136" class="CT" element="C" mass="12.01100"
         ↪  def="[C;X4](C)(C)(H)H" desc="alkane CH2"/>
        <Type name="opls_140" class="HC" element="H" mass="1.00800" def="H[C;X4]"
         ↪  desc="alkane H"/>
    </AtomTypes>
</ForceField>
```

indicates that the element of interest —always the first token in the SMARTS string —is a carbon atom (i.e., C) and this carbon atom has 4 total bonds (i.e., ;X4, where ; indicates the logical operator AND). The identities of the 4 bonded neighbors are 1 carbon atom and 3 hydrogen atoms, expressed as (C)(H)(H)H. Similarly, the opls136 atom type, which describes a methylene group in an alkane, is expressed in SMARTS notation as [C;X4](C)(C)(H)H. Here, the only change from the opls135 definition lies in the identity of the 4 bonded neighbors (2 carbon atoms and 2 hydrogen atoms). Increased chemical complexity can be described by adding details about each of the neighboring interaction sites within SMARTS. For example, the opls140 atom type, which describes a generic alkane hydrogen, is defined as H[C;X4] - a hydrogen atom bonded to a carbon atom with 4 bonds. Multiple valid SMARTS can be defined for each atom type, where, e.g., opls140 could be defined simply as def="H" since there is only a single hydrogen atom type defined in Listing 4.12. However, such a definition would not necessarily provide a user of the force field with a clear understanding of the chemical context for which this atom type applies, and may limit future evolution of the force field. Our extension of the XML file format also includes the desc attribute (e.g., shown in Listing 4.12) that allows for unstructured comments to be provided for each entry if desired.

We note that the parser in the `Foyer` libraries does not currently support the full SMARTS language, instead providing support for the subset that was found to be relevant to the definition of chemical context for atom types. Table 4.3 lists the currently supported primitives, Table 4.4 shows our extensions to the language, and Table 4.5 outlines the logical operators supported.

4.4.2.3 Establishing rule precedence

Rule precedence must be established when multiple atom type definitions can apply to a given interaction site. In typical hierarchical schemes, this is determined implicitly by the order in which rules are evaluated; in general, more specific rules are evaluated first and when a match is found, the code stops evaluating rules altogether. While this approach works, it becomes more challenging to maintain the correct ordering of rules

Table 4.3: Currently implemented SMARTS atomic primitives[1].

Symbol	Symbol name	Atomic property requirements	Default
*	wildcard	any atom	(no default)
A	aliphatic	aliphatic	(no default)
r <n>	ring size	in smallest SSSR[2] ring of size <n>	any ring atom
X <n>	connectivity	<n>total connections	exactly one
#n	atomic number	atomic number <n>	(no default)

[1] This table has been adapted from the Daylight SMARTS website.
[2] Smallest set of smallest rings.

Table 4.4: Extensions to SMARTS atomic primitives.

Symbol	Symbol name	Atomic property requirements	Default
_A	non-element	non-atomistic element	(no default)
%<type>	atomtype	of atomtype <type>	(no default)

as the number of atom types grows and as chemistries become more complex and specific. Users may find it difficult, if not impossible, to make even small additions to a larger force field without breaking existing behavior. `Foyer` allows rule precedence to be explicitly stated via the use of the overrides attribute added to the XML file format. This allows atom type usage rules to be encoded in any order within the file, eliminating incorrectly placed rule order as a source of error. `Foyer` iteratively evaluates all rules on all interaction sites in the system, maintaining for each interaction site a "whitelist" consisting of rules that evaluate to True and a "blacklist" consisting of rules that have been superseded by another rule (i.e., those that appear in the overrides attribute). The set difference between the white- and blacklists of an interaction site yields the correct atom type if the force field is implemented correctly (incorrect/incomplete definition of force fields is discussed later). As an example of a system where overrides need to be defined, consider describing alkenes and benzene in a single force field file, as shown in Listing 4.13 (note, only the < AtomTypes > section of the force field file is shown). The reader is again referred to Table 4.2 for visual depictions of the relevant atom types.

When atom-typing a benzene molecule, the carbon atoms in the ring will match the SMARTS patterns for both opls142 (an alkene carbon) and opls_145 (a benzene carbon). Without the overrides attribute, `Foyer` will find that multiple atom types apply to each carbon atom. Providing the overrides indicates that if the

Table 4.5: SMARTS Logical Operators[1].

Symbol	Expression	Meaning
exclamation	$!e_1$	not e1
ampersand	$e_1\&e_2$	e1 and e2 (high precedence)
comma	e_1,e_2	e1 or e2
semicolon	$e_1;e_2$	e1 and e2 (low precedence)

[1] This table has been adapted from the Daylight SMARTS website.

Listing 4.13: Atom type definitions for alkenes and benzene using the OPLS-AA force field highlighting the overrides syntax and mechanism for referencing other atom types. Note, only the section that applies to atom types is shown for clarity.

```
<ForceField>
    <AtomTypes>
        <Type name="opls_141" class="CM" element="C" mass="12.01100"
         ↪  def="[C;X3](C)(C)C" desc="alkene C (R2-C=)"/>
        <Type name="opls_142" class="CM" element="C" mass="12.01100"
         ↪  def="[C;X3](C)(C)H" desc="alkene C (RH-C=)"/>
        <Type name="opls_144" class="HC" element="H" mass="1.00800"
         ↪  def="[H][C;X3]" desc="alkene H"/>
        <Type name="opls_145" class="CA" element="C" mass="12.01100"
         ↪  def="[C;X3;r6]1[C;X3;r6][C;X3;r6][C;X3;r6][C;X3;r6][C;X3;r61"
         ↪  overrides="opls_142"/>
        <Type name="opls_146" class="HA" element="H" mass="1.00800"
         ↪  def="[H][C;%opls_145]" overrides="opls_144" desc="benzene H"/>
    </AtomTypes>
</ForceField>
```

opls145 pattern matches, it will supersede opls_142. Thus, the difference between the whitelist (containing opls142 and opls_145) and blacklist (containing only opls_142) would be opls_145.

Note that multiple atom types can be listed in a single overrides attribute. The approach taken here also allows atom types to inherit overrides from the atom types they override. For example, consider a case in which atom types 1, 2 and 3 each evaluate to True for an interaction site. If atom type 3 overrides atom type 2 (i.e., adds atom type 2 to the blacklist) and atom type 2 overrides atom type 1 (i.e., adds atom type 1 to the blacklist), then atom type 3 will implicitly override atom type 1. Additionally, in `Foyer`, the SMARTS grammar has been modified such that specific atom type names can also be included within the definition (see Table 4.4). For example, opls146, the hydrogen atom attached to carbon atoms in a benzene ring, has the SMARTS definition [H][C%opls_145], as shown in Listing 4.13; This states that the interaction site of interest is a hydrogen atom (H) and is bonded to a carbon atom that has atom type opls_145 (C;%opls_145). Because `Foyer` evaluates rules iteratively for each interaction site, such recursive definitions can be utilized without the need to explicitly define atom types in a chemical topology input file. For example, in this case, when `Foyer` identifies the interaction site of a carbon atom to be opls145, the next iteration to evaluate the hydrogen atom will find that opls_146 now evaluates to True. Similar to how an overrides statement clearly defines precedence, this recursive definition provides a clear way to identify chemical context and the relationship between different atom types for highly specific parameters. We note, that one could also replace the recursive reference to opls145 with its SMARTS string, although, in this case, it would result in a more complex, less human readable definition.

Because the logic used to define chemical context is separated from the source code used to evaluate it, one can construct a force field file that contains only the relevant subset of atom types need for a given application area. Using the above example of benzene and alkenes, if a system only contained benzene molecules, one could avoid specifying the overrides attributes altogether by simply creating a force field file containing only atom types relevant to benzene and eliminating those associated with alkenes. In many cases, considering smaller subsets is beneficial as the amount of effort required to differentiate and set rule precedence between atom types is reduced. Additionally, using smaller files will reduce the likelihood of errors related to defining chemical context and rule precedence, reduce the number of test molecules with known atom types required to fully validate the rules, and increase the readability of the force field files by limiting the number of entries.

4.4.2.4 Extension of SMARTS for non-atomistic systems

`Foyer` is able to atom-type systems in which an interaction site does not represent a single atom with a standard element, but instead may represent a group of atoms (relevant to united-atom and coarse-grained (CG) force fields) or a generic site (relevant to simplified models). Standard SMARTS notation does not support non-atomic species due to its reliance on the presence of an element specification for each interaction site. To circumvent this limitation, the `Foyer` SMARTS parser allows users to define custom "elements" by prefixing their string representation with an underscore (see Table 4.4). For example, CCC could represent a coarse-grained interaction site intended to model three carbon atoms. In its current implementation, `Foyer` makes a first pass through force field files to detect any custom element definitions. These are injected into the grammar that parses SMARTS strings and are given priority over standard elements. This allows non-atomistic and atomistic atom types to be used either separately or together.

In practice, united-atom and coarse-grained force fields can be defined in an almost identical fashion to all-atom force fields, where the only difference is that "elements" are user-defined strings prepended with an underscore. As an example, consider an alkane modeled with the united-atom TraPPE force field[40, 41]. An interaction site in this force field represents both carbon and the hydrogen atoms bonded to it. Thus, this force field contains two distinct atom types, one that represents (CH_3) and one that represents (CH_2). These can be encoded as shown in Listing 4.14.

Focusing on atom type CH3_sp3, usage is encoded with the definition [_CH3;X1][_CH3,_CH2] which states that the base "element" is _CH3 with one bond (i.e., ;X1) to either a _CH3 or a _CH2 group. In SMARTS, a comma indicates an "OR" logic statement and a semicolon is used to denote an "AND" logical statement (see Table 4.5 for a complete list of SMARTS logical operators). In this example, [CH3;X1] states the element must be _CH3 "AND" have only a single bond. Atom-type _CH2_sp3, which represents a "middle" alkane carbon and its 2 associated hydrogen atoms, is defined similarly as [_CH2;X2]([_CH3,_CH2])[_CH3,_CH2]. Here, the base "element" is a CH2 with two bonds (i.e., ;X2), each of which may be either "element" _CH3 or _CH2. Note, the interaction sites defined in the input chemical topology would need to follow the same

Listing 4.14: Atom type definitions for alkenes and benzene using the OPLS-AA force field highlighting the overrides syntax and mechanism for autoreferencing other atom types. Note, only the section that applies to atom types is shown for clarity.

```
<ForceField>
    <AtomTypes>
        <Type name="CH3_sp3" class="CH3" element="_CH3" mass="15.03500"
         ↪  def="[_CH3;X1][_CH3,_CH2]" desc="Alkane CH3, united atom"/>
        <Type name="CH2_sp3" class="CH2" element="_CH2" mass="14.02700"
         ↪  def="[_CH2;X2]([_CH3,_CH2])[_CH3,_CH2]" desc="Alkane CH2, united
         ↪  atom"/>
    </AtomTypes>
</ForceField>
```

naming convention as the force field file, labeled as CH3 and _CH2.

4.4.2.5 Determining bonded parameters

Once a chemical topology is atom-typed, bonded interactions can be determined by simply searching for the matching pairs, triplets, and quartets (bonds, angles, and torsions, respectively). In many force fields, the bonded parameters are not as specific as the non-bonded interactions, and thus are not defined directly based on atom types. Thus, rather than atom types, a more general class identifier (some- times referred to as the “bond family”) is used to identify these interactions. In Listing 4.15, both opls136 and opls_962 are part of the same class “CT”. Thus a bond between opls136-opls_962 would have the same parameters (defined as class1="CT" class2="CT" in Listing 4.15) as a bond between opls_136-opls_136 (also defined as class1="CT" class2="CT"). However, this general approach breaks down for certain chemical topologies.

For example, while the atom types for carbon atoms in alkanes[18] and perfluoralkanes[42] are both of class “CT” and share the same bond and angle parameters for carbon atoms, they differ in terms of torsional parameters. In order to handle this conflict, many codes require users to comment out the more general set of parameters or include statements within the code that accomplish the same task. However, in this approach, one would not be able to atom-type a system composed of a mixture of alkane and perfluoroalkane molecules, since only one set of parameters can be included simultaneously. Note that while one could define a new class to differentiate between alkanes and perfluoroalkanes, this would result in a force field file with many duplicate parameters sets that simply have different labels.

The `OpenMM` format allows bonded parameters to be defined using the type attribute in place of the class attribute, where type refers directly to the name attribute that stores the atom type, allowing for bonded interactions to be defined with increased specificity. Additionally, mixed use of type and class in the definition of these bonded interactions is supported. Referring to Listing 4.15, to provide the necessary distinction between torsional parameters for perfluoroalkanes and alkanes, one could define perfluoroalkane torsions using type

Listing 4.15: Force field XML snippet showing atom types defined for carbon in CH_2 and CF_2 substructures, a bonded definition between carbons, and C-C-C-C dihedral definitions for hydrogenated and perfluorinated alkanes.

```
        <ForceField>
        <AtomTypes>
            ...
            <Type name="opls_136" class="CT" element="C" mass="12.01100"
             ↪  def="[C;X4](C)(C)(H)H" desc="alkane CH2"/>
            <Type name="opls_962" class="CT" element="C" mass="12.01100"
             ↪  def="[C;X4](C)(C)(F)F" desc="perfluoroalkane CF2" />
            ...
        </AtomTypes>
        <HarmonicBondForce>
            ...
            <Bond class1="CT" class2="CT" length="0.1529" k="224262.4"/>
            ...
        </HarmonicBondForce>
        ...
        <RBTorsionForce>
            ...
            <Proper class1="CT" class2="CT" class3="CT" class4="CT" c0="2.9288"
             ↪  c1="-1.4644" c2="0.2092" c3="-1.6736" c4="0.0" c5="0.0"/>
            <Proper type1="opls_962" type2="opls_962" type3="opls_962" type4="opls_962"
             ↪  c0="14.91596" c1="-22.564312" c2="-39.41328" c3="11.614784" c4="35.446848"
             ↪  c5="0.0"/>
            ...
        </RBTorsionForce>
            ...
        </ForceField>
```

attributes (i.e., type1="opls962" type2="opls_962" type3="opls_962" type4="opls_962", where opls_962 is defined in the `< AtomTypes > XML section`), and alkane torsions with the more general quartet for alkanes of class1="CT" class2="CT" class3="CT" class4=CT" that uses class attributes. However, when iterating through bonded parameter definitions, `OpenMM` assigns parameters based on the first match found. In the example described above, perfluoroalkane torsional parameters would therefore need to be defined before alkane parameters in the torsional section, and thus the ordering shown in Listing 4.15 would result in the incorrect assignment of torsional parameters for perfluoroalkanes. Several approaches can be taken to address this. overrides statements could be used to set rule precedence for bonded topologies and thus eliminate the need to specify order in the file, however additional modification to the force field file format would be required because bonded parameters do not have a “name” attribute like atom types. In the approach taken by SMIRNOFF[34], bonded parameters are defined directly using their chemical context (i.e., via SMIRKS), eliminating this issue altogether; however, taking a similar approach would result in the duplication of many parameters in the same way as defining a new class attribute. A more simple approach taken by `Foyer` is to perform a preprocessing step on the bonded parameters. This step orders bonded parameters such that the most specific cases are sorted to the top of the list to set precedence. This accomplished by assigning a weight to each entry proportional to the number of type attributes included (as these are the most specific). For example, a torsion that explicitly defines the atom types for which it applies (i.e., has 4 type attributes) would be given the highest weight, and sorted to the top of the list, whereas an entry that specifies only class attributes would be given the lowest. Thus, for the force field XML shown in Listing 4.15, `Foyer` would reverse the order of the two defined dihedrals during preprocessing.

4.4.3 `Foyer` Software

In order to read the force field usage specification discussed above and perform atom-typing, the `Foyer` software has been developed as an open-source Python library. Python allows for portability between platforms and provides a wealth of freely available modules (e.g., `NumPy` [43], `SciPy` [44], `NetworkX` [45]) to facilitate many of the underlying operations. The source, documentation, tutorials, and examples of `Foyer` are freely available and can be found on the GitHub project repository[12], tutorial repository[46], and website[47]. Figure 4.6 provides an overview of the general software workflow, which we will discuss here.

4.4.3.1 Inputs and preprocessing

`Foyer` accepts, as input, the XML force field file and an input chemical topology for which to apply the force field. In addition to sorting bonded parameters by specificity as described in the previous section, the XML force field file undergoes a preprocessing and validation step via application of an XML schema definition. Here `Foyer` enforces which elements (e.g. HarmonicBondForce) are valid and how their attributes should be formatted. While this does not test the accuracy of the parameters, it does ensure that all of the expected parameters are defined. Additionally, the schema ensures that atom types are not defined more than once and that atom types referenced in other sections (e.g., < HarmonicBondForce >) are actually defined in the < AtomTypes > section. Next, the SMARTS strings defined by the def attribute for all atom types are parsed and checked for validity. This does not validate whether a SMARTS string is correctly defined for a given interaction site but simply ensures that the SMARTS string can be interpreted by `Foyer` and does not contain any erroneous characters. Parsing errors are captured and re-raised with error messages that allow a user to pin point the location of the problem in the XML file and within the SMARTS string. Wherever possible, `Foyer` attempts to provide helpful suggestions for fixing detected errors.

Input chemical topologies can be passed to `Foyer` through various data structures; the current version supports the `OpenMM` Topology object [36–38], the `ParmEd` Structure object[13], and the `mBuild` Compound object[1, 2, 39]. Each of `OpenMM`, `ParmEd`, and `mBuild` topologies support inputs from a variety of common molecular file formats, such as PDB and MOL2, and thus it is typically straightforward to convert

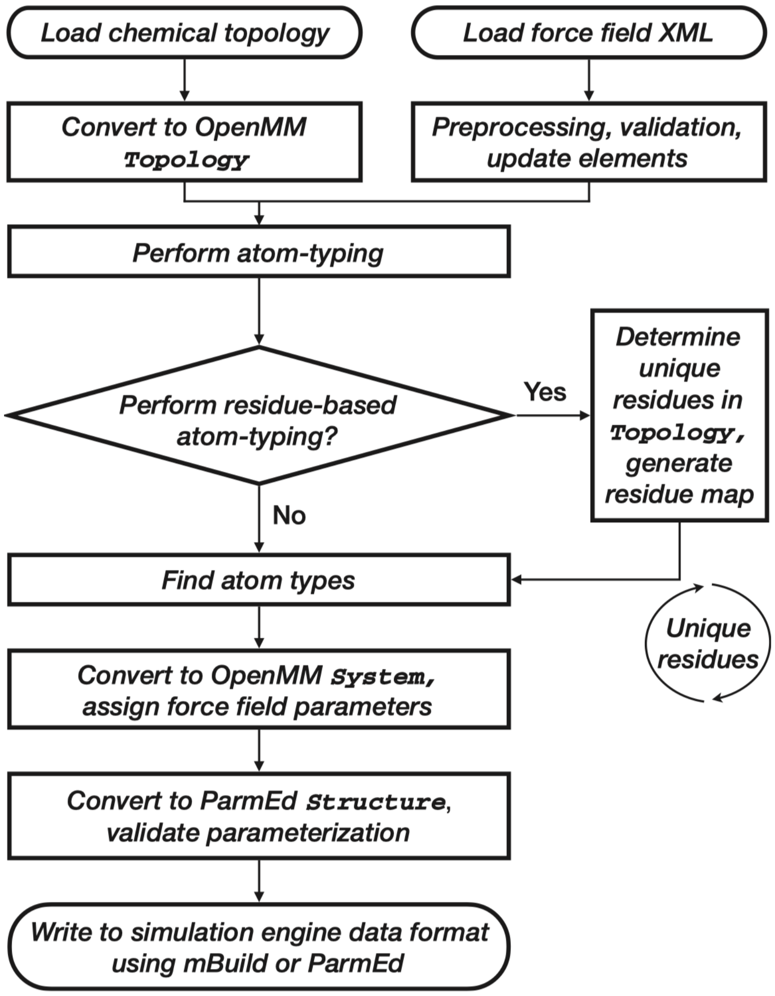

Load chemical topology
Load force field XML
Convert to OpenMM Topology
Preprocessing, validation, update elements
Perform atom-typing
Perform residue-based atom-typing?
Yes
No
Determine unique residues in Topology, generate residue map
Find atom types
Unique residues
Convert to OpenMM System, assign force field parameters
Convert to ParmEd Structure, validate parameterization
Write to simulation engine data format using mBuild or ParmEd

Figure 4.6: Flowchart of the `Foyer` software from chemical topology and force field XML inputs to a simulation data file output.

a given system into a data structure that `Foyer` can accept. Regardless of the input format, once read into `Foyer` the chemical topology is converted to an `OpenMM` Topology object. The `OpenMM Topology` object provides a standardized data container to store the necessary system information and allows for leveraging of routines already defined within `OpenMM`'s library.

4.4.3.2 Atom-typing

A flowchart of `Foyer`'s atom-typing procedure is shown in Figure 4.7. To perform atom-typing, `Foyer` constructs a graph of the complete system defined by the chemical topology (or alternatively a graph of each unique residue, see the Residue-based Atom-typing section below) and iteratively searches for SMARTS matches via subgraph isomorphism (where subgraphs are generated for each SMARTS definition). Graph construction and matching are performed using the `NetworkX` package[45], an open-source Python project that provides an intuitive interface for a multitude of graph-based algorithms and is the de facto standard network analysis library in Python. During this step, the iterative process of determining the atom type is

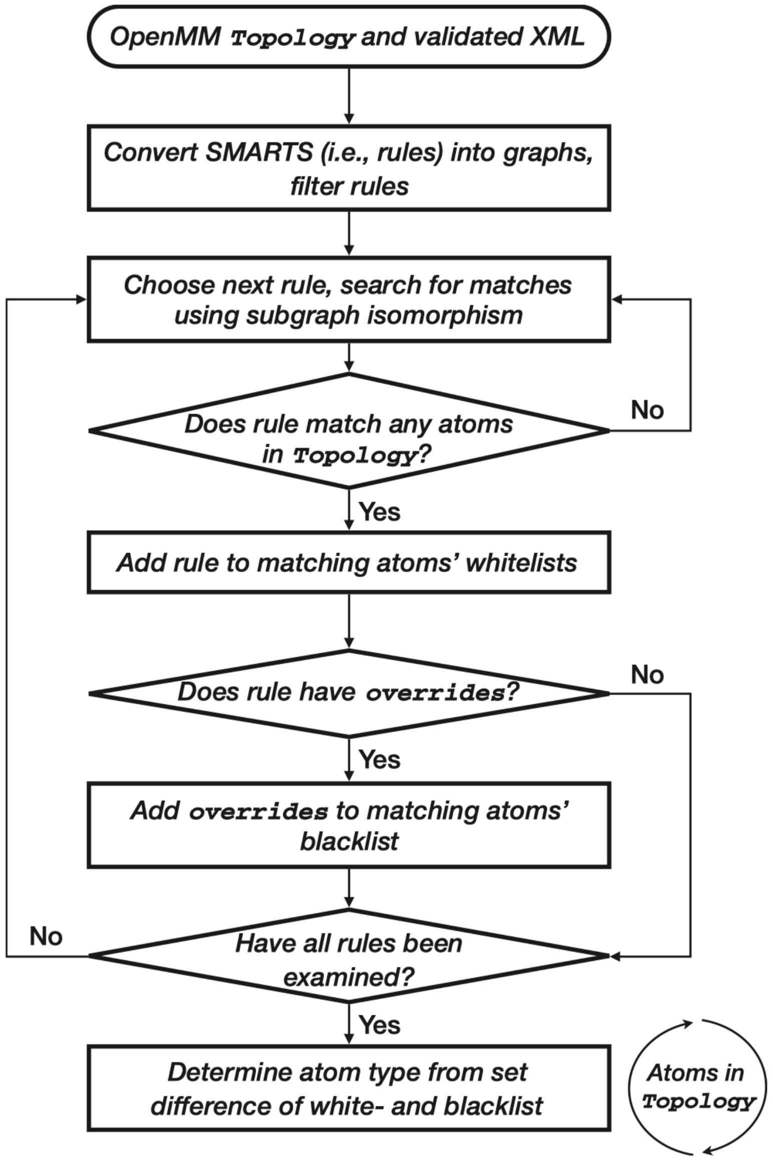

OpenMM Topology and validated XML
Convert SMARTS (i.e., rules) into graphs, filter rules
Choose next rule, search for matches using subgraph isomorphism
Does rule match any atoms in Topology?
No
Yes
Add rule to matching atoms' whitelists
Does rule have overrides?
No
Yes
Add overrides to matching atoms' blacklist
Have all rules been examined?
No
Yes
Determine atom type from set difference of white- and blacklist
Atoms in Topology

Figure 4.7: Flowchart of `Foyer`'s atom-typing process.

undertaken, adding rules to the white and back lists for each interaction site in the system.

The implementation of the SMARTS based atom-typing scheme is comprised of several steps and internally relies on a subgraph isomorphism to detect matches as highlighted in Figure 4.8. First, a SMARTS string is parsed into an abstract syntax tree (AST) from which we populate a SMARTSGraph object. This class inherits from the Graph class in the NetworkX package. Elements in this `SMARTSGraph` are represented as nodes and chemical bonds as edges. Inheriting from `NetworkX` is convenient in that it allows

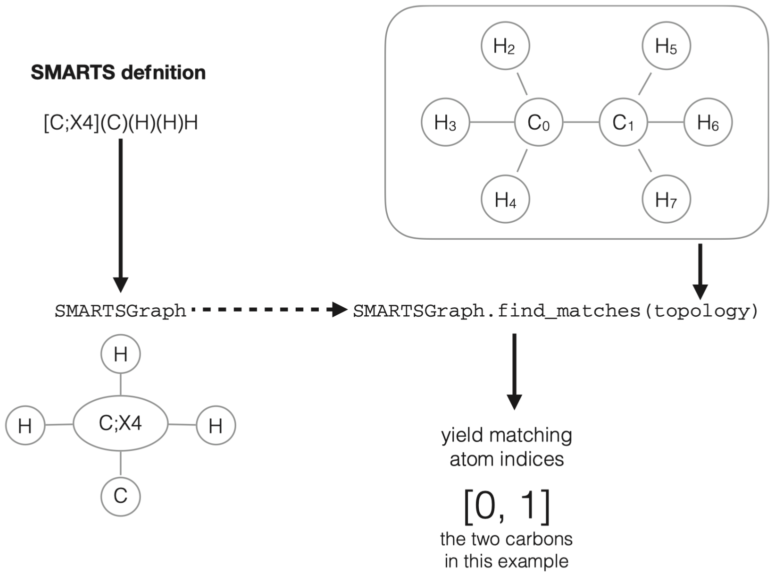

SMARTS defnition
[C;X4](C)(H)(H)H
SMARTSGraph
SMARTSGraph.find_matches(topology)
H2
H5
H3
C0
C1
H6
H4
H7
H
C;X4
C
yield matching atom indices
[0, 1]
the two carbons in this example

Figure 4.8: Schematic of the workflow to apply SMARTS patterns to chemical topologies. The SMARTS strings used to define atomtypes are read into a `SMARTSGraph` class which inherits from `NetworkX`'s core data structure. Using the `find_matches` method, a `SMARTSGraph` instance can search for subgraph isomorphisms of itself within a provided chemical topology and will yield all atoms that match the first token in the original SMARTS string —the atom type that we are looking for.

us to leverage most of the algorithms and visualization methods already implemented there. The primary distinguishing feature of the `SMARTSGraph` is the set of methods that encode the logic for matching the more complex SMARTS tokens. These methods can be directly used by `NetworkX`'s implementation of the VF2 subgraph isomorphism algorithm[7]. A thin wrapper provided by the `findmatches` method allows a `SMARTSGraph` instance to search for all subgraph isomorphisms within a bare chemical topology (an non-atom-typed graph of just elements and bonds). This method returns the indices of all elements that match the first token in the SMARTS string, which defines the atom type that we are looking for. Successfully matching elements have the atom type definition added to their whitelist and any overridden types added to their blacklist. The appropriate atom type for an interaction site is determined by examining the difference between white- and blacklists, where a sufficiently descriptive force field should yield only a single atom type as the difference between the two lists.

The use of white- and blacklists provides users with a means to validate the completeness of the chemical contexts defined by the set of SMARTS strings and overrides. For example, when considering a test molecule, if multiple valid atom types are found as the difference between white- and blacklists, this indicates the rules are not sufficiently unique and likely have incomplete information provided to the override attributes. `Foyer` provides the list of conflicting types to aid in resolving such issues. If no atom types exist as the difference, the interaction site of interest cannot be described by the force field rules as implemented. Typically, this will require adding a new atom type or amending an existing atom type's definition. This may also indicate, that there is an error in how rule precedence has been defined, such as, all the rules on the whitelist "overriding" each other. The efficacy of this type of validation in `Foyer` will depend on providing a sufficient range of systems to fully explore the combinations of atom types that can be applied, where, as a general rule, the set of systems chosen to perform validation tests should collectively utilize all atom types defined in the force field. Note, these validation tests can be done to identify conflicts and under-defined systems, but do not necessarily indicate that the force field has been implemented corrected; separate verification tests are needed to ensure proper implementation, whereby the atom types identified by `Foyer` are compared to that of molecules with known, validated atom types, as discussed later.

4.4.3.2.1 Residue-based Atom-typing

Many systems of interest to molecular simulation contain topologies that consist of duplicates of smaller molecules or repeat units, each with identical topologies. A brute-force implementation of the atom-typing process wastes time by repeating subgraph isomorphism computation on each repeat unit and thus would not scale well with system size. To eliminate unnecessary calculations, a map of atom-typed residues is saved after each unique residue is atom-typed the first time. Then, when an identical residue is found, it copies the atom-typed information from the residue map instead of repeating the subgraph isomorphism. This feature is enabled by default but can optionally be turned off.

As an example, consider a box of *N* hexane molecules. After the subgraph isomorphism is called on the first molecule, the result is copied and saved into a map. Then, when molecules 2 to *N* are encountered, those results are copied into the running topology. The time it takes for the apply function to finish is timed for each case and plotted in Figure 4.9 relative to the brute force approach, where significant speed improvements are observed for common system sizes.

4.4.3.3 Force field assignment and output

Once atom-typed, the `OpenMM` Topology, now containing atom types for all particles in the system, is used to create an `OpenMM` System object and bonded parameters of systems determined. This step can be accomplished by simply searching the list of bonded parameters for the appropriate pair, triplet, quartet of atom types for bonds, angles, and dihedrals, respectively; such routines exist within `OpenMM` and are utilized in this context, where again we note these interactions undergo a sorting to ensure more specific definitions appear first in the file. Additionally, validation checks are performed at this time to ensure all triplets and quartets of interaction sites have had angle and dihedral (proper and improper) parameters assigned (checks for bond parameterization of interaction site pairs are performed by `OpenMM` in the prior step). These validation checks

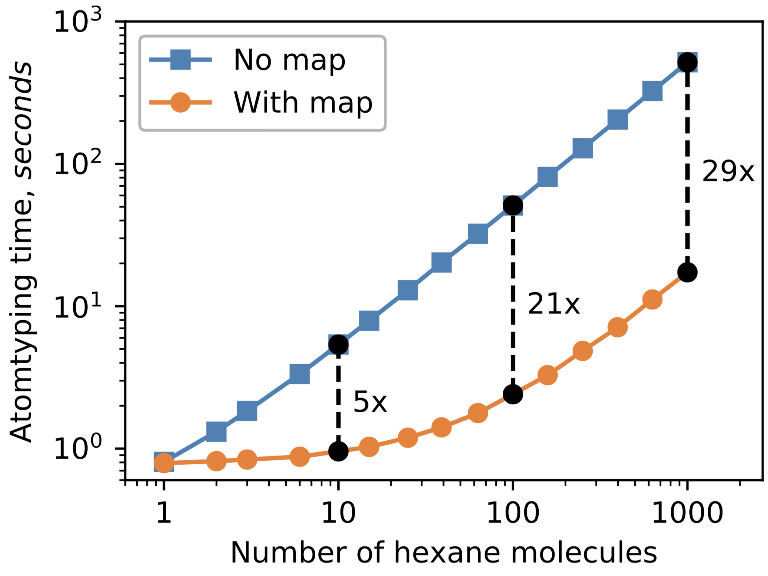


Figure 4.9: Comparison of atom-typing cost with and without the use of residue templates. Without a residue template map, the scaling is approximately linear with system size. With a map, the scaling is independent of system size for small systems and becomes approximately linear at larger system sizes due to operations other than the subgraph isomorphism. The speedup approaches a factor of approximately 30 as the system size becomes large. The times to atom-type each system were obtained with a 2013 MacBook Pro, 3 GHz Core i7, 8 GB RAM.

provide the user with an error (that can optionally be overridden) to help prevent the return of incorrectly parameterized Structures. To output the atom-typed system into a usable format for a simulation engine, the fully atom-typed and parameterized system is returned as a ParmEd Structure object. Through the use of the ParmEd Structure, `Foyer` has access to various additional functionality, such as I/O routines that properly parse the ParmEd Structure into common chemical file formats (MOL2, PDB, `NAMD` and `GROMACS` formats, among others[13]). For file formats not natively supported by ParmEd, custom I/O routines for outputting to these formats (e.g., the `LAMMPS` data file format) have been developed within the mBuild package.

It should be noted that by utilizing `ParmEd` to take advantage of the extensive I/O routines, the force fields that `Foyer` currently supports must match functional forms supported by the internals of the ParmEd Structure object. For example, non-bonded interactions are currently limited to a 12-6 Lennard-Jones functional form. However, due to the large amount of force fields that utilize this functional form (e.g., Amber[16], GAFF[48], OPLS[18], TraPPE[40, 41], CHARMM[17]), the current version of `Foyer` is still widely applicable; planned future development will include support for additional functional forms to better accommodate the diverse force field landscape that exists.

4.4.3.4 Validating/verifying Output

Foyer provides scripts to validate its output files by comparing against systems with known atom types (e.g., those determined by hand or reference molecules provided by a force field developer). Output validation requires (1) system(s) with known, validated atom types and (2) the force field XML file. The known systems are read into Foyer and atom types are determined using the rules in the XML file. The atom types generated by Foyer are then compared against the known atom-typed system(s). The pytest [49] library is used to provide a clear, descriptive output of the results of these validation tests. Implementing output validation tests is particularly useful to force field developers as they ensure that the desired output is retained if a force field file is evolved through the addition of new atom types definitions or merged with a separate force field file. The utility of these validation checks relies not only upon providing accurate reference systems, but also a sufficient variety of test systems that encompass all defined atom types, as discussed previously. An example of such a validation test suite is provided as part of the Foyer template repository freely available on GitHub[50].

4.4.4 Usage Examples

At the time of publication, Foyer includes example force field XML files with def, overrides, and doi statements for 110 OPLS atom types as well as parameters and atom types for the simulation of alkanes and primary alcohols using the TraPPE force field; a set of molecules with known atom types (162 for OPLS, 12 for TraPPE) are also included for automated testing of the code. As discussed later in Section 5, separate repositories have been created to demonstrate how to reproducibility distribute force fields. These include OPLS compatible sets of parameters for perfluoropolyethers[51, 52], perfluoroalkanes[53], and alkylsilanes grafted silica substrates[54]. Implementation of additional atom types for OPLS and TraPPE force fields and the implementation of other ParmEd compatible force fields is an active area of work.

Here, a basic overview of the usage of Foyer is provided, although we direct readers to the GitHub project repository[12] and tutorial repository[46] for additional usage examples. Consider constructing a bulk system of ethane molecules and applying the OPLS force field. Listing 4.16 shows a simple mBuild script to load an ethane molecule and fill a 2 nm x 2 nm x 2 nm box with 100 molecules. This defines the system's chemical topology to which the force field will be applied. As input, the force field file is identical to Listing 4.11 but with the < AtomType > information from Listing 4.12, as Listing 4.12 includes the usage rule definitions. Listing 4.16 demonstrates two different syntaxes for applying a force field using Foyer and saving the output, in this case to the file format required by GROMACS. The second option allows different forcefields to be applied to different topologies in the system. Listing 4.17 shows an example of creating two separate chemical topologies in the system, and applying two different force field files to each. The two atom-

Listing 4.16: Script to fill a box with ethane and apply the OPLS-AA force field to the system.

```python
import mbuild as mb
from mbuild.examples import Ethane
from foyer.test.utils import get_fn
from foyer import Forcefield

""" Approach 1 """
# create the chemical topology
ethane_fluid = mb.fill_box(compound=Ethane(), n_compounds=100, box=[2,2,2])
# apply and save the topology
ethane_fluid.save("ethane-box.top", forcefield_files=get_fn("oplsaa_alkane.xml"))
ethane_fluid.save("ethane-box.gro")

""" Approach 2 """
# create the chemical topology
ethane_fluid = mb.fill_box(compound=Ethane(), n_compounds=100, box=[2,2,2])
# load the forcefield
opls_alkane = Forcefield(forcefield_files=get_fn("oplsaa_alkane.xml"))
# Apply the forcefield to atom-type system
ethane_fluid_typed = opls_alkane.apply(ethane_fluid)

# Save the atom-typed system
ethane_fluid_typed.save("ethane-box.top", overwrite=True)
ethane_fluid_typed.save("ethane-box.gro", overwrite=True)
```

Listing 4.17: Script to build a system with an amorphous silica substrate in contact with a bulk ethane system and apply a different force field to the substrate and fluid respectively.

```python
import mbuild as mb
from mbuild.examples import Ethane
from mbuild.lib.atoms import H
from mbuild.lib.bulk_materials import AmorphousSilica
from foyer.test.utils import get_fn
from foyer import Forcefield

# create a silica substrate, capping surface oxygens with hydrogen
silica = mb.SilicaInterface(bulk_silica=AmorphousSilica())
silica_substrate = mb.Monolayer(surface=silica, chains=H(), guest_port_name="up")
# determine the box dimensions dictated by the silica substrate
box = mb.Box(mins=[0,0,max(silica.xyz[:,2])], maxs=silica.periodicity + [0,0,4])
# fill the box with ethane
ethane_fluid = mb.fill_box(compound=Ethane(), n_compounds=200, box=box)
# load the forcefields
opls_alkane = Forcefield(forcefield_files=get_fn("oplsaa_alkane.xml"))
opls_silica = Forcefield(forcefield_files=get_fn("oplsaa-silica.xml"))
# Apply the forcefields to atom-type system
ethane_fluid_typed = opls_alkane.apply(ethane_fluid)
silica_substrate_typed = opls_silica.apply(silica_substrate)
# merge the two structures
system = silica_substrate_typed + ethane_fluid_typed
# Save the atom-typed system
system.save("ethane-silica.top", overwrite=True)
system.save("ethane-silica.gro", overwrite=True)
```

typed structures that result (ethane fluid and silica_substrate) are then combined using a simple + operator and saved to any format supported by `ParmEd` (this assumes that cross interactions between ethane and silica are defined using standard mixing rules). Note that if the surface and polymers were bonded together (e.g., to create a surface-bound monolayer), the force field files would need to be combined into a single XML document.

4.4.5 Promoting reproducible force field dissemination

Force field files and associated documentation, examples, and validation tests, can be readily developed and distributed using standard software development approaches to improve quality and reproducibility. For example, the common git + GitHub/Bitbucket based distribution process allows force field creators to disseminate their force field files and associated content to the public via a version controlled repository that can be referenced from relevant publications. In this approach, a specific version of the force field used in a publication can be tagged in the git repository and a reference to this tagged version provided in the manuscript, allowing for a clear reference to the exact parameters and usage rules employed in the work. Other services, such as Zenodo[55], can additionally provide a digital object identifier (DOI) for the tagged record and a snapshot of the content of the archive. For example, the software and examples associated with this publication have been tagged on GitHub as "paperCOMMAT_2019" (see, Ref. [56]) with DOI: 10.5281/zenodo.2880526 archiving this tag (see Ref. [12]). A variety of other features of this standard software development process translate well to force field development. Version control systems like git are designed to facilitate distributed, collaborative software development and allow for changes to the files in the repository to be easily tracked in a transparent manner. For example, as a force field is evolved or corrected, revisions can be easily tracked, including the author(s) responsible for the changes, and the specific differences between force field versions clearly identified using standard tools such as DIFF and through the use of descriptive "commit" statements as the content of the repository is changed. The support for tracking issues in services such as GitHub/Bitbucket additionally allow the community to provide feedback, request clarification, or identify errors in a file in a transparent manner. Whenever the developers wish to they can create a new release of the force field that, as noted above, can be tagged or provided with a citable DOI. Verification and validation of a force field can also be simplified by using this software design approach, by implementing automated testing tools that can perform checks on every new iteration (i.e., commit) of the force field content, to ensure errors are not introduced as the force field is changed.

To promote these practices, we have created a template git repository on GitHub which contains the basic framework needed to create, test, and publish a new force field as well as a guided tutorial that introduces users to the SMARTS based atom-typing scheme[50]. This process was successfully used in recent work that derived force field parameters for perfluoropolyethers[52], a novel lubricant class. The force field was published in conjunction with the manuscript and made freely available on GitHub[53]. The specific version of the force field at time of publication is citable via a separate DOI[51]. Any adjustments or improvements to the force field could now be released under a new DOI while the old one would still exist and point to the originally published force field in order to maintain provenance.

4.4.5.1 Atom type DOI labels

While automated atom-typing and the containment of atom types and force field parameters within a single file helps reduce user error and promotes reproducibility, users also require knowledge of the original source of parameters, in order to ensure proper citation and validation that the parameters are appropriate for their system of interest. Foyer achieves this goal by adding a doi attribute to each Type definition within the AtomTypes block of a force field XML. Listing 4.18 shows the same atom-type definition for the OPLS-AA methyl carbon as in Listing 4.12 with the additional doi attribute providing the DOI to the original source where parameters for this atom type were derived.

Listing 4.18: Atom type definition for a methyl carbon tagged with the source DOI

```xml
    <ForceField>
        <AtomTypes>
            <Type name="opls_135" class="CT" element="C" mass="12.01100"
              ↪  def="[C;X4](C)(H)(H)H" desc="alkane CH3" doi="10.1021/ja9621760"/>
        </AtomTypes>
    </ForceField>
```

This feature eliminates ambiguity concerning the origin of parameters for a particular atom type. Furthermore, Foyer automatically logs associations between DOIs and atom types during the atom-typing process, providing a BibTeX file featuring the full citation for the sources of all parameters applied to a particular system, along with additional notes detailing precisely which atom types are contained within each source. For example, Listing 4.19 shows the BibTeX file generated for a nitropropane molecule using the OPLS-AA force field, which includes all reference information as well as notes describing which atom type parameters were obtained from each reference.

4.4.6 Conclusions

The Foyer Python library and annotation scheme for defining force field usage has been presented in this work. Foyer defines a general applicable approach for the specification of classical force fields in a format that is both human and machine readable and provides tools for automated atom-typing and validation. The force field annotation scheme used in Foyer defines parameters and their usage within an XML formatted force field, allowing force fields to be developed and evolved without the need to modify the source code used to evaluate them. This annotation scheme uses SMARTS to define the chemical context of an atom type and defines rule precedence via explicit override statements, allowing rules to appear in any order in a force

Listing 4.19: BibTeX file generated during atom-typing of nitropropane using the OPLS-AA force field (modified with line breaks for readability).

```
        @article{Price_2001,
        doi = {10.1002/jcc.1092},
        url = {https://doi.org/10.1002},
        year = {2001},
        publisher = {Wiley-Blackwell},
        volume = {22},
        number = {13},
        pages = {1340-1352},
        author = {Melissa L. P. Price and Dennis Ostrovsky and William L. Jorgensen},
        title = {Gas-phase and liquid-state properties of esters, nitriles, and nitro
    ↪ compounds with the OPLS-AA force field},
        journal = {Journal of Computational Chemistry},
        note = {Parameters for atom types: opls_761, opls_760, opls_764, opls_763}
        }

        @article{Jorgensen_1996,
        doi = {10.1021/ja9621760},
        url = {https://doi.org/10.1021},
        year = {1996},
        month = {jan},
        publisher = {American Chemical Society (ACS)},
        volume = {118},
        number = {45},
        pages = {11225-11236},
        author = {William L. Jorgensen and David S. Maxwell and Julian Tirado-Rives},
        title = {Development and Testing of the OPLS All-Atom Force Field on
    ↪ Conformational Energetics and Properties of Organic Liquids},
        journal = {Journal of the American Chemical Society},
        note = {Parameters for atom types: opls_135, opls_140, opls_136}
        }
```

field file. The Foyer software treats chemical topologies as graphs and rules as subgraphs, to identify atom types, using an iterative approach. This iterative approach allows force fields to be automatically evaluated for completeness and identify under specified force fields, helping to prevent the dissemination and usage of ambiguously defined force fields. Foyer is designed to be compatible with several common simulation engines and utilizes/extends several existing open-source Python tools and file formats, in order to maximize flexibility and general applicability. Collectively, the approach utilized by Foyer can used to improve the clarity of force field usage and dissemination within the molecular simulation community. The Foyer software is open-source and freely available via GitHub [12, 47, 57].

References

1. Klein, C., Sallai, J., Jones, T. J., Iacovella, C. R., McCabe, C. & Cummings, P. T. in *Foundations of Molecular Modeling and Simulation: Select Papers from FOMMS 2015* (eds Snurr, R. Q., Adjiman, C. S. & Kofke, D. A.) 79–92 (Springer Singapore, Singapore, 2016). ISBN: 978-981-10-1128-3.

2. *mBuild Github Repository*

3. Contributors, M. *mBuild Webpage* https://mbuild.mosdef.org. Accessed: 2022-03-13.

4. Thompson, M. W., Gilmer, J. B., Matsumoto, R. A., Quach, C. D., Shamaprasad, P., Yang, A. H., Iacovella, C. R., McCabe, C. & Cummings, P. T. Towards Molecular Simulations That Are Transparent, Reproducible, Usable by Others, and Extensible (TRUE). *Molecular Physics* **118,** e1742938. ISSN: 0026-8976, 1362-3028 (June 2020).

5. Klein, C., Summers, A. Z., Thompson, M. W., Gilmer, J. B., McCabe, C., Cummings, P. T., Sallai, J. & Iacovella, C. R. Formalizing Atom-Typing and the Dissemination of Force Fields with Foyer. *Computational Materials Science* **167,** 215–227. ISSN: 0927-0256. http://www.sciencedirect.com/science/article/pii/S0927025619303040 (Sept. 2019).

6. Daylight Chemical Information Systems, I. *Daylight Theory: SMARTS - A Language for Describing Molecular Patterns* http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html. Accessed: 2022-03-14.

7. Cordella, L., Foggia, P., Sansone, C. & Vento, M. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **26,** 1367–1372. ISSN: 0162-8828 (Oct. 2004).

8. Contributors, M. *MoSDeF Webpage* https://mosdef.org. Accessed: 2022-03-13.

9. *mBuild Github repository* https://github.com/mosdef-hub/mbuild (2018).

10. Berendsen, H. J. C., van der Spoel, D. & van Drunen, R. GROMACS: A message-passing parallel molecular dynamics implementation. *Comput. Phys. Commun.* **91,** 43–56. ISSN: 0010-4655 (Sept. 1995).

11. Abraham, M. J., Murtola, T., Schulz, R., Pá ll, S., Smith, J. C., Hess, B. & Lindahl, E. GROMACS: High Performance Molecular Simulations through Multi-Level Parallelism from Laptops to Supercomputers. *SoftwareX* **1-2,** 19–25. ISSN: 2352-7110 (Sept. 2015).

12. Contributors, M. *Foyer Github Repository* https://github.com/mosdef-hub/foyer.git. Accessed: 2022-03-13.

13. Contributors, P. *ParmEd: Parameter/topology editor and molecular simulator* https://github.com/ParmEd/ParmEd. Accessed: 2022-03-15.

14. Thompson, M., Wagner, J., Gilmer, J. B., Timalsina, U., Quach, C. D., Boothroyd, S. & Mitchell, J. A. *OpenFF Interchannge* version v0.1.4. Jan. 2022. https://github.com/openforcefield/openff-interchange.

15. Goldbaum, N. J., ZuHone, J. A., Turk, M. J., Kowalik, K. & Rosen, A. L. unyt: Handle, manipulate, and convert data with units in Python. *Journal of Open Source Software* **3,** 809. https://doi.org/10.21105/joss.00809 (Aug. 2018).

16. Weiner, S. J., Kollman, P. A., Case, D. A., Singh, U. C., Ghio, C., Alagona, G., Profeta, S. & Weiner, P. A New Force Field for Molecular Mechanical Simulation of Nucleic Acids and Proteins. *Journal of the American Chemical Society* **106,** 765–784. ISSN: 0002-7863, 1520-5126 (Feb. 1984).

17. MacKerell Jr., A. D., Banavali, N. & Foloppe, N. Development and Current Status of the CHARMM Force Field for Nucleic Acids. *Biopolymers* **56,** 257–265. ISSN: 0006-3525 (Jan. 2000).

18. Jorgensen, W. L., Maxwell, D. S. & Tirado-Rives, J. Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids. *J. Am. Chem. Soc.* **118,** 11225–11236. ISSN: 0002-7863 (Nov. 1996).

19. Siepmann, J. I., Karaborni, S. & Smit, B. Simulating the Critical Behaviour of Complex Fluids. *Nature* **365,** 330–332. ISSN: 0028-0836, 1476-4687 (Sept. 1993).

20. Potoff, J. J. & Siepmann, J. I. Vapor Liquid Equilibria of Mixtures Containing Alkanes, Carbon Dioxide, and Nitrogen. *AIChE Journal* **47,** 1676–1682. ISSN: 00011541, 15475905 (July 2001).

21. Sun, H. COMPASS: An Ab Initio Force-Field Optimized for Condensed-Phase ApplicationsOverview with Details on Alkane and Benzene Compounds. *The Journal of Physical Chemistry B* **102,** 7338–7364. ISSN: 1520-6106, 1520-5207 (Sept. 1998).

22. Oostenbrink, C., Villa, A., Mark, A. E. & Van Gunsteren, W. F. A Biomolecular Force Field Based on the Free Enthalpy of Hydration and Solvation: The GROMOS Force-Field Parameter Sets 53A5 and 53A6. *Journal of Computational Chemistry* **25,** 1656–1676. ISSN: 0192-8651, 1096-987X (Oct. 2004).

23. developers, G. *GROMACS OPLS-AA Forcefield Parameters* https://github.com/gromacs/gromacs/blob/e131e1d16c589fded5cad47bbd52b010d59c80a7/share/top/oplsaa.ff/atomtypes.atp. [Online; accessed 2022-03-16]. 2022.

24. Sandve, G. K., Nekrutenko, A., Taylor, J. & Hovig, E. Ten simple rules for reproducible computational research. *PLoS Comput. Biol.* **9** (ed Bourne, P. E.) e1003285. ISSN: 1553-734X (Oct. 2013).

25. Thompson, M. W., Matsumoto, R., Sacci, R. L., Sanders, N. C. & Cummings, P. T. Scalable Screening of Soft Matter: A Case Study of Mixtures of Ionic Liquids and Organic Solvents. *The Journal of Physical Chemistry B* **123,** 1340–1347. ISSN: 1520-6106 (Feb. 2019).

26. Bush, B. L. & Sheridan, R. P. PATTY: A Programmable Atom Type and Language for Automatic Classification of Atoms in Molecular Databases. *Journal of Chemical Information and Computer Sciences* **33,** 756–762. ISSN: 0095-2338 (Sept. 1993).

27. Schüttelkopf, A. W. & van Aalten, D. M. F. *PRODRG* : A Tool for High-Throughput Crystallography of Protein–Ligand Complexes. *Acta Crystallographica Section D Biological Crystallography* **60,** 1355–1363. ISSN: 0907-4449 (Aug. 2004).

28. Wang, J., Wang, W., Kollman, P. A. & Case, D. A. Automatic Atom Type and Bond Type Perception in Molecular Mechanical Calculations. *Journal of Molecular Graphics and Modelling* **25,** 247–260. ISSN: 1093-3263 (Oct. 2006).

29. Ribeiro, A. A. S. T., Horta, B. A. C. & de Alencastro, R. B. MKTOP: a program for automatic construction of molecular topologies. *J. Braz. Chem. Soc.* **19,** 1433–1435. ISSN: 0103-5053 (Aug. 2008).

30. Malde, A. K., Zuo, L., Breeze, M., Stroet, M., Poger, D., Nair, P. C., Oostenbrink, C. & Mark, A. E. An Automated Force Field Topology Builder (ATB) and Repository: Version 1.0. *J. Chem. Theory Comput.* **7,** 4026–4037. ISSN: 1549-9618 (Dec. 2011).

31. Vanommeslaeghe, K. & MacKerell Jr, A. D. Automation of the CHARMM General Force Field (CGenFF) I: bond perception and atom typing. *J. Chem. Inf. Model.* **52,** 3144–3154. ISSN: 1549-9596 (Dec. 2012).

32. Yesselman, J. D., Price, D. J., Knight, J. L. & Brooks, C. L. MATCH: An Atom-Typing Toolset for Molecular Mechanics Force Fields. *Journal of Computational Chemistry* **33,** 189–202. ISSN: 0192-8651 (Jan. 2012).

33. Weininger, D. SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. *Journal of Chemical Information and Modeling* **28,** 31–36. ISSN: 1549-9596 (Feb. 1988).

34. Mobley, D. L., Bannan, C. C., Rizzi, A., Bayly, C. I., Chodera, J. D., Lim, V. T., Lim, N. M., Beauchamp, K. A., Slochower, D. R., Shirts, M. R., Gilson, M. K. & Eastman, P. K. Escaping Atom Types in Force Fields Using Direct Chemical Perception. *Journal of Chemical Theory and Computation* **14,** 6076–6092. ISSN: 1549-9618, 1549-9626 (Nov. 2018).

35. In't Veld, P. J. *EMC: Enhanced Monte Carlo; A multi-purpose modular and easily extendable solution to molecular and mesoscale simulations* http://montecarlo.sourceforge.net/emc/Welcome.html. Accessed: 2022-03-14.

36. Eastman, P., Friedrichs, M. S., Chodera, J. D., Radmer, R. J., Bruns, C. M., Ku, J. P., Beauchamp, K. A., Lane, T. J., Wang, L.-P., Shukla, D., Tye, T., Houston, M., Stich, T., Klein, C., Shirts, M. R. & Pande, V. S. OpenMM 4: A Reusable, Extensible, Hardware Independent Library for High Performance Molecular Simulation. *Journal of Chemical Theory and Computation* **9,** 461–469. ISSN: 1549-9618, 1549-9626 (Jan. 2013).

37. Eastman, P., Swails, J., Chodera, J. D., McGibbon, R. T., Zhao, Y., Beauchamp, K. A., Wang, L.-P., Simmonett, A. C., Harrigan, M. P., Brooks, B. R. & Pande, V. S. *OpenMM 7: Rapid Development of High Performance Algorithms for Molecular Dynamics* 2016.

38. Developers, O. *OpenMM User Guide* http://docs.openmm.org/7.0.0/userguide/application.html. Accessed: 2022-03-14.

39. Sallai, J., Varga, G., Toth, S., Iacovella, C., Klein, C., McCabe, C., Ledeczi, A. & Cummings, P. T. Web- and Cloud-based Software Infrastructure for Materials Design. *Procedia Computer Science* **29,** 2034–2044. ISSN: 1877-0509 (2014).

40. Martin, M. G. & Siepmann, J. I. Transferable Potentials for Phase Equilibria. 1. United-Atom Description of *n* -Alkanes. *The Journal of Physical Chemistry B* **102,** 2569–2577. ISSN: 1520-6106, 1520-5207 (Apr. 1998).

41. Developers, T. *TraPPE Webpage* http://trappe.oit.umn.edu/. Accessed: 2022-03-10.

42. Watkins, E. K. & Jorgensen, W. L. Perfluoroalkanes: Conformational Analysis and Liquid-State Properties from Ab Initio and Monte Carlo Calculations. *The Journal of Physical Chemistry A* **105,** 4118–4125. ISSN: 1089-5639, 1520-5215 (Apr. 2001).

43. Harris, C. R., Millman, K. J., van der Walt, S. f. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. & Oliphant, T. E. Array Programming with NumPy. *Nature* **585,** 357–362 (Sept. 2020).

44. Virtanen, P. *et al.* SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17,** 261–272. ISSN: 1548-7091, 1548-7105 (Mar. 2020).

45. Scellato, S. NetworkX: Network Analysis with Python, 49.

46. Contributors, M. *Foyer Tutorials Webpage* https://github.com/mosdef-hub/foyer_tutorials.git. Accessed: 2022-03-13.

47. Contributors, M. *Foyer Webpage* https://foyer.mosdef.org. Accessed: 2022-03-13.

48. Wang, J., Wolf, R. M., Caldwell, J. W., Kollman, P. A. & Case, D. A. Development and Testing of a General Amber Force Field. *Journal of Computational Chemistry* **25,** 1157–1174. ISSN: 0192-8651, 1096-987X (July 2004).

49. Krekel, H., Oliveira, B., Pfannschmidt, R., Bruynooghe, F., Laugher, B. & Bruhin, F. *pytest* 2004. https://github.com/pytest-dev/pytest.

50. Contributors, M. *Foyer Forcefield Template* https://github.com/mosdef-hub/forcefield_template. Accessed: 2022-03-13.

51. Black, J., Silva, G., Klein, C., Iacovella, C., Morgado, P., Martins, L., Filipe, E. & McCabe, C. *Opls-Aa Compatible Parameters For Perfluoroethers* Zenodo. May 2017.

52. Black, J. E., Silva, G. M. C., Klein, C., Iacovella, C. R., Morgado, P., Martins, L. F. G., Filipe, E. J. M. & McCabe, C. Perfluoropolyethers: Development of an All-Atom Force Field for Molecular Simulations and Validation with New Experimental Vapor Pressures and Liquid Densities. *The Journal of Physical Chemistry B* **121,** 6588–6600. ISSN: 1520-6106 (July 2017).

53. *OPLSaa_perfluoroalkanes* Apr. 2018. https://github.com/chrisiacovella/OPLSaa%7B%5C_%7Dperfluoroalkanes (2018).

54. Contributors, M. *OPLS parameters for Alkylsilanes in Foyer format* https://github.com/summeraz/OPLSaa_alkylsilanes. Accessed: 2022-03-13.

55. Research, E. O. F. N. & OpenAIRE. *Zenodo* en. 2013. https://www.zenodo.org/.

56. Klein, C., Summers, A. Z., Thompson, M., Gilmer, J., Sallai, J., Yang, A., Volgyesi, P., Henry, M., Matsumoto, R., Moore, T., Gowers, R. & Tiet, F. *mosdef-hub/foyer: Foyer software and examples associated with our year 2019 publication in Computational Materials Science.* version paper_COMMAT_2019. May 2019.

57. Contributors, M. *GitHub tag of Foyer for Publication* https://github.com/mosdef-hub/foyer/tree/paper_COMMAT_2019. Accessed: 2022-03-13.

CHAPTER 5

Self-Assembly of Patchy Alkane-grafted Silica Nanoparticles

5.1 Introduction

In this work [1][2], the Molecular Simulation and Design Framework (MoSDeF) [2] and TRUE principals [3] are employed to examine the self-assembly of anisotropically coated "patchy" nanoparticles. Specifically, we use a coarse-grained model to examine silica nanoparticles coated with alkane chains, where the poles of the grafted nanoparticle are bare, resulting in strongly attractive patches. Through a systematic screening process leveraging MoSDeF [2, 4] and the Signac Framework [5–7] to sweep through parameter-space, the patchy nanoparticles are found to form dispersed, string-like, and aggregated phases, dependent on the combination of alkane chain length, coating chain density, and the fractional coated surface area. Correlation analysis is used to identify the ability of various particle descriptors to predict bulk phase behavior from more computationally efficient single grafted nanoparticle simulations and demonstrates that the solvent-accessible surface area of the nanoparticle core is a key predictor of bulk phase behavior. The results of this work enhance our knowledge of the phase space of patchy nanoparticles and provide a powerful approach for future screening of these or similar materials.

5.2 Background

The properties of systems containing nanoparticles are often highly correlated with the spatial arrangement of the underlying particles [8–11]. It has been demonstrated that the arrangement of nanoparticles is often dictated by the nanoparticle shape [12] and surface chemistry [13], as well as the properties of the solvent or polymer matrix [14]. Functionalization of nanoparticles with oligomer or polymer coatings [15–17] is one method that has been demonstrated to allow control over nanoparticle arrangement and phase behavior. For example, Akcora *et al.* demonstrated via a combination of experiment, molecular simulation, and theory that the phase behavior of polystyrene grafted silica nanoparticles could be tuned through the modification of the length and density of the polystyrene grafts [18]. Long grafts and high surface densities resulted in the formation of dispersed nanoparticle phases, gradually transitioning to strings, sheets, and spherical aggregates as the graft length and/or graft density was reduced. The use of anisotropic surface functionalization allows further control over the phase behavior and structural arrangement of the nanoparticles by introducing directional interactions between nanoparticles. Often referred to as patchy particles, these anisotropic coat-

[1]Portions of this chapter reproduced with permission from AIP Publishing from the following work:

[2]Craven, N. C., Gilmer, J. B., Spindel, C. J., Summers, A. Z., Iacovella, C. R. & McCabe, C. Examining the Self-Assembly of Patchy Alkane-Grafted Silica Nanoparticles Using Molecular Simulation. *The Journal of Chemical Physics* **154,** 034903. ISSN: 0021-9606, 1089-7690 (Jan. 2021).

ings can vastly increase the diversity and complexity of the structures formed by the nanoparticles [19–23]. For example, molecular simulations by Zhang *et al.* demonstrated that a variety of complex phases could be achieved by modifying the arrangement of attractive patches on the surface of simplified composite nanoparticles, including square-lattices, hexagonal-lattices, strings, rings, and polyhedral clusters [24]. Experiments by Fava *et al.* demonstrated that gold nanorods end-functionalized with polystyrene could be induced to form various phases depending on the solvent quality, including end-to-end, single-wide strings of nanorods when the solvent quality was good for the nanorods and spherical aggregates when the solvent quality was poor for the nanorods [25]. DNA grafted nanoparticles have also been demonstrated as a means of creating highly specific, directional interactions between nanoparticles and colloids [21, 26, 27]. In other work, molecular simulations and theoretical calculations by Bianchi *et al.* demonstrated that significant changes to the liquid phase envelope could be achieved by the modification of the number of short range, directional, attractive patches on the spherical particles, where a reduction in the number of patches could be seen to increase the stability of the "liquid" regime to very low concentrations [28].

Controlling the anisotropic effect for functionalized nanoparticles can be a complicated task as there is a nearly infinite design space of building blocks, where subtle features of the system (e.g., coating density, and length) may play a significant role in phase behavior. Molecular simulation enables the design space to be systematically probed by providing precise control over the individual building blocks and has been extensively used to study nanoparticle systems [20, 29–32]. However, to date, most studies of nanoparticles with anisotropic interactions have used simplified, generic phenomenological models that represent the patchiness of the particles as differently interacting beads on a particle surface [30, 33, 34]. While such models have provided tremendous insight and relate well to patterning approaches used by experimentalists on larger length scales [35], they do not necessarily relate to a specific chemistry and may not capture subtle details associated with polymer grafts (e.g., entropic interactions) [36, 37] which may significantly impact the final structure/phase formed [38]. Grafted nanoparticle systems have also been studied via coarse-grained (CG) models, although limited work has considered anisotropic systems [8, 18, 39–43]. Furthermore, in most cases, these studies have utilized generic models of the chains and nanoparticle cores, rather than chemistry-specific [44–46] and, therefore, may not be directly relatable to specific chemistries.

Here, we examine the properties of patchy, grafted nanoparticles, using a chemistry-specific CG modeling approach. Nanoparticle interactions are governed by a CG model for silica nanoparticles developed in earlier work that was shown to accurately reproduce the energetic interactions of the corresponding atomistic models [44]. Surface coatings are modeled as alkane chains using the CG model of Nielsen *et al.* with implicit solvent interactions, where chains are considered to be in a good solvent and, thus, do not attract [47]. Uncoated regions of the nanoparticles result in the directional, attraction at the two poles. Using this detailed CG model, screening of the nanoparticle coatings is performed using the Molecular Simulation Design Framework (MoSDef) [2, 3, 48, 49], to systematically examine the phase behavior of the nanoparticle systems, varying the density of the coating, fraction of surface coverage (FSA), and chain length of the alkane graft. The bulk phase behavior is used to plot phase diagrams to identify key relations between the screening parameters. Simulations of singular grafted nanoparticles are also performed and it is demonstrated that the solvent-accessible surface area (SASA) of the nanoparticle cores allows a priori prediction of the phase behavior of bulk systems.

5.3 Methods

5.3.1 Nanoparticle Model

Patchy grafted nanoparticles are modeled as silica nanospheres, featuring an anisotropic (non-uniform) coating of alkane grafts. The CG model used to describe the nanoparticle cores was parameterized in earlier work to model silica [43]. The nanoparticles are constructed as a composite particle of beads arranged on a spherical surface using the golden spiral algorithm (see Figure 5.1) following the work of Summers *et al.* [44]; in all cases, the nanoparticles are modeled as 5.0 nm in diameter, with 153 CG beads.

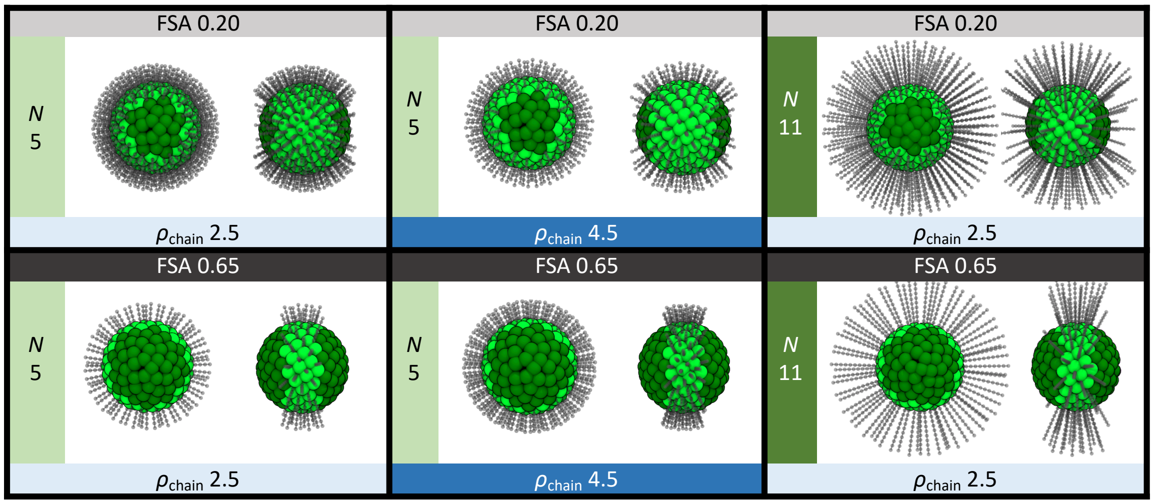


Figure 5.1: Visualizations of grafted nanoparticles based on tunable model parameters of fractional surface area (FSA), chain length in CG beads (N), and chain density (ρ_{chain}). All grafted nanoparticles have symmetry about the equator and are shown in two different orientations. Light green beads represent CG silica beads attached to a CG alkane graft (gray). Dark green represents CG silica beads around the poles without grafts.

A Mie pair potential Equation 5.1 was used to define the interaction between beads in the nanoparticles, where ε is the well depth and n and m are the potential exponents.

$$U(r) = (\frac{n}{n-m})(\frac{n}{m})^{(\frac{m}{n-m})} * \varepsilon[(\frac{\sigma}{r})^n - (\frac{\sigma}{r})^m] \tag{5.1}$$

This model of silica nanoparticles has been shown to accurately capture the energetic interactions between atomistic nanoparticles. We note that, while short range square well potentials are also widely used to model the interactions between nanoparticle cores in isotropically grafted nanoparticle simulations, Haley *et al.* demonstrated that the liquid phase envelope of polymer grafted nanoparticles was reduced as the range of the Lennard-Jones interaction between nanoparticle cores was reduced and began to more closely resemble a square well potential [50]. Similarly, Kalyuzhnyi *et al.* demonstrated that the liquid phase envelope of a four-site patchy particle model strongly depends on the core-core interactions. As such, accurately modeling these interactions is key for studying the assembly of nanoparticle systems [51].

For alkane grafts, a 3:1 CG mapping from Nielsen *et al.* was used where single beads represent groupings of three methyl units [47]. Parameters for the interactions between the silica cores and the cross-interactions between silica cores and alkane grafts are derived to match interactions from all-atom models using the procedure outlined by Summers *et al.* [44] A schematic of a model grafted nanoparticle is included in Figure C.1 of the supplementary material, and all model parameters are reported in Table 5.1. Bonds and angles for alkyl grafts are treated as harmonic bonds, with spring constants 1232.1 $\frac{kcal}{molnm^2}$ and 2.8346 $\frac{kcal}{mol}$, respectively.

Table 5.1: Mie interaction terms used to model CG nanoparticles and chains. Beads in the core are labeled "silica", beads along the length of the chain are labeled "chain", and beads that cap these chains are labeled "terminus."

Interactions	$\varepsilon[\frac{kcal}{mol}]$	$\sigma[nm]$	n	m
Silica-silica	0.9286	0.60	20.0087	4.7578
Silica-chain	0.6360	0.5291	29.6574	5.5835
Silica-terminus	0.6788	0.5331	28.0821	5.6179
Chain-chain	0.389092	0.4582	9.0	6.0
Terminus-terminus	0.434996	0.4662	9.0	6.0
Chain-terminus	0.411404	0.4662	9.0	6.0

The equilibrium length is set to 0.364 nm, and positions for angles are set for three chain groups to 3.019 42 radians and 3.054 33 radians for two chain groups and one terminus. Chains were constrained to the surface by a rigid bead; note that chains are not free to move about the surface of the nanoparticle and the nanoparticle core beads, and chain attachment beads, are integrated through time as a rigid body. The effects of solvent were treated implicitly through the use of a Langevin thermostat and by truncating chain-chain interactions at the minimum of the attractive well such that they are purely repulsive (i.e., the Weeks-Chandler-Andersen potential [52]), as has been done in prior studies [44, 53].

In this work, patchiness is introduced through a polar coating pattern where the grafted chains cover the surface of the silica core but are removed starting at the poles to expose the spherical core. This results in a band of grafts encircling the equator of the particle that varies in thickness. Three properties of the coating were investigated:

- Fractional surface area (FSA) of the exposed nanoparticle core, defined as the fraction of the nanoparticle core surface area that is not covered by alkane chains, i.e., the fraction of the nanoparticle that is "patchy."

- Chain density ρ_{chain} defined as chains per nanometer squared.
- Chain length N defined by the number of 3:1 CG alkane beads on each grafted chain.

The effect of the model parameters on the grafted nanoparticles can be seen in the visualizations in Figure 5.1 Each grafted nanoparticle model was constructed using the `mBuild` library and parameterized with the `Foyer` library within the MoSDeF suite of tools.

5.3.2 Simulation Details

Simulations were performed using the 1.3.3 version of the `HOOMD-Blue` molecular simulation engine [54, 55] and the Signac framework [5, 6, 56] for workflow management. For bulk simulations, nanoparticle self-assembly was examined by placing 25 nanoparticles in a $40*40*40nm^3$ box for 100 ns, for a given set of design variables (see Table C.1). In total 44 simulations (state points) were considered. The simulation workflow was as follows: First, a short energy minimization was performed to resolve any issues with overlapping particles from system construction, with a time step of 0.01 fs. This was followed by a 0.1 ns period to further relax the system, with a step size of 1 fs. Next, the system was annealed in four stages from 1000 K to 400 K, running for 10 ns with 4 fs time steps at each stage, to remove any dependence of the final self-assembled structure on the system's initial configuration. Finally, the system was run at 300 K for 50 ns, allowing a steady state configuration to be achieved. Data from the last 10 ns of the simulation were used for analysis. This analysis includes calculation of the radial distribution function (RDF), using the `Freud` Python package [57], analysis of local coordination using `MDTraj` [58], and visualization using the visual molecular dynamics (`VMD`) package [59].

Simulations of single grafted nanoparticles were also performed that screen over the range of chain length from 5 to 11 CG beads, chain density from 2.5 $\frac{chains}{nm^2}$ to 5.0 $\frac{chains}{nm^2}$, and FSA from 0.2 to 0.65, for a total of 539 simulations; 40 of these simulations correspond to the same design variables of the 44 bulk simulations. For each system, a brief energy minimization was performed followed by an NVT simulation run at 300 K for 10 ns with a 2 fs time step during which the nanoparticle core was held stationary, allowing only the alkane grafts to move. SASA was calculated with a 0.25 nm probe particle using `MDTraj` [58] and normalized by the idealized surface area of the 5.0 nm diameter nanoparticle (i.e., $2\pi D$). The radius of gyration (R_g) of the entire grafted nanoparticle ($R_{g_{NP}}$), R_g of each of the individual chains ($R_{g,chain}$), and asphericity of the entire grafted nanoparticle (i.e., including the core and chains) were calculated from the single particle trajectories using the `MDAnalysis` [60, 61] package. To be consistent with other prior work [19], we do not directly consider R_g but rather consider the ratio of nanoparticle radius (2.5 nm) to the two R_g measures, which are labeled $R_{g,NP}^{-1}$ and $R_{g,chain}^{-1}$. For analysis and comparison of the metrics computed in the bulk and single grafted nanoparticle simulations, Spearman's rank coefficients were determined using the `SciPy` computing package [62].

5.4 Results

5.4.1 Bulk Simulations

Simulations of bulk grafted nanoparticles were performed to examine the phase behavior as a function of chain length, surface grafted chain density, and FSA. The 44 systems simulated and the phases formed are listed in Table C.1 of the supplementary material. Qualitatively, three distinct phases were observed, as shown in Figure 5.2 "dispersed", "stringy", and "aggregated".

The phases were categorized both visually and by calculating the average number of nearest neighbors (i.e., the coordination number); nearest neighbors are defined as nanoparticles within a distance of 7.5 nm between the centers-of-mass of the nanoparticle core. Dispersed phases [Figure 5.2(a)] are defined as those whose average nearest neighbor coordination number is less than 1. String-like aggregates [Figure 5.2(b)] are defined as those with nearest neighbor coordination numbers ranging from 1 to 2 and a standard deviation of these values of less than one. The use of standard deviation ensures that systems that consist of multiple phases (e.g., higher-coordinated aggregated nanoparticles and lower-coordinated dispersed phases) are not misidentified as strings. The third phase observed is characterized by nanoparticles that aggregate but do not demonstrate significant string-like behavior or any long-range crystalline ordering [Figure 5.2(c)]. Such phases are characterized by high coordination numbers; specifically, a coordination number greater than 2 is used to define an aggregated phase or those where the standard deviation is above 1. We note that, visually, the aggregated phases tend to be more planar in nature, rather than forming a spherical aggregate, as shown in Figure 5.2(c). As evidenced by the coordination numbers (reported in the caption of Figure 5.2 and Table C.1 of the supplementary material), the dispersed phases show little-to-no aggregation of nanoparticles, whereas the aggregated phases show significant grouping, both at the uncoated poles and in the coated regions. This can also be seen in the radial distribution functions (RDFs) presented in the bottom row of Figure 5.2, where dispersed phases do not show a strong first neighbor peak that would indicate aggregation, whereas the aggregated phases show two strong short-ranged peaks; the first peak corresponds to nanoparticles directly in surface contact at the patches, and the second peak, shifted by $\sim$2 Å, corresponds to interactions between nanoparticles, buffered by the polymer grafts. A string-like phase appears as an intermediate between the dispersed and aggregated phases in which the nanoparticles aggregate, but only at the poles that are not grafted; this can be seen in Figure 5.2(b-4). In general, the RDF of the string-like phase shows only a single peak at the first neighbor separation, corresponding to direct nanoparticle-nanoparticle contact, although strings that have multiple branches [see Figure 5.2(b-1)] show a small second peak shifted by $\sim$2Å.

To more clearly identify the correlations between chain density, chain length, FSA, and bulk phase, phase diagrams of the bulk systems studied are plotted in Figure 5.3(a)-3(c). To simplify the ability to represent the data, one of the three variables (chain length, chain density, FSA) is held fixed, while the others are varied. From Figure 5.3(a), it is clear that, for the fixed chain length of 6, phase behavior is most strongly linked to FSA and less dependent on chain density. Figure 5.3(b) shows that the likelihood of forming dispersed phases significantly increases as graft length increases and FSA decreases. For shorter chain lengths, as the FSA of the exposed nanoparticle core increases, nanoparticles begin to associate, first forming strings for moderate values, then transitioning to aggregate phases as the "patches" become larger and less localized. The phase behavior in Figure 5.3(c) shows that nanoparticles with longer chain lengths and denser coatings are more likely to produce dispersed phases, as expected since such systems can more effectively shield the nanoparticle cores, while systems with shorter chain lengths and less dense coatings are more likely to form aggregate phases. The aforementioned work of Akcora *et al.* examined the phase behavior of isotropically grafted silica nanoparticles as a function of graft length and graft density, finding similar trends in terms of overall phase behavior as observed here [18]. We note that the string-like phases reported by Akcora *et al.* visually appear to be 1–2 nanoparticles in width, rather than the single nanoparticle-wide chains observed here, likely related to the more explicit directional interactions encoded in the patchy model. Additionally, we note that the aggregated phase observed in this work tends to a planar shape, in agreement with sheet-like structures observed by Akcora *et al.*; it was proposed that the underlying mechanism of sheet formation was kinetically controlled directional phase separation, supported by the analysis of the growth of the structures. It is likely that the same mechanism underlies the systems here, although our system sizes are too small to confidently quantify growth scaling. The simulation results reported herein are also in qualitative agreement with experiments of patchy spherical micelles formed from triblock copolymers that also formed string-like aggregates; such experiments considered the role of temperature and pH on the assembly, rather than grafting characteristics as reported herein, and thus, we cannot make a direct comparison of phase behavior [23]. We additionally note the clear agreement in terms of the formation of string-like phases in experiments of larger, patchy colloids that asssemble via charged particles at the poles [35].

5.4.2 Single Nanoparticle Simulations

The literature contains several proposed metrics for relating bulk phase behavior to the properties of single grafted nanoparticles; the ability to predict phase landscapes from single nanoparticle simulations would allow for substantially reduced computational cost compared to bulk simulations. Specifically, Lafitte *et al.*

related the asphericity of a single grafted nanoparticle to the bulk phase [63], where it was found that grafted nanoparticles with low asphericity yielded dispersed phases and nanoparticles with increasing asphericity resulted in aggregated phases, with some stringy phases intermediate. This aggregation was proposed to result from reduced shielding of the attractive cores due to an uneven spread of grafted chains on the nanoparticle, which could be measured through asphericity [64]. Bozorgui *et al.* presented a geometric argument relating the radius of gyration (R_g) of the grafted chains to the radius of the particle, with regard to the ability of nanoparticle cores to achieve close contact. In subsequent related work, mean field theory calculations by Pryamtisyn *et al.* showed that the particle radius, normalized by R_g , of the grafted chains was a key parameter dictating the anisotropic assembly of grafted nanoparticles [19] (here, referred to by the variable ($R^{-1}_{g,chain}$). In addition to these metrics, we consider R_g of the nanoparticle and chains, treated as a single entity, which implicitly captures some of the information of both asphericity and chain R_g; as discussed in the simulation details, we normalize the core nanoparticle radius by the R_g value of the nanoparticles and chains (here, referred to $R^{-1}_{g,NP}$). To quantify the relative amount of the nanoparticle core that can be accessed, SASA is calculated and normalized by the nanoparticle surface area to give the fractional SASA (f_{SASA}). We note that we would expect for short chains with dense surface coatings, f_{SASA} should be closely related to FSA, and longer chains may act to shield the uncoated regions, reducing or completely eliminating access to the patches.

Spearman's rank correlation coefficients [65] were calculated to ascertain if correlations exist between (1) behavior observed for the bulk simulations (i.e., phase and the average and standard deviation of the coordination number), (2) system parameters (i.e., chain length, chain density, and FSA), and (3) properties measured from single grafted nanoparticle simulations (i.e., asphericity, $R^{-1}_{g,chain}$, $R{-}1_{g,NP}$, and f_{SASA}). Figure 5.4 shows the correlations obtained, with larger and redder squares indicating correlations tending toward 1. Table C.2 of the supplementary material reports the numerical values, while Figure C.3–Figure C.2 show the correlation scatter plots used to determine the correlation value.

First, we can observe that the phase and coordination number (both average and standard deviation) are strongly correlated, as anticipated, given that this coordination number information was used to identify the bulk phase. We note that, individually, none of the system parameters are strongly correlated with phase information; this is as expected given that Figure 5.3 clearly shows the multi-parameter dependence. Of the four metrics calculated for the single nanoparticle simulations, f_{SASA} shows the largest correlation with phase. $R^{-1}_{g,NP}$ and asphericity also appear to show reasonable correlation. $R^{-1}_{g,chain}$ does not appear strongly correlated with phase. We note that high correlation values do not necessarily indicate that the metric can be used in a predictive capacity. Figure 5.5 plots normalized histograms of various metrics as a function of the resulting bulk phase. Figure 5.5(a) considers f_{SASA}, which again, demonstrated the strongest correlation with phase.

While there is some overlap between histograms, each histogram is sufficiently unique, suggesting that f_{SASA} can be used in a predictive capacity. For comparison, histograms of FSA are plotted in Figure 5.5(b), where a substantial overlap is seen between each histogram, underscoring the need to use SASA to measure the actual patch area that is accessible. Figure 5.5(c) considers asphericity; even though significant correlations were observed with phase, it is clear that this metric is not sufficiently sensitive to uniquely identify each phase; only the highest asphericity values have a single unique phase associated with them. Figure 5.5(d) plots nanoparticle $R_{g,NP}^{-1}$; while there appears to be clear correlations with phase, the overlap between histograms is larger than in the case of f_{SASA}, where we note that there are no values of $R_{g,NP}^{-1}$ that only result in a string-like phase, limiting the predictive power. However, $R_{g,NP}^{-1}$ and asphericity may still be useful in helping to identify trends in the bulk phase behavior. $R_{g,chain}^{-1}$ is plotted in Figure 5.5(e); substantial overlap between histograms is observed, and thus, as the correlation value suggests, this cannot be used in a predictive capacity for the systems considered here.

In order to test the ability of f_{SASA} to be used in a predictive capacity for predicting phase behavior, heatmaps were constructed using a mesh algorithm from the f_{SASA} values, as shown in Figure 5.6; data points for the bulk phase systems, as reported in Figure 5.3, are overlaid for comparison. Equivalent raw (i.e., unmeshed) heatmaps are included in Figure C.7 of the supplementary material.

Here, a color gradient is defined to capture transition values, where white is used to capture the value of f_{SASA} where the histograms cross in Figure 5.5(a) (i.e., f_{SASA} values of $\sim$0.11 for dispersed to stringy and $\sim$0.20 for stringy to aggregated), with the width of the gradients around these points correlating with the overlap of the histograms, suggestive of the uncertainty in identifying the phase. As such, regions of white can be considered to be estimates of the phase boundaries predicted by f_{SASA}. Close agreement is seen between the phase behaviors predicted from f_{SASA} calculated for the single grafted nanoparticle simulations in comparison with the phases identified from the bulk simulations. We note that the phase prediction in Figure 5.6(c), where FSA is fixed at 0.55, shows a much more gradual transition from dispersed to stringy phases, as evidenced by the broad gradient. For a value of FSA = 0.55, most of the chains are distributed along the equator of the nanoparticle, with a large portion of the nanoparticle core exposed (e.g., see FSA = 0.65 in Figure 5.1). In this regime (i.e., long chains and high FSA), chains will have a high degree of conformational freedom, which may result in larger variability in the measurement. Furthermore, the actual conformation may depend more strongly on the local environment of the grafted nanoparticle than for shorter chains with lower FSA, where chain conformations will likely change if they are near another nanoparticle (e.g., as discussed in the work of Bozorgui *et al.* [64] and Meng *et al.* [39]). The effects of interactions with neighbors are not captured by the single grafted nanoparticle simulations from which f_{SASA} is calculated. Nonetheless, f_{SASA} appears to be a strong predictor of phase. The supplementary material includes heatmaps generated from $R^{-1}_{g,chain}$, $R{-}1_{g,NP}$, and asphericity in Figure C.7–Figure C.12, including both raw and meshed histograms. As might be expected from Figure 5.5, $R^{-1}_{g,chain}$ is not able to predict the phase behavior. $R{-}1_{g,NP}$ overall provides a reasonable estimate of the phase behavior, although, due to the overlap of the histograms, it does not provide as clear of a definition of the transitions as f_{SASA} (i.e., the gradient regions are larger). Asphericity provides a reasonable estimate of phase behavior when considering phase as a function of FSA but does not seem to be predictive in the high FSA limit.

The results presented clearly demonstrate that for polar associating nanoparticle systems, f_{SASA} values obtained from simulations of single nanoparticles offer insight into the phase of the corresponding bulk system and provide sufficient accuracy to be used in a predictive manner. As such, this may significantly reduce the need to run as many computationally intensive simulations of the corresponding bulk systems to establish a clear picture of the phase behavior. This may also allow for prescreening of the parameter space to identify regions of interest, again further reducing the computational cost of examining possible systems. As currently constructed, f_{SASA} will not be able to help predict the structural arrangement for a particular coating pattern (e.g., that a hexagonal sheet would form by an equatorial pattern, as proposed by Zhang and Glotzer [24]), but it could help determine which combination of grafting variables are most likely to form the phase of interest (i.e., the region intermediate between low f_{SASA} systems that will disperse and high f_{SASA} systems that aggregate irrespective of the directional interactions).

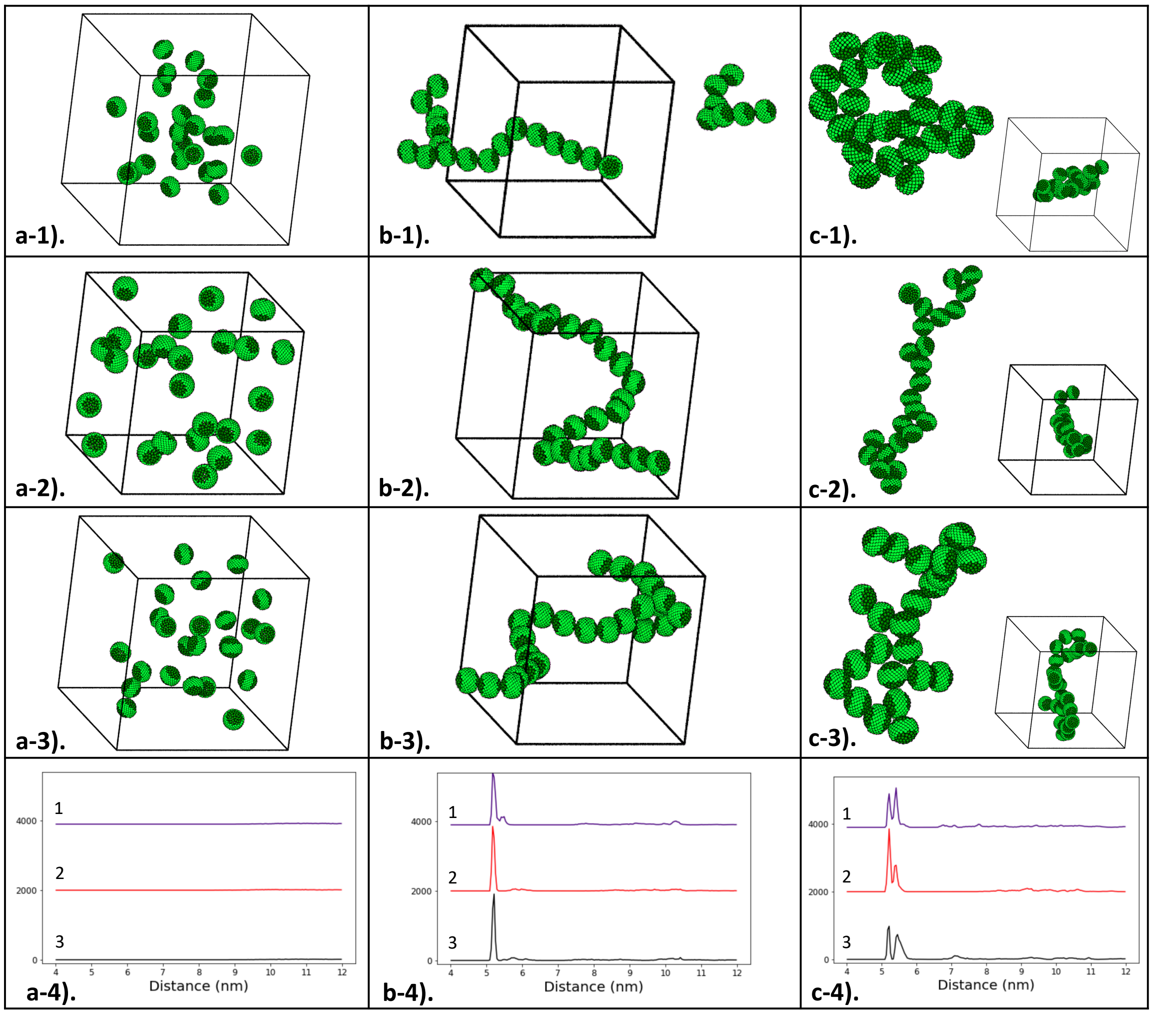


Figure 5.2: Examples of equilibrium phases with coordination numbers of [(a1)–(a3)] 0.00, 0.00, 0.00 in the "dispersed" phase, [(b1)–(b3)] 1.80, 1.71, 1.76 in the "stringy" phase, and [(c1)–(c3)] 2.33, 2.41, 2.23 in the "aggregated" phase, respectively. (4) represents the corresponding RDFs to the visualized systems, offset by a value of 2000 in the y direction for clarity. Note that the scale of the RDFs is the same for all systems. Grafts are omitted for clarity in the visualizations. Figure C.2 shows the same figure with grafted chains. Aggregated phases are visualized from two separate orientations to more clearly demonstrate the structure.

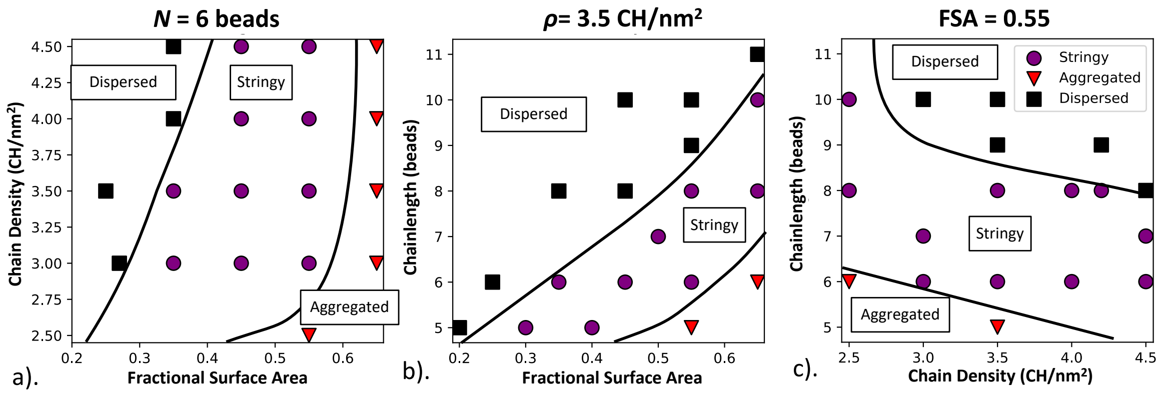


Figure 5.3: Bulk phase diagrams for (a). ρ_{chain} as a function of FSA with N held constant at six beads, (b) FSA vs N with ρ_{chain} held constant at 3.5 $\frac{chains}{nm^2}$, (c) ρ_{chain} vs N with FSA held constant at 0.55. Black solid lines are to guide the eye, delineating the stringy, aggregated, and dispersed phases

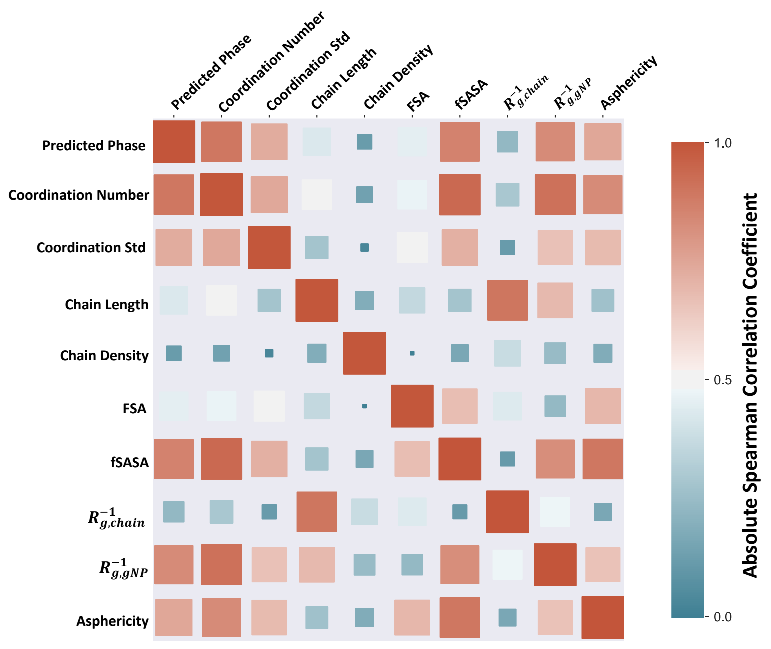
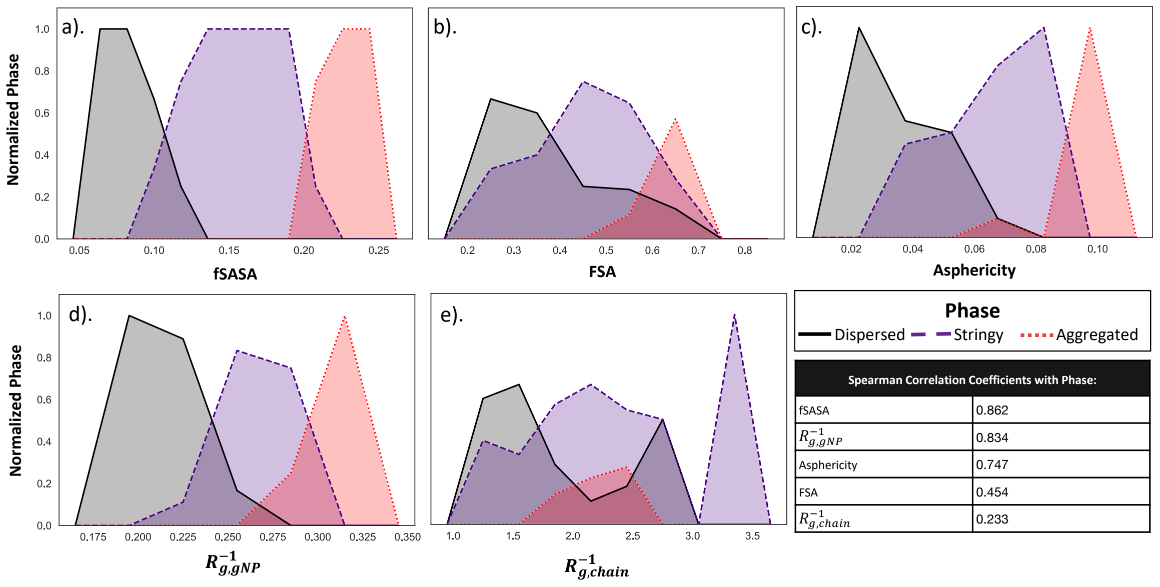


Figure 5.4: Correlation matrix of key descriptors for the nanoparticle systems studied. Chain length, chain density, and FSA are explicitly defined parameters. Coordination number, coordination number standard deviation, and predicted phase are resultant measures from the bulk simulations. f_{SASA}, $R^{-1}_{g,chain}$, $R^{-1}_{g,gNP}$,and asphericity are calculated from single nanoparticle simulations. $R^{-1}_{g,gNP}$ is defined as the nanoparticle radius divided by the radius of gyration of the grafted nanoparticle. $R^{-1}_{g,chain}$ is defined likewise, but with the radius of gyration of the grafts. Coefficient values close to unity correspond to stronger linear correlation, where strongly correlated systems are plotted as red, following the scale; note that the size of the markers also scales linearly with the correlation value.



Spearman Correlation Coefficients with Phase:	
fSASA	0.862
$R_{g,gNP}^{-1}$	0.834
Asphericity	0.747
FSA	0.454
$R_{g,chain}^{-1}$	0.233

Figure 5.5: Histograms of the fraction of simulations that result in a particular bulk phase binned by the metrics calculated from the single grafted nanoparticle simulations for (a) f_{SASA}, (b) FSA, (c) asphericity (1 corresponds to a flat plane), (d) $R_{g,NP}^{-1}$, and (e) $R_{g,chain}^{-1}$. Also included are Spearman's correlation coefficients of each parameter compare to the bulk phases reported. Higher correlation is an indication of better predictive nature using that metric.

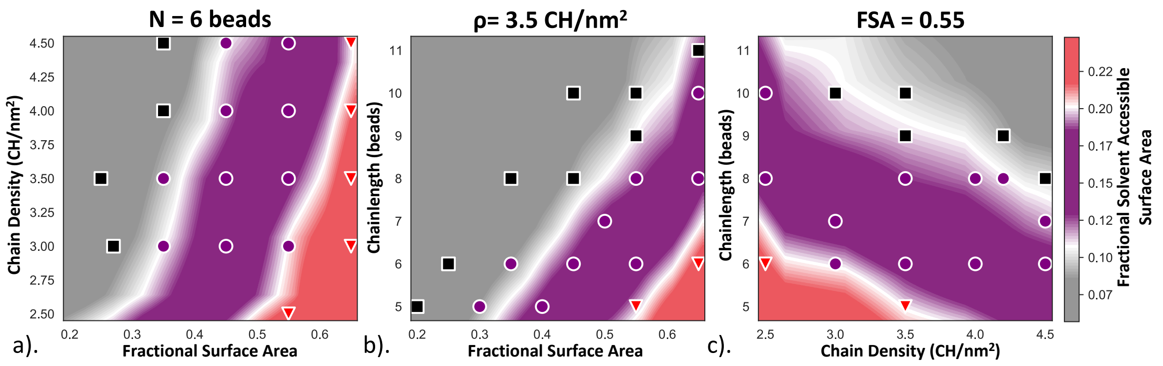


Figure 5.6: Phase diagrams from Figure 5.3 overlaid on fSASA heatmaps. White regions are mapped to the numerical value where histograms cross in Figure 5.5(a). Bulk simulation phase is represented as black squares (dispersed phase), purple circles, (stringy phase), and red triangles (aggregated phase), following the same scheme in Figure 5.3. Bilinear interpolation was used for meshing the f_{SASA} heatmap values. (a) Chain density as a function of FSA with N held constant at six beads, (b) FSA vs N with chain density held constant at 3.5 $\frac{chains}{nm^2}$, (c) chain density vs N with FSA held constant at 0.55.

5.5 Conclusion

Molecular dynamics simulations have been performed to study the self-assembly of grafted nanoparticles to gain insight into trends in phase behavior. A screening workflow was developed that leverages the MoSDeF [2] toolkit for system building and parameterization, the management of this dataspace was handled by the Signac [6, 7, 56] framework, and analysis was performed using `MDTraj` [58], `MDAnalysis` [60, 61], and `Freud` [57]. Patchy alkane-grafted nanoparticles with chains excluded from the poles form three main phases: dispersed, stringy, and aggregated. Nanoparticle phases trend from dispersed, to stringy, to aggregated phases through the increased fractional surface area, decreased chain density, and reduced chain length. The f_{SASA} of single-grafted nanoparticles was found to provide a predictive capability in terms of the equilibrium phase of the corresponding bulk systems of nanoparticles. Furthermore, the relationships explored in this work can likely be extended to other systems. For example, Asai *et al.* have shown that isotropic polymer grafted nanoparticles can behave like Janus particles due to surface fluctuations [66]. Although the explicit pattern patchy particles display may be challenging to replicate, the general principles that determine the phase separation should be translatable to isotropic nanoparticles. While this work considered patches created by exposed nanoparticle cores, this analysis could be easily adapted to capture systems where directional attraction arises due to interactions between polymers. For example, we again note the close agreement between our work and that of patchy micelles formed from triblock copolymer building blocks [23]; we would anticipate that a similar f_{SASA} analysis of model systems could be used to predict the phase behavior.

References

1. Craven, N. C., Gilmer, J. B., Spindel, C. J., Summers, A. Z., Iacovella, C. R. & McCabe, C. Examining the Self-Assembly of Patchy Alkane-Grafted Silica Nanoparticles Using Molecular Simulation. *The Journal of Chemical Physics* **154,** 034903. ISSN: 0021-9606, 1089-7690 (Jan. 2021).

2. Contributors, M. *MoSDeF Webpage* https://mosdef.org. Accessed: 2022-03-13.

3. Thompson, M. W., Gilmer, J. B., Matsumoto, R. A., Quach, C. D., Shamaprasad, P., Yang, A. H., Iacovella, C. R., McCabe, C. & Cummings, P. T. Towards Molecular Simulations That Are Transparent, Reproducible, Usable by Others, and Extensible (TRUE). *Molecular Physics* **118,** e1742938. ISSN: 0026-8976, 1362-3028 (June 2020).

4. Klein, C., Sallai, J., Jones, T. J., Iacovella, C. R., McCabe, C. & Cummings, P. T. in *Foundations of Molecular Modeling and Simulation: Select Papers from FOMMS 2015* (eds Snurr, R. Q., Adjiman, C. S. & Kofke, D. A.) 79–92 (Springer Singapore, Singapore, 2016). ISBN: 978-981-10-1128-3.

5. Contributors, S. *Signac Framework Webpage* https://signac.io. Accessed: 2022-03-13.

6. Dice, B. D., Butler, B. L., Ramasubramani, V., Travitz, A., Henry, M. M., Ojha, H., Wang, K. L., Adorf, C. S., Jankowski, E. & Glotzer, S. C. *Signac: Data Management and Workflows for Computational Researchers* in *Proceedings of the 20th Python in Science Conference* (2021), 23–32.

7. Adorf, C. S., Dodd, P. M., Ramasubramani, V. & Glotzer, S. C. Simple data and workflow management with the signac framework. *Computational Materials Science* **146,** 220–229. ISSN: 0927-0256. https://authors.elsevier.com/a/1WbPL3In-uhIuD%20http://linkinghub.elsevier.com/retrieve/pii/S0927025618300429%20https://www.mendeley.com/research-papers/simple-data-workflow-management-signac-framework/ (Feb. 2018).

8. Kumar, S. K. & Krishnamoorti, R. Nanocomposites: Structure, Phase Behavior, and Properties. *Annual Review of Chemical and Biomolecular Engineering* **1,** 37–58. ISSN: 1947-5438, 1947-5446 (June 2010).

9. Mann, S. Self-Assembly and Transformation of Hybrid Nano-Objects and Nanostructures under Equilibrium and Non-Equilibrium Conditions. *Nature Materials* **8,** 781–792. ISSN: 1476-1122, 1476-4660 (Oct. 2009).

10. Jayaraman, A. Polymer Grafted Nanoparticles: Effect of Chemical and Physical Heterogeneity in Polymer Grafts on Particle Assembly and Dispersion. *Journal of Polymer Science Part B: Polymer Physics* **51,** 524–534. ISSN: 0887-6266 (Apr. 2013).

11. Kadulkar, S., Banerjee, D., Khabaz, F., Bonnecaze, R. T., Truskett, T. M. & Ganesan, V. Influence of Morphology of Colloidal Nanoparticle Gels on Ion Transport and Rheology. *The Journal of Chemical Physics* **150,** 214903. ISSN: 0021-9606, 1089-7690 (June 2019).

12. Chen, S., Ingram, R. S., Hostetler, M. J., Pietron, J. J., Murray, R. W., Schaaff, T. G., Khoury, J. T., Alvarez, M. M. & Whetten, R. L. Gold Nanoelectrodes of Varied Size: Transition to Molecule-Like Charging. *Science* **280,** 2098–2101. ISSN: 0036-8075, 1095-9203 (June 1998).

13. Cho, K.-S., Talapin, D. V., Gaschler, W. & Murray, C. B. Designing PbSe Nanowires and Nanorings through Oriented Attachment of Nanoparticles. *Journal of the American Chemical Society* **127,** 7140–7147. ISSN: 0002-7863, 1520-5126 (May 2005).

14. Verma, M., Nehra, K. & Kumar, P. S. Plasmonic Oligomers: The Role of Polymer-Solvent Interactions. *Advanced Science Letters* **22,** 3860–3862. ISSN: 1936-6612 (Nov. 2016).

15. Grzelczak, M., Vermant, J., Furst, E. M. & Liz-Marz án, L. M. Directed Self-Assembly of Nanoparticles. *ACS Nano* **4,** 3591–3605. ISSN: 1936-0851, 1936-086X (July 2010).

16. Chevigny, C., Dalmas, F., Di Cola, E., Gigmes, D., Bertin, D., Boué, F. & Jestin, J. Polymer-Grafted-Nanoparticles Nanocomposites: Dispersion, Grafted Chain Conformation, and Rheological Behavior. *Macromolecules* **44,** 122–133. ISSN: 0024-9297, 1520-5835 (Jan. 2011).

17. Ekeroth, S., Ikeda, S., Boyd, R., Münger, P., Shimizu, T. & Helmersson, U. Impact of Nanoparticle Magnetization on the 3D Formation of Dual-Phase Ni/NiO Nanoparticle-Based Nanotrusses. *Journal of Nanoparticle Research* **21,** 228. ISSN: 1388-0764, 1572-896X (Nov. 2019).

18. Akcora, P., Liu, H., Kumar, S. K., Moll, J., Li, Y., Benicewicz, B. C., Schadler, L. S., Acehan, D., Panagiotopoulos, A. Z., Pryamitsyn, V., Ganesan, V., Ilavsky, J., Thiyagarajan, P., Colby, R. H. & Douglas, J. F. Anisotropic Self-Assembly of Spherical Polymer-Grafted Nanoparticles. *Nature Materials* **8,** 354–359. ISSN: 1476-1122, 1476-4660 (Apr. 2009).

19. Pryamtisyn, V., Ganesan, V., Panagiotopoulos, A. Z., Liu, H. & Kumar, S. K. Modeling the Anisotropic Self-Assembly of Spherical Polymer-Grafted Nanoparticles. *The Journal of Chemical Physics* **131,** 221102. ISSN: 0021-9606, 1089-7690 (Dec. 2009).

20. Iacovella, C. R., Keys, A. S. & Glotzer, S. C. Self-Assembly of Soft-Matter Quasicrystals and Their Approximants. *Proceedings of the National Academy of Sciences* **108,** 20935–20940. ISSN: 0027-8424, 1091-6490 (Dec. 2011).

21. Duguet, É., Hubert, C., Chomette, C., Perro, A. & Ravaine, S. Patchy Colloidal Particles for Programmed Self-Assembly. *Comptes Rendus Chimie* **19,** 173–182. ISSN: 1631-0748 (Jan. 2016).

22. Choueiri, R. M., Galati, E., Thérien-Aubin, H., Klinkova, A., Larin, E. M., Querejeta-Fernández, A., Han, L., Xin, H. L., Gang, O., Zhulina, E. B., Rubinstein, M. & Kumacheva, E. Surface Patterning of Nanoparticles with Polymer Patches. *Nature* **538,** 79–83. ISSN: 0028-0836, 1476-4687 (Oct. 2016).

23. Nghiem, T.-L., Löbling, T. I. & Gröschel, A. H. Supracolloidal Chains of Patchy Micelles in Water. *Polymer Chemistry* **9,** 1583–1592. ISSN: 1759-9954, 1759-9962 (2018).

24. Zhang, Z. & Glotzer, S. C. Self-Assembly of Patchy Particles. *Nano Letters* **4,** 1407–1413. ISSN: 1530-6984, 1530-6992 (Aug. 2004).

25. Fava, D., Nie, Z., Winnik, M. A. & Kumacheva, E. Evolution of Self-Assembled Structures of Polymer-Terminated Gold Nanorods in Selective Solvents. *Advanced Materials* **20,** 4318–4322. ISSN: 09359648, 15214095 (Nov. 2008).

26. Liu, W., Mahynski, N. A., Gang, O., Panagiotopoulos, A. Z. & Kumar, S. K. Directionally Interacting Spheres and Rods Form Ordered Phases. *ACS Nano* **11,** 4950–4959. ISSN: 1936-0851, 1936-086X (May 2017).

27. Tigges, T., Heuser, T., Tiwari, R. & Walther, A. 3D DNA Origami Cuboids as Monodisperse Patchy Nanoparticles for Switchable Hierarchical Self-Assembly. *Nano Letters* **16,** 7870–7874. ISSN: 1530-6984, 1530-6992 (Dec. 2016).

28. Bianchi, E., Tartaglia, P., Zaccarelli, E. & Sciortino, F. Theoretical and Numerical Study of the Phase Diagram of Patchy Colloids: Ordered and Disordered Patch Arrangements. *The Journal of Chemical Physics* **128,** 144504. ISSN: 0021-9606, 1089-7690 (Apr. 2008).

29. Kalb, J., Dukes, D., Kumar, S. K., Hoy, R. S. & Grest, G. S. End Grafted Polymernanoparticles in a Polymeric Matrix: Effect of Coverage and Curvature. *Soft Matter* **7,** 1418–1425. ISSN: 1744-683X, 1744-6848 (2011).

30. Iacovella, C. R. & Glotzer, S. C. Phase Behavior of Ditethered Nanospheres. *Soft Matter* **5,** 4492. ISSN: 1744-683X, 1744-6848 (2009).

31. Ndoro, T. V. M., Voyiatzis, E., Ghanbari, A., Theodorou, D. N., Böhm, M. C. & Mü ller-Plathe, F. Interface of Grafted and Ungrafted Silica Nanoparticles with a Polystyrene Matrix: Atomistic Molecular Dynamics Simulations. *Macromolecules* **44,** 2316–2327. ISSN: 0024-9297, 1520-5835 (Apr. 2011).

32. Hattemer, G. D. & Arya, G. Viscoelastic Properties of Polymer-Grafted Nanoparticle Composites from Molecular Dynamics Simulations. *Macromolecules* **48,** 1240–1255. ISSN: 0024-9297, 1520-5835 (Feb. 2015).

33. Zhang, Horsch, M. A., Lamm, M. H. & Glotzer, S. C. Tethered Nano Building Blocks: Toward a Conceptual Framework for Nanoparticle Self-Assembly. *Nano Letters* **3,** 1341–1346. ISSN: 1530-6984, 1530-6992 (Oct. 2003).

34. Iacovella, C. R. & Glotzer, S. C. Complex Crystal Structures Formed by the Self-Assembly of Ditethered Nanospheres. *Nano Letters* **9,** 1206–1211. ISSN: 1530-6984, 1530-6992 (Mar. 2009).

35. Song, P., Wang, Y., Wang, Y., Hollingsworth, A. D., Weck, M., Pine, D. J. & Ward, M. D. Patchy Particle Packing under Electric Fields. *Journal of the American Chemical Society* **137,** 3069–3075. ISSN: 0002-7863, 1520-5126 (Mar. 2015).

36. Singh, C., Ghorai, P. K., Horsch, M. A., Jackson, A. M., Larson, R. G., Stellacci, F. & Glotzer, S. C. Entropy-Mediated Patterning of Surfactant-Coated Nanoparticles and Surfaces. *Physical Review Letters* **99,** 226106. ISSN: 0031-9007, 1079-7114 (Nov. 2007).

37. Liu, Z., Guo, R., Xu, G., Huang, Z. & Yan, L.-T. Entropy-Mediated Mechanical Response of the Interfacial Nanoparticle Patterning. *Nano Letters* **14,** 6910–6916. ISSN: 1530-6984, 1530-6992 (Dec. 2014).

38. Gröschel, A. H., Walther, A., Löbling, T. I., Schacher, F. H., Schmalz, H. & Müller, A. H. E. Guided Hierarchical Co-Assembly of Soft Patchy Nanoparticles. *Nature* **503,** 247–251. ISSN: 0028-0836, 1476-4687 (Nov. 2013).

39. Meng, D., Kumar, S. K., D. Lane, J. M. & Grest, G. S. Effective Interactions between Grafted Nanoparticles in a Polymer Matrix. *Soft Matter* **8,** 5002. ISSN: 1744-683X, 1744-6848 (2012).

40. Martin, T. B., Seifpour, A. & Jayaraman, A. Assembly of Copolymer Functionalized Nanoparticles: A Monte Carlo Simulation Study. *Soft Matter* **7,** 5952. ISSN: 1744-683X, 1744-6848 (2011).

41. Moore, T. C., Iacovella, C. R. & McCabe, C. Derivation of Coarse-Grained Potentials via Multistate Iterative Boltzmann Inversion. *The Journal of Chemical Physics* **140,** 224104. ISSN: 0021-9606, 1089-7690 (June 2014).

42. Shi, K., Santiso, E. E. & Gubbins, K. E. Bottom-Up Approach to the Coarse-Grained Surface Model: Effective Solid-Fluid Potentials for Adsorption on Heterogeneous Surfaces. *Langmuir* **35,** 5975–5986. ISSN: 0743-7463, 1520-5827 (Apr. 2019).

43. Chremos, A., Panagiotopoulos, A. Z., Yu, H.-Y. & Koch, D. L. Structure of Solvent-Free Grafted Nanoparticles: Molecular Dynamics and Density-Functional Theory. *The Journal of Chemical Physics* **135,** 114901. ISSN: 0021-9606, 1089-7690 (Sept. 2011).

44. Summers, A. Z., Iacovella, C. R., Cane, O. M., Cummings, P. T. & MCabe, C. A Transferable, Multi-Resolution Coarse-Grained Model for Amorphous Silica Nanoparticles. *Journal of Chemical Theory and Computation* **15,** 3260–3271. ISSN: 1549-9618, 1549-9626 (May 2019).

45. Cheng, L. & Cao, D. Aggregation of Polymer-Grafted Nanoparticles in Good Solvents: A Hierarchical Modeling Method. *The Journal of Chemical Physics* **135,** 124703. ISSN: 0021-9606, 1089-7690 (Sept. 2011).

46. Chan, E. R., Striolo, A., McCabe, C., Cummings, P. T. & Glotzer, S. C. Coarse-Grained Force Field for Simulating Polymer-Tethered Silsesquioxane Self-Assembly in Solution. *The Journal of Chemical Physics* **127,** 114102. ISSN: 0021-9606, 1089-7690 (Sept. 2007).

47. Nielsen, S. O., Lopez, C. F., Srinivas, G. & Klein, M. L. A Coarse Grain Model for *n* -Alkanes Parameterized from Surface Tension Data. *The Journal of Chemical Physics* **119,** 7043–7049. ISSN: 0021-9606, 1089-7690 (Oct. 2003).

48. Klein, C., Summers, A. Z., Thompson, M. W., Gilmer, J. B., McCabe, C., Cummings, P. T., Sallai, J. & Iacovella, C. R. Formalizing Atom-Typing and the Dissemination of Force Fields with Foyer. *Computational Materials Science* **167,** 215–227. ISSN: 0927-0256. http://www.sciencedirect.com/science/article/pii/S0927025619303040 (Sept. 2019).

49. Summers, A. Z., Gilmer, J. B., Iacovella, C. R., Cummings, P. T. & McCabe, C. MoSDeF, a Python Framework Enabling Large-Scale Computational Screening of Soft Matter: Application to Chemistry-Property Relationships in Lubricating Monolayer Films. *Journal of Chemical Theory and Computation* **0.** PMID: 32004433, null. ISSN: 1549-9618. eprint: https://doi.org/10.1021/acs.jctc.9b01183 (Mar. 0).

50. Haley, J. D., Iacovella, C. R., Cummings, P. T. & McCabe, C. Examining the Aggregation Behavior of Polymer Grafted Nanoparticles Using Molecular Simulation and Theory. *The Journal of Chemical Physics* **143,** 054904. ISSN: 0021-9606, 1089-7690 (Aug. 2015).

51. Kalyuzhnyi, Y. V., Iacovella, C. R., Docherty, H., Holovko, M. & Cummings, P. T. Network Forming Fluids: Yukawa Square-Well m-Point Model. *Journal of Statistical Physics* **145,** 481–506. ISSN: 0022-4715, 1572-9613 (Oct. 2011).

52. Weeks, J. D., Chandler, D. & Andersen, H. C. Role of Repulsive Forces in Determining the Equilibrium Structure of Simple Liquids. *The Journal of Chemical Physics* **54,** 5237–5247. ISSN: 0021-9606, 1089-7690 (June 1971).

53. Peters, B. L., Lane, J. M. D., Ismail, A. E. & Grest, G. S. Fully Atomistic Simulations of the Response of Silica Nanoparticle Coatings to Alkane Solvents. *Langmuir* **28,** 17443–17449. ISSN: 0743-7463, 1520-5827 (Dec. 2012).

54. Glaser, J., Nguyen, T. D., Anderson, J. A., Lui, P., Spiga, F., Millan, J. A., Morse, D. C. & Glotzer, S. C. Strong Scaling of General-Purpose Molecular Dynamics Simulations on GPUs. *Computer Physics Communications* **192,** 97–107. ISSN: 0010-4655 (July 2015).

55. Anderson, J. A., Lorenz, C. D. & Travesset, A. General Purpose Molecular Dynamics Simulations Fully Implemented on Graphics Processing Units. *Journal of Computational Physics* **227,** 5342–5359. ISSN: 0021-9991 (May 2008).

56. Adorf, C. S., Ramasubramani, V., Dice, B. D., Henry, M. M., Dodd, P. M. & Glotzer, S. C. *glotzerlab/signac* version 1.0.0. Development and deployment supported by MICCoM, as part of the Computational Materials Sciences Pro- gram funded by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, Materials Sciences and Engineering Division, under Subcontract No. 6F-30844.

Project conceptualization and implementation supported by the National Science Foundation, Award DMR 1409620. Feb. 2019.

57. Ramasubramani, V., Dice, B. D., Harper, E. S., Spellings, M. P., Anderson, J. A. & Glotzer, S. C. Freud: A Software Suite for High Throughput Analysis of Particle Simulation Data. *Computer Physics Communications* **254,** 107275. ISSN: 0010-4655 (Sept. 2020).

58. McGibbon, R. T., Beauchamp, K. A., Harrigan, M. P., Klein, C., Swails, J. M., Hernández, C. X., Schwantes, C. R., Wang, L.-P., Lane, T. J. & Pande, V. S. MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. *Biophysical Journal* **109,** 1528–1532. ISSN: 0006-3495 (Oct. 2015).

59. Humphrey, W., Dalke, A. & Schulten, K. VMD: Visual molecular dynamics. *Journal of Molecular Graphics* **14,** 33–38. ISSN: 0263-7855. http://www.ncbi.nlm.nih.gov/pubmed/8744570%20http://linkinghub.elsevier.com/retrieve/pii/0263785596000185 (Feb. 1996).

60. Michaud-Agrawal, N., Denning, E. J., Woolf, T. B. & Beckstein, O. MDAnalysis: A Toolkit for the Analysis of Molecular Dynamics Simulations. *Journal of Computational Chemistry* **32,** 2319–2327. ISSN: 0192-8651 (July 2011).

61. Gowers, R., Linke, M., Barnoud, J., Reddy, T., Melo, M., Seyler, S., Domański, J., Dotson, D., Buchoux, S., Kenney, I. & Beckstein, O. *MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations* in *Python in Science Conference* (Austin, Texas, 2016), 98–105.

62. Virtanen, P. *et al.* SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17,** 261–272. ISSN: 1548-7091, 1548-7105 (Mar. 2020).

63. Lafitte, T., Kumar, S. K. & Panagiotopoulos, A. Z. Self-Assembly of Polymer-Grafted Nanoparticles in Thin Films. *Soft Matter* **10,** 786–794. ISSN: 1744-683X, 1744-6848 (2014).

64. Bozorgui, B., Meng, D., Kumar, S. K., Chakravarty, C. & Cacciuto, A. Fluctuation-Driven Anisotropic Assembly in Nanoscale Systems. *Nano Letters* **13,** 2732–2737. ISSN: 1530-6984, 1530-6992 (June 2013).

65. Kokoska, S. & Zwillinger, D. *CRC Standard Probability and Statistics Tables and Formulae* (Crc Press, 2000).

66. Asai, M., Cacciuto, A. & Kumar, S. K. Quantitative Analogy between Polymer-Grafted Nanoparticles and Patchy Particles. *Soft Matter* **11,** 793–797. ISSN: 1744-683X, 1744-6848 (2015).

CHAPTER 6

High-Throughput Screening of Tribological Properties of Monolayer Films using Molecular Dynamics and Machine Learning

6.1 Introduction

As feature lengths in micro- and nano- electromechanical devices (MEMs and NEMs) continues to decrease, the surface area (A_{surface}) to volume ratio ($\frac{A_{\text{surface}}}{V}$) of these devices increases. As a result, the operation of MEMs and NEMs devices tend to be dominated by surface effects as compared to their macroscale counterparts [1, 2], resulting in significant friction and wear (tribology). Traditional hydrocarbon-based lubricants are too viscous to effectively lubricate MEMs/NEMs devices leading to the need to develop alternative lubrication strategies. As such, thin film coatings have emerged as a popular way to reduce the severity of frictional effects in MEMs and NEMs [3]. Understanding and being able to minimize/engineer preferred nanotribological effects is important to improve nanotechnology. To do this requires a molecular-level understanding of these surface and interfacial effects. With the ability of atomic force microscopy (AFM) to inspect and investigate thin films, experimentalists can study and characterize nanotribological effects [4] with increasing precision. Also, computational chemistry gives us unparalleled tunability and the ability to isolate many molecular-level effects, especially in the field of nanotribology.

These thin film systems have many tunable parameters, from chain length, film composition, polymer chemistry, and the structure and chemistry of the substrate material to name a few. This presents a rich chemical parameter space to evaluate, which is extremely suitable for computational chemistry. Using computational chemistry and tools to accelerate the design and implementation of this vast parameter space, computational studies can accelerate the identification and molecular-level understanding of candidate thin films faster and cheaper than attempting to do this through experimental means alone. In a similar effort to the Materials Genome Initiative (MGI) [5], which leverages computational screening of many hard materials to pre-screen for potential candidate materials for experimental synthesis, being able to apply similar approaches to thin films and other soft material designs could have massive impacts on nanotribology and other disciplines. However, the computational study of soft materials (e.g., molecular liquids, polymers, and other biomaterial that are easily deformed at relevant conditions and can only be described by an ensemble of structures) is quite challenging to simulate compared to hard materials (e.g., crystals, alloys, salts, and other materials which does not easily deform at biologically–relevant conditions). For example, in soft materials, the intermolecular forces are of the same order of magnitude as the thermal motion ($k_B T$, where k_B is Boltzmann's constant and T is the absolute temperature), in contrast to hard materials where the interatomic forces are much greater than $k_B T$ (typically resulting in crystalline structures). This means that for soft materials, very long simulation times are needed to ensure proper thermodynamic sampling as well as quite large systems to capture the mesoscale order, drastically different to hard materials. Like the MGI, specialized tools are necessary to better facilitate the screening and study of candidate soft matter systems. These tools need to be readily available to computational chemists, expose many tunable parameters of the systems to support screening, be extensible by others, and also promote and support reproducible soft matter simulations.

In this chapter [1] MoSDeF and the TRUE[8] principals are used to explore the scalable screening of soft matter thin films for tribological properties. Then, predictive random forest (RF) models are developed to explore the effect of the thin film coating terminal group chemistry and its effects on coefficient of friction (μ) and the force of adhesion (F_0). The work covered in this chapter span two journal articles [6, 7] and provides a useful workflow for high throughput screening of soft matter films with non-equilibrium molecular dynamics (NEMD).

6.2 Background

Monolayer films have shown promise as a means of reducing friction and wear of mechanical devices with nanoscale surface separations (e.g., nano- and micro-electromechanical systems, NEMS and MEMS) [9, 10]. Such films are highly tunable through modification of their terminal group chemistries, backbone chain length, backbone chemistry, and film composition, all of which have been demonstrated to impact their tribological effectiveness along with other properties, such as durability, solvent interactions, and thermal response [9–13]. This tunability presents a rich chemical parameter space that can be explored for the optimization of film properties as well as gleaning useful information about the quantitative structure-property relationships (QSPR) of these systems, to develop better predictive models for design considerations [14, 15]. Of these various modifications, the chemical and physical characteristics (descriptors) of the terminal groups plays a dominant role in the tribological response. For example, Yu et al. [12] showed that phenyl-terminated monolayer thin films yield higher frictional forces than methyl-terminated films, explained by the phenyl groups' ability to twist and impede movement during shear (these can be thought of as structural molecular descriptors according to QSPR) [14]. Hydroxylated and carboxylated thin films have been shown to have high frictional and adhesive forces relative to methyl-terminated films, attributed to their ability to form inter-monolayer hydrogen bonds during contact (physicochemical descriptors according to QSPR) [13,

[1]Reprinted with permission from (Summers, A. Z., Gilmer, J. B., Iacovella, C. R., Cummings, P. T. & McCabe, C. MoSDeF, a Python Framework Enabling Large-Scale Computational Screening of Soft Matter: Application to Chemistry-Property Relationships in Lubricating Monolayer Films. *Journal of Chemical Theory and Computation* **0.** PMID: 32004433, null. ISSN: 1549-9618. eprint: https://doi.org/10.1021/acs.jctc.9b01183 [Mar. 0]), (Quach, C. D., Gilmer, J. B., Pert, D. O., Mason-Hogans, A., Iacovella, C. R., Cummings, P. T. & McCabe, C. High-Throughput Screening of Tribological Properties of Monolayer Films Using Molecular Dynamics and Machine Learning. *The Journal of Chemical Physics,* 5.0080838. ISSN: 0021-9606, 1089-7690 [Feb. 2022]). Copyright 2022 American Chemical Society.

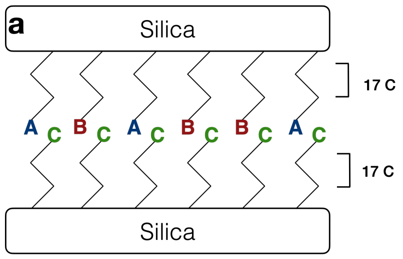
14]. Similar trends found in molecular dynamics (MD) simulations further support these relationships[6, 16, 17]. Moreover, this interfacial region may feature not just a single terminal group chemistry but instead multiple chemistries. For example, experiments by Brewer et al. [13] demonstrated that a methyl-functionalized microscope tip in contact with either hydroxyl or carboxyl terminated monolayers results in a lower coefficient of friction (COF) compared to the same tip in contact with a methyl terminated monolayer. Monolayers composed of two or more terminal group chemistries within the same layer, at varied relative compositions, may also provide a means to further tune and improve performance. For example, computational studies by Lewis et al. [18] for monolayers composed of methyl terminated alkanes mixed with perfluoroalkanes showed a regime where the COF was reduced compared to either pure component system. There are a multitude of complex relationships between these various chemical/molecular descriptors when translated to monolayer polymer systems as hinted at by Le et al. [14]

MD simulations are a useful tool to perform large-scale sweeps of the accessible parameter space to create an in silico self-consistent data set. Computational examination avoids the need to develop experimental synthesis techniques, which may be non-trivial and time intensive. Simulations can also more effectively reveal the intrinsic properties associated with defect-free films on pristine contaminant-free surfaces. This approach has been utilized to study and optimize various parameters describing the monolayer, such as backbone chain length and chain densities. Our recent development of the Molecular Simulation and Design Framework (MoSDeF [19]) and the development of the Signac Framework[20–23] by Glotzer et al., enables the large-scale screening of soft matter systems, allowing the reproducible initialization and parameterization of systems, and the management of large dataspaces. These tools have been used to perform large scale screening studies of soft matter systems in several recent papers [6, 24] as well as used to fully capture the provenance of simulation workflows for increased reproducibility in other work [8, 25].

However, the vast parameter space to be explored for monolayer films would still make brute force computational screening impractical. Instead, a promising approach is to combine computational screening with machine learning (ML) techniques in order to accelerate and better direct the exploration of the parameter landscape [6]. That is, use computational screening to gather sufficient data to train predictive ML models (thus minimizing the number of computationally expensive simulations), and subsequently using the ML models to predict the properties of new systems and guide the screening towards film chemistries with desirable properties. This approach has been utilized in other various studies with great success, such as developing predictive models that have errors lower than hybrid Density Functional Theory (DFT) methods [26, 27], that learn and predict various protein folding events and structures [28–32], that use active learning to direct iterative optimizations and uncover optimal targets like structures [33–37] and to accelerate the discovery of novel monomers and polymers for favorable macroscopic properties [33, 38–43]. Different ML models and techniques are applied but all demonstrate a useful approach to leverage in silico data from molecular simulations and experimental data (when available). As the power of in silico data is further realized for ML models and predictive design, being able to rapidly generate, screen, learn, and predict from these data will be powerful tools for computational and experimental researchers alike.

In prior work, we developed a screening framework that enable computational screening studies to be performed over a multitude of terminal group chemistries for contacting monolayers undergoing shear using non-equilibrium molecular dynamics (NEMD) simulations [6]. Uniform monolayers with 16 different terminal group chemistries were examined, with each monolayer terminal group chemistry independently varied, allowing key trends and several chemistry combinations that provided favorable tribological performance, i.e., both low COF and low adhesion force (F_0), to be identified. Furthermore, data from 100 different monolayer terminal group combinations were used to develop, train, and test ML models that allow COF and F_0 to be predicted with good accuracy solely from the chemistry of the terminal groups expressed as SMILES strings (discussed in detail in the Methods section). In addition, additional studies were performed to evaluate the model's predictive ability and extensibility to non-uniform terminal group monolayers. See Section D.1 for a summary of the previous work related to chemically dissimilar films. The success of these models, trained from a relatively small dataset, suggests that this approach can be used to prescreen computational space and accelerate the identification of films with favorable properties, assuming the dataspace is diverse enough to minimize the chance of overfitting. However, to determine if tribological performance could be further improved —for example, by mixing terminal groups together within a film —several key questions regarding the use of ML in a predictive capacity for monolayer films remain. Specifically, (1) what is the minimal data set required in order to adequately train ML models for tribological properties; (2) how transferrable are the ML models to systems with other chemistries and film compositions not included in the training data; and (3) can the models be used to prescreen the design space and if so, what level of accuracy can be achieved with a reasonable amount of training data?

To address these questions and further probe the tribological design space of thin films and the relationships between terminal group chemistry, thin film composition, and tribology, here we consider systems in which one monolayer consists of a single unique terminal group, and the other monolayer contains a mix of two unique terminal groups, allowing the relative composition of the two groups to be varied. For these designs, a pool of 19 different terminal groups were considered resulting in 9747 unique monolayer combinations, when considering the mixing ratio of terminal groups in the mixed monolayer, as will be discussed in detail in the Methods section and Figure 6.1. In the Methods section we provide an overview of the computational approach, focusing on the simulation workflow, analysis methods, and the ML model. In the results section, we present the data generated from the MD screening and identify key terminal groups and combi-



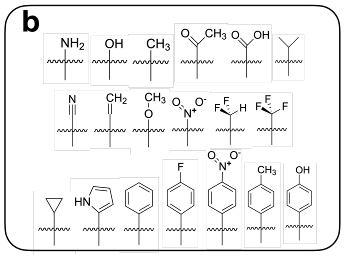


Figure 6.1: (a) Simplified schematic of the systems studied. The top monolayer is a mixture of two types of terminal groups chemistries (A and B), studied at two different mixing ratios (25:75 and 50:50), while the bottom monolayer is homogeneous (chemistry C). (b) Depiction of the 19 different chemistries considered. From top to bottom, left to right, the terminal groups are amino, hydroxyl, methyl, acetyl, carboxyl, isopropyl, cyano, ethylene, methoxy, nitro, difluoromethyl, perfluoromethyl, cyclopropyl, pyrrole, phenyl, fluorophenyl, nitrophenyl, toluene, and phenol.

nations associated with improved tribological performance. We then develop and evaluate the dependence of the ML models on the training set used, assessing the effectiveness of using a ML algorithm to predict properties. Finally, we utilize the ML model to demonstrate the feasibility of screening large data spaces (193,131 unique systems, created from 621 chemistries from the `CheMBL` library [44, 45]), investigating suitable strategies for utilizing ML models to guide future work. All relevant information to reproducibly generate this data and workflow is readily available and adaptable for others to use, following the principle of TRUE (Transferable, Reproducible, Usable by others and Extensible) simulations described by Thompson *et al.* [8] See Appendix D for more details.

6.3 Methods

6.3.1 Simulation Model and Workflow

In all cases, the simulated system consists of two opposing amorphous silica surfaces, each coated with an alkylsilane monolayer film. Specifically, each surface has dimensions of 5x5nm^2, constructed using the procedure outlined by Summers et al. [46], and available as an `mbuild` script provided in the supplemental GitHub repository [47]. The silica surfaces have an average surface roughness of 0.11nm, closely approximating the more computationally intensive synthesis mimetic simulation approach by Black *et al.* [48] 100 alkylsilane chains are chemically bonded to each surface, with an in-plane surface density of 4 chains nm^{-2}; this surface density is consistent with prior computational studies [6, 46, 48] and experiments [49–51] that estimate chain surface densities to be between 4.0–5.0chains nm^{-2}. Each alkylsilane chain is composed of a fully saturated 17 carbon backbone that is capped with a terminal group that can be easily varied computationally (see Figure 6.1b). The remaining undercoordinated oxygens at the surface are changed to hydroxyl groups to mimic surface oxidation. Of the two surfaces in the dual monolayer systems, the bottom surface is homogeneous (singular terminal group), while the top surface contains a mixture of two types of alkylsilane chains, differing by their terminal groups. The mixing ratios for the top monolayers considered in this study are 25 : 75 and 50 : 50. The pool of 19 different terminal group chemistries investigated are shown in Figure 6.1b; this adds 3 additional terminal group chemistries to those considered by Summers *et al.* [6] The uniform bottom monolayer and the mixed top monolayer can be composed of any combination of groups from Figure 6.1b, with the constraint that the two groups in the mixed monolayer must be different. In total 12,996 combinations ([19 terminal groups in uniform layer] * [19 * 18 terminal group combinations in mixed layer] * [2 composition ratios]) were considered; this translates to a total of 116,964 simulations (12,996 * 3 * 3) when factoring in the composition ratios studied, the 3 normal loads, and 3 replicates considered for each system. Of the 12,996 systems considered, 3249 systems with the mixing ratio in the top monolayer of 50 : 50 were duplicated during the screening and thus such combinations had 3 additional replicates; in total 9747 unique combinations ($19*19*18$ of 25:75 systems $+\frac{1}{2}*19*19*18$ of 50:50 systems) were considered. We also note that a small subset of simulations (less than 1% of the total) failed to complete due to unstable initial configurations, but in all cases, each unique system composition reported includes at least 3 replicates.

Each monolayer system was prepared using the MoSDeF software suite[19, 52–55] (see Appendix D for additional information). The initialization of the monolayer structure is encapsulated as an `mbuild` recipe [53, 56], which preserves the entire process used to construct the monolayer structure. The `foyer` library [52, 57] was used to atom type and parameterize each system with the Optimized Potential for Liquid Simulation - All Atoms (OPLS-AA) forcefield[58]. Parameters for the alkylsilane chains were taken from GROMACS 5.1 [59–61], and those for the silica surface from Lorenz et al [62]. The force field XML (extensible markup language) file used by `foyer` is provided in Appendix D and can be accessed from the Supplemental Repository[47]. The project workflow as a whole was managed using the `Signac` Framework [21, 22]. The use of MoSDeF in addition to the *`Signac` Framework*, ensures that all scripts and input parameters used to initialize the systems, submit the systems for simulation, and analyze the systems are captured and preserved, ensuring the simulations are TRUE (Transparent, Reproducible, Usable by Others, and Extensible) [8]. All scripts and parameter files are available in the associated GitHub repository [47] (see Appendix D).

MD simulations were performed using the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) and GROMACS simulation engines [59, 61, 63, 64]. LAMMPS was solely used to relieve the system of initial high-energy configurations and possible overlaps [63, 64]. The more stable structure generated was then fed to GROMACS to perform the rest of the simulation workflow, starting with energy minimization following a steepest descent algorithm, followed by a 1 ns equilibration in the canonical (NVT) ensemble at 298K using the Nóse–Hoover thermostat[59, 61, 65]. An NVT simulation was then performed at

298 K in which the two surfaces were brought into contact by applying a constant normal force of 5nN along the z direction to the bottom surface over 0.5ns, allowing for the distance between the two surfaces to reach a steady state. After compression, shearing simulations (with surfaces moving at relative speed of $10\,\mathrm{m\,s^{-1}}$) were performed at 3 different normal loads of 5nN, 15 nN, and 25 nN. Specifically, the shearing process is simulated by pulling a ghost particle, which is coupled to the top surface via a harmonic spring with a spring constant of $10{,}000\,\mathrm{kJ\,mol^{-1}\,nm^{-2}}$, in the x direction at $10\,\mathrm{m\,s^{-1}}$. The shear is simulated for 10 ns, and the last 5 ns is used for analysis (production regime). The particle-particle particle-mesh (PPPM) algorithm was used to calculate the long-range electrostatic interactions, using a force and pressure correction in the z-dimension to support slab geometries; systems are periodic in the monolayer plane [61, 66].

6.3.2 Calculation of Tribological Properties

The coefficient of friction (COF), μ, and adhesion force, F_0, were calculated from the last 5 ns of the simulation trajectory for each system under shear using the modified version of Amonton's law of friction given by Equation 6.1. Here F_f, F_0, μ, and F_N represent the frictional force, the adhesive force, the coefficient of friction, and the normal force, respectively [67]. A linear regression of the average friction force (ordinate) versus normal load (abscissa) can be used to calculate the COF from the slope and F_0 from the intercept of the regression line with the ordinate axis. The friction force is calculated by summing all the forces in the direction of shear on one of the monolayers every 1ps and averaged over the last 5ns of the simulation.

$$F_f = F_0 + \mu F_N \tag{6.1}$$

6.3.3 Machine Learning Model

The dataset is analyzed using the random forest regressor as implemented in the `scikit-learn` library, consistent with our prior work in Summers *et al.* [6, 68] Provided a training set, a set of input parameters and expected outputs, the random forest ensemble model will create a series of decision trees, each generated from a sub-sample of the training data set [69, 70]. The trained random forest model then makes predictions by averaging the predicted values from the component decision trees. Each predictive model will rank the importance of each of the input parameters based on how a given input affects the final prediction. This ranking unveils information regarding properties that play an important role in predicting the tribological properties of the monolayer (feature importance), making this method advantageous for screening/discovery research. All of the random forest models in this study have 1000 trees, ensuring the predictions converge in a reasonable amount of time[70]. In the forest, each decision tree is allowed to expand until all leaves are pure (choosing splits that decrease impurity defined by the Gini impurity). All models used mean squared error (MAE) as error criterion during training. Each random forest model, and its subsequent decision trees, are trained with 32–42 features, which are molecular descriptors calculated through `RDKit`[71]. This setup is consistent with previous study by Summers *et al.* [6] allowing for direct comparison between these studies, focusing on the accuracy of the models and feature importance ranking determined from the two sets of data. An effort to optimize the parameters of the random forest model resulted in insignificant improvement in model accuracy and thus the original values were retained for better comparability with prior work; further details can be found in Appendix D (see Figure D.9–Figure D.11).

The chemical and physical input parameters for the ML model are supplied by the `RDKit` cheminformatics library [71]. The COF and F_0 calculated from the simulations are the expected outputs (i.e., targeted properties) for the random forest ensemble to predict. The training procedure of these predictive models is adapted from that described in previous work [6]. Briefly, each of the systems in the training set can be represented by a set of SMILES strings [72], describing the terminal group chemistry. Each terminal group is represented by two SMILES strings: one of a hydrogen capped structure and one of a methyl capped structure. The SMILES strings are used to calculate molecular descriptors that characterize the chemical and physical properties of chemical structures, via the `RDKit` [71] cheminformatics library. These descriptors fit into four categories: size (e.g., molecular weight), shape (e.g., inertial shape factor), complexity (e.g., degree of branching), and charge distribution (e.g., topological polar surface area). The SMILES string of the hydrogen- capped terminal group is used to calculate descriptors relating to shape, while the SMILES string of the methyl-capped terminal group is used to calculate the remaining descriptors. While shape characteristics can be sufficiently modeled with a hydrogen-capped structure, properties that involve charge distribution among others are better represented if they mimic the actual structure the terminal groups are attached to; a methyl terminus was found to be a sufficient approximation of the alkyl chain for these measurements [6]. Through this process, each chemical structure is represented by 53 descriptors, summarized in Appendix D (see Table D.19). The molecular descriptors for the top and bottom monolayers are first calculated independently. Descriptors for the top monolayer, with two terminal groups, are the weighted average (by relative composition) of its component terminal groups' descriptors, while descriptors for the bottom monolayer are the molecular descriptors of its singular terminal group. This representation can have limitations when used to describe monolayers, since it does not encode information regarding connectivity of constituent chains and distribution pattern [73]. However, since we are mainly interested in the contribution of different terminal group chemistries, making up the intermonolayer regions/interfaces, our method of calculating the molecular descriptor "fingerprint" was found to be sufficient to encapsulate information for the region of interest, with an assumption that the two terminal groups in the top monolayer are randomly distributed. These are then combined, storing the mean and minimum of each descriptor as the “molecular fingerprint” of the entire system, totaling 106 descriptors ([53 metrics]*[2 corresponding to min and mean]), which has been demonstrated in previous work to encapsulate the most important features of these systems [6]. These “molecular fingerprints” later undergo a dimensionality reduction step that removes descriptors whose values have low variance and reduces highly correlated descriptors. Specifically, descriptors whose values are at least 90% correlated will be reduced to only one attribute (descriptors are sorted alphabetically), while descriptors whose variance is below 2% are also removed. This step reduced the number of descriptors (or effective fingerprint) of each system to be between 32 and 45, with a mean of 37 ±3. The number of effective fingerprint descriptors decrease as the size of the training data increase, since some correlations may not manifest with a smaller data set. This dimensionality reduction process is consistent with that used in Summers *et al.* [6] using the source code hosted in a GitHub repo [74]. The reduced list of molecular descriptors are then used as the input parameters to the ML models. The process of determining molecular “fingerprint” of each system is summarized in Figure 6.2.

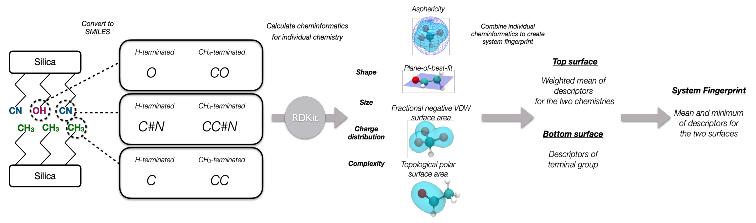

Convert to SMILES
Silica
CN
OH
CN
CH3
CH3
CH3
Silica
H-terminated
CH3-terminated
O
CO
C#N
CC#N
C
CC
Calculate cheminformatics for individual chemistry
RDKit
Shape
Size
Charge distribution
Complexity
Asphericity
Plane-of-best-fit
Fractional negative VDW surface area
Topological polar surface area
Combine individual cheminformatics to create system fingerprint
Top surface
Weighted mean of descriptors for the two chemistries
Bottom surface
Descriptors of terminal group
System Fingerprint
Mean and minimum of descriptors for the two surfaces

Figure 6.2: Process of generating the molecular descriptors (fingerprints) of the dual monolayer systems. Component terminal group chemistries of each top and bottom monolayer are represented by an H-terminated and methyl (CH_3)-terminated SMILES string, which can be used by `RDKit` to calculate corresponding molecular descriptors. For this study, we consider a total of 53 descriptors (listed in Table D.19), which can be grouped into four categories, namely, shape, size, charge distribution, and complexity. The weighted averages of these descriptors are then calculated to represent their corresponding surface, which, in turn, will be used to determine the fingerprint of each system. Figure adapted from Summers et al. [6]

The total simulation data set is split into subsets as shown graphically in Figure 6.3, as a means of examining different aspects of the training and transferability of ML models. In all cases, the data are split such that approximately 20% of the data are reserved for testing purposes, while the remaining 80% are used for training the model. When considering all composition ratios, the training and test sets are labeled total-train and total-test, respectively. We note that the most favorable systems, defined as those with both low COF and F_0 values, of the entire simulation dataset (see Table 6.1) are included in the total-test set and removed from the training procedure. This allows us to evaluate the ML models’ ability to re-identify these systems from a test set, which will be explored in the body of this work. As shown in Figure 6.3, the dataset can be further broken down by composition ratio, with the 50:50 and 25:75 mixing ratios considered

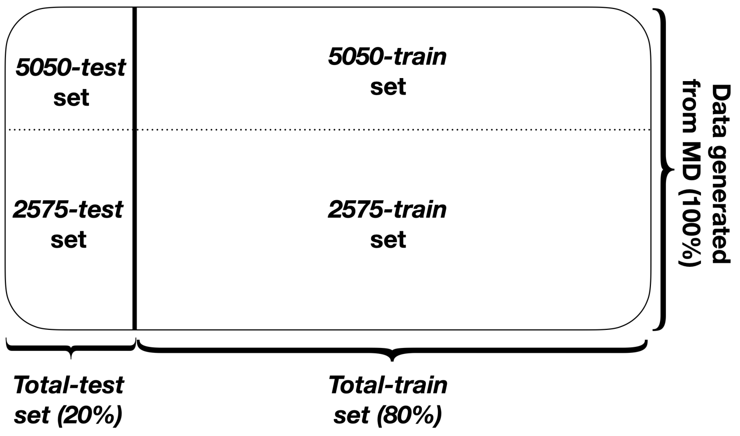


Figure 6.3: Summary of the data splitting process. The MD simulation data are split into two subsets: a *total–test* set (20% of the dataset) and a *total–train* set (80% of the dataset). The testing set is subdivided to create the *5050–test* and *2575–test* sets. The training set is also subdivided to create the *5050–train* and *2575–train* sets.

separately, e.g., 5050-test/5050-train and 2575-test/2575-train. To examine the role of training set size on the performance of the ML model the total-train and 5050-train sets are further subdivided to create training sets with fewer data points. Since COF and F_0 of these films have been shown to have little correlation [6], they are considered independently in the development of the ML models. To ensure that that the entire range of COF and F_0 are being properly sampled (Figure 6.4), each training set is binned based upon the values of the COF or F_0 into 10 equal size quantiles. Data are then randomly selected from each bin to create training sets

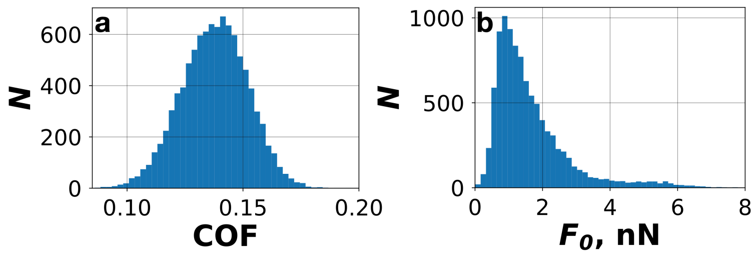


Figure 6.4: Distribution of (a) COF and (b) F_0 for systems considered in this study, obtained from MD simulations.

of varying size. For each training set size, 5 variations are created that differ only by the random seed used when sampling from the corresponding master training set, e.g., total-train and 5050-train. Distributions of individual training sets are examined to ensure that they resemble the distribution of the full data set. Each training set is then used to train and create a predictive ML model that predicts either COF or F_0.

6.3.4 Results

Considering first the results of the high throughput screening MD simulations, including those performed in the current study and in Summers et al. [6], we identify 22 monolayer designs that provide favorable frictional properties, e.g., those that have low simulated COF and F_0 values (see Table 6.1 and Figure 6.5). This list was

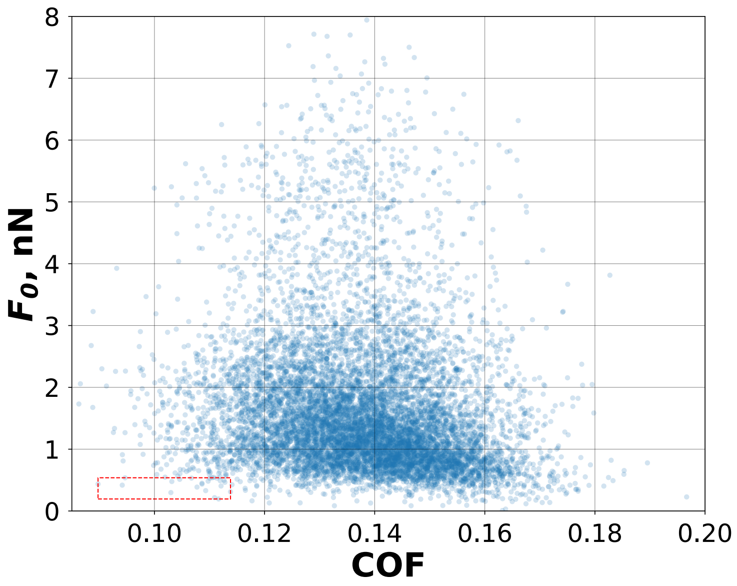


Figure 6.5: Distribution of simulated systems based on their COF and F_0 values. The 22 most–favorable systems, listed in Table 6.1, correspond to data points confined within the red dashed box in the lower left quadrant of the figure.

created by the intersection of the best 500 systems ranked from lowest to highest COF (values ranging from 0.074 to 0.114) with the best 500 systems ranked from lowest to highest F_0 (values ranging from 0.007 nN to 0.541 nN). We first note that, in general, these designs are in agreement with the conclusion obtained by Summers et al. from a study of a considerably smaller dataset, where it was noted that the COF of monolayers is primarily affected by the shape and size of the terminal group, with chemistries of small sizes and simple shapes (e.g., sp hybridization) exhibiting the lowest COF [6]. Summers et al. also noted that the F_0 is most strongly affected by charge distribution, with polarity and hydrogen bonding both increasing the F_0 [6]. In agreement with these findings, we observe that a majority of the systems identified (19 out of 22) consists of a cyano homogeneous monolayer. The cyano group is small in size, has sp hybridization and does not readily form hydrogen bonds, characteristics that fit with previous work to identify chemistries that can lower the COF and F_0 of monolayers. We also note most systems in Table 6.1 are made up of 3 components and only one system that consists of two homogeneous monolayers (System 1 in Table 6.1), which was simulated in the Summers et al. work [6]. This result suggests a slight advantage to having mixed monolayer designs. However, we also recognize that the data set is dominated with mixed monolayers compared to homogeneous monolayers, therefore the best performing systems are likely the result of the much larger representation of mixed monolayer systems compared to the homogeneous systems. Nonetheless, mixed monolayer systems could provide extra flexibility during the design process and allow for the optimization of other properties, such as thermal stability or environmental interactions, depending on the specific application, giving these designs advantages over homogeneous monolayers.

Using the simulation data, we now explore combining ML techniques with MD simulations to perform high-throughput screening of monolayer systems. In Summers et al., the random forest regressor algorithm was applied to create predictive ML models to estimate the frictional properties of alkylsilane monolayers capped with different terminal group chemistries [6]. The ML models were trained on simulation data for homogeneous monolayers with 16 distinct terminal groups, resulting in a relatively small data set, containing only 100 data points. The models were then applied to a test set and compared to the COF and F_0 results obtained for the same systems directly from MD simulation to determine the accuracy of the ML models. This comparison can be readily visualized by plotting the tribological properties obtained from the ML models against the values calculated from the MD simulations; the coefficient of determination (R^2) and the mean absolute percentage error (MAPE) of the plots are used to quantitatively measure the accuracy of the ML predictions. The R^2 is commonly used/reported to quantify the correlation between the simulated and predicted values, and MAPE provides error metrics that scale by the prediction values[75]. Using the additional simulation data generated herein, we can now better assess the feasibility of using ML to predict tribological properties and determine the amount of data necessary to create models that can make sufficiently precise estimations, as described below.

We first train a new set of ML models using 1000 data points from the 5050-train set. The models are then applied to the 5050-test set and, as described in the Methods section, compared to the COF and F_0 results obtained directly from the MD simulations in order to determine the accuracy of the ML models (see Figure 6.6). Results for the ML models trained with the Summers et al. [6] data set applied to the 5050-test set are also shown for comparison. When applied to the same testing set, the Summers et al. [6] models provide R^2 values of 0.472 and 0.657 for COF and F_0 respectively, compared to 0.822 and 0.899 for COF and F_0 from the 5050-train set. While the R^2 values are lower for the Summers et al. ML models, it is worth noting that the training data set did not include any information regarding mixed monolayer compositions; as such, the Summers et al. ML models still demonstrate impressive efficacy. This point is further demonstrated by their MAPE, where the Summers et al. models could predict COF of system with 0.056 (5.6%) error and predict F_0 with 0.266 (26.6%) error. These MAPE values are higher but are still comparable with those produced by the new set of models, trained with 10-fold amount of data. Clearly our prediction of F_0 is less accurate in the higher adhesion regime than the lower adhesion regime, as seen in Figure 6.6b and Fig-

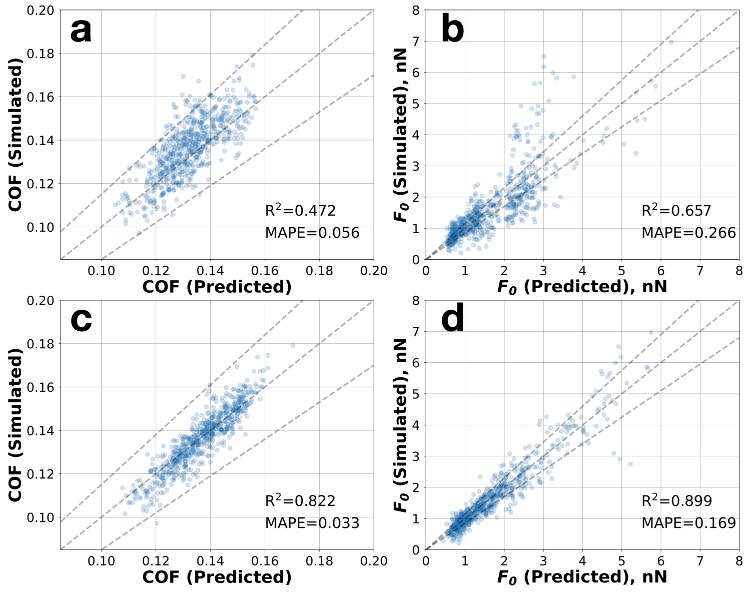


Figure 6.6: Predicted-versus-simulated plots for COF and F_0 for models trained with 100 simulation data points for uniform monolayers from the dataset of Summers et al. [6] (a) and (b) and trained with 1000 data points randomly chosen from the 5050-train set (c) and (d). The dotted line in the middle represents perfect prediction ($y = x$). The outer two lines represent the 15% variation from a perfect prediction ($y = 1.15x$ and $y = 0.85x$). The Coefficient of determination (R^2) and mean absolute percentage error (MAPE) are included. For each system (data point), the predicted properties are averaged from the five predictions made by the five ML model replicates, and the simulated properties are averaged from at least three simulation replicates.

ure 6.6d, as was also observed in the prior work of Summers et al. [6] This is likely related to the challenges associated with capturing the ability of systems to form hydrogen bonds between contacting layers, although, since our primary goal in this work is to identify systems with low adhesion values, quantitative agreement in the higher adhesion regime is not required, as previously discussed in Summers et al. [6] Nonetheless, this result suggests that ML models trained with limited data could still provide meaningful estimation, and that the use of the random forest regressor may lead to models that are predictive for chemistries and compositions outside of the training set. It should be noted, however, that this relationship could be solely related to these monolayer systems and specifically to non-equilibrium shearing and may not be applicable to other non-equilibrium studies. The prediction deviation plots for these models are shown in Figure 6.7. In general, we see that for lower values of COF or F_0, both models deviate slightly in the positive direction, meaning they predict a slightly higher value compared to simulation; as the value of either COF or F_0 increases, a negative deviation is observed with the ML models predicting slightly better performance than is observed in the MD simulations (see Figure 6.7a, b). This skew in the predictions suggests that for favorable tribological conditions (i.e., low COF and low F_0), the model will tend to overestimate the values, thus reducing the likelihood of incorrectly identifying poor performing films as viable options. This trend appears to be correlated

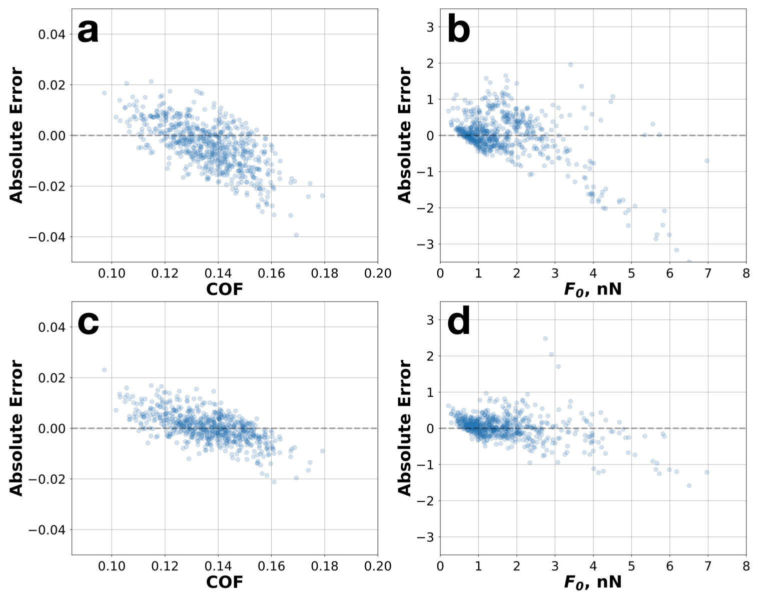


Figure 6.7: Deviations in predictions of COF and F_0 from ML models trained with 100 simulation data points for uniform monolayers from Summers et al. [6] [(a) and (b)] and trained with 1000 simulation data points randomly chosen from the *5050–train* set [(c) and (d)].

to the size and distribution of the training set provided, with the trend becoming less apparent as the size of the training data set is increased (see Figure 6.7c, d). Given that this behavior of the model minimizes the chances of exaggerating the performance of high performing systems (i.e., those with low COF and F_0), this suggests the predictive ML models can be more confidently used to screen over potential film candidates for possible applications. We also note that while the R^2 values for COF models are substantially smaller than those of F_0 models, which might suggest the latter models outperform their COF counterparts, their MAPE values indicate the opposite, where the F_0 models exhibit significantly greater percentage errors. This disparity could be attributed to the difference in the range of these two properties; while COF values span a small range of values from roughly 0.085 to 0.2, F_0 can take values from 0 nN to 8 nN (see Figure 6.4), which may affect how these metrics are calculated. Hence, it is important to recognize that that neither R^2 or MAPE values can directly relate to the predictive ability of the COF and F_0 models, though they can still be used to compare the performance of ML models of a similar type. Comparison of the feature importance of the two models at this point can be misleading, since the two set of models are trained with feature vectors of various size and components, as discussed in the Methods section. An extensive comparison feature importance of different models in this study is further discussed in Appendix D (see Figure D.12–Figure D.15).

We further examine the quality of the ML model estimations as a function of training set size by gradually increasing the amount of data used to train the ML models. We start with only using data from the 5050-train set; by doing so, we can later evaluate the transferability of the model to the 25:75 systems, i.e., can these predictive models provide similarly precise estimation of the frictional properties of systems with different designs. The R^2 and MAPE values for models trained on data sets of increasing size from the 5050-train set are reported in Figure 6.8. The results for each data set size are calculated from 5 models, each differing by

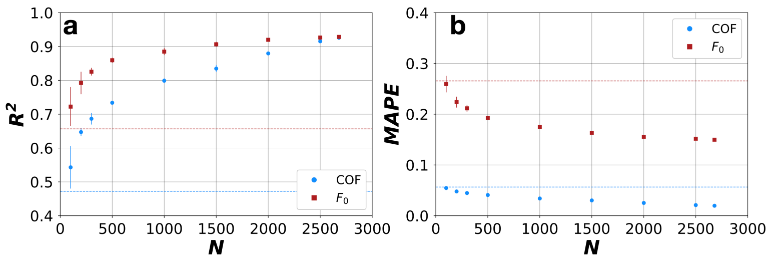


Figure 6.8: Correlation between the amount of data in the training dataset (N) and the predictive ability of the models trained using the *5050–train* sets when applied to the *5050-test* set to predict the COF (blue circles) and F_0 (red squares) quantified by R^2 (a) and MAPE (b). The dashed, horizontal lines, colored to correspond to its respective property in the key, show the predictive ability of the models trained using the dataset of Summers et al. [6] For each data point, the metrics (R^2/MAPE) are averaged from the metric of individual ML models (five replicates) when applied to the test set. The error bar of each point represents the standard deviation of the five ML models, each trained with different combinations of data.

the random seeds used when sampling from the 5050-train set, and the standard deviation of the predictions of the 5 models are represented as error bars. The R^2 and MAPE of estimations made by the Summers et al. [6] models are also shown in dotted lines for reference. From Figure 6.8a, we can see that the accuracy of the ML model predictions improves rapidly as the training set size is increased, with the ML predictions roughly plateauing between 1000 and 1500 data points for which the R^2 values for COF are 0.799 ±0.007 and 0.835 ±0.011 and for F_0 are 0.885 ±0.001 and 0.907 ±0.007, respectively. Similarly, in Figure 6.8b, the MAPE of both COF and F_0 models also decrease gradually from 0.0546 ±0.003 (5.46% ±0.3%) for COF and 0.259 ±0.016 (25.9% ±1.6%) for F_0 at 100 data points, and level off near 1000 data points, at 0.0339 ±0.001 (3.39% ±0.01%) for COF and 0.175 ±0.003 (17.5% ±0.3%) for F_0. While the predictive ability of the models does increase with the size of the training set beyond 1000 datapoints, the gains in accuracy are much less significant, suggesting one could achieve sufficiently accurate ML models even with a modest amount of data. We now examine the transferability of the ML algorithms in terms of their ability to predict the frictional properties of systems with different designs, i.e., systems with a different mixing ratio on the top monolayer in the testing set than in the training set. Results from applying the ML models described above, trained solely with data from the 5050-train set and the Summers et al. data set, on the 2575-test set are reported in Figure 6.9. From Figure 6.9a, we observe that the R^2 values for COF and F_0 increase rapidly before plateauing, again at a training set size of approximately 1000 points, with values of 0.647 ±0.001 and 0.848 ±0.007 for COF and F_0, respectively. Similar trends are observed in Figure 6.9b, where the MAPE

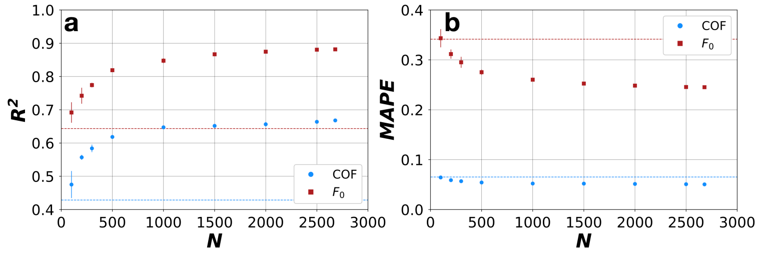


Figure 6.9: Correlation between the amount of data in the training dataset (N) and the predictive ability of the models trained using the *5050-train* sets when applied to the *2575-test* set to predict the COF (blue circles) and F_0 (red squares) quantified by R^2 (a) and MAPE (b). The dashed, horizontal lines, colored to correspond to its respective property in the key, show the predictive ability of the models trained using the dataset of Summers et al. [6] For each data point, the metrics (R^2/MAPE) are averaged from the metric of individual ML models (five replicates) when applied to the test set. The error bar of each point represents the standard deviation of the five ML models, each trained with different combinations of data.

of both COF and F_0 models rapidly decrease up until1000 data points before leveling-off, with an error of 0.0521 ±0.0 (5.21% ±0.0%) for COF and 0.26 ±0.003 (26.0% ±0.3%) for F_0. While the accuracy is lower, i.e., lower R^2 and higher MAPE, than that observed for the 5050-test set (see Figure 6.8), the agreement is promising, considering the ML models were not trained with data at these composition ratios. The plateau of the accuracy of the models is important, as it shows that improvements in accuracy of the models with larger data sets (as seen in Figure 6.9) do not necessarily manifest themselves when the model is transferred to compositions outside of the original training set. Moreover, from Figure 6.8 and Figure 6.9, we notice the 5050-train models trained with 100 data points also exhibit slightly better accuracy than the model trained with Summers et al. [6] data set, likely due to the inclusion of mixed-monolayer systems in their training sets. We note, for random forest regressors, the variety of data is more important in determining the quality of the predictions compared to the amount of data beyond a certain point, which is dependent on the complexity of the systems of interest; this point is further examined for the specific systems studied herein in Appendix D (see Figure D.16–Figure D.18). That is, there may be limited utility in using large training sets when trying to develop ML models to prescreen systems outside of the design space of the original training set. However, generating a set of systems with well distributed properties can be challenging and hard to estimate a priori, and hence may require more thoughtful design of the initial screening space as well as a more active learning approach to direct the screening space as the initial data is used to train the models. The results in Figure 6.9 suggest that the ML models are likely effective in predicting frictional properties of systems with design variations, i.e., despite the noticeable decline in performance the models could likely be used as a high-level screen to sieve the parameter space. To further examine the capability of ML models in shortening the list of potential candidates to be simulated/synthesized, we examine the ability of the models to identify systems that exhibit favorable tribological performance (i.e., low COF or F_0). To quantify the ability of the model to predict favorable solutions, we calculate the intersection of the top performing systems predicted by the ML model and those via simulation; an ideally performing ML model would be able to identify all of, or a majority of, the best performing systems determined through simulation. The ability of the ML model to accurately predict the systems with the favorable properties can be considered to be proportional to the percentage of the overlapping systems compiled from MD simulation and ML prediction. An overlap of 100% indicates complete agreement between the two methods, i.e., ML and MD, while a low overlap value indicates lower agreement, and by extension, poorer predictive ability of the ML models. We note that this metric describes the ability of the ML model to accurately capture relative differences between systems of interest and does not necessarily require quantitative agreement between the ML model and corresponding MD simulations.

First, we first consider the ability of 5050-train models in determining the best performing systems in the 5050-test set in Figure 6.10a. Systems in the set are first sorted separately, by the numerical value of

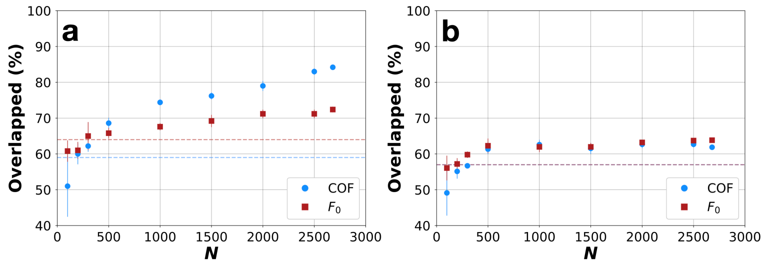


Figure 6.10: Intersection between the top 15% performing systems predicted by ML models at various training set sizes and top 15% performing systems calculated by MD simulations (*in silico* data) of the *5050–test* set (a) and the *2575–test* set (b). The systems are ranked by COF (blue circles) or F_0 (red squares). The dashed, horizontal lines, colored to correspond with its respective property, show the predictive ability of the models trained using the dataset of Summers et al. [6]

their COF and F_0 calculated from the simulations; the top 15% of these systems (i.e., systems with the lowest COF or F_0 values) of each set are considered, corresponding to 100 chemistries for 5050-test and 193 for 2575-test. These lists are then compared to the top 15% of systems predicted by the ML models, as a function of training set size, and the overlapping percentages are calculated. For COF, as the training set size of the model increases, so too does the fraction of top performing solutions predicted, achieving 74.4% ±0.8% accuracy for models trained using 1000 data points and 83.0 ±0.9% accuracy for models trained with 2500 systems; adhesion shows a weaker dependence on training set size, reaching 67.6% ±1.1% at 1000 data points and 71.2% ±1.2% at 2500 data points (see Figure 6.10a). Putting these results into perspective, if we had utilized the 5050-train model of 1000 data points to predict systems with the best performing properties from the 5050-test set and only simulated those in the top 15%, we would have reduced the total number of additional screening simulations by 85%, while still identifying 74.4% of the best performing systems as ranked by COF, or 83% of the best performing systems ranked by F_0. On the other hand, if we reduced the number of systems to be simulated at random, we would only expect to detect 15% of the best performing systems, ranked by COF or F_0. In other words, this approach can significantly increase the odds of finding systems with the top performing tribological properties. We conduct the same analysis of the ability of the ML models to identify the best performing systems in the 2575-test set, i.e., focusing on the transferability of the models, and show the results in Figure 6.10b. We observe predictions with an accuracy of roughly 61% for both the best performing systems ranked by COF or by F_0, almost independent of training set size (see Figure 6.10b). Even though these accuracies are not as high as the models used on the 5050-test set, they are still considerably higher than the 15% accuracy we would have expected if selecting systems at random. This suggests reasonable efficacy of using this approach to prescreen design space. Even for models trained with limited amounts of data, i.e., models trained with 500 data points whose accuracies are 61.3% ±1.3% for COF models and 62.3% ±2.0% for F_0, the predictions made should still be useful enough for prescreening and provide focused guidance to perform the next round of simulations, while reducing the computational costs substantially.

In Figure 6.10 we have compared an equipercentile of the top performers determined through different techniques, i.e., MD and ML. This approach, however, can potentially result in missing out in potential candidates, e.g., a model that has an accuracy of 60% in determining the true best 15% systems will miss potentially 40% of the candidates. In practice, considering a larger list of top performing systems proposed by the ML models is likely to increase the number of top performers identified in the given parameter space. This is especially relevant for a quantity such as COF, where the overall numerical range is relatively small, e.g., for 5050-test the top 15% of systems range from 0.0972 ±0.016 to 0.121 ±0.010 and the top 30% increases the upper bound very modestly to 0.129 ±0.008. For F_0 the range for the top 15% is 0.216 ±0.377 nN to 0.779 ±0.135 nN, with the upper bound increases to 1.002 ±0.171 for the top 30%. For 2575-test the values are similar where the top 15% for COF ranging from 0.090 ±0.012 to 0.122 ±0.014, with the upper bound increasing to 0.130 ±0.011 for the top 30%. For F_0, 0.085 ±0.462 nN to 0.706 ±0.410 for 15%, with the upper bound increasing to 0.951 ±0.436 when considering the top 30%. Figure 6.11 plots the overlap between the top 15% systems (MD data), ranked by their simulated tribological properties and the top 30% performing systems predicted by the 5050-train models. This set up demonstrates a significant increase in accuracy in predicting the best performing systems in both test sets. Specifically, in Figure 6.11a, the accuracy of predicting top performing systems starts at 78.0% ±7.5% and 92.4% ±2.1% (for models trained with only 100 data points) to 91.6% ±2.1% and 97.0% ±0.6% (for models trained with 1000 data points) when predicting top systems ranked by COF and F_0, respectively. In Figure 6.10b, the overlap percentages start at

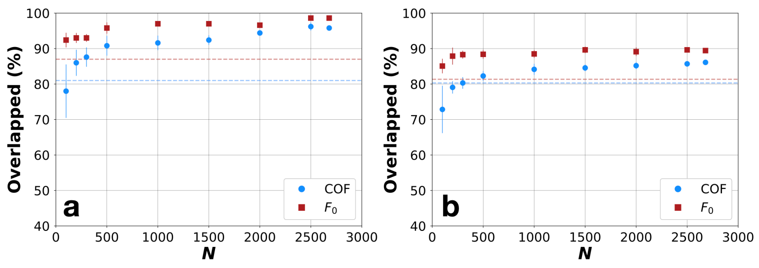


Figure 6.11: Intersection between the top 15% performing systems predicted by ML models at various training set sizes and top 30% performing systems calculated by MD simulations (*in silico* data) of the *5050–test* set (a) and the *2575–test* set (b). The systems are ranked by COF (blue circles) or F_0 (red squares). The dashed, horizontal lines, colored to correspond with its respective property, show the predictive ability of the models trained using the dataset of Summers et al. [6]

72.8% ±6.7% and 85.1% ±2.1% (for models trained with 100 data points) to 84.1% ±1.6% and 88.5% ±1.1% (for models trained with 1000 data points). In other words, by considering the top 30%, the ML models can reduce the number of additional systems to be considered by 70%, while being able to accurately predict the bulk of the best performing systems, regardless of the mixing ratio and even for very small training set sizes. These results also present other criteria to consider in terms model accuracy and computational cost in terms of using ML models to prescreen a dataspace, as these results show that it may be more efficient to train a model with fewer datapoints but consider a larger range of predictions from the ML model (e.g., simulating the top 30% predicted by the ML model). These results suggest a general approach combining ML techniques with MD simulations, namely simulating a small set of systems, e.g., about 5-10% of systems in the design space, using the simulation results to train predictive ML models, and then utilizing the ML models to screen over a wider range of potential systems, determining systems worthy of further investigation. Such an approach could drastically minimize the number of total simulations needed, decrease the throughput time to scan the parameter space, enabling higher quality candidates to be screened much faster.

Thus far we have considered models trained only using the 5050-train data. One would assume that accuracy could be improved by training the models on a data set that also included those systems in the 2575-train data set (i.e., using the total-train set). Figure 6.12 plots the scaling for R^2 (Figure 6.12a) and MAPE (Figure 6.12b) as a function of training set size when sampled from the total-train set and applied to the total-test set. As seen earlier, the rapid increase in accuracy occurs as the training set size is increased to 1000 data points; the improvements in adhesion are relatively minimal beyond that point, although considerable gains are observed for COF, with both measures attaining an R^2 value of >0.9 for a training set of size 7816 (see Figure 6.12a). We also observe similar trends for MAPE in Figure 6.12b, where the error quickly drops from 0.0593 ±0.001 (5.93% ±0.1%), for COF predictions, and 0.324 ±0.033 (32.4% ±3.3%), for F_0 predictions,

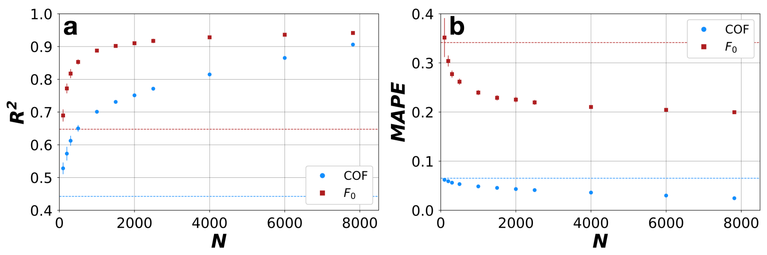


Figure 6.12: Correlation between the amount of data in the training dataset (N) and the predictive ability of the models trained using the *total–train* sets when applied to the *total–test* set to predict the COF (blue circles) and F_0 (red squares) quantified by R^2 (a) and MAPE (b). The dashed, horizontal lines, colored to correspond to its respective property in the key, show the predictive ability of the models trained using the dataset of Summers et al. [6] For each data point, the metrics (R^2/MAPE) are averaged from the metric of individual ML models (five replicates) when applied to the test set. The error bar of each point represents the standard deviation of the five ML models, each trained with different combinations of data.

at 100 data points to 0.023 ±0.0 (2.30% ±0.0%) for COF predictions, and 0.179 ±0.0 (17.9% ±0.0%) for F_0 predictions, at the maximum number of training data (7816). Focusing on the models trained with 1000 data points, predictions from the total-train set ML models applied to the total-test set achieve R^2 values of 0.701 ±0.005 for COF and 0.888 ±0.005 for F_0, which are slightly lower than the values obtained for the models trained with the 5050-train set when applied on the 5050-test set (see Figure 6.8). This may appear to indicate that this set of ML models require more data to attain a similar level of predictive ability, especially for COF models. However, it is worth noting that these evaluations are done on two test sets of differing size and composition. Moreover, the 5050-test set is a strict subset of the total-test set, so the latter includes a wider range of systems, and hence is deemed more challenging for the ML models. Thus, the relative performance of these models could not be directly compared at this point.

To evaluate the accuracy of using the total-train models for prescreening parameter space, i.e., their ability to determine best performing systems, we again calculate the agreement between the top 15% performing systems (398 total systems) from the total-test set as determined by Figure 6.13. Intersection between the (a) top 15% performing systems, ranked by COF (blue, circle) or F_0 (red, square) or combined (purple, cross), of the total-test set determined from MD simulations and predicted by ML models trained with the total-train sets. The overlapped fraction of most-favorable systems is determined by comparing those predicted by ML models against those listed in Table 6.1. The error bar of each point represents the standard deviation of the 5 ML models, each trained with different combination of data. The dashed, horizontal lines, colored to correspond with its respective property, show the predictive ability of the models trained using the Summers et al. [6] data set.

MD simulations and the top 15% and 30% top performing systems predicted by our ML models, plotted in Figure 6.13a and b. In Figure 6.13a, we see a steady improvement in the accuracy of the COF predictions

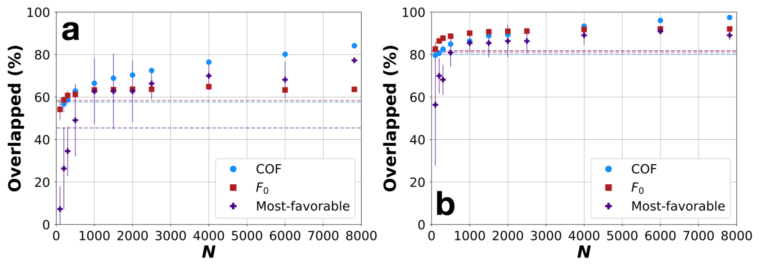


Figure 6.13: Intersection between the (a) top 15% performing systems, ranked by COF (blue circles) or F_0 (red squares) or combined (purple crosses), of the *total–test* set determined from MD simulations and predicted by ML models trained with the *total–train* sets. The overlapped fraction of most-favorable systems is determined by comparing those predicted by ML models against those listed in Table 6.1. The error bar of each point represents the standard deviation of the five ML models, each trained with different combinations of data. The dashed, horizontal lines, colored to correspond with its respective property, show the predictive ability of the models trained using the dataset of Summers et al. [6]

as nearly 8000 data points in the training set are used, achieving an overlapping of 84.2% ±0.3%; F_0, exhibits a similar trend observed previously in Figure 6.9b, with little dependence on training set size beyond 1000 points, maintaining an overlapping value of approximately 63%. Hence, increasing the number of data points to train the ML models can have a positive effect on the predictive ability for some properties, but the large amount of training data needed to improve accuracy may ultimately negate potential performance gains in terms of screening. Recall that for a training set size of 1000 data points, the 5050-train model could predict >65% of high performing 50:50 systems and >60% of high-performing 25:75 systems, ranked by either COF or F_0 when conducting similar analyses (see Figure 6.10). In Figure 6.13b, we perform similar overlap analyses with an extended list of top performing systems predicted by the ML models (top 30%). When determining top COF/F_0 systems, we observe that the overlapping fraction rapidly increases to above 80% at N=500, and subsequently plateaus, displaying minimal increase in their accuracy past this point, similar to what was previous observed in Figure 6.10. We now consider a new metric, most-favorable systems, defined as those with both low COF and F_0. This list of favorable systems is generated from the intersection of the top 15% (Figure 6.13a) or 30% (Figure 6.13b) of systems ranked by their predicted COF and F_0 values. This list is compared directly to those in Table 6.1 to determine their overlapping fraction, which indicates the ability of the ML models to identify the most-favorable systems. Using this metric, we can see that the accuracy of the prediction of these ML models as a function of training points used. In Figure 6.13a, we can see that the overlapping percentage rapidly increases until around 1000 training data points, at which point the overlapping fraction values are maintained at $\tilde{6}5\%$. Meanwhile, if we increase the intersection to 30%, we can see that the overlapping fraction can surpass 80% with as few as N=500 data points and reach as high as 90.9% when N increases. These results reassert the feasibility of the combinatorial approach described earlier, where users can use MD to generate small sets of data necessary to build a minimally functional baseline ML model to screen over a wider set of potential candidates. The results from such baseline models, can be utilized for various efforts, such as focusing on only simulating systems with the most favorable properties as predicted from the model or building more varied and evenly sampled data sets based on the predictive deficiencies from the baseline model to further improve its robustness. Either approach can improve the quality of results obtained with finite computing resources. Depending on the specific application, complexity of the systems of interest, and computing power, one can choose an optimal strategy to employ, e.g., how many data should be collected to train ML models, and how much of the dataspace to be truncated based on suggestions by the models. However, it is worth recalling that the performance of the random forest regressor algorithm is dependent on the distribution of properties in the training data, and identifying a priori which systems to simulation to ensure appropriate distribution may be challenging. Solving this issue may require additional iterations of training ML models, where we create multiple predictive models as simulation data become available, using these ML models to suggest additional systems to ensure appropriate sampling, rather than identifying favorable candidates at this stage. Although such intermediate ML models, may not have high accuracy, they can provide valuable information needed to improve the performance of the subsequent ML models (Figure 6.14).

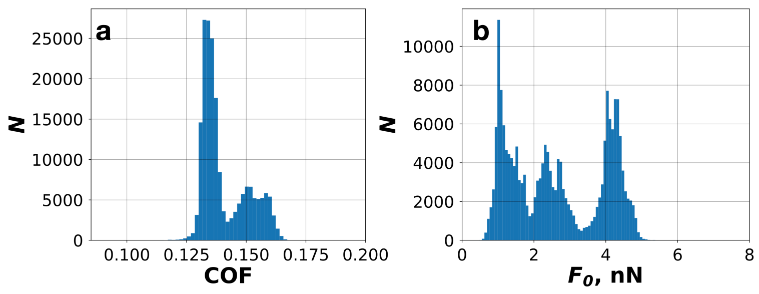


Figure 6.14: Distribution of (a) COF and (b) F_0 predicted by the ML models for 193,131 unique systems created with molecules from the `ChEMBL` small molecule library.[45]

In Table 6.2 and 3 we summarize the outcome, R^2 and MAPE, of applying all of the ML models, i.e., Summers et al., 5050-train, and total-train models, on all available test sets, i.e., 5050-test, 2575-test, and total-test; summary of other common metrics, i.e., mean absolute error (MAE) and mean squared error (MSE) is also included in Appendix D (see Table D.20 and Table D.21) [75]. Table 6.2 directly compares of the performance of the different ML models.

We note, of all models trained with 100 data points, the performance of the total-train models exhibits the highest accuracy, followed by the 5050-train, and finally models trained by Summers et al. data. The difference in performance likely results from the inclusion of mixed-monolayer systems, resulting in better

Table 6.1: 22 most-favorable systems determined by the intersection of the top 500 systems ranked by their COF and the top 500 systems ranked by their F_0. The COF and F_0 mean values and standard deviation (std.) are calculated from the three replicates.

	Terminal group A	Terminal group B	Terminal group C	A fraction	B fraction	COF[1] - mean	COF[1] - std.	F_0[2] (nN) – mean	F_0[2] (nN) std.
0	Cyano	Cyano	Isopropyl	0.5	0.5	0.1032	0.0063	0.4768	0.1075
1	Cyclopropyl	Ethylene	Cyano	0.5	0.5	0.1086	0.0142	0.4498	0.2820
2	Difluoromethyl	Isopropyl	Cyano	0.5	0.5	0.1100	0.0098	0.5292	0.4261
3	Difluoromethyl	Methyl	Cyano	0.5	0.5	0.1128	0.0036	0.4640	0.1585
4	Ethylene	Isopropyl	Cyano	0.5	0.5	0.1109	0.0108	0.2161	0.3768
5	Ethylene	Perfluoromethyl	Cyano	0.5	0.5	0.1093	0.0119	0.3297	0.3124
6	Isopropyl	Methoxy	Cyano	0.5	0.5	0.1113	0.0144	0.5313	0.5019
7	Isopropyl	Methyl	Cyano	0.5	0.5	0.1121	0.0182	0.4041	0.3252
8	Isopropyl	Perfluoromethyl	Cyano	0.5	0.5	0.1030	0.0181	0.2978	0.1960
9	Difluoromethyl	Carboxyl	Isopropyl	0.25	0.75	0.1117	0.0021	0.1944	0.8319
10	Difluoromethyl	Isopropyl	Carboxyl	0.25	0.75	0.1105	0.0078	0.5375	0.7366
11	Difluoromethyl	Methyl	Cyano	0.25	0.75	0.1126	0.0041	0.4967	0.0876
12	Ethylene	Isopropyl	Cyano	0.25	0.75	0.1046	0.0184	0.5122	0.1374
13	Isopropyl	Cyclopropyl	Cyano	0.25	0.75	0.0898	0.0119	0.4354	0.1522
14	Isopropyl	Perfluoromethyl	Cyano	0.25	0.75	0.1077	0.0073	0.3868	0.1580
15	Methoxy	Cyano	Ethylene	0.25	0.75	0.1086	0.0053	0.4346	0.4340
16	Methyl	Isopropyl	Cyano	0.25	0.75	0.0942	0.0148	0.4181	0.2017
17	Perfluoromethyl	Cyclopropyl	Cyano	0.25	0.75	0.1138	0.0070	0.3104	0.1799
18	Perfluoromethyl	Ethylene	Cyano	0.25	0.75	0.1067	0.0002	0.5315	0.1965
19	Perfluoromethyl	Isopropyl	Cyano	0.25	0.75	0.0947	0.0114	0.5366	0.2037
20	Phenyl	Isopropyl	Cyano	0.25	0.75	0.1098	0.0156	0.4825	0.3196
21	Toluene	Ethylene	Cyano	0.25	0.75	0.1119	0.0134	0.4907	0.6589

[1] Coefficient of friction.
[2] Adhesion force.

Table 6.2: Summary of the performance of each ML models when predicting COF and F_0 for all different test sets, measured by R^2. For each data point, the R^2 value is averaged from R^2 of individual ML models (five replicates) when applied to the test set.

		5050-test		*2575-test*		*Total-test*	
	N	COF[1]	F_0[2]	COF[1]	F_0[2]	COF[1]	F_0[2]
Summers *et al.* models	100	0.4720	0.6570	0.4290	0.6430	0.4430	0.6480
5050–train models	100	0.543 ± 0.063	0.722 ± 0.058	0.475 ± 0.041	0.692 ± 0.031	0.496 ± 0.046	0.701 ± 0.032
	200	0.647 ± 0.012	0.792 ± 0.033	0.557 ± 0.008	0.742 ± 0.024	0.584 ± 0.008	0.757 ± 0.026
	300	0.687 ± 0.017	0.826 ± 0.011	0.584 ± 0.011	0.774 ± 0.007	0.615 ± 0.013	0.790 ± 0.007
	500	0.734 ± 0.004	0.859 ± 0.008	0.618 ± 0.005	0.819 ± 0.002	0.653 ± 0.004	0.831 ± 0.004
	1000	0.799 ± 0.007	0.885 ± 0.010	0.647 ± 0.001	0.848 ± 0.007	0.693 ± 0.002	0.859 ± 0.005
	1500	0.835 ± 0.011	0.907 ± 0.007	0.652 ± 0.004	0.867 ± 0.006	0.706 ± 0.005	0.879 ± 0.003
	2000	0.880 ± 0.004	0.920 ± 0.002	0.656 ± 0.007	0.875 ± 0.004	0.723 ± 0.006	0.888 ± 0.003
	2500	0.915 ± 0.003	0.927 ± 0.001	0.664 ± 0.003	0.881 ± 0.001	0.739 ± 0.002	0.895 ± 0.001
	2680	0.926	0.928	0.668	0.882	0.745	0.896
Total–train models	100	0.567 ± 0.019	0.701 ± 0.027	0.511 ± 0.019	0.684 ± 0.023	0.528 ± 0.018	0.690 ± 0.019
	200	0.614 ± 0.032	0.792 ± 0.008	0.555 ± 0.020	0.763 ± 0.020	0.573 ± 0.021	0.772 ± 0.015
	300	0.652 ± 0.021	0.818 ± 0.017	0.595 ± 0.017	0.817 ± 0.014	0.612 ± 0.015	0.817 ± 0.014
	500	0.693 ± 0.020	0.849 ± 0.008	0.631 ± 0.011	0.855 ± 0.011	0.65 ± 0.01	0.853 ± 0.008
	1000	0.738 ± 0.007	0.883 ± 0.007	0.684 ± 0.007	0.890 ± 0.006	0.701 ± 0.005	0.888 ± 0.005
	1500	0.768 ± 0.006	0.897 ± 0.007	0.715 ± 0.005	0.904 ± 0.003	0.731 ± 0.004	0.902 ± 0.004
	2000	0.784 ± 0.006	0.908 ± 0.005	0.736 ± 0.003	0.911 ± 0.006	0.751 ± 0.003	0.910 ± 0.006
	2500	0.807 ± 0.006	0.915 ± 0.009	0.756 ± 0.004	0.918 ± 0.006	0.771 ± 0.003	0.917 ± 0.007
	4000	0.851 ± 0.006	0.928 ± 0.006	0.799 ± 0.004	0.928 ± 0.004	0.815 ± 0.003	0.928 ± 0.004
	6000	0.894 ± 0.003	0.937 ± 0.003	0.853 ± 0.005	0.936 ± 0.001	0.865 ± 0.004	0.936 ± 0.001
	7816	0.925	0.943	0.898	0.941	0.906	0.942

[1] R^2 when predicting the coefficient of friction.
[2] R^2 when predicting adhesion force.

Table 6.3: Summary of the performance of each ML models when predicting COF and F_0 for all different test sets, measured by MAPE. For each data point, the MAPE value is averaged from the MAPE of individual ML models (five replicates) when applied to the test set.

		5050-test		*2575-test*		*Total-test*	
	N	COF[1]	F_0[2]	COF[1]	F_0[2]	COF[1]	F_0[2]
Models of Summers et al.	100	0.0565	0.266	0.0653	0.341	0.0623	0.315
5050-train models	100	0.0546 ± 0.0030	0.259 ± 0.016	0.0643 ± 0.0020	0.343 ± 0.018	0.061 ± 0.003	0.315 ± 0.016
	200	0.0478 ± 0.0010	0.224 ± 0.011	0.0589 ± 0.0010	0.312 ± 0.009	0.0551 ± 0.0010	0.282 ± 0.010
	300	0.0447 ± 0.0010	0.212 ± 0.007	0.0568 ± 0.0010	0.295 ± 0.011	0.0527 ± 0.0010	0.267 ± 0.009
	500	0.0407 ± 0.0010	0.193 ± 0.004	0.0542	0.275 ± 0.005	0.0496	0.247 ± 0.004
	1000	0.0339 ± 0.0010	0.175 ± 0.003	0.0521	0.260 ± 0.003	0.0459	0.231 ± 0.002
	1500	0.0301 ± 0.0010	0.163 ± 0.001	0.052	0.253 ± 0.004	0.0445 ± 0.0010	0.222 ± 0.002
	2000	0.0251	0.155 ± 0.002	0.0513 ± 0.0010	0.248 ± 0.002	0.0423	0.217 ± 0.001
	2500	0.0209	0.152 ± 0.002	0.0508	0.246 ± 0.001	0.0406	0.213 ± 0.001
	2680	0.0196	0.15	0.0505	0.245	0.0399	0.213
Total-train models	100	0.0539 ± 0.0012	0.270 ± 0.022	0.0621 ± 0.0010	0.352 ± 0.040	0.0593 ± 0.0008	0.324 ± 0.033
	200	0.0504 ± 0.0025	0.228 ± 0.011	0.0592 ± 0.0010	0.304 ± 0.011	0.0562 ± 0.0016	0.278 ± 0.011
	300	0.0478 ± 0.0018	0.214 ± 0.005	0.0562 ± 0.0010	0.277 ± 0.007	0.0533 ± 0.0011	0.256 ± 0.006
	500	0.0447 ± 0.0017	0.197 ± 0.006	0.0533 ± 0.0010	0.262 ± 0.006	0.0504 ± 0.0009	0.240 ± 0.006
	1000	0.0408 ± 0.0005	0.178 ± 0.002	0.0486	0.240 ± 0.005	0.0460 ± 0.0004	0.219 ± 0.003
	1500	0.0379 ± 0.0007	0.167 ± 0.003	0.0456	0.229 ± 0.005	0.0430 ± 0.0004	0.208 ± 0.005
	2000	0.0359 ± 0.0009	0.162 ± 0.003	0.0433	0.225 ± 0.006	0.0407 ± 0.0004	0.204 ± 0.005
	2500	0.0336 ± 0.0005	0.157 ± 0.004	0.0411	0.220 ± 0.005	0.0385 ± 0.0002	0.198 ± 0.005
	4000	0.0287 ± 0.0006	0.148 ± 0.002	0.0361	0.210 ± 0.004	0.0336 ± 0.0003	0.189 ± 0.003
	6000	0.0236 ± 0.0004	0.145 ± 0.002	0.0300 ± 0.0005	0.204 ± 0.002	0.0278 ± 0.0004	0.184 ± 0.002
	7816	0.0198	0.14	0.0246 ± 0.0001	0.2	0.023	0.179

[1] R^2 when predicting the coefficient of friction.
[2] R^2 when predicting adhesion force.

distributed training sets provided by total-train. However, this trend does not persist for models trained with more data. Interestingly, we note that the models trained on the 5050-train data appear to have better performance compared to the total-train models when predicting the 5050-test set, since they require less data to acquire similar predictive ability. Focusing on the performance of the different models on the total-test set, which is expected to be more difficult to predict, the 5050-train models show comparable accuracy to the total-train models. These observations trends are also confirmed by the MAPE values in Table 6.3. This result suggests that as long as the models are trained on a sufficiently large data set that captures the distribution of the population well, the models are extensible to untested regimes. That is, for significant variations, in our case composition, it may be more computationally efficient and accurate to train models with different compositions separately, rather than aggregating data into a large training set. Furthermore, all models trained on a relatively small amount of data, e.g., 1000 data points, exhibit similar performance across all test sets. This reaffirms the transferability of these ML models from a more general perspective and highlights the feasibility of utilizing ML algorithms to estimate properties of systems whose designs may be dissimilar to the training data used. Expectedly, the ML models trained with 7816 data points from the total-train data set (80% of the total data set) demonstrate the best performance across all test sets, but of course, require the highest computational in terms of gathering training data.

As a proof-of-concept of using ML to pre-screen the design space, we perform a screening study using one of the total-train models (model-0) for COF and F_0 (trained with 7816 data points each). The chemical space for this screening was constructed by querying the ChEMBL small molecules library,36,37 and identifying chemistries whose molecular weight ranges from 4 to 99 amu; this list of chemistries (981) underwent further filtering to remove those containing metallic elements and those that cannot be processed by the `RDKit` library [71], e.g., chiral or charged molecules, resulting in 621 unique terminal group chemistries (provided in the Supplemental Repository [47]). With these 621 chemistries, 193,131 unique systems can be created in which each monolayer is homogeneous (i.e., containing only one type of terminal group); mixed monolayer chemistries were not considered at the moment due to the vast amount of data that would be generated, a dynamically pruning approach to help drive the screening towards a smaller subset of the mixed monolayer systems is necessary to explore this space in any feasible time and memory requirements. This simple system design (dual homogeneous monolayers) was chosen to allow more unique chemistries to be considered in a reasonable time frame, since introduction of mixed monolayers would scale up the number of systems to be considered by several orders of magnitude. Descriptors for the 621 terminal groups were determined using the SMILES strings for each chemistry (as described in the Method section); these descriptors were then provided as input to the ML models which in turn predicted tribological properties for the 193,131 unique systems. This screening process, which evaluated 385,641 systems since duplicate systems (i.e., systems in which chemistry A was the top monolayer and chemistry B on the bottom monolayer and vice versa) were not removed, took approximately 24 hours to predict the COF and F_0 values on a standard desktop computer ($\tilde{0}$.22 seconds per system), which is orders of magnitudes faster than the time required to perform a single MD simulation and without the need for expansive computational resources.

The distribution of COF and F_0 of the systems predicted by the ML model are shown in Figure 6.14. We note the distributions differ from that of the data set screened using MD simulations (see Figure 6.3 and Figure 6.7), which is expected given the vastly expanded chemical design space. Using the first quartile of the COF distribution (0.1280) and F_0 distribution (0.8966 nN) obtained from MD as a reference, the data set contains 5121 systems that can be considered to have good COF values and 10,598 systems with good F_0 values. To further reduce the number of systems of interest, a list of 2000 best performing systems ranked by their COF values and a list of the 2000 best systems ranked by their F_0 are compiled, with the top 20 systems at the intersection of these lists reported in Table 6.3. We note that many of the same chemistries that were identified by our initial MD simulations (see Table 6.1) are also observed in this list; specifically, Systems 2, 3, and 16 have been considered in Summer et al.8; along with several other chemistries that may be worth future consideration, such as various alkenes (allyl, propene), alkynes (acetylene, but-2-yne), halocarbons (1,1-difluoroethyl, bromoethyl, vinyl chloride), and nitriles (cyano, malononitrile, acrylonitrile). We note that none of the systems reported in Table 6.3 outperform those previously identified in Table 6.1 from the MD simulations (in terms of COF and F_0), although this might be expected since the systems in Table 6.4 consist of 2 homogeneous monolayers, and hence, do not include the benefits offered by the mixed monolayers, as we discussed earlier. Nonetheless, this highlights the feasibility of combining ML with MD screening to reduce computational cost and identify favorable candidates for further study, in particular for reducing the vast design space of mixed monolayer systems using such a database for screening.

Table 6.4: 20 best performing systems determined by the intersection of the top 2000 systems ranked by their COF and the top 2000 systems ranked by their F_0 . The properties were predicted using one of the total-train models (model-0).

	Terminal group A	Terminal group B	COF	F_0 (nN)
1	Cyano	Propyl	0.1144	0.7257
2	Cyano	Cyclopropyl	0.1151	0.4631
3	Methyl	Cyano	0.1153	0.5532
4	Acetylene	1,1-difluoroethyl	0.1180	0.7699
5	Cyano	Ethyl	0.1206	0.6490
6	Fulminic acid	Cyclopropyl	0.1236	0.7117
7	Ethylene	1,1-difluoroethyl	0.1244	0.7341
8	Bromoethyl	1,2-diformylhydrazine	0.1250	0.7695
9	Methyl	Fulminic acid	0.1260	0.7704
10	Cyano	Difluoroethyl	0.1265	0.7279
11	Bromoethyl	Malononitrile	0.1269	0.7098
12	Acetylene	Ethyl	0.1270	0.7254
13	1,1-difluoroethane	Propene	0.1271	0.7290
14	Propyl	2,2-difluoroacetamide	0.1280	0.7423
15	Acetylene	Propyl	0.1281	0.7777
16	Methyl	Acetylene	0.1281	0.7405
17	Bromoethyl	1,2-dicyanoethyl	0.1282	0.7737
18	Fulminic acid	Ethyl	0.1283	0.7725
19	Cyclopropyl	Acrylonitrile	0.1283	0.7152
20	Allyl	But-2-yne	0.1283	0.7546

6.4 Conclusion

Utilizing the MoSDeF software suite with the Signac Framework [20, 22, 23], a workflow to initialize, parametrize, convert to MD engine inputs, perform MD simulations, calculate the tribological properties of interest, and then train a predictive ML model for these soft matter monolayer systems was developed and extended from our previous smaller scale study [6]. The tribological properties of nearly 10,000 unique system designs have been screened. From the MD simulations, we identified systems that exhibited both low COF and F_0. The bulk of these systems consisted of a cyano group monolayer contacting a heterogeneous monolayer, suggesting mixed monolayer systems may provide a viable route for further tuning tribological performance. However, we note that no clear trends were observed for different mixing ratios in the monolayer, suggesting that their effects strongly depend on the chemistries involved. Although using high-throughput screening we were able to much more readily determine systems with favorable properties than could be accomplished through experiment, the process still requires a significant amount of time and computing resources. However, coupling MD simulations with machine learning can guide the screening process and reduce the simulations needed in order to optimize system designs. Using this approach, we have assessed the dependence of the random forest regressor's predictive ability based on the training set provided. The results suggest a positive correlation between the performance of machine learning models with the size of the training set, along with factors such as the distribution of the data; however, performance typically plateaus once a modest data set size is reached. For the type of systems considered in this study, a training set of 1000 data points is found to be sufficient to train an efficient predictive model. We note that at small training set sizes, i.e., fewer than 500 data points, the machine learning models were still quite successful in determining the best and worst performing systems. Moreover, the models were shown to have high transferability when applied to predict properties of dissimilar systems and that improvements in accuracy seen for larger training sets often do not necessarily equate to improved performance when models are transferred to systems outside of the training set. These findings suggest a synergistic approach of using MD simulations and machine learning to build high quality predictive models and minimize computing resources needed: MD can be used to generate a small set of data to train baseline ML models, which can then be utilized to quickly evaluate possible candidates and narrow the parameter space. The performance of the baseline model is dependent on the distribution of training data provided; however, the accuracy of the model can be improved via a few iterations of training, using earlier, less accurate, models to guide simulations toward creating well distributed training data. In addition, the baseline model could help confirm/provide insight about the connection between chemical intuition with properties of interest. We also note, that care must be taken to ensure that the data set is not overtly biased and that the further trained models are not overfit to the provided data. This work follows guidelines suggested by the TRUE standard, emphasizing the reproducibility and extensibility of the study; accordingly, the Supplementary Information contains all the information needed to reproduce the simulations and machine learning models described in this work. The code and data are distributed via GitHub.

References

1. Bhushan, B. Nanotribology and Nanomechanics of MEMS/NEMS and BioMEMS/ BioNEMS Materials and Devices. *Microelectronic Engineering* **84,** 387–412. ISSN: 0167-9317 (2007).
2. Song, Y., Premachandran Nair, R., Zou, M. & Wang, Y. Adhesion and Friction Properties of Micro/Nano-Engineered Superhydrophobic/Hydrophobic Surfaces. *Thin Solid Films* **518,** 3801–3807. ISSN: 0040-6090 (May 2010).
3. Ulman, A. *An Introduction to Ultrathin Organic Films: From Langmuir-Blodgett to Self-Assembly* ISBN: 978-0-08-092631-5 ().
4. Bîrleanu, C., Pustan, M., Șerdean, F. & Merie, V. AFM Nanotribomechanical Characterization of Thin Films for MEMS Applications. *Micromachines* **13,** 23. ISSN: 2072-666X (Dec. 2021).
5. OSTP. *Materials Genome Initiative for Global Competitiveness* tech. rep. (2011).
6. Summers, A. Z., Gilmer, J. B., Iacovella, C. R., Cummings, P. T. & McCabe, C. MoSDeF, a Python Framework Enabling Large-Scale Computational Screening of Soft Matter: Application to Chemistry-

Property Relationships in Lubricating Monolayer Films. *Journal of Chemical Theory and Computation* **0.** PMID: 32004433, null. ISSN: 1549-9618. eprint: https://doi.org/10.1021/acs.jctc.9b01183 (Mar. 0).

7. Quach, C. D., Gilmer, J. B., Pert, D. O., Mason-Hogans, A., Iacovella, C. R., Cummings, P. T. & McCabe, C. High-Throughput Screening of Tribological Properties of Monolayer Films Using Molecular Dynamics and Machine Learning. *The Journal of Chemical Physics,* 5.0080838. ISSN: 0021-9606, 1089-7690 (Feb. 2022).

8. Thompson, M. W., Gilmer, J. B., Matsumoto, R. A., Quach, C. D., Shamaprasad, P., Yang, A. H., Iacovella, C. R., McCabe, C. & Cummings, P. T. Towards Molecular Simulations That Are Transparent, Reproducible, Usable by Others, and Extensible (TRUE). *Molecular Physics* **118,** e1742938. ISSN: 0026-8976, 1362-3028 (June 2020).

9. Bhushan, B. & Sundararajan, S. Micro/Nanoscale Friction and Wear Mechanisms of Thin Films Using Atomic Force and Friction Force Microscopy. *Acta Mater.* **46,** 3793 (1998).

10. Bhushan, B., Kasai, T., Kulik, G., Barbieri, L. & Hoffmann, P. AFM Study of Perfluoroalkylsilane and Alkylsilane Self-Assembled Monolayers for Anti-Stiction in MEMS/NEMS. *Ultramicroscopy* **105,** 176–188. ISSN: 0304-3991 (2005).

11. Vilt, S. G., Leng, Z., Booth, B. D., McCabe, C. & Jennings, G. K. Surface and Frictional Properties of Two-Component Alkylsilane Monolayers and Hydroxyl-Terminated Monolayers on Silicon. *The Journal of Physical Chemistry C* **113,** 14972–14977. ISSN: 1932-7447. eprint: https://doi.org/10.1021/jp904809h (Aug. 2009).

12. Yu, B., Qian, L., Yu, J. & Zhou, Z. Effects of Tail Group and Chain Length on the Tribological Behaviors of Self-Assembled Dual-Layer Films in Atmosphere and in Vacuum. *Tribology Letters* **34,** 1–10. ISSN: 1023-8883, 1573-2711 (Apr. 2009).

13. Brewer, N. J., Beake, B. D. & Leggett, G. J. Friction Force Microscopy of Self-Assembled Monolayers: Influence of Adsorbate Alkyl Chain Length, Terminal Group Chemistry, and Scan Velocity. *Langmuir* **17,** 1970–1974. ISSN: 0743-7463 (Mar. 2001).

14. Le, T., Epa, V. C., Burden, F. R. & Winkler, D. A. Quantitative Structure-Property Relationship Modeling of Diverse Materials Properties. *Chemical Reviews* **112,** 2889–2919. ISSN: 0009-2665 (May 2012).

15. Bereau, T. Computational Compound Screening of Biomolecules and Soft Materials by Molecular Simulations. *Modelling and Simulation in Materials Science and Engineering* **29,** 023001. ISSN: 0965-0393, 1361-651X (Mar. 2021).

16. Rivera, J. L., Jennings, G. K. & McCabe, C. Examining the Frictional Forces between Mixed Hydrophobic Hydrophilic Alkylsilane Monolayers. *The Journal of Chemical Physics* **136,** 244701. ISSN: 0021-9606, 1089-7690 (June 2012).

17. Mazyar, O. A., Jennings, G. K. & McCabe, C. Frictional Dynamics of Alkylsilane Monolayers on SiO $_2$: Effect of 1- *n* - Butyl-3-methylimidazolium Nitrate as a Lubricant. *Langmuir* **25,** 5103–5110. ISSN: 0743-7463, 1520-5827 (May 2009).

18. Lewis, J. B., Vilt, S. G., Rivera, J. L., Jennings, G. K. & McCabe, C. Frictional Properties of Mixed Fluorocarbon/Hydrocarbon Silane Monolayers: A Simulation Study. *Langmuir* **28,** 14218–14226. ISSN: 0743-7463, 1520-5827 (Oct. 2012).

19. Contributors, M. *MoSDeF Webpage* https://mosdef.org. Accessed: 2022-03-13.

20. Dice, B. D., Butler, B. L., Ramasubramani, V., Travitz, A., Henry, M. M., Ojha, H., Wang, K. L., Adorf, C. S., Jankowski, E. & Glotzer, S. C. *Signac: Data Management and Workflows for Computational Researchers* in *Proceedings of the 20th Python in Science Conference* (2021), 23–32.

21. Ramasubramani, V., Adorf, C. S., Dodd, P. M., Dice, B. D. & Glotzer, S. C. *Signac: A Python Framework for Data and Workflow Management* in *Proceedings of the 17th Python in Science Conference* (eds Akici, F., Lippa, D., Niederhut, D. & Pacer, M.) (2018), 152–159.

22. Contributors, S. *Signac Framework Webpage* https://signac.io. Accessed: 2022-03-13.

23. Adorf, C. S., Dodd, P. M., Ramasubramani, V. & Glotzer, S. C. Simple data and workflow management with the signac framework. *Computational Materials Science* **146,** 220–229. ISSN: 0927-0256. https://authors.elsevier.com/a/1WbPL3In-uhIuD%20http://linkinghub.elsevier.com/retrieve/pii/S0927025618300429%20https://www.mendeley.com/research-papers/simple-data-workflow-management-signac-framework/ (Feb. 2018).

24. Thompson, M. W., Matsumoto, R., Sacci, R. L., Sanders, N. C. & Cummings, P. T. Scalable Screening of Soft Matter: A Case Study of Mixtures of Ionic Liquids and Organic Solvents. *The Journal of Physical Chemistry B* **123,** 1340–1347. ISSN: 1520-6106 (Feb. 2019).

25. Cummings, P. T., McCabe, C., Iacovella, C. R., Ledeczi, A., Jankowski, E., Jayaraman, A., Palmer, J. C., Maginn, E. J., Glotzer, S. C., Anderson, J. A., Ilja Siepmann, J., Potoff, J., Matsumoto, R. A., Gilmer, J. B., DeFever, R. S., Singh, R. & Crawford, B. Open-Source Molecular Modeling Software in Chemical Engineering Focusing on the Molecular Simulation Design Framework. *AIChE Journal* **67,** e17206. eprint: https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.17206 (2021).

26. Faber, F. A., Hutchison, L., Huang, B., Gilmer, J., Schoenholz, S. S., Dahl, G. E., Vinyals, O., Kearnes, S., Riley, P. F. & von Lilienfeld, O. A. Prediction Errors of Molecular Machine Learning Models Lower than Hybrid DFT Error. *Journal of Chemical Theory and Computation* **13,** 5255–5264. ISSN: 1549-9618, 1549-9626 (Nov. 2017).

27. Smith, J. S., Isayev, O. & Roitberg, A. E. ANI-1: An Extensible Neural Network Potential with DFT Accuracy at Force Field Computational Cost. *Chemical Science* **8,** 3192–3203. ISSN: 2041-6520, 2041-6539 (2017).

28. Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P. & Hassabis, D. Highly Accurate Protein Structure Prediction with AlphaFold. *Nature* **596,** 583–589. ISSN: 1476-4687 (Aug. 2021).

29. Doerr, S., Majewski, M., Pérez, A., Kr ämer, A., Clementi, C., Noe, F., Giorgino, T. & De Fabritiis, G. TorchMD: A Deep Learning Framework for Molecular Simulations. *Journal of Chemical Theory and Computation* **17,** 2355–2363. ISSN: 1549-9618, 1549-9626 (Apr. 2021).

30. Smith, J. S., Nebgen, B., Lubbers, N., Isayev, O. & Roitberg, A. E. Less Is More: Sampling Chemical Space with Active Learning. *The Journal of Chemical Physics* **148,** 241733. ISSN: 0021-9606, 1089-7690 (June 2018).

31. Tian, H., Trozzi, F., Zoltowski, B. D. & Tao, P. Deciphering the Allosteric Process of the Phaeodactylum Tricornutum Aureochrome 1a LOV Domain. *The Journal of Physical Chemistry B* **124,** 8960–8972. ISSN: 1520-6106 (Oct. 2020).

32. Wang, F., Shen, L., Zhou, H., Wang, S., Wang, X. & Tao, P. Machine Learning Classification Model for Functional Binding Modes of TEM-1 β-Lactamase. *Frontiers in Molecular Biosciences* **6,** 47. ISSN: 2296-889X (July 2019).

33. Shmilovich, K., Mansbach, R. A., Sidky, H., Dunne, O. E., Panda, S. S., Tovar, J. D. & Ferguson, A. L. Discovery of Self-Assembling π-Conjugated Peptides by Active Learning-Directed Coarse-Grained Molecular Simulation. *The Journal of Physical Chemistry B* **124,** 3873–3891. ISSN: 1520-6106, 1520-5207 (May 2020).

34. Kelkar, A. S., Dallin, B. C. & Van Lehn, R. C. Identifying Nonadditive Contributions to the Hydrophobicity of Chemically Heterogeneous Surfaces via Dual-Loop Active Learning. *The Journal of Chemical Physics* **156,** 024701. ISSN: 0021-9606 (Jan. 2022).

35. Janet, J. P., Ramesh, S., Duan, C. & Kulik, H. J. Accurate Multiobjective Design in a Space of Millions of Transition Metal Complexes with Neural-Network-Driven Efficient Global Optimization. *ACS Central Science* **6,** 513–524. ISSN: 2374-7943, 2374-7951 (Apr. 2020).

36. Gowers, R. J., Farmahini, A. H., Friedrich, D. & Sarkisov, L. Automated Analysis and Benchmarking of GCMC Simulation Programs in Application to Gas Adsorption. *Molecular Simulation* **44,** 309–321. ISSN: 0892-7022, 1029-0435 (Mar. 2018).

37. Jablonka, K. M., Jothiappan, G. M., Wang, S., Smit, B. & Yoo, B. Bias Free Multiobjective Active Learning for Materials Design and Discovery. *Nature Communications* **12,** 2312. ISSN: 2041-1723 (Dec. 2021).

38. Afzal, M. A. F., Haghighatlari, M., Ganesh, S. P., Cheng, C. & Hachmann, J. Accelerated Discovery of High-Refractive-Index Polyimides via First-Principles Molecular Modeling, Virtual High-Throughput Screening , and Data Mining. *The Journal of Physical Chemistry C* **123,** 14610–14618. ISSN: 1932-7447 (June 2019).

39. Kuenneth, C., Schertzer, W. & Ramprasad, R. Copolymer Informatics with Multitask Deep Neural Networks. *Macromolecules* **54,** 5957–5961. ISSN: 0024-9297, 1520-5835 (July 2021).

40. Kim, C., Batra, R., Chen, L., Tran, H. & Ramprasad, R. Polymer Design Using Genetic Algorithm and Machine Learning. *Computational Materials Science* **186,** 110067. ISSN: 0927-0256 (Jan. 2021).

41. Gormley, A. J. & Webb, M. A. Machine Learning in Combinatorial Polymer Chemistry. *Nature Reviews Materials* **6,** 642–644. ISSN: 2058-8437 (Aug. 2021).

42. Statt, A., Kleeblatt, D. C. & Reinhart, W. F. Unsupervised Learning of Sequence-Specific Aggregation Behavior for a Model Copolymer. *Soft Matter* **17,** 7697–7707. ISSN: 1744-6848 (Aug. 2021).

43. Webb, M. A., Jackson, N. E., Gil, P. S. & de Pablo, J. J. Targeted Sequence Design within the Coarse-Grained Polymer Genome. *Science Advances* **6,** eabc6216. ISSN: 2375-2548 (Oct. 2020).

44. Gaulton, A., Hersey, A., Nowotka, M., Bento, A. P., Chambers, J., Mendez, D., Mutowo, P., Atkinson, F., Bellis, L. J., Cibrián-Uhalte, E., Davies, M., Dedman, N., Karlsson, A., Magariños, M. P., Overington, J. P., Papadatos, G., Smit, I. & Leach, A. R. The ChEMBL Database in 2017. *Nucleic Acids Research* **45,** D945–D954. ISSN: 1362-4962 (Jan. 2017).

45. Davies, M., Nowotka, M., Papadatos, G., Dedman, N., Gaulton, A., Atkinson, F., Bellis, L. & Overington, J. P. ChEMBL Web Services: Streamlining Access to Drug Discovery Data and Utilities. *Nucleic Acids Research* **43,** W612–620. ISSN: 1362-4962 (July 2015).

46. Summers, A. Z., Iacovella, C. R., Cummings, P. T. & McCabe, C. Investigating Alkylsilane Monolayer Tribology at a Single-Asperity Contact with Molecular Dynamics Simulation. *Langmuir* **33,** 11270–11280. ISSN: 0743-7463 (Oct. 2017).

47. Quach, C. D. & Gilmer, J. B. *High-Throughput Screening of Tribological Properties of Monolayer Films Using Molecular Dynamics and Machine Learning: Supplemental Repository* https://github.com/daico007/iMoDELS-supplements/tree/6054ccd91b19575c8d5ccbd5a672b9b55664f0c8. [Online; accessed 2022-03-16]. 2022.

48. Black, J. E., Iacovella, C. R., Cummings, P. T. & McCabe, C. Molecular Dynamics Study of Alkylsilane Monolayers on Realistic Amorphous Silica Surfaces. *Langmuir* **31,** 3086–3093. ISSN: 0743-7463, 1520-5827 (Mar. 2015).

49. Chandross, M., Grest, G. S. & Stevens, M. J. Friction between Alkylsilane Monolayers: Molecular Simulation of Ordered Monolayers. *Langmuir* **18,** 8392–8399. ISSN: 0743-7463, 1520-5827 (Oct. 2002).

50. Mikulski, P. T. & Harrison, J. A. Packing-Density Effects on the Friction of *n* - Alkane Monolayers. *Journal of the American Chemical Society* **123,** 6873–6881. ISSN: 0002-7863, 1520-5126 (July 2001).

51. Kojio, K., Ge, S., Takahara, A. & Kajiyama, T. Molecular Aggregation State of N-Octadecyltrichlorosilane Monolayer Prepared at an Air/Water Interface. *Langmuir* **14,** 971–974. ISSN: 0743-7463 (Mar. 1998).

52. Contributors, M. *Foyer Webpage* https://foyer.mosdef.org. Accessed: 2022-03-13.

53. Contributors, M. *mBuild Webpage* https://mbuild.mosdef.org. Accessed: 2022-03-13.

54. *mBuild Github Repository*

55. Contributors, M. *Foyer Github Repository* https://github.com/mosdef-hub/foyer.git. Accessed: 2022-03-13.

56. Klein, C., Sallai, J., Jones, T. J., Iacovella, C. R., McCabe, C. & Cummings, P. T. in *Foundations of Molecular Modeling and Simulation: Select Papers from FOMMS 2015* (eds Snurr, R. Q., Adjiman, C. S. & Kofke, D. A.) 79–92 (Springer Singapore, Singapore, 2016). ISBN: 978-981-10-1128-3.

57. Klein, C., Summers, A. Z., Thompson, M. W., Gilmer, J. B., McCabe, C., Cummings, P. T., Sallai, J. & Iacovella, C. R. Formalizing Atom-Typing and the Dissemination of Force Fields with Foyer. *Computational Materials Science* **167,** 215–227. ISSN: 0927-0256. http://www.sciencedirect.com/science/article/pii/S0927025619303040 (Sept. 2019).

58. Jorgensen, W. L., Maxwell, D. S. & Tirado-Rives, J. Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids. *J. Am. Chem. Soc.* **118,** 11225–11236. ISSN: 0002-7863 (Nov. 1996).

59. Berendsen, H. J. C., van der Spoel, D. & van Drunen, R. GROMACS: A message-passing parallel molecular dynamics implementation. *Comput. Phys. Commun.* **91,** 43–56. ISSN: 0010-4655 (Sept. 1995).

60. Van Der Spoel, D., Lindahl, E., Hess, B., Groenhof, G., Mark, A. E. & Berendsen, H. J. GROMACS: Fast, flexible, and free. *Journal of Computational Chemistry* **26,** 1701–1718. ISSN: 0192-8651 (Dec. 2005).

61. Abraham, M. J., Murtola, T., Schulz, R., Pá ll, S., Smith, J. C., Hess, B. & Lindahl, E. GROMACS: High Performance Molecular Simulations through Multi-Level Parallelism from Laptops to Supercomputers. *SoftwareX* **1-2,** 19–25. ISSN: 2352-7110 (Sept. 2015).

62. Lorenz, C., Webb, E., Stevens, M., Chandross, M. & Grest, G. Frictional Dynamics of Perfluorinated Self-Assembled Monolayers on Amorphous SiO2. *Tribology Letters* **19,** 93–98. ISSN: 1023-8883, 1573-2711 (June 2005).

63. Plimpton, S. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics* **117,** 1–19. ISSN: 0021-9991. http://www.sciencedirect.com/science/article/pii/S002199918571039X (Mar. 1995).

64. Thompson, A. P., Aktulga, H. M., Berger, R., Bolintineanu, D. S., Brown, W. M., Crozier, P. S., in 't Veld, P. J., Kohlmeyer, A., Moore, S. G., Nguyen, T. D., Shan, R., Stevens, M. J., Tranchida, J., Trott, C. & Plimpton, S. J. LAMMPS - a Flexible Simulation Tool for Particle-Based Materials Modeling at the Atomic, Meso, and Continuum Scales. **271,** 108171 (2022).

65. Hoover, W. G. Canonical Dynamics: Equilibrium Phase-Space Distributions. *Physical Review A: Atomic, Molecular, and Optical Physics* **31,** 1695–1697 (3 Mar. 1985).

66. Darden, T., York, D. & Pedersen, L. Particle Mesh Ewald: An $N \cdot \log(N)$ Method for Ewald Sums in Large Systems. *The Journal of Chemical Physics* **98,** 10089–10092. ISSN: 0021-9606, 1089-7690. eprint: https://doi.org/10.1063/1.464397 (June 1993).

67. Clear, S. C. & Nealey, P. F. Chemical Force Microscopy Study of Adhesion and Friction between Surfaces Functionalized with Self-Assembled Monolayers and Immersed in Solvents. *Journal of Colloid and Interface Science* **213,** 238–250. ISSN: 0021-9797 (May 1999).

68. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, É. d. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research* **12,** 2825–2830. ISSN: 1532-4435. arXiv: 1201.0490 (2012).

69. Pavlov, Y. L. *Random Forests* ISBN: 978-3-11-094197-5 (2019).

70. Svetnik, V., Liaw, A., Tong, C., Culberson, J. C., Sheridan, R. P. & Feuston, B. P. Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling. *Journal of Chemical Information and Computer Sciences* **43,** 1947–1958. ISSN: 0095-2338 (Nov. 2003).

71. Landrum, G. *et al.* RDKit: Open-source cheminformatics (2006).

72. Weininger, D. SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. *Journal of Chemical Information and Modeling* **28,** 31–36. ISSN: 1549-9596 (Feb. 1988).

73. Patel, R., Borca, C. & Webb, M. *Featurization Strategies for Polymer Sequence or Composition Design by Machine Learning* Preprint (Chemistry, Nov. 2021).

74. Summers, A. Z. *Atools*

75. Vishwakarma, G., Sonpal, A. & Hachmann, J. Metrics for Benchmarking and Uncertainty Quantification: Quality, Applicability, and Best Practices for Machine Learning in Chemistry. *Trends in Chemistry* **3,** 146–156. ISSN: 2589-5974 (Feb. 2021).

CHAPTER 7

Conclusions

While there is a concerted effort among researchers to enforce/encourage best practices for reproducible science, additional work remains to ensure that novel research is performed in a transparent, reproducible, usable by others, and extensible manner. Especially so in the field of molecular simulation, where many aspects of the experiment that might not be as controllable in experimental studies (monolayer composition, energy breakdowns, etc.) can be easily tuned and the full provenance of the data can be captured. Unfortunately, this is easier said than done, as many researchers have noted the challenges of reproducible molecular simulation. This also means that there is clear need for guiding principals and best practices to make molecular simulations more reproducible. By emphasizing and designing studies with reproducibility and extensibility in mind, researchers will build a sturdy foundation to accelerate new discoveries and extend previous work, leading to new insights at a much faster rate than before. In this work, it is shown that designing computational studies with reproducibility and extensibility in mind greatly increases the ability of others to reproduce said studies, while also serving as a template or springboard for additional studies.

The chapters hitherto describe work to better define best practices, a software framework, and design principles for reproducible molecular simulations. In addition, two example studies employing these methods were also performed. In Chapter 2, best practices for molecular simulation were examined and condensed into a set of guiding principles and information that simulationists are encouraged to follow. Best practices are a key component of reproducible science and are a great way to summarize the knowledge and possible issues that many researchers will encounter. It should be noted that documents such as these are never meant to be a replacement for a deeper understanding of the source material itself, but instead a useful sanity check for researchers to employ before, during, and after the simulation to spot early and quickly possible sources of error. Chapter 3 provides a set of guiding principals for researchers to design their molecular simulation studies such that their work will be easier for someone to replicate and validate their work and extend this work even further. Case studies are also provided for many kinds of molecular simulation (lipid bilayers, non-equilibrium molecular dynamics, vapor-liquid equilibria, etc.). Chapter 4 describes the new features and developments of the Molecular Simulation and Design Framework (MoSDeF), and how these tools help increase the reproducibility of the initialization of a molecular system for simulation. In Chapter 5, MoSDeF and these best practices above were used to perform a screening study investigating the self-assembly of coarse-grained nanoparticles with patches of CG alkane chains. It was found that measurable properties of a single patchy nanoparticle (e.g. solvent-accessible surface area, radius of gyration, and asphericity) were capable of predicting the phase space of a larger system when allowed to self-assemble. This provides a computationally cheaper alternative to explore and inform larger scale studies of self-assembly behavior, while also serving as a reusable workflow for nanoparticle simulation studies. In Chapter 6, a workflow to initialize, parametrize, convert to MD engine inputs, perform MD simulations, calculate the tribological properties of interest, and then train a predictive ML model was developed and then massively scaled up, showcasing the usefulness of designing studies with TRUE in mind. Through the use of MoSDeF and TRUE principles, scalable, reproducible, and extensible molecular simulations are much more accessible and this serves as a useful template and guide for future researchers.

Appendices

Appendix A

Best Practices for Molecular Simulation

A.1 License

To begin, the `GitHub` repository containing this work is freely available under a Creative Commons Attribution 4.0 International license. The full contents of this license is provided below and is from git commit `cb3d58f0aa479a52d42f9dfbe36dde8033f6c764`.

License Contents

d. Effective Technological Measures means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. Exceptions and Limitations means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. Licensed Material means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. Licensed Rights means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. Licensor means the individual(s) or entity(ies) granting rights under this Public License.

i. Share means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. Sui Generis Database Rights means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. You means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

Section 2 -- Scope.

a. License grant.

   1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

      a. reproduce and Share the Licensed Material, in whole or in part; and

      b. produce, reproduce, and Share Adapted Material.

   2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

   3. Term. The term of this Public License is specified in Section 6(a).

   4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created,

and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

    a. Offer from the Licensor -- Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

    b. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

b. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

c. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 -- Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 -- Disclaimer of Warranties and Limitation of Liability.

a. UNLESS OTHERWISE SEPARATELY UNDERTAKEN BY THE LICENSOR, TO THE EXTENT POSSIBLE, THE LICENSOR OFFERS THE LICENSED MATERIAL AS-IS AND AS-AVAILABLE, AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE LICENSED MATERIAL, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHER. THIS INCLUDES, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OR ABSENCE OF ERRORS, WHETHER OR NOT KNOWN OR DISCOVERABLE. WHERE DISCLAIMERS OF WARRANTIES ARE NOT ALLOWED IN FULL OR IN PART, THIS DISCLAIMER MAY NOT APPLY TO YOU.

b. TO THE EXTENT POSSIBLE, IN NO EVENT WILL THE LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY (INCLUDING, WITHOUT LIMITATION, NEGLIGENCE) OR OTHERWISE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, EXEMPLARY, OR OTHER LOSSES, COSTS, EXPENSES, OR DAMAGES ARISING OUT OF THIS PUBLIC LICENSE OR USE OF THE LICENSED MATERIAL, EVEN IF THE LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES, COSTS, EXPENSES, OR DAMAGES. WHERE A LIMITATION OF LIABILITY IS NOT ALLOWED IN FULL OR IN PART, THIS LIMITATION MAY NOT APPLY TO YOU.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 -- Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 -- Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 -- Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose

A.2 Checklist for Molecular Simulation

Molecular Dynamics Simulation Checklist

Take stock of your plans

- ☐ **Count the cost:** Think about what you know about the timescales of what you want to observe and determine whether it is tractable to simulate this given the size of your system, your computational resources, and the expense of the simulation. Would the questions you want to answer be better addressed a different way?
- ☐ Pick the desired ensemble (NVT, NPT, NVE, μVT)[1]
- ☐ Determine reference states that you are trying to emulate/discover.
- ☐ What temperature, pressure, etc. are you interested in?
- ☐ What force field properly describes your system?
- ☐ What is already known in the literature and what data do you wish to compare to?

Prepare to implement your plans and make critical decisions about the system

- ☐ Choose a simulation package suitable for simulating that ensemble with your target force field
- ☐ Determine whether you are simulating a bulk (typically periodic) or finite system and choose the appropriate cutoff types and periodicity (full periodicity for bulk systems, partial periodicity for interfaces, etc.) as discussed in subsection 2.5.2
- ☐ Prepare your system, paying particular attention to ensuring it contains the chemical components you want with the structures you want, and that force field parameters are assigned as intended (it is good practice to ensure that you properly implemented the force field by replicating energies, forces, or other observables from prior publications)

Determine handling of cutoffs

- ☐ As a general rule, electrostatics are long-range enough that either the cutoff needs to be larger than the system size (for finite systems) or periodicity is needed along with full treatment of long-range electrostatics (subsection 2.4.4)
- ☐ Nonpolar interactions can often be safely treated with cutoffs of 1-1.5 nm as long as the system size is at least twice that, but long-range dispersion corrections may be needed (subsection 2.5.1)

[1] For mixtures, the semi-grand ensemble (or osmotic ensemble) may be of interest, where the number of particles is fixed but their identities can change [1] allowing, e.g., a constant chemical potential for salt ions to be maintained [2]

Choose appropriate settings for the desired ensemble

☐ Pick a thermostat that gives the correct distribution of temperatures, not just the correct average temperature; if you have a small system or a system with weakly interacting component choose one which works well even in the small-system limit.

☐ Pick a barostat that gives the correct distribution of pressures

☐ Consider the known shortcomings and limitations of certain integrators and thermostats/barostats and whether your choices will impact the properties you are calculating

Choose an appropriate timestep for stability and avoiding energy drift

☐ Determine the highest-frequency motion in the system (typically bond vibrations unless bond lengths are constrained)

☐ As a first guess, set the timestep to approximately one tenth of the highest-frequency motion's characteristic period

☐ Test this choice by running a simulation in the microcanonical ensemble, and ensure that energy is conserved

Determine your run protocol

☐ Plan how you will minimize and equilibrate your system and test that your equilibration protocol actually allows you to reach equilibrium in the target ensemble (subsection 2.5.3)

☐ Determine production settings, how many steps to run, and how often to store data/what data to store

☐ Ensure you have sufficient storage, memory, and computer time to complete the planned calculations

Bibliography

1. Allen, M. P. & Tildesley, D. J. *Computer Simulation of Liquids* 2nd ed. (Oxford University Press, New York, NY, Aug. 2017).

2. Ross, G. A., Rustenburg, A. S., Grinaway, P. B., Fass, J. & Chodera, J. D. Biomolecular Simulations under Realistic Macroscopic Salt Conditions. *J. Phys. Chem. B* **122,** 5466–5486. ISSN: 1520-6106 (May 2018).

Appendix B

Towards Molecular Simulations that are Transparent, Reproducible, Usable By Others, and Extensible (TRUE)

B.1 Packages and Libraries Necessary to Run Example TRUE Simulations

To successfully run these examples in a TRUE fashion requires the methodology to be reproducible as well as the software used for *all* steps of the example/study. Without this, changes in various software packages and their dependencies can introduce another source of irreproducibility to a TRUE study. Contained below is a detailed listing of the main software packages and libraries used throughout the examples. This suite of software is intended to be installed partly with the `conda` scientific software package manager, other python modules not accessible through `conda` will be installed from their source code, and finally, any simulation engine/extraneous packages will be compiled from their source code as well. Comprehensive installation instructions will be provided for each step of this process and annotated. Due to limited molecular simulation engines and other libraries being accessible with the `Windows` operating system, these next steps are only expected to run successfully on `GNU/Linux` and Apple `MacOS` operating systems.

The following text assumes the reader intends to install these packages on their local machine or compute node and can access a terminal emulator.

B.2 Installation of the `conda` Package Manager

To install the `conda` package manager, run the following commands in your shell session if you are using a MacOS operating system.

The $ denotes a line in your terminal emulator and is not part of the command.

```
$ cd ${HOME}
$ curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
$ /bin/bash Miniconda3-latest-MacOSX-x86_64.sh
```

If you are using a local GNU/Linux machine, the following commands should be executed.

```
$ cd ${HOME}
$ curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ /bin/bash Miniconda3-latest-Linux-x86_64.sh
```

Follow the prompts according to your installation preferences, although the default location (your home directory) can be expected to work. Please refer to the documentation (https://conda.io/projects/conda/en/latest/user-guide/index.html) for any additional help or if your installation is on a computing cluster.

B.3 Creating the `conda` Environment

After following the previous instructions and initalized the `conda init` shell support (if you are unsure what that is, refer to the user guide above). Now we will install the software and libraries needed to follow along with the TRUE examples.

Begin by creating a new `conda` environment with `Python 3.7` as the base `Python` interpreter, and activating into that environment. A `conda` environment is a directory that contains all of the necessary libraries and software needed based on your installation instructions. For example, you can have multiple environments, all of which use a different version of `Python`, and `conda` allows you to swap between these environments by *activating* (switching to) or *deactivating* (exiting) an environment. These environments are independent of one another, so modifying a certain environment will not change any other environments' installed packages. All of the examples in this paper are expected to be ran while you are in the `true37` python environment.

```
$ conda create -n true37 python=3.7
$ conda activate true37
```

Next, we will install all of our conda-installable packages and dependencies.

```
$ conda install -c conda-forge -c mosdef -c omnia -c bioconda mbuild foyer signac
 ↪  signac-flow hoomd gromacs=2018.4 lammps pandas matplotlib unyt py3dmol scipy
 ↪  openbabel gsd
```

An alternate option is to add the channels that conda will search to resolve the installation procedure to the `.condarc` file located in your home directory.

```
$ conda config --add channels conda-forge
$ conda config --add channels mosdef
$ conda config --add channels omnia
$ conda config --add channels bioconda
$ conda install mbuild foyer signac signac-flow hoomd gromacs=2018.4 lammps pandas
 ↪  matplotlib unyt py3dmol scipy openbabel gsd
```

To list all of the installed packages in your current `conda` environment, run:

```
conda list
```

B.4 Create a Temporary Workspace

Note that we are also creating a master directory where all of these TRUE examples will be stored, do not run a `git clone` command while inside another `git` repository. The commands to make the master directory and changing to that directory are idempotent, so you can copy and paste the 3 commands below as many times as desired.

```
$ export TRUE_EXAMPLES=${HOME}/true_examples
$ mkdir -p ${TRUE_EXAMPLES}
$ cd ${TRUE_EXAMPLES}
```

B.5 Installation of the mosdef_trappe TRUE example

This example makes use of the Monte Carlo engine GOMC (https://github.com/GOMC-WSU/GOMC.git).

GOMC also requires a working c/c++ compiler, please consult the user manual: (https://gomc-wsu.github.io/Manual/software_requirements.html) for additional help.

The instructions below assume you have an accessible c++ compiler.

The next step is to download a version of this sample workflow and install any dependencies, and to do that we will use the git version control tool. MacOS and GNU/Linux ship with a version of git, the commands to run are listed below.

To begin, we must compile and install GOMC.

```
$ export TRUE_EXAMPLES=${HOME}/true_examples
$ mkdir -p ${TRUE_EXAMPLES}
$ cd ${TRUE_EXAMPLES}

# GOMC requires cmake for compilation, we will install it from conda
$ conda activate true37
$ conda install -c conda-forge cmake

# clone GOMC
$ git clone https://github.com/GOMC-WSU/GOMC.git
$ cd GOMC
$ chmod u+x metamake.sh
$ ./metamake.sh
# once compiled, the executable should be located in the bin directory
$ ls ./bin

# add the gomc bin folder to our path, so we can find the executable no matter the
 ↪  directory
$ LOC_GOMC="$(pwd)/bin"
$ export PATH="${LOC_GOMC}:$PATH"
```

After installing GOMC, we can finally install mosdef_trappe.

```
$ export TRUE_EXAMPLES=${HOME}/true_examples
$ mkdir -p ${TRUE_EXAMPLES}
$ cd ${TRUE_EXAMPLES}
$ LOC_GOMC="${TRUE_EXAMPLES}/GOMC/bin"
$ export PATH="${LOC_GOMC}:$PATH"

$ git clone https://github.com/ahy3nz/mosdef_trappe.git
$ cd mosdef_trappe
$ conda activate true37
$ python -m pip install -e .
```

B.6 Installation of the `true_graphene` Example

This TRUE example requires a few dependencies, including a `mbuild` plugin as well

```
$ export TRUE_EXAMPLES=${HOME}/true_examples
$ mkdir -p ${TRUE_EXAMPLES}
$ cd ${TRUE_EXAMPLES}

# install the mbuild plugin
$ git clone https://github.com/rmatsum836/Pore-Builder.git
$ cd Pore-Builder
$ conda activate true37
$ conda install -c conda-forge -c mosdef -c omnia --file ./requirements.txt
$ python -m pip install -e .
# now clone the graphene_pore example
$ git clone https://github.com/rmatsum836/true_graphene.git
```

B.7 Installation of the `true-lipids` example

By installing the other packages above, all of the dependencies for this example should be installed.

```
$ export TRUE_EXAMPLES=${HOME}/true_examples
$ mkdir -p ${TRUE_EXAMPLES}
$ cd ${TRUE_EXAMPLES}

$ conda activate true37
$ git clone https://github.com/uppittu11/true_lipids.git
```

If running this example does not work, follow this installation step below.

```
$ export TRUE_EXAMPLES=${HOME}/true_examples
$ mkdir -p ${TRUE_EXAMPLES}
$ cd ${TRUE_EXAMPLES}

$ conda activate true37
$ cd true_lipids
$ conda install -c conda-forge -c mosdef -c omnia mbuild mdtraj py3dmol
```

B.8 Installation of the TRUE-nanotribology Project

To access and install the necessary software for this repository, the following steps should be taken.

```
$ export TRUE_EXAMPLES=${HOME}/true_examples
$ mkdir -p ${TRUE_EXAMPLES}
$ cd ${TRUE_EXAMPLES}

$ conda activate true37
$ git clone https://github.com/daico007/TRUE-nanotribology.git
$ cd TRUE-nanotribology
$ conda install -c conda-forge -c mosdef -c omnia -c bioconda --file ./requirements.txt
```

B.9 Removing the Installed Software

Listed below are all the steps needed to remove the examples, as well as the conda environment that was created. Refer to `miniconda`'s site for assistance unsintalling miniconda.

```
$ export TRUE_EXAMPLES=${HOME}/true_examples
# The rm -f (force) command might be needed to remove the directories
$ rm -r ${TRUE_EXAMPLES}

# remove the conda environment, and clean up
$ conda activate base
$ conda remove -n true37 --all

$ conda clean --index-cache --packages --tarballs --yes
```

Appendix C

Self–assembly of patchy alkane–grafted silica nanoparticles

Grafted nanoparticle systems were initialized using `mBuild` [1, 2]. A schematic of a grafted system is shown in Figure C.1 for a single chain, with labels for each bead type: the silica coarse grained (CG) core nanoparticle beads, the 3:1 CG alkane grafted chains which corresponds to a $CH_2CH_2CH_2$ subunit, and the terminus of those chains which corresponds to a $CH_2CH_2CH_3$ subunit.

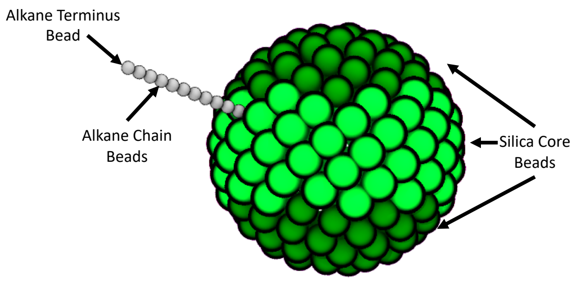


Figure C.1: A schematic of a nanoparticle, showing a single graft for clarity. Dark green beads demonstrate the area of the surface with no alkane grafted chains (this makes up the patch) at the poles of the particle. Light green beads around the equator will have chains grafted to the surface at the density ρ_{chain} that is predetermined. The fractional surface area, FSA, of the particle presented here is 0.55, and totaled from the area of the dark green beads.

44 bulk simulations were performed to identify the system structure. The simulation workflow is discussed in the main article; the Python script `Analysis/escale-model0.5.py` included in the project GitHub repository includes the exact `HOOMD-Blue` information to run the simulations. See the section titled Section C.5 to access this script and related data. Figure C.2 visualizes several example systems of the 3 representative phases (aggregated, stringy, and dispersed), including radial distribution functions (RDFs); Figure C.2 is identical to Figure 5.2 in the main text, however, surface grafted chains are also rendered.

Table C.1 shows the results of the bulk simulation, as well as the phases that form. Also appended are the single nanoparticle simulation values that are associated with a given bulk simulation statepoint. A full csv of these bulk simulation values, with the matching simulation identifiers is available in the previously mentioned GitHub repository titled `DataFull_TNP_Dataset.csv`.

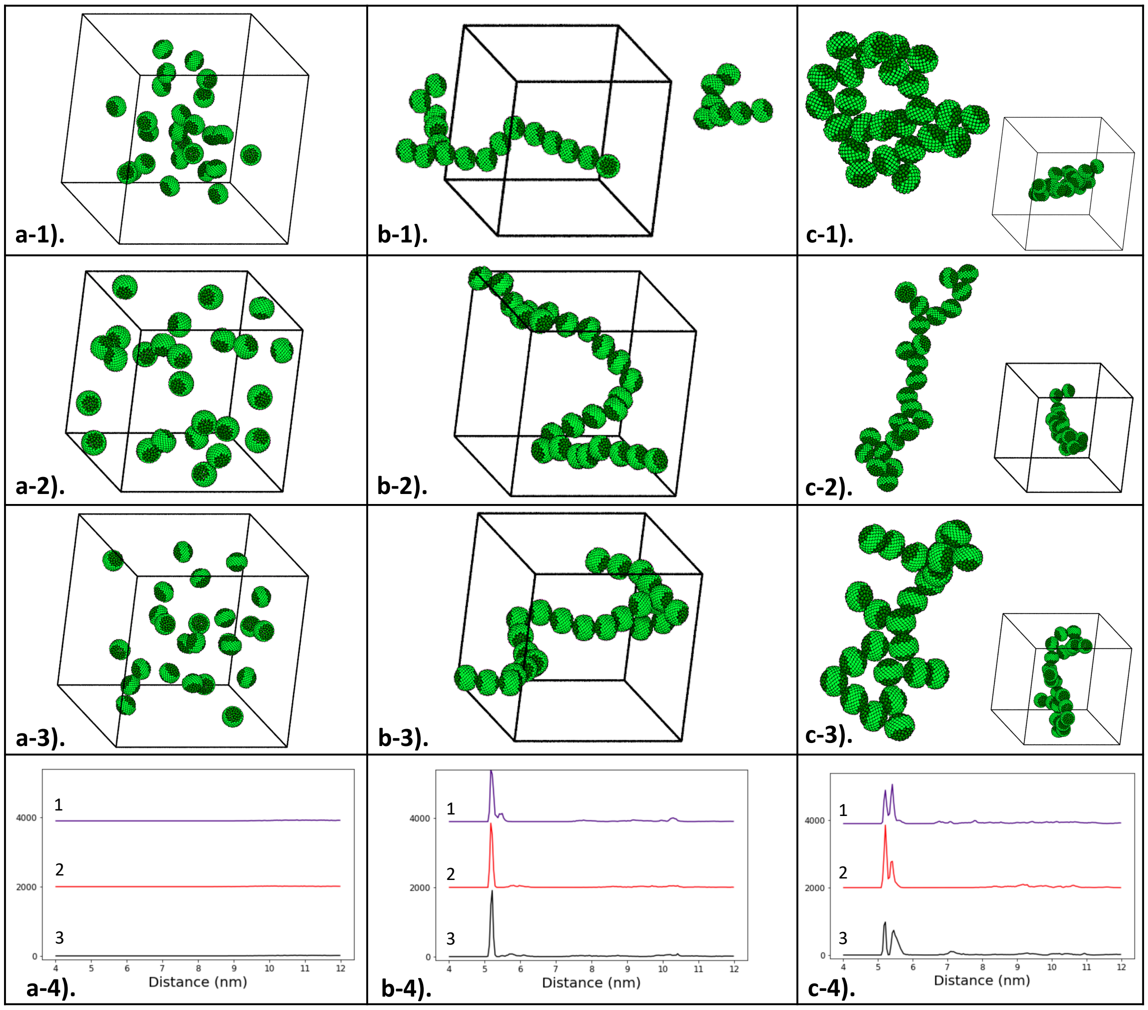


Figure C.2: Examples of equilibrium phases with statepoints of a1). $N = 10, r_{chain} = 3.5, FSA = 0.45$ a2). $N = 8, r_{chain} = 3.5, FSA = 0.35$ and a3). $N = 10, r_{chain} = 3.5, FSA = 0.55$ in the dispersed phase. b1). $N = 6, r_{chain} = 3.0, FSA = 0.45$ b2). $N = 6, r_{chain} = 4.0, FSA = 0.45$ and b3). $N = 5, r_{chain} = 3.5, FSA = 0.40$ in the stringy phase. c1). $N = 6, r_{chain} = 2.5, FSA = 0.55$ c2). $N = 6, r_{chain} = 3.0, FSA = 0.65$ and c3). $N = 5, r_{chain} = 3.5, FSA = 0.55$ in the aggregated phase. (4) are the corresponding RDFs to the visualized systems, offset by a value of 2000 in the y direction for clarity. Note, the scale of the RDFs is the same for all systems. Aggregated phases are visualized from two orientations to more clearly demonstrate the structure. This figure is a replicate of Figure 2 in the paper with chains visible.

Table C.1: Bulk Simulation Data. Data from the last three columns are matched to data generated at the same chain length, chain density, and FSA values for single particle simulations. Note that values of NaN in these columns mean no single, grafted nanoparticle simulation was completed for that statepoint. Of the 44 total bulk simulations, 40 simulations were matched to single grafted nanoparticle simulations. For Phase, a value of 0 is dispersed, 1 is stringy, and 2 is aggregated.

Chain Length (beads)	Chain Density ($\frac{Chains}{nm^2}$)	FSA	Avg Coord Number	Stdev Coord Number	Phase	f_{SASA}	$R^{-1}_{g,chain}$	$R^{-1}_{g,gNP}$	Asphericity
6	4.0	0.65	2.00	1.01	2	0.22	2.5	0.29	0.098
10	3.0	0.55	0.68	0.52	0	0.11	1.5	0.23	0.054
6	2.5	0.55	2.50	1.32	2	0.21	2.1	0.32	0.075
6	4.0	0.45	1.39	0.65	1	0.13	2.7	0.26	0.062
8	3.5	0.45	0.67	0.69	0	0.10	1.7	0.23	0.048
7	4.5	0.55	1.39	0.68	1	0.12	2.2	0.24	0.068
6	3.5	0.25	0.00	0.01	0	0.06	2.4	0.24	0.022
6	3.5	0.55	1.99	0.77	1	0.18	2.3	0.28	0.082
6	3.0	0.35	1.41	0.58	1	0.11	2.1	0.26	0.042
11	3.5	0.65	0.73	0.66	0	0.12	1.2	0.23	0.051
6	3.5	0.35	1.30	0.62	1	0.10	2.5	0.25	0.043
6	3.0	0.27	0.05	0.12	0	NaN	NaN	NaN	NaN
6	4.5	0.35	0.04	0.13	0	0.07	2.7	0.24	0.045
8	4.0	0.55	1.14	0.66	1	0.12	1.8	0.24	0.071
8	3.5	0.35	0.00	0.00	0	0.07	1.7	0.22	0.032
6	4.5	0.55	1.65	0.69	1	0.15	2.5	0.26	0.076
5	3.5	0.40	1.81	0.58	1	0.16	2.8	0.28	0.066
9	4.2	0.55	0.08	0.27	0	NaN	NaN	NaN	NaN
6	3.0	0.55	1.84	0.88	1	0.21	2.3	0.30	0.081
10	3.5	0.55	0.00	0.00	0	0.10	1.4	0.22	0.041
7	3.0	0.55	1.70	0.80	1	0.17	2.2	0.28	0.068
12	3.5	0.55	0.00	0.00	0	NaN	NaN	NaN	NaN
6	3.0	0.65	2.42	1.02	2	0.25	1.9	0.32	0.094
6	4.5	0.65	2.14	1.03	2	0.20	2.3	0.28	0.104
8	3.5	0.65	1.62	0.71	1	0.18	1.9	0.27	0.082
8	2.5	0.45	1.06	0.70	1	0.11	1.7	0.26	0.036
10	3.5	0.65	1.28	0.64	1	0.14	1.3	0.24	0.058
8	4.5	0.55	0.77	0.68	0	0.10	2.1	0.23	0.062
6	3.5	0.45	1.64	0.67	1	0.15	2.4	0.27	0.064
6	3.5	0.65	2.20	1.29	2	0.24	2.4	0.31	0.103
6	4.5	0.45	1.59	0.84	1	0.12	2.4	0.25	0.064
5	3.5	0.20	0.01	0.02	0	0.06	2.8	0.26	0.017
9	3.5	0.55	0.86	0.61	0	0.11	1.5	0.23	0.059
6	4.0	0.55	1.71	0.53	1	0.17	2.4	0.27	0.086
10	2.5	0.55	1.19	0.63	1	0.12	1.3	0.24	0.059
8	2.5	0.55	1.86	0.69	1	0.15	1.6	0.28	0.056
7	3.5	0.50	1.38	0.64	1	0.14	2.1	0.26	0.060
8	4.2	0.55	1.15	0.55	1	NaN	NaN	NaN	NaN
8	3.5	0.55	1.22	0.65	1	0.13	1.8	0.25	0.062
10	3.5	0.45	0.00	0.00	0	0.07	1.3	0.21	0.042
6	3.0	0.45	1.81	0.62	1	0.16	2.5	0.28	0.064
5	3.5	0.55	2.70	1.14	2	0.21	2.3	0.31	0.094
5	3.5	0.30	1.37	0.71	1	0.11	3.3	0.27	0.035
6	4.0	0.35	0.61	0.65	0	0.09	2.6	0.24	0.043

C.1 Analysis of single nanoparticles

All single nanoparticle values at tested statepoints (11 FSA's, 7 chain lengths, and 7 chain densities) are available in a csv file titled `Data/Single_Tether_Data.csv` available in the associated GitHub repository. The Python scripts used to calculate the metrics, discussed below, are also included (`Analysis/rg_asphere.py`).

C.1.1 Solvent Accessible Surface Area (SASA)

SASA is calculated using the `MDTraj` [3] package based upon the algorithm of Shrake and Rupley [4] . SASA values are calculated with a probe point radius of 0.25 nm. The nanoparticle core is modeled as a single bead with 2.5 nm in radius. The chain molecules are modeled with a radius of 0.2291 nm and the terminus beads have a radius of 0.2331 nm. 1000 points are generated over the surface of the particles and the total SASA of the nanoparticle core is summed. The code for this calculation can be found in the file `Analysis/calculate_SASA.py` in the GitHub repo. In this work, all SASA values are normalized by the total spherical surface area ($SASA_{sphere}$), which is $4\pi R_{NP}^2$. This normalized SASA value is f_{SASA} (Equation C.1).

$$f_{SASA} = \frac{SASA}{SASA_{sphere}} \tag{C.1}$$

C.1.2 Radius of Gyration (R_g)

Radius of Gyration values are calculated using `MDAnalysis` [5, 6]. Two metrics are considered; the radius of gyration of the entire grafted nanoparticle and the average radius of gyration of each of the chain grafts, considered individually. Normalization of this value is done by dividing nanoparticle radius (2.5 nm) by R_g and reported as $R_{g,NP}^{-1}$(Equation C.2) and $R_{g,chain}^{-1}$(Equation C.3) in the main text.

$$R_{g,NP}^{-1} = \frac{r_{\text{nanoparticle}}}{R_{g,\text{single grafted nanoparticle}}} \tag{C.2}$$

$$R_{g,chain}^{-1} = \frac{r_{\text{nanoparticle}}}{R_{g,\text{averaged from individual grafted chains}}} \tag{C.3}$$

C.1.3 Asphericity

Asphericity is calculated from single, grafted nanoparticle simulations using `MDAnalysis` [5, 6]. Values are already unitless, so normalization is not required. Asphericity is calculated for the entire grafted nanoparticle i.e., nanoparticle core and chains together, similar to $R_{g,NP}^{-1}$.

C.2 Variable Correlations

Figure C.3–Figure C.2 compare the various metrics considered in this work to establish if there are correlations. Figure C.3 plots asphericity. Asphericity has a linear positive correlation with the metrics indicative of the structure formed; phase, coordination number, and its standard deviation, as discussed in the main text. Of the user-defined grafted nanoparticle parameters (i.e. FSA, chain length, and chain density), asphericity seems to show the most positive correlation with FSA. Since FSA was predicted to be the strongest indicator of phase from these three variables, it is sensible that asphericity is a good predictor of phase and highly correlated with FSA. Asphericity and f_{SASA} also show strong positive correlation, presumably for similar reasons. A larger asphericity value of the nanoparticle is an effect of a greater portion of the chains being located at the equator, which contributes to a less spherical particle. This is indicative of a larger FSA, and likewise a larger f_{SASA} because more chains at the equator results in less covering of the nanoparticle core at the poles.

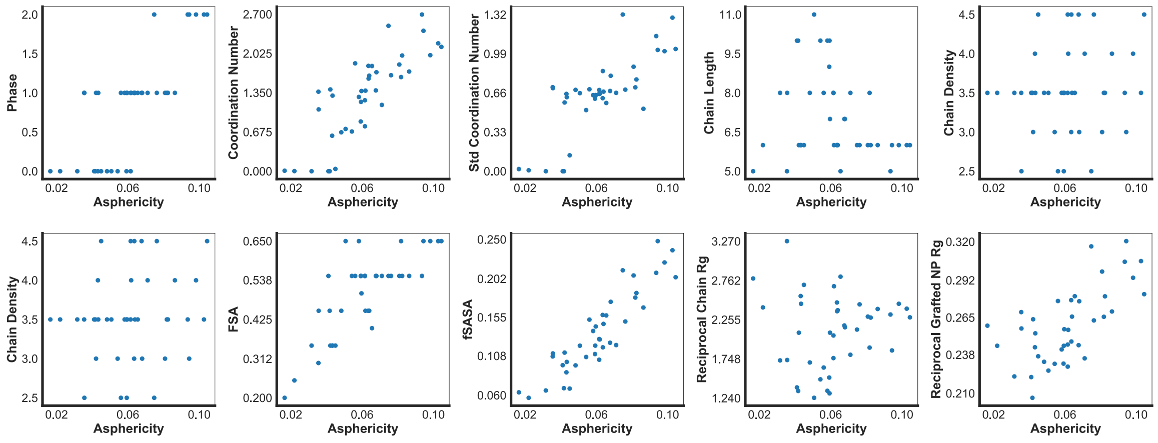


Figure C.3: Correlation plots of Asphericity versus other nanoparticle descriptors. This is done for 40 out of the total 44 bulk simulations. **Reciprocal Chain** R_g is $R_{g,chain}^{-1}$ and **Reciprocal Grafted NP** R_g is $R_{g,NP}^{-1}$.

Figure C.4 shows $R_{g,chain}^{-1}$ correlations. The highest correlation is to chain length, which is negative because the inverse of the R_g values are plotted. Correlations between $R_{g,chain}^{-1}$ and the rest of the variables of interest are relatively weak.

Figure C.5 plots correlations of $R_{g,NP}^{-1}$. The correlations observed are very similar to the asphericity results from Figure C.3 since they are inherently related. Both are calculated from the principle moments of the gyration tensor.

Figure C.2 considers f_{SASA}. f_{SASA} has the strongest linear correlation with phase and coordination number. When compared to standard deviation of the coordination number, there is a small dip where f_{SASA} values all correlate to deviation values of 0.4-0.9, then a rise as values become greater than 1.0. This can be attributed

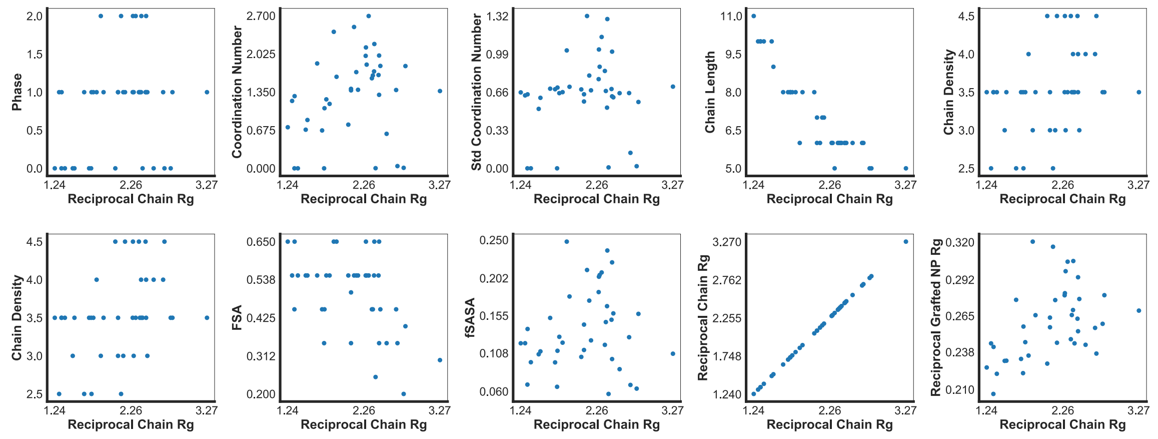


Figure C.4: Correlation plots of $R_{g,chain}^{-1}$ versus other nanoparticles descriptors. This is done for 40 out of the total 44 bulk simulations. **Reciprocal Chain** R_g is $R_{g,chain}^{-1}$ and **Reciprocal Grafted NP** R_g is $R_{g,NP}^{-1}$.

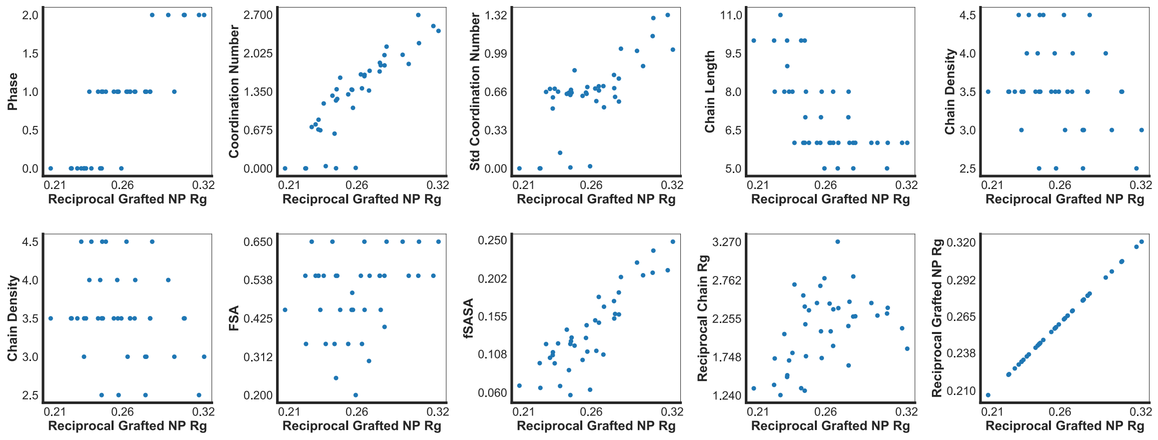


Figure C.5: Correlation plots of $R_{g,NP}^{-1}$ versus other nanoparticles descriptors. This is done for 40 out of the total 44 bulk simulations. **Reciprocal Chain** R_g is $R_{g,chain}^{-1}$ and **Reciprocal Grafted NP** R_g is $R_{g,NP}^{-1}$.

to the transition of stringy phases, which have relatively consistent neighbors, to aggregated phases which can have a larger variability in number of neighbors (hence a larger standard deviation). f_{SASA} also shows closer correlation to FSA than $R_{g,NP}^{-1}$. Table C.2 summarizes this data with calculations of the Spearman Rank Coefficients.

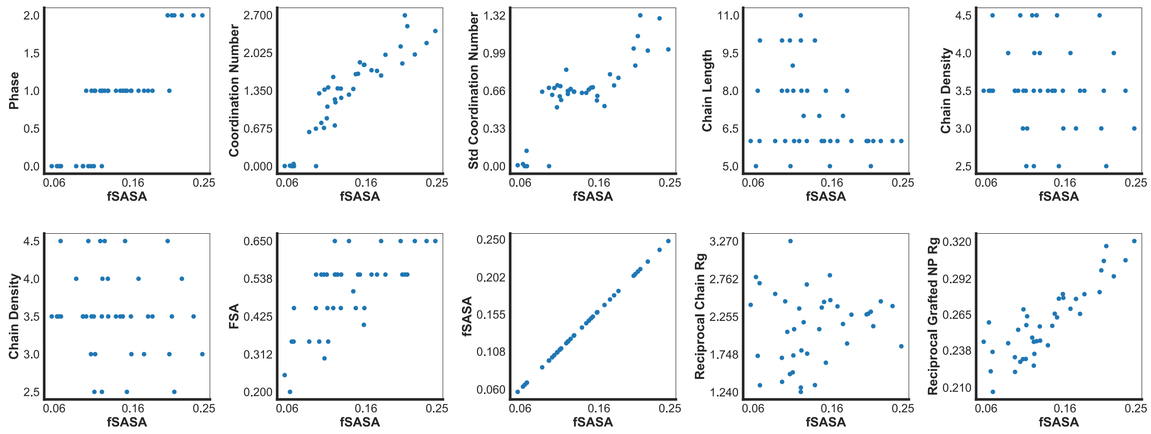


Figure C.6: Correlation plots of f_{SASA} versus other nanoparticles descriptors. This is done for 40 out of the total 44 bulk simulations. **Reciprocal Chain** R_g is $R^{-1}_{g,chain}$ and **Reciprocal Grafted NP** R_g is $R^{-1}_{g,NP}$
.

C.3 Table of Correlations

Correlation values were calculated using the `SciPy` python library [7, 8]. The calculation and scripts for visualizing these data are included in the Python Jupyter Notebook `Analysis/Plot_Data.ipynb` included in the associated GitHub repository.

Table C.2: Table of Spearman Rank Coefficients. These values are determined by comparing each key variable to understand their relations in these grafted nanoparticle systems. These values are used to generate Figure 5.4.

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
(1) Asphericity	1.000000	0.182819	-0.270345	0.833928	0.694420	0.746683	0.161351	0.661163	0.684181	0.897749
(2) Chain Density	0.182819	1.000000	-0.185443	-0.138837	-0.006178	-0.118768	0.376715	-0.245788	-0.029444	-0.161563
(3) Chain Length	-0.270345	-0.185443	1.000000	-0.498713	0.363046	-0.428529	-0.898545	-0.688980	-0.280594	-0.281531
(4) Coordination Number	0.833928	-0.138837	-0.498713	1.000000	0.471201	0.896691	0.293113	0.914806	0.744932	0.937512
(5) FSA	0.694420	-0.006178	0.363046	0.471201	1.000000	0.453747	-0.437929	0.237459	0.517765	0.677731
(6) Phase	0.746683	-0.118768	-0.428529	0.896691	0.453747	1.000000	0.232920	0.833741	0.729451	0.861574
(7) Reciprocal Chain Rg	0.161351	0.376715	-0.898545	0.293113	-0.437929	0.232920	1.000000	0.480863	0.114093	0.110131
(8) Reciprocal Grafted NP Rg	0.661163	-0.245788	-0.688980	0.914806	0.237459	0.833741	0.480863	1.000000	0.665603	0.825704
(9) Std Coordination Number	0.684181	-0.029444	-0.280594	0.744932	0.517765	0.729451	0.114093	0.665603	1.000000	0.721336
(10) fSASA	0.897749	-0.161563	-0.281531	0.937512	0.677731	0.861574	0.110131	0.825704	0.721336	1.000000

C.4 Data Analysis and Plotting

The following plots overlay phase information from the bulk simulations upon heatmaps that are based on calculations from the single, grafted nanoparticle simulations. Figure C.7–Figure C.10 are the raw heat maps plotted on a mesh grid, and Figure C.11–Figure C.13 are created using the same data, but with a bilinear interpolation to provide a smoother heatmap. The colormap is created such that, to the best approximation, the white regions of each plot align with the transition regions predicted by the phase histograms in Figure 5.5 in the main text; for metrics that are strongly correlated with the predicted phase, these should represent reasonable estimates of the phase boundaries. The code for plotting these can be found in the associated GitHub repository, where `Analysis/Plot_Data.ipynb` and `Analysis/plotting.py` include the relevant functions for replicating these plots. Note, Figure C.7 contains the same information as Figure 5.6 in the main text, but without averaging of the heatmap.

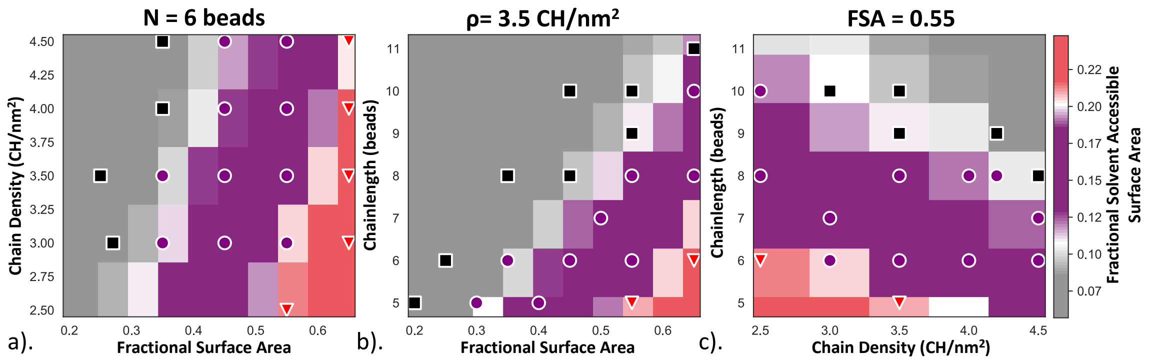


Figure C.7: Phase diagrams overlaid on f_{SASA} ($\frac{\text{SASA}}{\text{spherical surface area}}$) as a predictor. White regions indicate areas from the f_{SASA} histogram (Figure 5.5a) where overlap of the curves occur. Bulk simulation phase is represented as black squares dispersed, purple circles– stringy, red triangles– aggregated. Left). chain density as a function of FSA with N held constant at 6 beads Center). FSA versus N with chain density held constant at 3.5 $\frac{chains}{nm^2}$. Right). Chain density versus N with FSA held constant at 0.55.

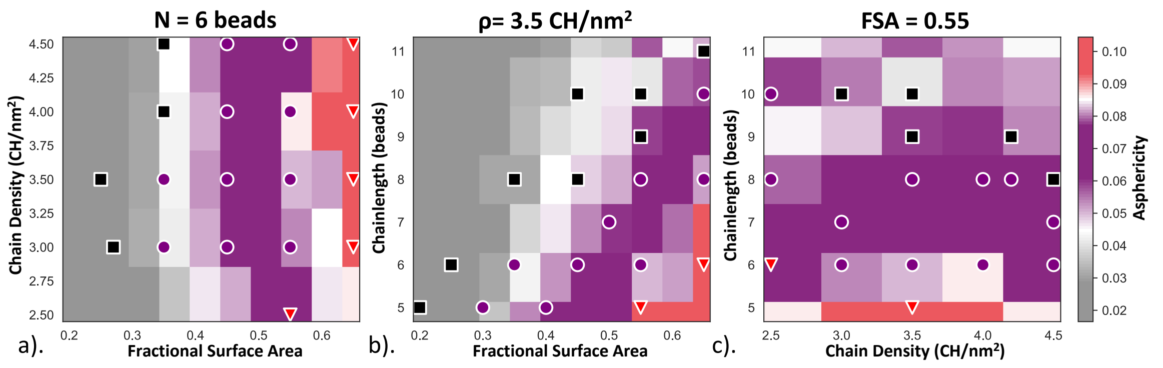


Figure C.8: Phase diagrams overlaid on asphericity as a predictor. White regions indicate areas from the asphericity histogram (Figure 5.5c) where overlap of the curves occur. Bulk simulation phase is represented as black squares– dispersed, purple circles– stringy, red triangles– aggregated. Left). Chain density as a function of FSA with N held constant at 6 beads. Center). FSA versus N with chain density held constant at 3.5 $\frac{chains}{nm^2}$. Right). Chain density versus N with FSA held constant at 0.55.

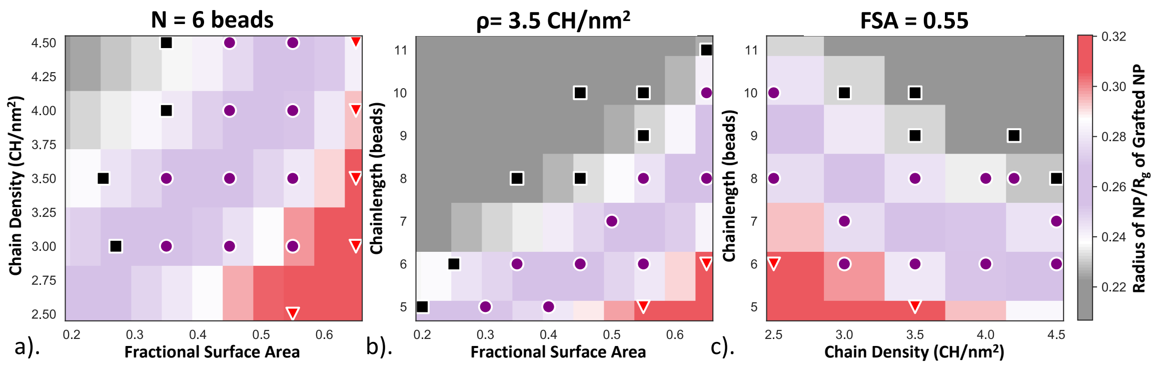


Figure C.9: Phase Diagrams from Figure 5.3 overlaid on $R^{-1}_{g,NP}$ (Equation C.2) as a predictor. White regions indicate areas from the $R^{-1}_{g,NP}$ histogram (Figure 5.5d) where overlap of the curves occur. Bulk simulation phase is represented as black squares– dispersed, purple circles– stringy, red triangles– aggregated. Left). Chain density as a function of FSA with N held constant at 6 beads. Center). FSA versus N with chain density held constant at 3.5 $\frac{chains}{nm^2}$. Right). Chain density versus N with FSA held constant at 0.55.

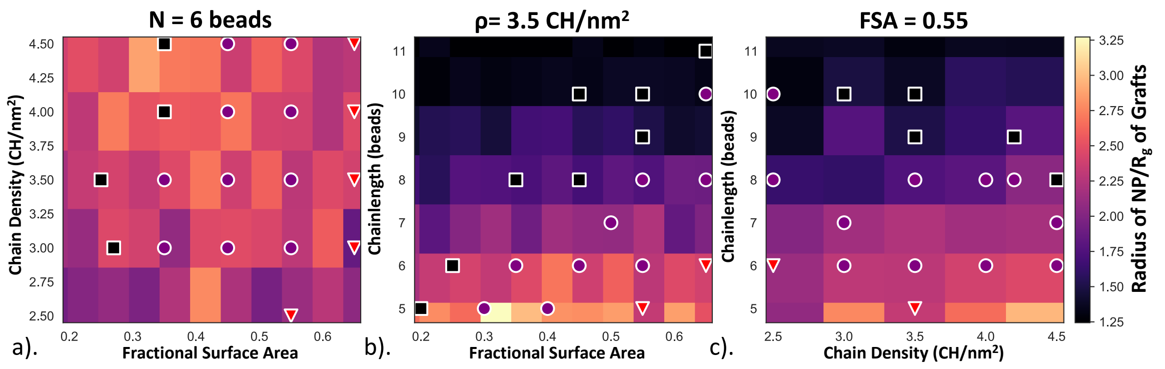


Figure C.10: Phase Diagrams from Figure 5.3 overlaid with $R^{-1}_{g,chain}$ (Equation C.3) as a predictor. Bulk simulation phase is represented as black squares– dispersed, purple circles– stringy, red triangles– aggregated. Left). Chain density as a function of FSA with N held constant at 6 beads. Center). FSA versus N with chain density held constant at 3.5 $\frac{chains}{nm^2}$. Right). Chain density versus N with FSA held constant at 0.55.

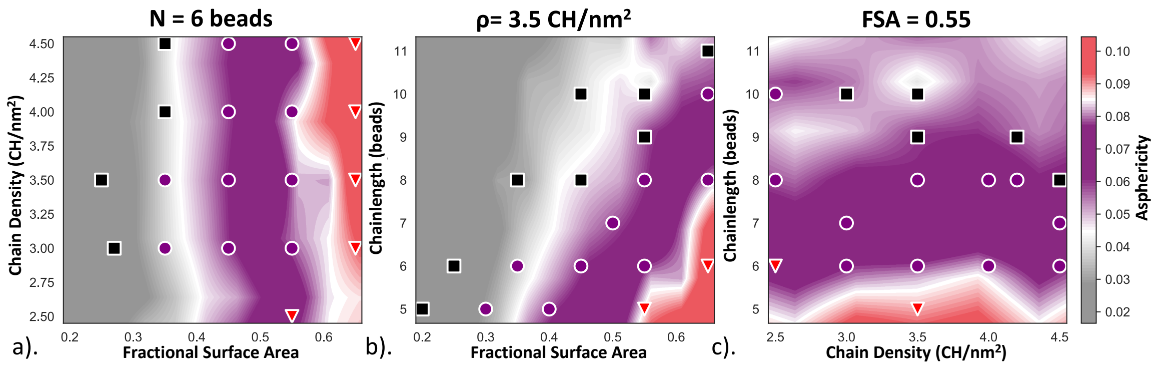


Figure C.11: Phase Diagrams from Figure 5.3 overlaid with asphericity as a predictor. White regions indicate areas from the asphericity histogram (Figure 5.5c) where overlap of the curves occur. Bulk simulation phase is represented as: black squares– dispersed, purple circles– stringy, red triangles– aggregated. Bilinear interpolation was used for meshing the asphericity values. Left). Chain density as a function of FSA with N held constant at 6 beads. Center). FSA versus N with chain density held constant at 3.5 $\frac{chains}{nm^2}$. Right). Chain density versus N with FSA held constant at 0.55.

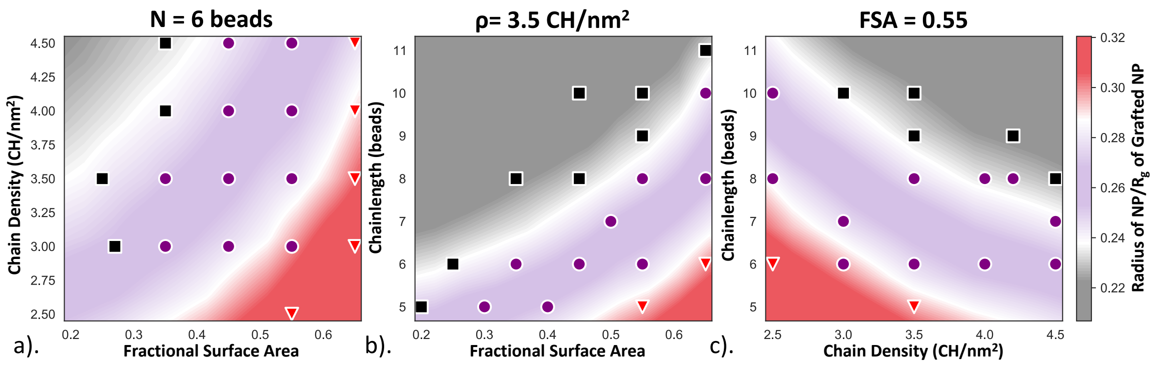


Figure C.12: Phase Diagrams from Figure 5.3 overlaid with $R_{g,NP}^{-1}$ (Equation C.2) as a predictor. White regions indicate areas from the $R_{g,NP}^{-1}$ histogram (Figure 5.5d) where overlap of the curves occur. Bulk simulation phase is represented as black squares– dispersed, purple circles– stringy, red triangles– aggregated. Bilinear interpolation was used for meshing the $R_{g,NP}^{-1}$ values. Left). Chain density as a function of FSA with N held constant at 6 beads. Center). FSA versus N with chain density held constant at 3.5 $\frac{chains}{nm^2}$. Right). Chain density versus N with FSA held constant at 0.55.

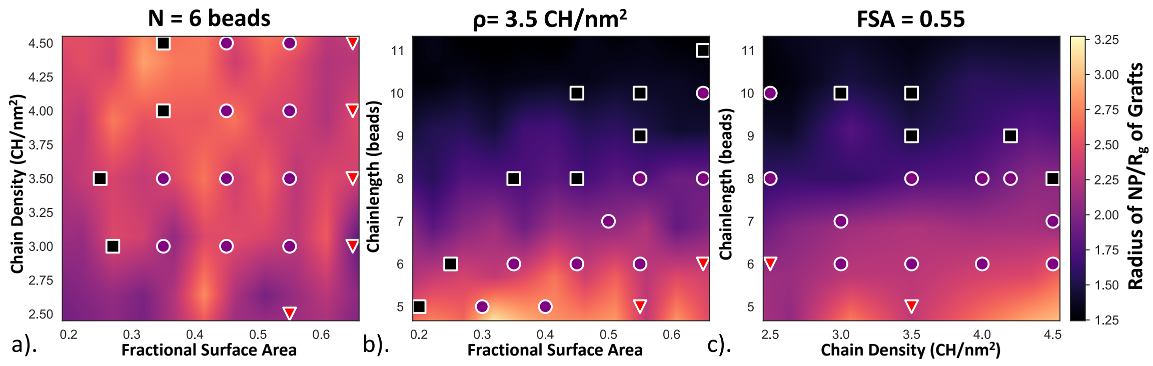


Figure C.13: Phase Diagrams from Figure 5.3 overlaid with $R_{g,chain}^{-1}$ (Equation C.3) as a predictor. Bulk simulation phase is represented as black squares– dispersed, purple circles– stringy, red triangles– aggregated. Bilinear interpolation was used for meshing the $R_{g,chain}^{-1}$ values. Left). Chain density as a function of FSA with N held constant at 6 beads. Center). FSA versus N with chain density held constant at 3.5 $\frac{chains}{nm^2}$. Right). Chain density versus N with FSA held constant at 0.55.

C.5 Downloading and Working with Datasets

C.5.1 Packages and Libraries Necessary to Run Simulations

Contained below is a detailed listing of the main software packages and libraries used throughout the work. This suite of software is intended to be installed partly with the conda scientific software package manager, other python modules not accessible through conda will be installed from their source code, and finally, any simulation engine/extraneous packages will be compiled from their source code as well. Comprehensive installation instructions will be provided for each step of this process and annotated. Due to limited molecular simulation engines and other libraries being accessible with the `Windows` operating system, these next steps are only expected to run successfully on `GNU/Linux` and Apple `MacOS` operating systems.

Text from this section of the SI is from the supplementary information of Thompson *et al.* [9] The following text assumes the reader intends to install these packages on their local machine or compute node and can access a terminal emulator.

C.5.2 Installation of the `conda` Package Manager

To install the `conda` package manager, run the following commands in your shell session if you are using a MacOS operating system.

The $ denotes a line in your terminal emulator and is not part of the command.

```
$ cd ${HOME}
$ curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
$ /bin/bash Miniconda3-latest-MacOSX-x86_64.sh
```

If you are using a local GNU/Linux machine, the following commands should be executed.

```
$ cd ${HOME}
$ curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ /bin/bash Miniconda3-latest-Linux-x86_64.sh
```

Follow the prompts according to your installation preferences, although the default location (your home directory) can be expected to work. Please refer to the documentation (https://conda.io/projects/conda/en/latest/user-guide/index.html) for any additional help or if your installation is on a computing cluster.

C.5.3 Create a Temporary Workspace

Note that we are also creating a master directory where all of the relevent data and code will be stored, do not run a `git clone` command while inside another `git` repository. The commands to make the master directory and changing to that directory are idempotent, so you can copy and paste the 3 commands below as many times as desired.

```
$ export NP_ASSEMBLY=${HOME}/patchy_np_analysis
$ mkdir -p ${NP_ASSEMBLY}
$ cd ${NP_ASSEMBLY}
```

C.5.4 Installation of the `nanoparticle assembly` GitHub Repo

(https://github.com/PTC-CMC/nanoparticle_assembly).

```
$ git clone https://github.com/PTC-CMC/nanoparticle_assembly.git
$ cd nanoparticle_assembly
```

C.5.5 Creating the `conda` Environment

After following the previous instructions and initialized the `conda init` shell support (if you are unsure what that is, refer to the user guide above). Now we will install the software and libraries needed to do the data analysis.

Begin by creating a new `conda` environment from the *environment.yml* file on the `nanoparticle_assembly` GitHub Repo, and activating into that environment. A `conda` environment is a directory that contains all of the necessary libraries and software needed based on your installation instructions. For example, you can have multiple environments, all of which use a different version of `Python`, and `conda` allows you to swap between these environments by *activating* (switching to) or *deactivating* (exiting) an environment. These environments are independent of one another, so modifying a certain environment will not change any other environments' installed packages. All of the examples in this paper are expected to be ran while you are in the `patchy_np36` python environment.

```
$ cd Setup
$ conda env create -f environment.yml
$ conda activate patchy_np36
```

To list all of the installed packages in your current `conda` environment, run:

```
conda list
```

C.5.6 Navigating the Cloned Directory

Included in this repo is the code to generate the figures in both the main text and the supplementary information. The `Analysis` directory contains the notebooks to open and see this information. Included notebooks and their use are:

- `Plot_Data.ipynb`

  This notebook generates plots from data read in from .csv files in the `Data` directory. It creates phase scatter plots, heatmaps, correlation plots, phase histograms, and the correlation matrix. Some slight modifications were made to these plots such as changing labels for the figures presented in the main text. This notebook can run just off of the .csv files and doesn't need access to the Zenodo DOI (Digital Object Identifier).

- `COG_Single_NP's.ipynb`
  This notebook is used to generate centers of geometry for the nanoparticle systems. It saves trajectories named COG.xyz that can be read in for nearest neighbor calculations. Note that one will need the bulk simulation data installed from this projects Zenodo source.

- `Read_Trajectory.ipynb`
  This notebook reads in the center of geometry data and does coordination number calculations. This data is saved out in `Full_TNP_Dataset.csv`. This must be run after COG_Single_NP's.ipynb because it needs the COG.xyz files.

- `vis_scripts.ipynb`
  This file takes the full nanoparticle dcd files, and generates .xml files that show the full structures shown in the paper. It prevents particles from being cut off at boundaries and gives a better visualization of the full string structures that can be built.

- `escale0.5-model3.py`
  This file will give relevant simulation parameters used in *Hoomd-Blue*. This is built for a deprecated version of *Hoomd-Blue*, so replicating the simulation with this file is not possible within this repository. However, the simulation parameters still can be used with updated scripts to replicate the simulations performed.

Other important directories include `Figures` which are where generated plots will be saved, and `Data` which is where output .csv files are kept from trajectory data.

Raw trajectory data can be imported from the Zenodo DOI. This can be located with the DOI *10.5281/zenodo.4042749* and installed into the `nanoparticle_assembly/trajectory/` directory. The following commands must be run.

```
$ cd ${NP_ASSEMBLY}
$ cd nanoparticle_assembly/trajectories/bulk
$ mv ../bulk_simulation_data.zip ./
$ unzip bulk_simulation_data.zip
$ cd ../single_particle
$ mv ../single_particle_data.zip ./
$ unzip single_particle_data.zip
```

These files can then be analyzed using the scripts discussed above. However, recreating the plots from `Plot_Data.ipynb` is still possible since that data is included in the .csv files in the `Data` folder.

C.5.7 Removing the Installed Software

Listed below are all the steps needed to remove the examples, as well as the conda environment that was created. Refer to `miniconda`'s site for assistance uninstalling miniconda.

```
$ export NP_ASSEMBLY=${HOME}/patchy_np_analysis
# The rm -f (force) command might be needed to remove the directories
$ rm -r ${NP_ASSEMBLY}

# remove the conda environment, and clean up
$ conda activate base
$ conda remove -n patchy_np36 --all

$ conda clean --index-cache --packages --tarballs --yes
```

Bibliography

1. Contributors, M. *MoSDeF Webpage* https://mosdef.org. Accessed: 2022-03-13.

2. Klein, C., Sallai, J., Jones, T. J., Iacovella, C. R., McCabe, C. & Cummings, P. T. in *Foundations of Molecular Modeling and Simulation: Select Papers from FOMMS 2015* (eds Snurr, R. Q., Adjiman, C. S. & Kofke, D. A.) 79–92 (Springer Singapore, Singapore, 2016). ISBN: 978-981-10-1128-3.

3. McGibbon, R. T., Beauchamp, K. A., Harrigan, M. P., Klein, C., Swails, J. M., Hernández, C. X., Schwantes, C. R., Wang, L.-P., Lane, T. J. & Pande, V. S. MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. *Biophysical Journal* **109,** 1528–1532. ISSN: 0006-3495 (Oct. 2015).

4. Shrake, A. & Rupley, J. Environment and Exposure to Solvent of Protein Atoms. Lysozyme and Insulin. *Journal of Molecular Biology* **79,** 351–371. ISSN: 0022-2836 (Sept. 1973).

5. Michaud-Agrawal, N., Denning, E. J., Woolf, T. B. & Beckstein, O. MDAnalysis: A Toolkit for the Analysis of Molecular Dynamics Simulations. *Journal of Computational Chemistry* **32,** 2319–2327. ISSN: 0192-8651 (July 2011).

6. Gowers, R., Linke, M., Barnoud, J., Reddy, T., Melo, M., Seyler, S., Domański, J., Dotson, D., Buchoux, S., Kenney, I. & Beckstein, O. *MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations* in *Python in Science Conference* (Austin, Texas, 2016), 98–105.

7. Virtanen, P. *et al.* SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17,** 261–272. ISSN: 1548-7091, 1548-7105 (Mar. 2020).

8. Kokoska, S. & Zwillinger, D. *CRC Standard Probability and Statistics Tables and Formulae* (Crc Press, 2000).

9. Thompson, M. W., Gilmer, J. B., Matsumoto, R. A., Quach, C. D., Shamaprasad, P., Yang, A. H., Iacovella, C. R., McCabe, C. & Cummings, P. T. Towards Molecular Simulations That Are Transparent, Reproducible, Usable by Others, and Extensible (TRUE). *Molecular Physics* **118,** e1742938. ISSN: 0026-8976, 1362-3028 (June 2020).

Appendix D

High-Throughput Screening of Tribological Properties of Monolayers Films using Molecular Dynamics and Machine Learning

D.1 Previous work of Summers *et al.* [1], chemically dissimilar monolayers

D.1.1 Screening of Chemically Dissimilar Monolayer Films as a Function of Terminal Group Chemistry

In Figure D.2a the COF and adhesion results for systems of both chemically identical (16 in total) and chemically dissimilar (84 in total) films with a backbone length of 17 carbons are shown; numerical values for chemically dissimilar films are included in Table D.2. From a tribological standpoint, the ideal monolayer chemistry should feature both a low COF and low adhesive force, and thus favorable chemistries would exist in the lower left-hand corner of Figure D.2a. Several chemically dissimilar systems appear to provide favorable tribological properties as compared to chemically identical systems, as highlighted in Figure D.2b and Figure D.2c. The majority of these systems features one monolayer that is polar/hydrophilic and another that is nonpolar/hydrophobic. As an example, while carboxyl-terminated monolayers are found to yield the highest adhesive forces for chemically identical systems (see Figure D.3), when paired with a nonpolar counter monolayer the ability to form intermonolayer hydrogen bonds is eliminated, and these combinations of terminal groups are able to provide favorable COF and adhesion values, consistent with prior studies in the literature [2–4]. In particular, favorable tribological properties are found for monolayers featuring nitrile and vinyl terminal groups, as shown in Figure D.2c. While nitrile and vinyl have several differences (chemical composition, polarity), both groups are linear and feature a cylindrical shape with a small VDW radius, and a more rigid nature owed to the presence of a double or triple bonds.

Screening of dual-monolayer systems was performed over two distinct parameter spaces: (1) chemically-identical systems (Figure D.1b, where the top and bottom monolayer films feature the same chemistry) as a function of terminal group and chain length and (2) chemically-dissimilar systems (Figure D.1c, where the top and bottom monolayer films feature different chemistries) for a single fixed chain length of 17 carbons. For each chemistry, five systems were generated, each corresponding to a unique arrangement of chains on the available binding sites on the silica surface, by changing the “seed” provided to the monolayer builder. Properties were evaluated for each of the five replicas and averaged to obtain values for each chemistry that are independent of chain arrangement. For systems featuring chemically-identical films, all 16 terminal group chemistries from Figure D.1a were considered along with five chain lengths (5, 8, 11, 14, and 17 backbone carbons, excluding the terminal group), and three different normal loads (5, 15, 25 nN) resulting in 80 unique chemistries and 1200 simulations in total. Chemically-dissimilar films feature backbone chain lengths of 17 carbons and include select combinations of seven terminal groups (carboxyl, fluorophenyl, hydroxyl, isopropyl, methyl, nitro, and perfluoromethyl) with the 16 terminal groups in Figure D.1a (ignoring duplicates already considered for chemical identical films), for an additional 84 unique chemistries and 1260 total simulations. For the development of machine learning models, the set of 100 unique terminal group combinations (both chemically identical and dissimilar) with a chain back length of 17 carbons are considered. To evaluate the predictive nature of the models, toluene and phenol terminal groups are considered and placed in contact with opposing monolayers composed of the original 16 terminal groups (see Figure D.1a) with chain length 17 carbons. This provides 30 additional unique chemistries (phenyl-phenol and pyrrole-phenol are not considered due to instability of the simulations under high normal loads); 3 normal loads (5, 15, 25 nN) are considered and 3 replicates for each of these systems, for a total of 900 additional simulations; In sum, 3360 simulations are therefore performed, with 194 unique dual monolayer chemistries with the aid of MoSDeF.

To further evaluate the utility of the models in Table D.1, in particular, their ability to pre-screen parameter space, simulations are performed for toluene and phenol terminal groups (not part of the original training set) in contact with the initial 16 terminal groups (see Figure D.1a). Figure D.4c,d plots predicted vs. simulated values of COF and adhesion, where the predicted values represent the average of the predictions of the 5 models in Table D.1 and error bars correspond to the standard deviation of these 5 predictions. The models predict the COF with reasonable accuracy, where most of the predictions are within the standard error of the simulated values and data points visual track the line $y = x$. The ability to predict the force of adhesion values is similar to that observed in Figure D.4b, in that values are well predicted in the lower adhesion regime but underpredicted for higher values; specifically, quantitative agreement is not observed for phenol under conditions where significant hydrogen bonding can occur between the two layers, although, as before, the model does capture the appropriate trends. This validation provides evidence as to the predictive nature of the models and that correlations between chemistry and tribology used by the machine learning model are appropriate. As such, the "feature importances" are extracted from the models to evaluate which elements of the system fingerprint have the most influence over each variable.

Table D.1: Evaluation of random forest regression models for COF and F_0 for systems with carbon backbone length of 17

Model	Target Variable	Training set			Cross-validation (OOB)			Test set		
Number		R^2	RMSE	MAE	R^2	RMSE	MAE	R^2	RMSE	MAE
1	COF	0.9496	0.0037	0.0029	0.6306	0.0100	0.0079	0.6524	0.0090	0.0068
	F_0 (nN)	0.9600	0.2644	0.1383	0.6985	0.7259	0.3782	0.6905	0.9530	0.4237
2	COF	0.9473	0.0036	0.0027	0.6067	0.0099	0.0074	0.6049	0.0110	0.0086
	F_0 (nN)	0.9539	0.3067	0.1473	0.6586	0.8351	0.4046	0.9422	0.3252	0.2439
3	COF	0.9418	0.0037	0.0028	0.5594	0.0101	0.0078	0.6571	0.0114	0.0087
	F_0 (nN)	0.9747	0.2283	0.1233	0.7978	0.6452	0.3410	0.4066	1.0031	0.5579
4	COF	0.9471	0.0038	0.0029	0.6029	0.0104	0.0081	0.8204	0.0062	0.0048
	F_0 (nN)	0.9435	0.2974	0.1406	0.5740	0.8165	0.3844	0.7787	0.8453	0.6076
5	COF	0.9484	0.0038	0.0030	0.6043	0.0106	0.0081	0.5199	0.0096	0.0069
	F_0 (nN)	0.9613	0.2642	0.1388	0.7119	0.7213	0.3797	0.6367	1.0031	0.4699
Avg.	COF	0.9468 ±0.0027	0.0037 ±0.0001	0.0029 ±0.0001	0.6008 ±0.0230	0.0102 ±0.0003	0.0079 ±0.0003	0.6509 ±0.0980	0.0094 ±0.0018	0.0072 ±0.0014
	F_0 (nN)	0.9587 ±0.0102	0.2722 ±0.0278	0.1377 ±0.0079	0.6882 ±0.0729	0.7488 ±0.0693	0.3776 ±0.0206	0.6909 ±0.1759	0.8259 ±0.2569	0.4606 ±0.1261

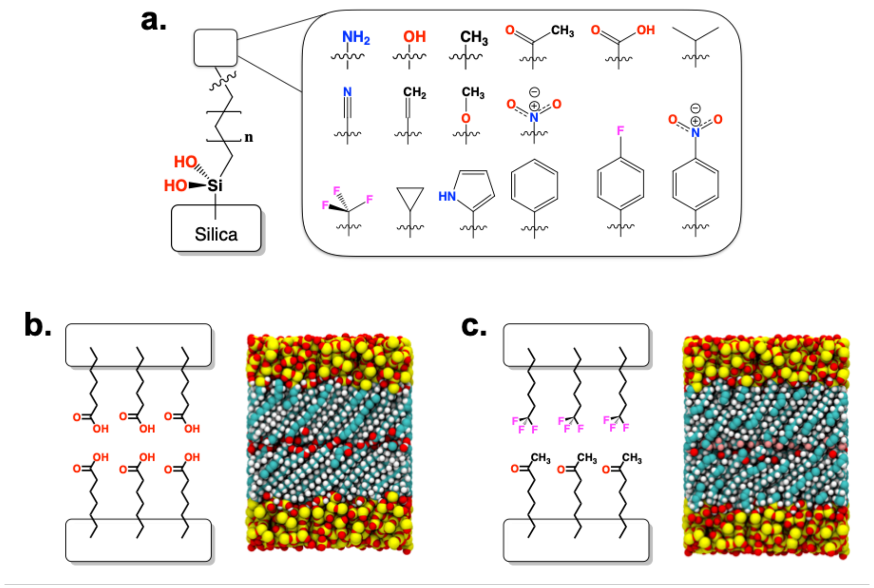


Figure D.1: a.) Overview of the chemical parameter space examined in this work. The pool of terminal group chemistries consists of (row 1) amino, hydroxyl, methyl, acetyl, carboxyl, isopropyl; (row 2) nitrile, vinyl, methoxy, nitro; (row 3) perfluoromethyl, cyclopropyl, 2-pyrrole, phenyl, fluorophenyl, and nitrophenyl. Representative dual-monolayer systems include both b.) chemically-identical and c.) chemically-dissimilar compositions, shown as both schematics and simulation renderings, where oxygen is colored red, silicon yellow, carbon cyan, hydrogen white, and fluorine green.

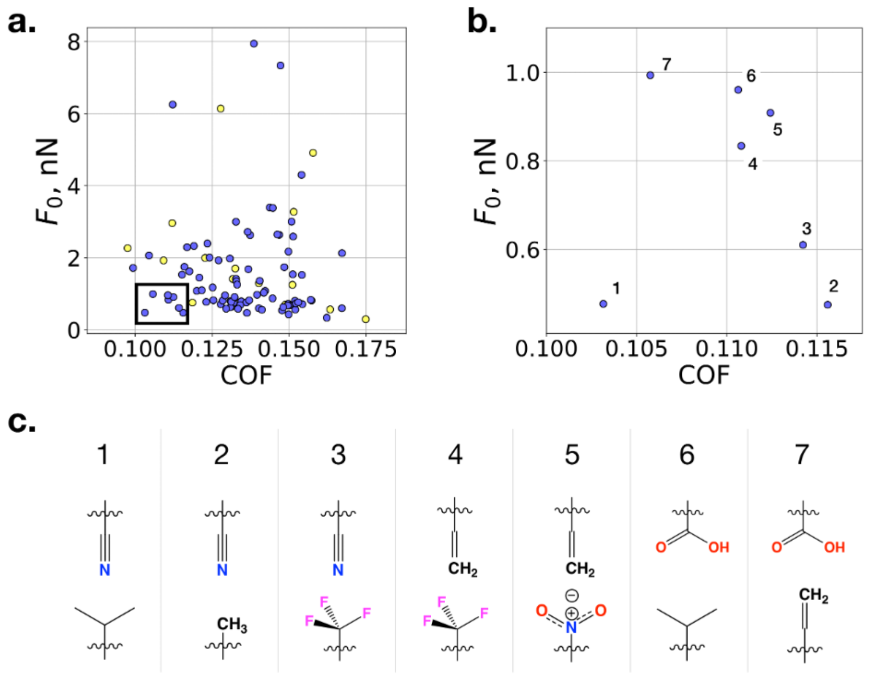


Figure D.2: (a) Scatter plot of COF and adhesion force data for chemically-identical systems (yellow) and chemically-dissimilar systems (blue). The highlighted region indicates the area featuring the most tribologically favorable systems. (b) Plots the data from the highlighted region, where the seven systems are annotated and (c) provides the corresponding terminal group chemistries in the favorable region. These include (1) nitrile-isopropyl, (2) nitrile-methyl, (3) nitrile-perfluoromethyl, (4) vinyl-perfluoromethyl, (5) vinyl-nitro, (6) carboxyl-isopropyl, and (7) carboxyl-vinyl.

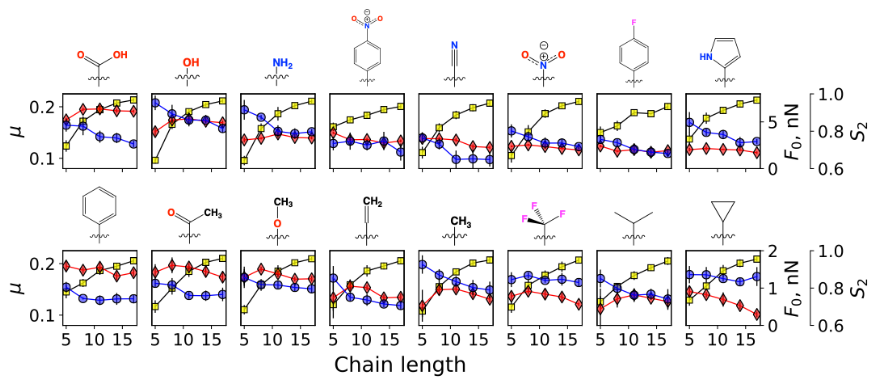


Figure D.3: Coefficient of friction (μ, blue circles), adhesive force (F_0, red diamonds), and nematic order (S_2, yellow squares) for chemically-identical systems shown for each terminal group chemistry as a function of chain length. Note the scale for F_0 differs between the two rows in order to clearly display trends for both polar and nonpolar systems. Error bars represent a single standard deviation calculated from the average of the five chain attachment configurations for each system.

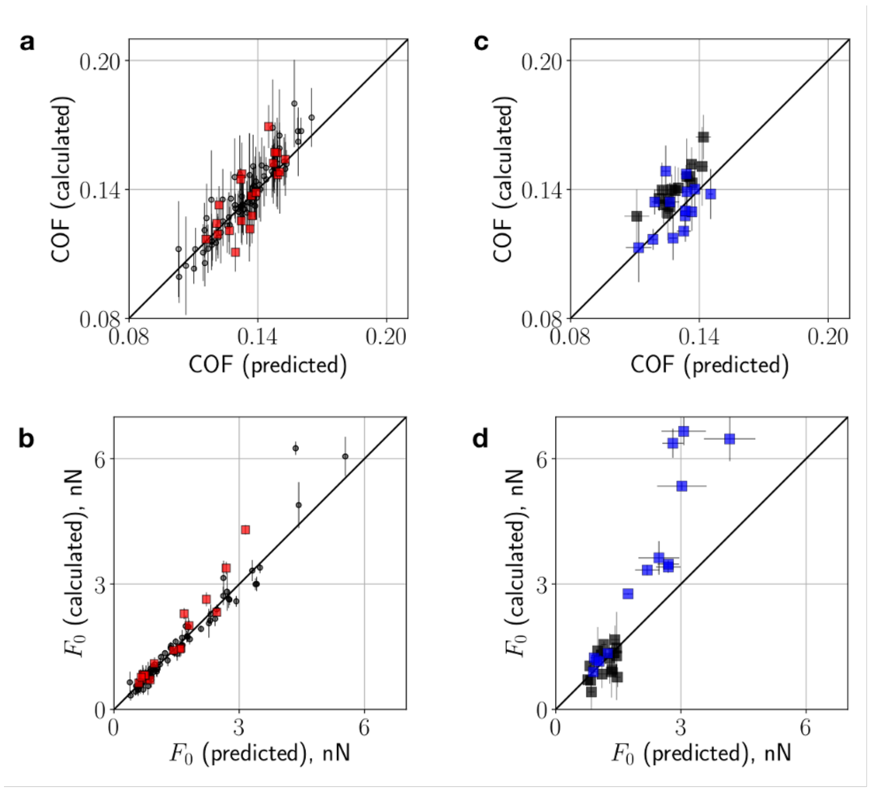


Figure D.4: Values predicted by RF model 1 for a.) COF and b.) adhesive force compared to the values calculated from simulation for systems with 17 carbon backbones. The $y = x$ line is drawn in black for reference. Black circles denote data used as part of the training set, while red squares denote data that was part of the test set. Comparison of predicted values and those calculated from simulation of c.) COF and d.) adhesive force for phenol (black) and toluene (blue).

D.1.2 Summary of chemically dissimilar films and model predictions

Table D.2: Mean and standard deviation of the coefficient of friction (COF) and adhesive force (F_0, nN) of chemically dissimilar films with c17 backbone length. Predictions from the models are also provided, representing the mean and standard deviation of the 5 models.

System	COF Calc.	COF Pred.	F_o Calc. (nN)	F_o Pred. (nN)
acetyl-acetyl	0.147 ± 0.015	0.137 ± 0.003	1.467 ± 0.188	1.642 ± 0.307
acetyl-carboxyl	0.145 ± 0.005	0.137 ± 0.003	3.382 ± 0.150	3.070 ± 0.254
acetyl-fluorophenyl	0.121 ± 0.011	0.125 ± 0.002	1.448 ± 0.202	1.587 ± 0.102
acetyl-hydroxyl	0.150 ± 0.010	0.147 ± 0.001	2.171 ± 0.176	2.562 ± 0.332
acetyl-isopropyl	0.132 ± 0.008	0.132 ± 0.001	0.749 ± 0.141	0.743 ± 0.015
acetyl-methyl	0.137 ± 0.010	0.137 ± 0.001	0.819 ± 0.106	0.772 ± 0.038
acetyl-nitro	0.137 ± 0.005	0.134 ± 0.004	1.683 ± 0.124	1.892 ± 0.146
acetyl-perfluoromethyl	0.150 ± 0.005	0.143 ± 0.003	0.677 ± 0.045	0.695 ± 0.029
amino-amino	0.165 ± 0.022	0.151 ± 0.001	3.323 ± 0.252	3.631 ± 0.348
amino-carboxyl	0.139 ± 0.012	0.139 ± 0.001	7.942 ± 0.140	5.360 ± 1.361
amino-fluorophenyl	0.131 ± 0.005	0.133 ± 0.001	1.978 ± 0.118	1.696 ± 0.208
amino-hydroxyl	0.154 ± 0.012	0.154 ± 0.001	4.300 ± 0.111	3.664 ± 0.313
amino-isopropyl	0.136 ± 0.010	0.140 ± 0.001	0.758 ± 0.105	0.745 ± 0.012
amino-methyl	0.152 ± 0.011	0.149 ± 0.002	0.732 ± 0.144	0.773 ± 0.020
amino-nitro	0.137 ± 0.005	0.137 ± 0.001	2.629 ± 0.132	3.160 ± 0.435
amino-perfluoromethyl	0.157 ± 0.016	0.152 ± 0.003	0.832 ± 0.169	0.751 ± 0.050
carboxyl-carboxyl	0.147 ± 0.018	0.136 ± 0.005	6.055 ± 0.470	5.594 ± 0.239
carboxyl-cyano	0.112 ± 0.010	0.115 ± 0.003	6.251 ± 0.159	3.965 ± 0.928
carboxyl-cyclopropyl	0.136 ± 0.030	0.135 ± 0.001	0.471 ± 0.148	0.580 ± 0.012
carboxyl-ethylene	0.106 ± 0.011	0.115 ± 0.002	0.993 ± 0.102	1.032 ± 0.013
carboxyl-fluorophenyl	0.124 ± 0.012	0.123 ± 0.001	2.005 ± 0.127	1.880 ± 0.075
carboxyl-hydroxyl	0.147 ± 0.014	0.147 ± 0.001	7.335 ± 0.060	5.952 ± 0.647
carboxyl-isopropyl	0.111 ± 0.009	0.125 ± 0.006	0.834 ± 0.110	0.742 ± 0.033
carboxyl-methoxy	0.137 ± 0.014	0.138 ± 0.003	2.718 ± 0.219	2.859 ± 0.245
carboxyl-methyl	0.134 ± 0.013	0.134 ± 0.001	0.689 ± 0.118	0.743 ± 0.030
carboxyl-nitro	0.151 ± 0.015	0.136 ± 0.007	3.002 ± 0.172	3.702 ± 0.280
carboxyl-nitrophenyl	0.133 ± 0.006	0.131 ± 0.002	2.998 ± 0.142	3.257 ± 0.109
carboxyl-perfluoromethyl	0.141 ± 0.014	0.139 ± 0.001	0.558 ± 0.212	0.768 ± 0.042
carboxyl-phenyl	0.118 ± 0.010	0.125 ± 0.003	1.623 ± 0.086	1.478 ± 0.081
carboxyl-pyrrole	0.123 ± 0.004	0.128 ± 0.002	2.395 ± 0.113	2.706 ± 0.309
cyano-cyano	0.112 ± 0.022	0.109 ± 0.006	2.820 ± 0.455	2.477 ± 0.410
cyano-fluorophenyl	0.099 ± 0.012	0.110 ± 0.006	1.717 ± 0.183	1.660 ± 0.075
cyano-hydroxyl	0.144 ± 0.016	0.135 ± 0.004	3.394 ± 0.131	3.295 ± 0.655
cyano-isopropyl	0.103 ± 0.007	0.114 ± 0.006	0.477 ± 0.120	0.613 ± 0.054
cyano-methyl	0.116 ± 0.013	0.125 ± 0.005	0.475 ± 0.114	0.682 ± 0.099
cyano-nitro	0.104 ± 0.023	0.112 ± 0.006	2.063 ± 0.343	2.254 ± 0.298
cyano-perfluoromethyl	0.114 ± 0.010	0.126 ± 0.007	0.610 ± 0.099	0.721 ± 0.071
cyclopropyl-cyclopropyl	0.173 ± 0.014	0.165 ± 0.001	0.650 ± 0.252	0.382 ± 0.007
cyclopropyl-fluorophenyl	0.128 ± 0.013	0.132 ± 0.003	0.711 ± 0.163	0.793 ± 0.042
cyclopropyl-hydroxyl	0.150 ± 0.013	0.153 ± 0.002	0.675 ± 0.153	0.690 ± 0.018
cyclopropyl-isopropyl	0.152 ± 0.015	0.148 ± 0.001	0.561 ± 0.185	0.529 ± 0.010
cyclopropyl-methyl	0.148 ± 0.019	0.151 ± 0.001	0.539 ± 0.151	0.539 ± 0.021

cyclopropyl-nitro	0.131 ± 0.012	0.132 ± 0.001	0.629 ± 0.203	0.648 ± 0.018
cyclopropyl-perfluoromethyl	0.162 ± 0.016	0.160 ± 0.001	0.335 ± 0.123	0.411 ± 0.019
ethylene-ethylene	0.135 ± 0.023	0.120 ± 0.002	0.859 ± 0.278	0.858 ± 0.033
ethylene-fluorophenyl	0.112 ± 0.011	0.118 ± 0.002	0.908 ± 0.111	1.136 ± 0.183
ethylene-hydroxyl	0.132 ± 0.006	0.138 ± 0.001	0.891 ± 0.039	0.975 ± 0.023
ethylene-isopropyl	0.130 ± 0.010	0.130 ± 0.001	0.587 ± 0.076	0.635 ± 0.010
ethylene-methyl	0.128 ± 0.014	0.132 ± 0.000	0.812 ± 0.188	0.787 ± 0.025
ethylene-nitro	0.111 ± 0.013	0.114 ± 0.001	0.960 ± 0.183	1.050 ± 0.024
ethylene-perfluoromethyl	0.133 ± 0.007	0.137 ± 0.002	0.583 ± 0.135	0.662 ± 0.024
fluorophenyl-fluorophenyl	0.121 ± 0.013	0.117 ± 0.002	1.996 ± 0.251	1.731 ± 0.141
fluorophenyl-hydroxyl	0.148 ± 0.012	0.145 ± 0.002	1.738 ± 0.119	1.723 ± 0.104
fluorophenyl-isopropyl	0.131 ± 0.007	0.129 ± 0.001	0.778 ± 0.045	0.840 ± 0.020
fluorophenyl-methoxy	0.140 ± 0.007	0.135 ± 0.003	1.360 ± 0.103	1.479 ± 0.066
fluorophenyl-methyl	0.122 ± 0.008	0.131 ± 0.004	1.098 ± 0.076	0.968 ± 0.063
fluorophenyl-nitro	0.116 ± 0.009	0.119 ± 0.002	1.748 ± 0.138	1.838 ± 0.069
fluorophenyl-nitrophenyl	0.117 ± 0.006	0.117 ± 0.001	2.292 ± 0.132	1.927 ± 0.152
fluorophenyl-perfluoromethyl	0.142 ± 0.005	0.137 ± 0.003	1.031 ± 0.051	0.972 ± 0.049
fluorophenyl-phenyl	0.125 ± 0.011	0.125 ± 0.002	1.175 ± 0.138	1.319 ± 0.109
fluorophenyl-pyrrole	0.115 ± 0.009	0.121 ± 0.002	1.532 ± 0.169	1.640 ± 0.061
hydroxyl-hydroxyl	0.180 ± 0.020	0.156 ± 0.001	4.891 ± 0.550	4.288 ± 0.390
hydroxyl-isopropyl	0.150 ± 0.018	0.148 ± 0.001	0.721 ± 0.133	0.740 ± 0.010
hydroxyl-methoxy	0.167 ± 0.006	0.160 ± 0.001	2.128 ± 0.073	2.414 ± 0.121
hydroxyl-methyl	0.153 ± 0.011	0.152 ± 0.001	0.765 ± 0.151	0.822 ± 0.054
hydroxyl-nitro	0.151 ± 0.012	0.147 ± 0.003	2.588 ± 0.128	3.201 ± 0.283
hydroxyl-nitrophenyl	0.146 ± 0.014	0.143 ± 0.001	2.646 ± 0.220	2.759 ± 0.026
hydroxyl-perfluoromethyl	0.152 ± 0.006	0.154 ± 0.001	0.807 ± 0.144	0.833 ± 0.032
hydroxyl-phenyl	0.154 ± 0.005	0.145 ± 0.003	1.524 ± 0.066	1.329 ± 0.048
hydroxyl-pyrrole	0.147 ± 0.016	0.147 ± 0.001	2.636 ± 0.168	2.521 ± 0.259
isopropyl-isopropyl	0.147 ± 0.019	0.137 ± 0.001	0.668 ± 0.293	0.626 ± 0.020
isopropyl-methoxy	0.145 ± 0.008	0.143 ± 0.002	0.873 ± 0.180	0.766 ± 0.040
isopropyl-methyl	0.140 ± 0.015	0.141 ± 0.001	0.598 ± 0.177	0.607 ± 0.015
isopropyl-nitro	0.123 ± 0.013	0.124 ± 0.003	0.774 ± 0.080	0.759 ± 0.030
isopropyl-nitrophenyl	0.120 ± 0.014	0.123 ± 0.001	1.082 ± 0.148	1.134 ± 0.028
isopropyl-perfluoromethyl	0.150 ± 0.012	0.147 ± 0.002	0.429 ± 0.096	0.547 ± 0.058
isopropyl-phenyl	0.132 ± 0.013	0.131 ± 0.001	0.910 ± 0.141	0.839 ± 0.049
isopropyl-pyrrole	0.131 ± 0.007	0.130 ± 0.001	0.797 ± 0.085	0.801 ± 0.034
methoxy-methoxy	0.159 ± 0.012	0.151 ± 0.002	1.332 ± 0.166	1.440 ± 0.066
methoxy-methyl	0.157 ± 0.014	0.151 ± 0.003	0.813 ± 0.152	0.781 ± 0.021
methoxy-nitro	0.151 ± 0.013	0.137 ± 0.007	1.540 ± 0.163	1.683 ± 0.249
methoxy-perfluoromethyl	0.167 ± 0.006	0.157 ± 0.003	0.599 ± 0.113	0.692 ± 0.045
methyl-methyl	0.169 ± 0.023	0.148 ± 0.001	0.801 ± 0.261	0.668 ± 0.016
methyl-nitro	0.125 ± 0.010	0.130 ± 0.002	0.822 ± 0.082	0.805 ± 0.056
methyl-nitrophenyl	0.133 ± 0.013	0.131 ± 0.002	1.255 ± 0.063	1.175 ± 0.032
methyl-perfluoromethyl	0.148 ± 0.019	0.150 ± 0.001	0.630 ± 0.178	0.598 ± 0.030
methyl-phenyl	0.142 ± 0.015	0.138 ± 0.002	1.080 ± 0.167	0.944 ± 0.021
methyl-pyrrole	0.140 ± 0.011	0.138 ± 0.000	0.972 ± 0.141	0.917 ± 0.048
nitro-nitro	0.135 ± 0.014	0.123 ± 0.001	2.283 ± 0.231	2.276 ± 0.091
nitro-nitrophenyl	0.119 ± 0.010	0.122 ± 0.001	2.326 ± 0.134	2.329 ± 0.089
nitro-perfluoromethyl	0.129 ± 0.015	0.134 ± 0.001	0.957 ± 0.132	0.912 ± 0.032

nitro-phenyl	0.133 ± 0.009	0.126 ± 0.003	1.416 ± 0.109	1.449 ± 0.049
nitro-pyrrole	0.127 ± 0.012	0.127 ± 0.002	1.928 ± 0.072	2.475 ± 0.351
nitrophenyl-nitrophenyl	0.127 ± 0.016	0.121 ± 0.004	3.146 ± 0.414	2.378 ± 0.251
nitrophenyl-perfluoromethyl	0.133 ± 0.006	0.133 ± 0.001	1.347 ± 0.062	1.224 ± 0.022
perfluoromethyl-perfluoromethyl	0.169 ± 0.010	0.150 ± 0.007	0.772 ± 0.148	0.586 ± 0.059
perfluoromethyl-phenyl	0.154 ± 0.010	0.143 ± 0.005	0.712 ± 0.189	0.872 ± 0.128
perfluoromethyl-pyrrole	0.134 ± 0.015	0.137 ± 0.001	0.791 ± 0.121	0.925 ± 0.094
phenyl-phenyl	0.136 ± 0.013	0.129 ± 0.002	1.473 ± 0.140	1.385 ± 0.062
pyrrole-pyrrole	0.146 ± 0.018	0.129 ± 0.001	1.985 ± 0.205	1.891 ± 0.208

D.1.3 Summary of chemically dissimilar models

Table D.3: Mean and standard deviation of the coefficient of friction (COF) and adhesive force (F_0, nN) of chemically dissimilar films with c17 backbone length. Predictions from the models are also provided, representing the mean and standard deviation of the 5 models.

System	**Model 1**	**Model 2**	**Model 3**	**Model 4**	**Model 5**
acetyl - acetyl	Test	**Train**	**Train**	Test	**Train**
acetyl - carboxyl	Test	**Train**	**Train**	**Train**	**Train**
acetyl - fluorophenyl	Test	Test	**Train**	Test	**Train**
acetyl - hydroxyl	**Train**	**Train**	**Train**	Test	**Train**
acetyl - isopropyl	**Train**	**Train**	**Train**	**Train**	**Train**
acetyl - methyl	Test	**Train**	**Train**	**Train**	**Train**
acetyl - nitro	**Train**	**Train**	**Train**	**Train**	Test
acetyl - perfluoromethyl	**Train**	Test	**Train**	**Train**	**Train**
amino - amino	**Train**	Test	**Train**	**Train**	**Train**
amino - carboxyl	Test	**Train**	**Train**	**Train**	Test
amino - fluorophenyl	**Train**	**Train**	Test	**Train**	**Train**
amino - hydroxyl	Test	**Train**	**Train**	**Train**	**Train**
amino - isopropyl	**Train**	**Train**	**Train**	**Train**	**Train**
amino - methyl	Test	Test	**Train**	Test	**Train**
amino - nitro	**Train**	**Train**	**Train**	Test	**Train**
amino - perfluoromethyl	Test	**Train**	**Train**	**Train**	Test
carboxyl - carboxyl	**Train**	Test	**Train**	Test	**Train**
carboxyl - cyano	**Train**	**Train**	Test	**Train**	**Train**
carboxyl - cyclopropyl	**Train**	**Train**	**Train**	**Train**	**Train**
carboxyl - ethylene	**Train**	**Train**	**Train**	**Train**	**Train**
carboxyl - fluorophenyl	Test	**Train**	**Train**	**Train**	**Train**
carboxyl - hydroxyl	**Train**	**Train**	**Train**	Test	**Train**
carboxyl - isopropyl	Test	Test	**Train**	**Train**	**Train**
carboxyl - methoxy	**Train**	**Train**	**Train**	Test	Test
carboxyl - methyl	**Train**	Test	**Train**	Test	**Train**
carboxyl - nitro	**Train**	**Train**	**Train**	**Train**	Test
carboxyl - nitrophenyl	**Train**	Test	**Train**	**Train**	**Train**
carboxyl - perfluoromethyl	**Train**	**Train**	**Train**	Test	**Train**
carboxyl - phenyl	**Train**	**Train**	**Train**	Test	**Train**

carboxyl - pyrrole	**Train**	**Train**	**Train**	Test	**Train**
cyano - cyano	**Train**	Test	**Train**	Test	**Train**
cyano - fluorophenyl	**Train**	**Train**	Test	**Train**	**Train**
cyano - hydroxyl	**Train**	**Train**	Test	**Train**	**Train**
cyano - isopropyl	**Train**	Test	Test	**Train**	**Train**
cyano - methyl	**Train**	**Train**	**Train**	**Train**	Test
cyano - nitro	**Train**	**Train**	**Train**	**Train**	**Train**
cyano - perfluoromethyl	**Train**	**Train**	Test	**Train**	**Train**
cyclopropyl - cyclopropyl	**Train**	**Train**	**Train**	**Train**	**Train**
cyclopropyl - fluorophenyl	Test	**Train**	**Train**	**Train**	**Train**
cyclopropyl - hydroxyl	**Train**	**Train**	Test	**Train**	**Train**
cyclopropyl - isopropyl	**Train**	**Train**	**Train**	**Train**	**Train**
cyclopropyl - methyl	**Train**	**Train**	**Train**	**Train**	**Train**
cyclopropyl - nitro	**Train**	**Train**	**Train**	**Train**	Test
cyclopropyl - perfluoromethyl	**Train**	**Train**	**Train**	**Train**	**Train**
ethylene - ethylene	**Train**	**Train**	Test	**Train**	**Train**
ethylene - fluorophenyl	**Train**	**Train**	**Train**	**Train**	Test
ethylene - hydroxyl	**Train**	**Train**	**Train**	**Train**	**Train**
ethylene - isopropyl	**Train**	**Train**	**Train**	**Train**	**Train**
ethylene - methyl	**Train**	**Train**	**Train**	**Train**	**Train**
ethylene - nitro	**Train**	**Train**	**Train**	**Train**	**Train**
ethylene - perfluoromethyl	**Train**	**Train**	**Train**	**Train**	**Train**
fluorophenyl - fluorophenyl	**Train**	**Train**	Test	**Train**	**Train**
fluorophenyl - hydroxyl	**Train**	**Train**	Test	**Train**	**Train**
fluorophenyl - isopropyl	**Train**	Test	**Train**	**Train**	**Train**
fluorophenyl - methoxy	**Train**	Test	**Train**	**Train**	**Train**
fluorophenyl - methyl	Test	Test	**Train**	**Train**	**Train**
fluorophenyl - nitro	**Train**	**Train**	Test	Test	**Train**
fluorophenyl - nitrophenyl	Test	**Train**	**Train**	**Train**	**Train**
fluorophenyl - perfluoromethyl	**Train**	**Train**	**Train**	**Train**	**Train**
fluorophenyl - phenyl	**Train**	**Train**	**Train**	**Train**	Test
fluorophenyl - pyrrole	**Train**	**Train**	**Train**	**Train**	**Train**
hydroxyl - hydroxyl	**Train**	**Train**	**Train**	Test	**Train**
hydroxyl - isopropyl	**Train**	**Train**	**Train**	**Train**	**Train**
hydroxyl - methoxy	**Train**	**Train**	**Train**	**Train**	**Train**
hydroxyl - methyl	**Train**	**Train**	**Train**	**Train**	Test
hydroxyl - nitro	**Train**	**Train**	Test	**Train**	**Train**
hydroxyl - nitrophenyl	**Train**	**Train**	**Train**	**Train**	**Train**
hydroxyl - perfluoromethyl	**Train**	**Train**	**Train**	**Train**	Test
hydroxyl - phenyl	**Train**	**Train**	**Train**	**Train**	**Train**
hydroxyl - pyrrole	Test	**Train**	Test	**Train**	**Train**

isopropyl - isopropyl	**Train**	**Train**	**Train**	**Train**	**Train**
isopropyl - methoxy	**Train**	**Train**	**Train**	**Train**	Test
isopropyl - methyl	**Train**	**Train**	**Train**	**Train**	**Train**
isopropyl - nitro	**Train**	Test	**Train**	**Train**	**Train**
isopropyl - nitrophenyl	**Train**	**Train**	**Train**	**Train**	**Train**
isopropyl - perfluoromethyl	**Train**	Test	**Train**	**Train**	Test
isopropyl - phenyl	**Train**	**Train**	**Train**	Test	**Train**
isopropyl - pyrrole	**Train**	**Train**	**Train**	**Train**	Test
methoxy - methoxy	**Train**	Test	Test	**Train**	**Train**
methoxy - methyl	Test	**Train**	**Train**	Test	**Train**
methoxy - nitro	**Train**	**Train**	Test	**Train**	**Train**
methoxy - perfluoromethyl	**Train**	**Train**	Test	**Train**	**Train**
methyl - methyl	**Train**	**Train**	Test	**Train**	Test
methyl - nitro	Test	**Train**	**Train**	Test	Test
methyl - nitrophenyl	**Train**	**Train**	Test	**Train**	**Train**
methyl - perfluoromethyl	Test	**Train**	**Train**	**Train**	**Train**
methyl - phenyl	**Train**	Test	**Train**	**Train**	**Train**
methyl - pyrrole	**Train**	**Train**	**Train**	**Train**	**Train**
nitro - nitro	**Train**	**Train**	**Train**	**Train**	**Train**
nitro - nitrophenyl	Test	**Train**	**Train**	**Train**	**Train**
nitro - perfluoromethyl	**Train**	**Train**	**Train**	**Train**	**Train**
nitro - phenyl	Test	**Train**	**Train**	**Train**	**Train**
nitro - pyrrole	**Train**	Test	Test	Test	**Train**
nitrophenyl - nitrophenyl	**Train**	Test	Test	**Train**	Test
nitrophenyl - perfluoromethyl	**Train**	**Train**	**Train**	**Train**	**Train**
perfluoromethyl - perfluoromethyl	Test	Test	**Train**	**Train**	**Train**
perfluoromethyl - phenyl	**Train**	**Train**	**Train**	**Train**	Test
perfluoromethyl - pyrrole	**Train**	**Train**	**Train**	**Train**	Test
phenyl - phenyl	**Train**	**Train**	**Train**	**Train**	Test
pyrrole - pyrrole	**Train**	**Train**	**Train**	Test	**Train**

D.1.4 Summary of properties of toluene and phenol containing films

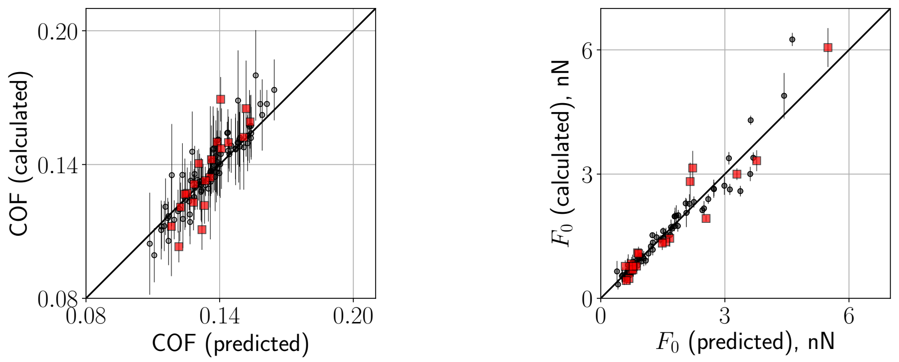


Figure D.5: Plots of the predicted values vs. values calculated from simulation for COF and F_0 for model 2; red data points represent those reserved for testing. Data points represent the mean and error bars represent the standard deviation of the calculated values (see Table D.2).

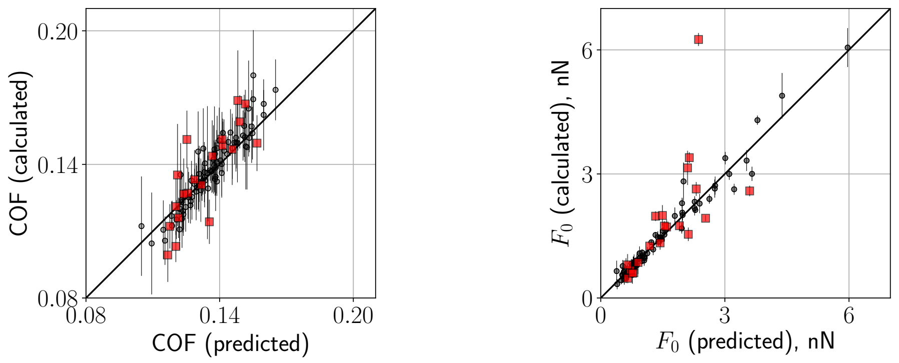


Figure D.6: Plots of the predicted values vs. values calculated from simulation for COF and F_0 for for model 3; red data points represent those reserved for testing. Data points represent the mean and error bars represent the standard deviation of the calculated values (see Table D.2).

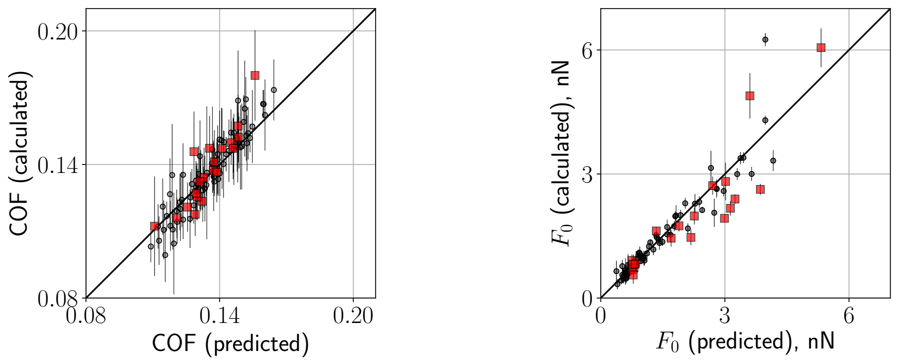


Figure D.7: Plots of the predicted values vs. values calculated from simulation for COF and F_0 for model 4; red data points represent those reserved for testing. Data points represent the mean and error bars represent the standard deviation of the calculated values (see Table D.2).

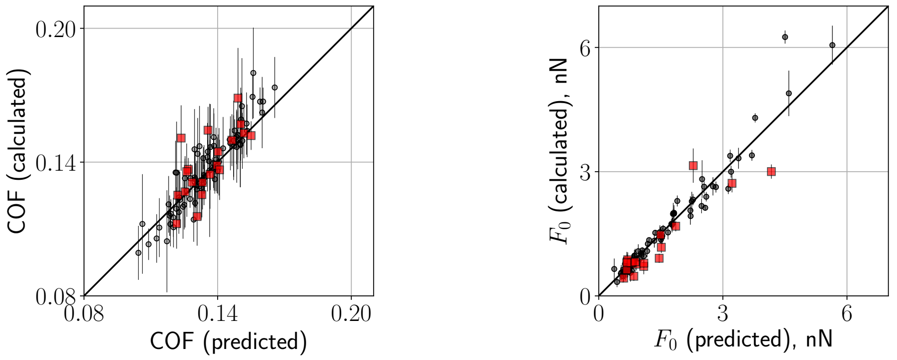


Figure D.8: Plots of the predicted values vs. values calculated from simulation for COF and F_0 for model 5; red data points represent those reserved for testing. Data points represent the mean and error bars represent the standard deviation of the calculated values (see Table D.2).

Table D.4: Mean and standard deviation of the COF and F_0 for chemically dissimilar films containing toluene and phenol, comparing the calculated values from simulation to those predicted by the models. Values represents the mean and standard deviation.

System	COF Calc.	COF Pred.	F_o Calc. (nN)	F_o Pred. (nN)
acetyl-toluene	0.140 ± 0.007	0.127 ± 0.001	1.305 ± 0.161	1.095 ± 0.090
acetyl-phenol	0.128 ± 0.006	0.133 ± 0.002	3.627 ± 0.403	2.474 ± 0.491
fluorophenyl-toluene	0.140 ± 0.013	0.123 ± 0.003	0.958 ± 0.302	1.352 ± 0.111
fluorophenyl-phenol	0.134 ± 0.006	0.119 ± 0.001	2.766 ± 0.013	1.734 ± 0.115
nitro-toluene	0.133 ± 0.009	0.123 ± 0.002	1.280 ± 1.054	1.460 ± 0.083
nitro-phenol	0.148 ± 0.012	0.124 ± 0.002	3.408 ± 0.137	2.698 ± 0.295
amino-toluene	0.143 ± 0.004	0.137 ± 0.001	1.559 ± 0.071	1.155 ± 0.063
amino-phenol	0.130 ± 0.009	0.136 ± 0.002	6.656 ± 0.333	3.068 ± 0.532
methyl-toluene	0.146 ± 0.007	0.135 ± 0.002	0.700 ± 0.374	0.840 ± 0.043
methyl-phenol	0.130 ± 0.014	0.134 ± 0.002	1.180 ± 0.189	0.980 ± 0.075
phenyl-toluene	0.139 ± 0.011	0.129 ± 0.002	0.903 ± 0.617	1.338 ± 0.085
ethylene-toluene	0.129 ± 0.013	0.125 ± 0.001	0.847 ± 0.336	1.109 ± 0.141
ethylene-phenol	0.117 ± 0.005	0.118 ± 0.002	1.345 ± 0.110	1.250 ± 0.142
isopropyl-toluene	0.141 ± 0.009	0.130 ± 0.001	1.041 ± 0.091	0.818 ± 0.036
isopropyl-phenol	0.117 ± 0.010	0.128 ± 0.002	1.234 ± 0.069	0.925 ± 0.089
nitrophenyl-toluene	0.136 ± 0.006	0.121 ± 0.001	1.668 ± 0.339	1.421 ± 0.105
nitrophenyl-phenol	0.134 ± 0.009	0.126 ± 0.004	3.477 ± 0.092	2.700 ± 0.257
carboxyl-toluene	0.136 ± 0.005	0.126 ± 0.001	1.468 ± 0.098	1.459 ± 0.070
carboxyl-phenol	0.121 ± 0.002	0.133 ± 0.003	6.476 ± 0.532	4.170 ± 0.616
perfluoromethyl-toluene	0.151 ± 0.018	0.141 ± 0.003	0.419 ± 0.558	0.855 ± 0.080
perfluoromethyl-phenol	0.140 ± 0.013	0.138 ± 0.003	1.152 ± 0.064	1.038 ± 0.079
hydroxyl-toluene	0.164 ± 0.010	0.142 ± 0.004	1.219 ± 0.601	1.299 ± 0.051
hydroxyl-phenol	0.138 ± 0.012	0.145 ± 0.001	6.372 ± 0.349	2.808 ± 0.251
cyclopropyl-toluene	0.152 ± 0.008	0.136 ± 0.002	0.709 ± 0.097	0.768 ± 0.046
cyclopropyl-phenol	0.147 ± 0.004	0.134 ± 0.003	0.899 ± 0.068	0.902 ± 0.104
pyrrole-toluene	0.132 ± 0.002	0.126 ± 0.002	1.351 ± 0.245	1.401 ± 0.066
methoxy-toluene	0.146 ± 0.018	0.134 ± 0.002	1.406 ± 0.567	1.005 ± 0.038
methoxy-phenol	0.139 ± 0.006	0.134 ± 0.002	3.337 ± 0.122	2.195 ± 0.291
cyano-toluene	0.128 ± 0.012	0.111 ± 0.006	0.773 ± 0.235	1.483 ± 0.075
cyano-phenol	0.113 ± 0.016	0.112 ± 0.006	5.346 ± 0.096	3.024 ± 0.592

D.2 Using the released source code

Following the best practices described by TRUE principles (Transferable, Reproducible, Usable by others, and Extensible) [5], the code used to perform the molecular dynamics (MD) simulations and data analysis with machine learning (ML) are released along with the publication. The code is archived and uploaded to Zenodo for reference while a more light-weight, accessible version is hosted on GitHub [6]. To replicate the study utilizing this repository in a TRUE fashion, one must install all software used during this study. Below are the necessary details to set up and install a Python environment needed to operate on the repository. The environment is managed partly with the scientific software manager, `conda`, while a few python libraries need to be compiled from source. Since the simulation portion of this study was conducted on `GNU/Linux` and the data analysis was performed on Apple `MacOS` operating systems, these operating systems will be the main target in the instructions below.

D.3 Installation of the `conda` Package Manager

To install `conda` to a local machine, run the following commands in your terminal based on your operating system (note that the initial "$" is meant to denote a line on the command line):

D.3.1 `MacOS`

```
$ cd ${HOME}
$ curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
$ /bin/bash Miniconda3-latest-MacOSX-x86_64.sh
```

D.3.2 `Linux`

```
$ cd ${HOME}
$ curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ /bin/bash Miniconda3-latest-Linux-x86_64.sh
```

D.4 Cloning the repository and creating the python environment

Once the `conda` package manager has been installed, the reader can proceed to clone the repository and create a working environment. We note, due to the sheer size of the simulation data, only the analysis routines, along with the summarized data, is hosted on GitHub for easy access; the full simulation workflow is archived and uploaded Zenodo.

`MacOS` and `Linux`:

```
$ git clone https://github.com/daico007/iMoDELS-supplements.git
$ cd iModels-supplements
$ conda install -c conda-forge mamba
$ mamba env create --file env.yml
$ conda activate screening
```

D.5 Utilizing the repository

The `iModels-supplements` repository includes the summarized data generated from the MD simulations and routines used to train and evaluate the efficacy of the ML models. The raw data contains information regarding the systems as well as the calculated tribological properties, while the analysis routines can be found in a collection of Python scripts (used to perform the training and evaluation of the ML models) and `Jupyter` notebooks (used to plot and visualize data). We have set up a `Jupyter` notebook, named `Data-Lookup.ipynb`, to specifically assist with 1) Looking up data generated from MD and 2) utilize the trained ML models to predict tribological properties of new systems. More details of the structure of the repository can be found on the GitHub page [6].

D.6 Additional force field details

This study utilized the Optimized Potential for Liquid System - All Atom, consistent with that used in Summers *et al.*, study [1, 7, 8]. Beyond parameters for chemistries studied in the previous work, interaction parameters for the three new chemistries (toluene, phenol, and difluoromethyl) are presented below. The complete list of parameters is stored in a `foyer`-compatible XML file, named `oplsaa.xml`, and included with the workflow repository [6].

D.6.1 Force field parameters

D.6.1.1 Toluene

Table D.5: Toluene nonbonded parameters.

Nonbonded parameters					
Atom Type	Element	Charge	Sigma, Å	Epsilon, kcal mol^{-1}	Reference
opls_140	H	0.06	2.5	0.03	[7]
opls_148	C	-0.065	3.5	0.066	[7]
opls_145	C	-0.115	3.55	0.07	[7]
opls_146	H	0.115	2.42	0.03	[7]

Table D.6: Toluene bonded parameters.

Harmonic Bond parameters				
Bond	Elements	k, kcal mol^{-1} Å^{-2}	r_0, Å	Reference
opls_149-opls_140	C-H	340	1.09	[7]
opls_145-opls_148	C-C	317	1.51	[7]
opls_145-opls_145	C-C	469	1.4	[7]
opls_145-opls_149	C-C	317	1.51	[7]
opls_140-opls_148	H-C	340	1.09	[7]

Table D.7: Toluene angle parameters.

Harmonic Angle parameters				
Angle	Elements	k, $\text{kcal}\,\text{mol}^{-1}\,\text{deg}^{-2}$	θ_0, deg	Reference
opls_149-opls_145-opls_145	C-C-C	70	120	[7]
opls_140-opls_149-opls_145	H-C-C	35	109.5	[7]
opls_136-opls_149-opls_145	C-C-C	63	114	[7]
opls_148-opls_145-opls_145	C-C-C	70	120	[7]
opls_145-opls_148-opls_140	C-C-H	35	109.5	[7]
opls_145-opls_145-opls_145	C-C-C	63	120	[7]
opls_145-opls_145-opls_146	C-C-H	35	120	[7]
opls_140-opls_148-opls_140	H-C-H	33	107.8	[7]

Table D.8: Toluene dihedral parameters.

Dihedral parameters						
Dihedral	Elements	k_1	k_2	k_3	k_4	Reference
				$\text{kcal}\,\text{mol}^{-1}$		
opls_149-opls_145-opls_145-opls_145	C-C-C-C	0	7.25	0	0	[7]
opls_149-opls_145-opls_145-opls_146	C-C-C-H	0	7.25	0	0	[7]
opls_140-opls_149-opls_145-opls_145	H-C-C-C	0	0	0	0	[7]
opls_136-opls_149-opls_145-opls_145	C-C-C-C	0	0	0	0	[7]
opls_140-opls_136-opls_149-opls_145	H-C-C-C	-1.2×10^{-6}	0	0.462	0	[7]
opls_148-opls_145-opls_145-opls_145	C-C-C-C	0	7.25	0	0	[7]
opls_148-opls_145-opls_145-opls_146	C-C-C-H	0	7.25	0	0	[7]
opls_145-opls_145-opls_145-opls_145	C-C-C-C	0	7.25	0	0	[7]
opls_145-opls_145-opls_145-opls_146	C-C-C-H	0	7.25	0	0	[7]
opls_145-opls_145-opls_148-opls_140	C-C-C-H	0	0	0	0	[7]
opls_146-opls_145-opls_145-opls_146	H-C-C-H	0	7.25	0	0	[7]

Table D.9: Toluene improper parameters.

Improper parameters[1]					
Improper[1]	Elements	K_ϕ, $\text{kcal}\,\text{mol}^{-1}$	n	γ, deg	Reference
opls_148-opls_145-opls_145-opls_145	C-C-C-C	1.1	2	180	[7]
opls_145-opls_145-opls_145-opls_146	C-C-C-H	1.1	2	180	[7]
opls_149-opls_145-opls_145-opls_145	C-C-C-C	1.1	2	180	[7]

[1] Dihedral OPLS parameters are converted from Ryckaert-Bell parameters stored in the “oplsaa.xml”.

D.6.1.2 Phenol

Table D.10: Phenol nonbonded parameters.

Nonbonded parameters					
Atom Type	Element	Charge	Sigma, Å	Epsilon, $\mathrm{kcal\,mol^{-1}}$	Reference
opls_145	C	-0.115	3.55	0.07	[7]
opls_166	C	0.15	3.55	0.07	[7]
opls_167	O	-0.585	3.07	0.17	[7]
opls_146	H	0.115	2.42	0.03	[7]
opls_168	H	0.435	10	0.0	[7]

Table D.11: Phenol bonded parameters.

Harmonic Bond parameters				
Bond	Elements	k, $\mathrm{kcal\,mol^{-1}\,Å^{-2}}$	r_0, Å	Reference
opls_145-opls_149	C-C	317	1.51	[7]
opls_145-opls_145	C-C	469	1.4	[7]
opls_145-opls_166	C-C	469	1.4	[7]
opls_167-opls_166	O-C	450	1.364	[7]
opls_146-opls_145	H-C	367	1.08	[7]
opls_168-opls_167	H-O	553	0.945	[7]

Table D.12: Phenol angle parameters.

Harmonic Angle parameters				
Angle	Elements	k, $\mathrm{kcal\,mol^{-1}\,deg^{-2}}$	θ_0, deg	Reference
opls_149-opls_145-opls_145	C-C-C	70	120	[7]
opls_140-opls_149-opls_145	H-C-C	35	109.5	[7]
opls_136-opls_149-opls_145	C-C-C	63	114	[7]
opls_145-opls_145-opls_145	C-C-C	63	120	[7]
opls_145-opls_145-opls_146	C-C-H	35	120	[7]
opls_145-opls_145-opls_166	C-C-C	63	120	[7]
opls_145-opls_166-opls_145	C-C-C	63	120	[7]
opls_145-opls_166-opls_167	C-C-O	70	120	[7]
opls_166-opls_145-opls_146	C-C-H	35	120	[7]
opls_166-opls_145-opls_145	C-C-C	63	120	[7]
opls_166-opls_167-opls_168	C-O-H	35	113	[7]

Table D.13: Phenol dihedral parameters.

Dihedral parameters						
Dihedral	Elements	k_1	k_2	k_3	k_4	Reference
		$\mathrm{kcal\,mol^{-1}}$				
opls_149-opls_145-opls_145-opls_145	C-C-C-C	0	7.25	0	0	[7]
opls_149-opls_145-opls_145-opls_146	C-C-C-H	0	7.25	0	0	[7]
opls_140-opls_149-opls_145-opls_145	H-C-C-C	0	0	0	0	[7]
opls_136-opls_149-opls_145-opls_145	C-C-C-C	0	0	0	0	[7]
opls_140-opls_136-opls_149-opls_145	H-C-C-C	-1.20E-06	0	0.46199928	0	[7]
opls_145-opls_145-opls_145-opls_166	C-C-C-C	0	7.25	0	0	[7]
opls_145-opls_145-opls_145-opls_146	C-C-C-H	0	7.25	0	0	[7]
opls_145-opls_145-opls_145-opls_145	C-C-C-C	0	7.25	0	0	[7]
opls_145-opls_145-opls_166-opls_145	C-C-C-C	0	7.25	0	0	[7]
opls_145-opls_145-opls_166-opls_167	C-C-C-O	0	7.25	0	0	[7]
opls_145-opls_166-opls_145-opls_145	C-C-C-C	0	7.25	0	0	[7]
opls_145-opls_166-opls_145-opls_146	C-C-C-H	0	7.25	0	0	[7]
opls_145-opls_166-opls_167-opls_168	C-C-O-H	0	1.68200048	0	0	[7]
opls_166-opls_145-opls_145-opls_146	C-C-C-H	0	7.25	0	0	[7]
opls_167-opls_166-opls_145-opls_146	O-C-C-H	0	7.25	0	0	[7]
opls_146-opls_145-opls_145-opls_146	H-C-C-H	0	7.25	0	0	[7]

Table D.14: Phenol improper parameters.

Improper[1]	Elements	K_ϕ, kcal mol^{-1}	n	γ, deg	Reference
opls_149-opls_145-opls_145-opls_145	C-C-C-C	1.1	2	180	[7]
opls_145-opls_145-opls_145-opls_146	C-C-C-H	1.1	2	180	[7]
opls_145-opls_166-opls_145-opls_146	C-C-C-H	1.1	2	180	[7]
opls_145-opls_145-opls_166-opls_167	C-C-C-O	1.1	2	180	[7]
opls_166-opls_145-opls_145-opls_146	C-C-C-H	1.1	2	180	[7]

Improper parameters[1]

[1] Dihedral OPLS parameters are converted from Ryckaert-Bell parameters stored in the “oplsaa.xml”.

D.6.1.3 Difluoromethyl

Table D.15: Difluoromethyl nonbonded parameters.

Atom Type	Element	Charge	Sigma, Å	Epsilon, kcal mol^{-1}	Reference
opls_140	H	0.06	2.5	0.03	[7]
opls_962	C	0.24	3.5	0.066	[7]
opls_965	F	-0.12	2.95	0.053	[7]

Nonbonded parameters

Table D.16: Difluoromethyl bonded parameters.

Bond	Elements	k, kcal mol^{-1} Å^{-2}	r_0, Å	Reference
opls_136-opls_140	C-H	340	1.09	[7]
opls_140-opls_136	H-C	340	1.09	[7]
opls_136-opls_140	C-H	340	1.09	[7]
opls_140-opls_136	H-C	340	1.09	[7]
opls_1004-opls_140	C-H	340	1.09	[7]
opls_140-opls_1004	H-C	340	1.09	[7]
opls_962-opls_136	C-C	268	1.529	[7]
opls_965-opls_962	F-C	367	1.332	[7]
opls_965-opls_962	F-C	367	1.332	[7]
opls_140-opls_962	H-C	340	1.09	[7]

Harmonic Bond parameters

Table D.17: Difluoromethyl angle parameters.

Angle	Elements	k, kcal mol^{-1} deg^{-2}	θ_0, deg	Reference
opls_136-opls_962-opls_965	C-C-F	50	109.5	[7]
opls_136-opls_962-opls_140	C-C-H	37.5	110.7	[7]
opls_140-opls_136-opls_962	H-C-C	37.5	110.7	[7]
opls_136-opls_136-opls_962	C-C-C	58.3500239	112.7	[7]
opls_965-opls_962-opls_965	F-C-F	77	109.1	[7]
opls_965-opls_962-opls_140	F-C-H	40	107	[7]

Harmonic Angle parameters

Table D.18: Difluoromethyl dihedral parameters.

Dihedral	Elements	k_1	k_2	k_3	k_4	Reference
		kcal mol^{-1}				
opls_140-opls_136-opls_962-opls_965	H-C-C-F	0	0	0.4	0	[7]
opls_140-opls_136-opls_962-opls_140	H-C-C-H	0	0	0.3	0	[7]
opls_136-opls_136-opls_962-opls_965	C-C-C-F	0.3	0	0.4	0	[7]
opls_136-opls_136-opls_962-opls_140	C-C-C-H	0	0	0.3	0	[7]
opls_140-opls_136-opls_136-opls_962	H-C-C-C	0	0	0.3	0	[7]

Dihedral parameters

In this study, we used the Random Forest Regressor algorithm [9] as implemented by `scikit-learn` as the `RandomForestRegressor` [10] and hence, we can optimize parameters that are exposed to the users. Of those, we have considered the number of trees, max depth, and max features, which are known to have effects on the accuracy as well as the speed/performance of the algorithm. We perform optimization by training and evaluating the models, which are trained with three data amounts (100, 1000, 7816), as a function of different parameters. The results are shown in the following figures, with Figure D.9 showing the effects of different number of trees (`n_estimators`), Figure D.10 showing the maximum depth of each tree (`max_depth`), and the maximum number of features to consider when looking for the best split (`max_features`) in Figure D.11. In each plot, the dashed, horizontal lines represent the performance of the models trained with the parameters used by Summers *et al.*, and by the models in the body of this paper. We can see that, for all plots, at all training sizes, the performance of the models trained with the parameters utilized previously Summers *et al.* are highly performant even when compared to these optimized parameters. There is minor improvement for using a smaller number of trees when training models with 100 training points, though the improvement is not significant. Hence, for the sake of consistency and ability to directly compare the modes from the Summers *et al.* paper, we have re-used the parameters utilized by Summers *et al* [1].

D.7 Feature importance

An advantage provided by the random forest algorithm is the ability to interrogate how each model utilized the provided features, called feature importance. This allows for interpretation and classification of features that strongly affect the predictions made by the model, and by extension, determine the molecular descriptors that strongly affect the system's property of interest. As data are provided to train the model, we observe improvement in the model regarding its ability to predict tribological properties of dual-monolayer systems from different test sets, as pointed out in the main text. However, the driving force behind those improvements have not been fully discussed. Here, we provide a comparison between the feature importances' of several models, trained with various training set sizes, to note any changes during this process.

The feature importance rankings of COF and F_0 for different models, trained with varying amounts of training data are shown in Figure D.12-Figure D.15. These set of plots show the process in which the random forest algorithm ranks each parameter and show how the feature importance rankings shift as more/less data are used to train the models. Comparing between COF models, while the highest-ranking feature, the *hk-alpha*, is quite consistent across the models, the other feature positions shuffle. The adjustments in the feature importance ranking reflect the changes that occur as the forest of decision trees finds descriptors that more evenly split the data. This explains why there was still a steady positive correlation between the amount of data and the predictive ability of the COF models. Between F_0 models, we note the position of the two highest ranking features, *hbonds* and *tpsa-min* remain unchanged, though there are some shuffling among the lesser ranked features. Among these two features, we observed a gradual adjustment in their relative importance. However, the conclusions drawn from these different feature importance rankings remain consistent with Summers *et al.* [1], for both the COF and F_0 models, that the COF is mostly impacted by the shape and size of the terminal group chemistries, while F_0 is strongly affected by the terminal group charge distribution.

During the testing and evaluation of the random forest regressor model, an additional study on the effect of training set distribution was explored. Since a random forest regressor depends on randomly sampling from the training data, if the training data is from a biased dataset, the predictions from the model will also be biased. To explore this, we prepared 3 biased datasets of 1000 points selected from extrema of our data. Specifically, we created biased models using (1) 500 lowest values and 500 highest values, (2) 1000 highest values, (3) 1000 lowest values of the *total-train* set described in the main text. We then employed the same training methodology as previously described. Note that training set size was held constant at 1000 points. The models' performance was quantified by their ability to capture the trends/distribution of their test set. The difference between the data distribution predicted by ML models against reference data generated through MD simulations can be quantified by using KL divergence, which measures the distance between two distributions, and visualized by the violin plots in Figure D.16. The reference data, those calculated directly from MD simulations, are represented by the blue portion of the violin plot. Well-performing models are the violin plots that reproduce a similar overall data distribution to the blue portion of the violin plot. It is very apparent that biased distributions of data dramatically impact the predictive ability of these models. For example, the dataset emulating a bimodal distribution from the very extreme values of COF and F_0 led to a predictive model that also approximated a bimodal distribution.

This effect may make it difficult for certain strategies that involve developing ML models from minimal data, depending on the distribution. We compare the distribution of frictional properties of systems in the *5050-test* set computed from simulations and the distribution predicted by the ML models (one trained with the Summers *et al.* data set and one trained with 100 data points from the *5050-train* data set) and present the results in Figure D.17. We note that the Summers *et al.* models do not fully capture the MD-defined distributions for either COF or F_0; and, in both cases, fails to mimic the tails of the distributions. ML models trained on a subset of the *5050-train*, on the other hand, closely follow the simulation data. This improvement is likely due to the inclusion of systems with mixed monolayers, which subsequently improve the distribution of the utilized training set. The improvement in distribution induced by inclusion of more diverse systems explains the increase in accuracy between the ML models trained with 100 data points from the *total-train* or the *5050-train* sets with those trained with data from the Summers *et al.* data set.

However, we note, even for the training sets with well-distributed data, including more data could still help improve the accuracy of the ML models. This point is demonstrated in Figure D.18 where we compare the ability of the model trained with 100 data points and the model trained with 1000 data points from the *5050-train* data set. These training sets are created such that they capture a similar distribution, closely following the distribution of the complete data (see the *Methods* section). We can see that the models trained with 1000 data points still demonstrate improvement in their ability to capture the distribution of the *5050-test* set, both by the shapes of their distributions and the alignment of the medians and quantiles value, indicating data training set size still greatly affect the accuracy of the ML models. However, we note, larger training set sizes would also increase variability, and thus it is not possible to fully separate out these two effects. The benefit of adding more data in the training set is further discussed in the body of this work (the *Results* section).

D.8 Additional metrics

In addition to the two metrics reported in previously in the body of this work (coefficient of determination, R^2, and mean average percentage error, MAPE), we also include other popular metrics, mean absolute error (MAE) and mean squared error (MSE) below in Table D.20 and Table D.21 [11]. We note the trends presented in these metrics are consistent with conclusions drawn using the other metrics, i.e., R^2 and MAPE, as presented in the body of this work.

Table D.19: Molecular descriptors from `RDKit` [12] .

Molecular descriptor	Description	Category
Approximate Surface Area	Approximation of molecular surface area using the approach defined by Labute [13]	Size
Asphericity	Measure of molecular shape (from Baumgartner [14]); $A = 0$ for spherical shape, $A = 1$ for highly prolate shapes, and $A = 0 : 25$ for oblate shapes	Shape
Balaban J	Related to connectivity, degree of branching [15]	Complexity
Bertz CT	Measure of molecular complexity through connectivity [16]	Complexity
Chi0, Chi1	Connectivity indices [17]	Complexity
Chi0n - Chi4n	Connectivity indices over various molecular fragments (0=atoms, 1=one bond fragments, 2=two bond fragments, etc.) [17]	Complexity
Chi0v - Chi4v	Valence connectivity indices (0=atoms, 1=one bond fragments, 2=two bond fragments, etc.) [17]	Complexity
Eccentricity	Shape descriptor calculated from the inertiamatrix (0=spherical, 1=linear), from Arteca [18]	Shape
Hall-Kier alpha	Modifying term for kappa descriptors, related to shape/flexibility [19]	Shape

Hall-Kier kappa1	Alpha-modified topological shape descriptor; related to complexity/number of cycles (rings) in the bond graph [19]	Shape
Hall-Kier kappa2	Alpha-modified topological shape descriptor; related to degree of star-like bond graph vs.linearity [19]	Shape
Hall-Kier kappa3	Alpha-modified topological shape descriptor; related to "centrality" of branching [19]	Shape
Hydrogen bond factor	Developed in Summers *et al.* [1] work; related to ability for formation of inter-monolayer hydrogen bonds	Charge distribution/ Misc.
IPC	Complexity/connectivity descriptor estimated from adjacency matrix of bond graph [20]	Complexity
Inertial shape factor	Characterization of molecular shape from principal moments of inertia (pm2/(pm1 *pm3), where pm1–3 are the three principal moments), from Todeschini and Consoni [19]	Shape
logP	Octanol - water partition coefficient estimated through the method of Wildman and Crippen; [21] measure of hydrophobicity	Charge distribution/ Misc.
Molar refractivity	Estimation of molecular polarizability; calculated through the method of Wildman and Crippen [21]	Size
Molecular weight	-	Size
Molecular weight (heavy atoms)	Molecular weight excluding hydrogens	Size
Normalized principal moments ratios (NPR1, NPR2)	Used to characterize molecular shape, from Sauer and Schwarz [22]	Shape
Number of heavy atoms	Number of non-hydrogen atoms	Size
Number of rotatable bonds	-	Size/Shape
Number of valence electrons	-	Size
Plane of best fit	Measure of molecular planarity (0=planar, in-creasing with less planarity) [23]	Shape
Principal moments of inertia (PMI1, PMI2, PMI3)	Three principal moments of inertia for the molecule (1=smallest, 3=largest)	Shape
Radius of gyration	(From Arteca [18]) Characterizes molecular shape, specifically, elongation	Shape/Size
Sphericity	Measure of molecular shape (0=spherical, 1=flat), from Robinson *et al.* [24]	Shape
Topological polar surface area	Estimation of surface area of only polar atoms, from Ertl *et al.* [25]	Charge distribution
Total hydrophobic VSA	Sum of SA contributions from atoms with $-0.20 \leq q < 0.20$	Charge distribution
Total negative van der Waals surface area (VSA)	Sum of SA contributions from atoms with $q < 0.0$	Charge distribution
Total negative polar VSA	Sum of SA contributions from atoms with $q < 0.20$	Charge distribution
Total polar VSA	Sum of SA contributions from atoms with $\lvert q \rvert > 0.20$	Charge distribution
Total positive VSA	Sum of SA contributions from atoms with $q > 0.0$	Charge distribution
Total positive polar VSA	Sum of SA contributions from atoms with $q \geq 0.20$	Charge distribution
Fractional hydrophobic VSA	Total hydrophobic VSA / Total VSA	Charge distribution

Fractional negative VSA	Total negative VSA / Total VSA	Charge distribution
Fractional negative polar VSA	Total negative polar VSA / Total VSA	Charge distribution
Fractional polar VSA	Total polar VSA / Total VSA	Charge distribution
Fractional positive VSA	Total positive VSA / Total VSA	Charge distribution
Fractional positive polar VSA	Total positive polar VSA / Total VSA	Charge distribution

Table D.20: Summary of the performance of each ML models when predicting COF and F_0 for all different test sets, measured by MAE. For each data point, the MAE value is averaged from MAE of individual ML models (5 replicates) when applied to the test set.

	N	*5050-test*		*2575-test*		*Total-test*	
		COF[1]	F_0[2]	COF[1]	F_0[2]	COF[1]	F_0[2]
Summers *et al.* models	100	0.0078308	0.4131868	0.0091615	0.4516796	0.0087057	0.4384945
5050-train models	100	0.0072 ± 0.0004	0.3792 ± 0.0303	0.0087 ± 0.0003	0.4254 ± 0.0158	0.0082 ± 0.0003	0.4096 ± 0.0172
	200	0.0064 ± 0.0001	0.3231 ± 0.0208	0.0080 ± 0.0001	0.3882 ± 0.0132	0.0074 ± 0.0001	0.3659 ± 0.0157
	300	0.0059 ± 0.0002	0.3034 ± 0.0083	0.0077 ± 0.0001	0.3662 ± 0.0088	0.0071 ± 0.0001	0.3447 ± 0.0086
	500	0.0054 ± 0.0001	0.2711 ± 0.0059	0.0074 ± 0.0001	0.3332 ± 0.0027	0.0067 ± 0.0001	0.3119 ± 0.0033
	1000	0.0045 ± 0.0001	0.2417 ± 0.0054	0.0071	0.3096 ± 0.0053	0.0062	0.2864 ± 0.0032
	1500	0.0040 ± 0.0001	0.2227 ± 0.0035	0.0071	0.2951 ± 0.0053	0.0060 ± 0.0001	0.2703 ± 0.0029
	2000	0.0034 ± 0.0001	0.2100 ± 0.0032	0.0070 ± 0.0001	0.2877 ± 0.0036	0.0058 ± 0.0001	0.2611 ± 0.0031
	2500	0.0028	0.2027 ± 0.0016	0.0069	0.2831 ± 0.0008	0.0055	0.2556 ± 0.0010
	2680	0.0026	0.2000 ± 0.0004	0.0069	0.2822 ± 0.0004	0.0054	0.2540 ± 0.0002
Total-train models	100	0.0071 ± 0.0002	0.3988 ± 0.0171	0.0084 ± 0.0001	0.4402 ± 0.0255	0.0079 ± 0.0001	0.4260 ± 0.0203
	200	0.0067 ± 0.0003	0.3362 ± 0.0085	0.0080 ± 0.0002	0.3816 ± 0.0091	0.0075 ± 0.0002	0.3660 ± 0.0074
	300	0.0064 ± 0.0002	0.3158 ± 0.0098	0.0076 ± 0.0002	0.3417 ± 0.0097	0.0072 ± 0.0001	0.3328 ± 0.0094
	500	0.0059 ± 0.0002	0.2841 ± 0.0064	0.0072 ± 0.0001	0.3086 ± 0.0097	0.0068 ± 0.0001	0.3002 ± 0.0069
	1000	0.0054 ± 0.0001	0.2508 ± 0.0064	0.0066 ± 0.0001	0.2729 ± 0.0049	0.0062 ± 0.0001	0.2653 ± 0.0032
	1500	0.0051 ± 0.0001	0.2356 ± 0.0063	0.0062 ± 0.0001	0.2561 ± 0.0035	0.0058	0.2491 ± 0.0039
	2000	0.0048 ± 0.0001	0.2234 ± 0.0032	0.0059 ± 0.0001	0.2492 ± 0.0052	0.0055	0.2404 ± 0.0045
	2500	0.0045 ± 0.0001	0.2162 ± 0.0066	0.0056 ± 0.0001	0.2400 ± 0.0045	0.0052	0.2318 ± 0.0052
	4000	0.0038 ± 0.0001	0.2002 ± 0.0055	0.0049 ± 0.0001	0.2267 ± 0.0041	0.0045	0.2177 ± 0.0033
	6000	0.0032	0.1916 ± 0.0030	0.0041 ± 0.0001	0.2177 ± 0.0020	0.0038 ± 0.0001	0.2087 ± 0.0013
	7816	0.0026	0.1845 ± 0.0004	0.0033	0.2115 ± 0.0003	0.0031	0.2023 ± 0.0003

[1] MAE when predicting coefficient of friction
[2] MAE when predicting adhesion force

Table D.21: Summary of the performance of each ML models when predicting COF and F_0 for all different test sets, measured by MSE. For each data point, the MSE value is averaged from MSE of individual ML models (5 replicates) when applied to the test set

	N	5050-test		2575-test		Total-test	
		COF[1]	F_0[2]	COF[1]	F_0[2]	COF[1]	F_0[2]
Summers et al. models	100	0.0001	0.39463	0.00013	0.48673	0.00012	0.45518
5050-train models	100	$(0.05456 \pm 1.00000) \times 10^{-5}$	0.25929 ± 0.06627	$(0.06433 \pm 1.00000) \times 10^{-5}$	0.34339 ± 0.04205	$(0.06098 \pm 1.00000) \times 10^{-5}$	0.31459 ± 0.04180
	200	0.04782	0.22383 ± 0.03822	0.0589	0.31164 ± 0.03278	0.05511	0.28156 ± 0.03380
	300	0.04467	0.21159 ± 0.01304	0.05682	0.29515 ± 0.00948	0.05266	0.26653 ± 0.00966
	500	0.04074	0.19252 ± 0.00901	0.05424	0.27528 ± 0.00243	0.04961	0.24693 ± 0.00455
	1000	0.03394	0.17486 ± 0.01200	0.05213	0.26029 ± 0.01002	0.0459	0.23103 ± 0.00700
	1500	0.03013	0.16331 ± 0.00844	0.05196	0.25252 ± 0.00792	0.04448	0.22197 ± 0.00393
	2000	0.02515	0.15540 ± 0.00258	0.05131	0.24841 ± 0.00561	0.04235	0.21655 ± 0.00420
	2500	0.02091	0.15167 ± 0.00154	0.0508	0.24559 ± 0.00128	0.04056	0.21342 ± 0.00119
	2680	0.01963	0.14971 ± 0.00050	0.05046	0.24523 ± 0.00046	0.0399	0.21251 ± 0.00021
Total-train models	100	0.05389	0.27016 ± 0.03076	0.06209	0.35161 ± 0.03185	0.05928	0.32371
	200	0.05041 ± 0.00001	0.22817 ± 0.00870	0.05919	0.30398 ± 0.02796	0.05618	0.27801
	300	0.04779	0.21422 ± 0.02009	0.0562	0.27722 ± 0.01965	0.05332	0.25564
	500	0.04469	0.1971 ± 0.0089	0.05334	0.26177 ± 0.01527	0.05038	0.23962
	1000	0.04084	0.17797 ± 0.00812	0.04864	0.23967 ± 0.00789	0.04597	0.21853
	1500	0.03791	0.16722 ± 0.00851	0.04564	0.22900 ± 0.00464	0.04299	0.20784
	2000	0.03588	0.16195 ± 0.00559	0.04327	0.22530 ± 0.00843	0.04074	0.20360
	2500	0.03357	0.15685 ± 0.01087	0.04113	0.21964 ± 0.00774	0.03854	0.19813
	4000	0.0287	0.14824 ± 0.00641	0.03609	0.21033 ± 0.00514	0.03356	0.18906
	6000	0.02363	0.14455 ± 0.00310	0.03004	0.20447 ± 0.00188	0.02784	0.18394
	7816	0.01982	0.13985 ± 0.00013	0.02459	0.19966 ± 0.00036	0.02296	0.17917

[1] MAE when predicting coefficient of friction

[2] MAE when predicting adhesion force

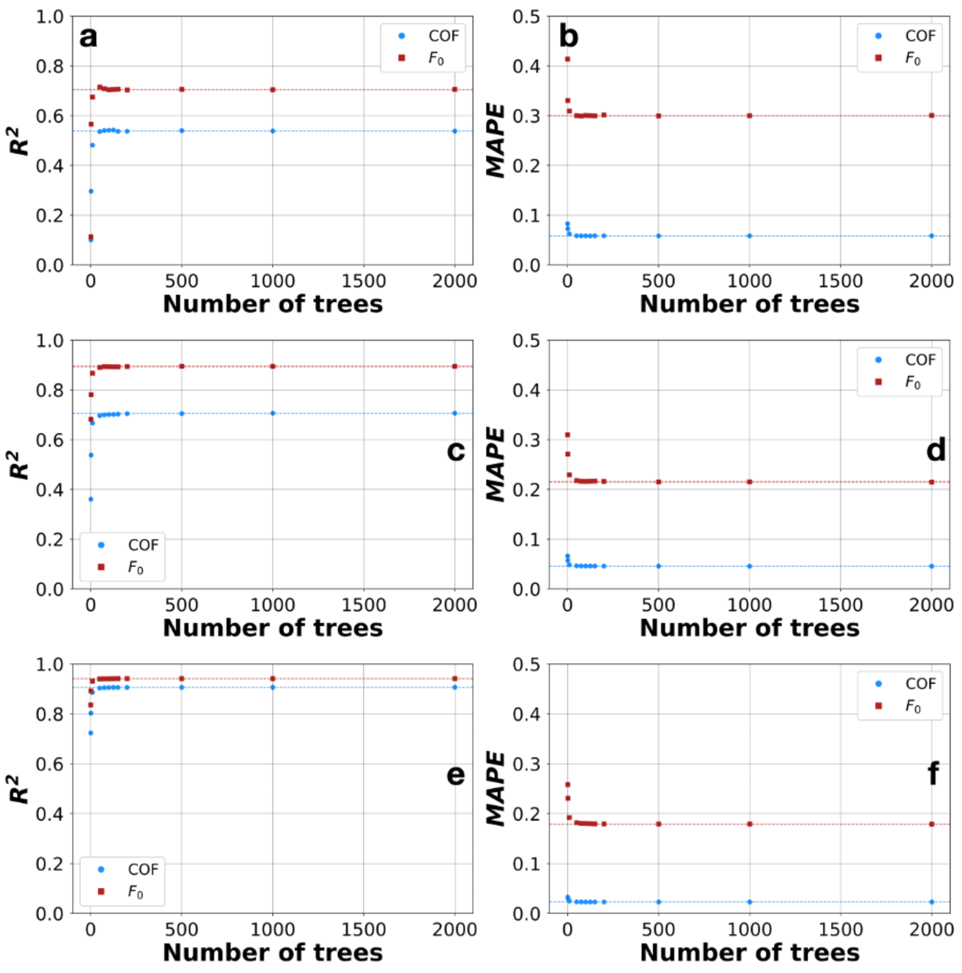


Figure D.9: Performance of predictive models, measured with R^2 and MAPE, as a function of number of trees. Models are evaluated at tree training set sizes, 100 (a and b), 1000 (c and d), and 7816 (e and f). The dotted, horizontal line represents the accuracy of models trained with 1000 trees, which is used to train ML models in the body of this work.

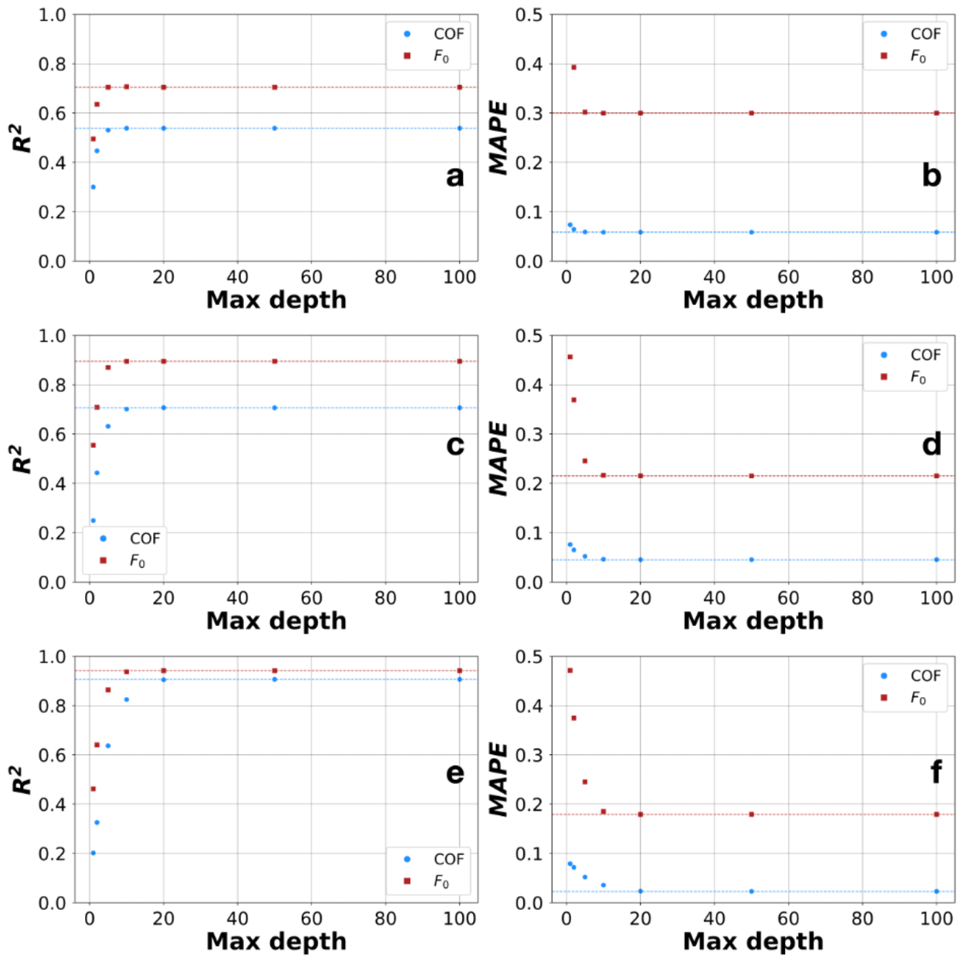


Figure D.10: Performance of predictive models, measured with R^2 and MAPE, as a function of maximum depth. Models are evaluated at three training set sizes, 100 (a and b), 1000 (c and d), and 7816 (e and f). The dotted, horizontal line represents the accuracy of models trained with no maximum depth, which is used to train ML models in the body of this work.

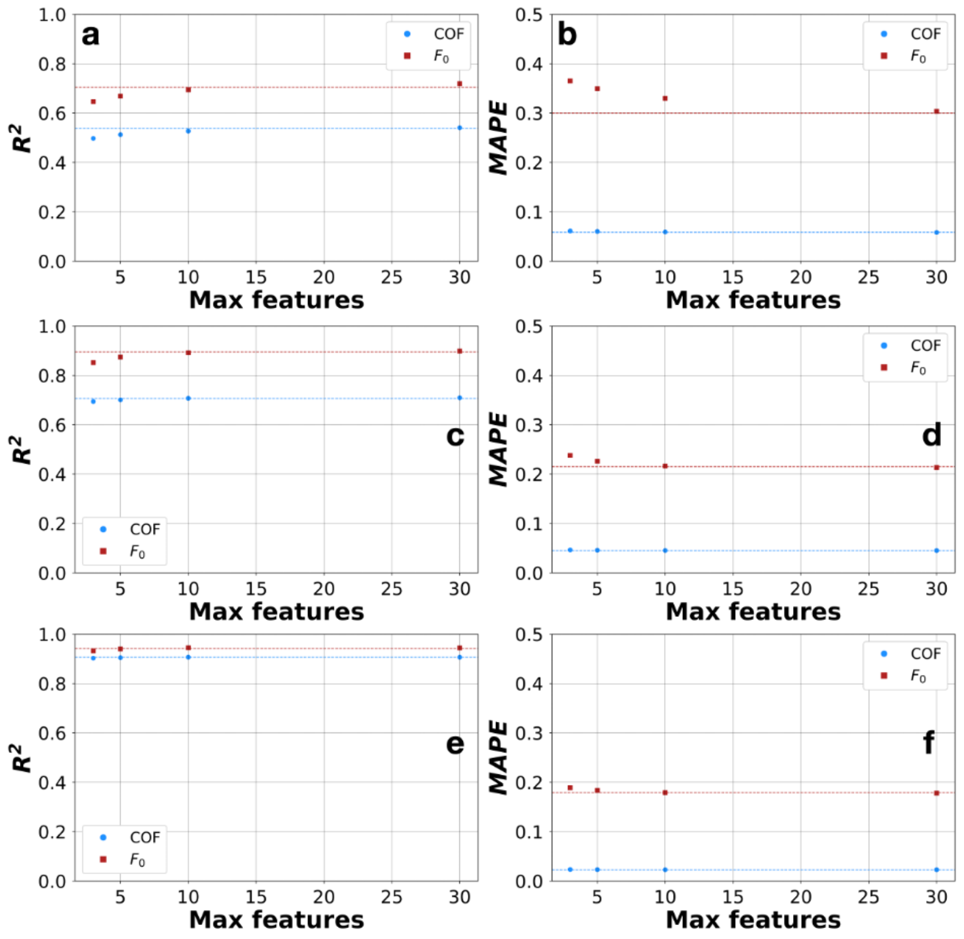


Figure D.11: Performance of predictive models, measured with R^2 and MAPE, as a function of maximum features. Models are evaluated at three training set sizes, 100 (a and b), 1000 (c and d), and 7816 (e and f). The dotted, horizontal line represents the accuracy of models trained with the maximum number of features provided (32–45 features), which is used to train ML models in the body of this work.

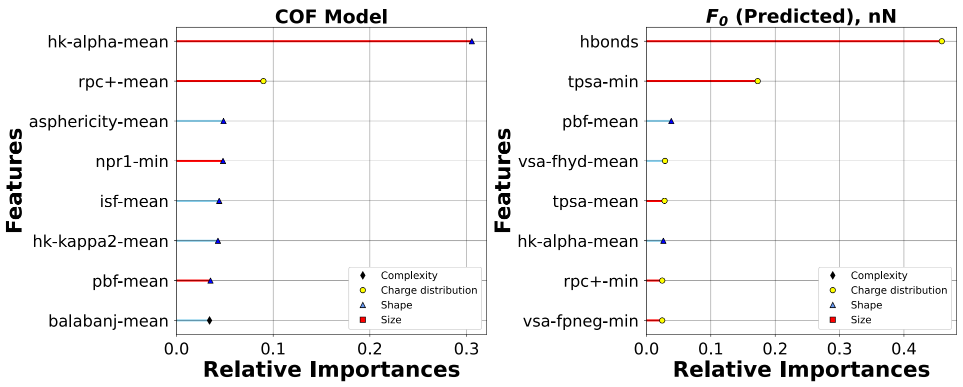


Figure D.12: Feature importance of models trained with 100 data points from the Summers *et al.* [1] data when predicting COF (left) and F_0 (right).

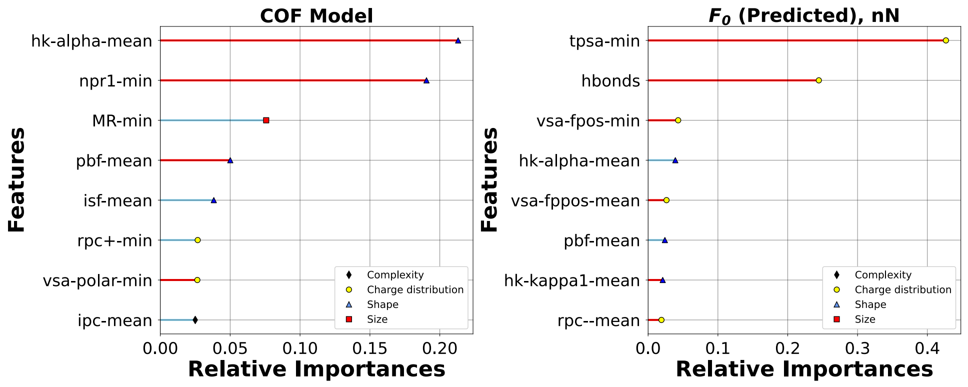


Figure D.13: Feature importance of models trained with 1000 data points from the *5050-train* data, COF (left) and F_0 (right).

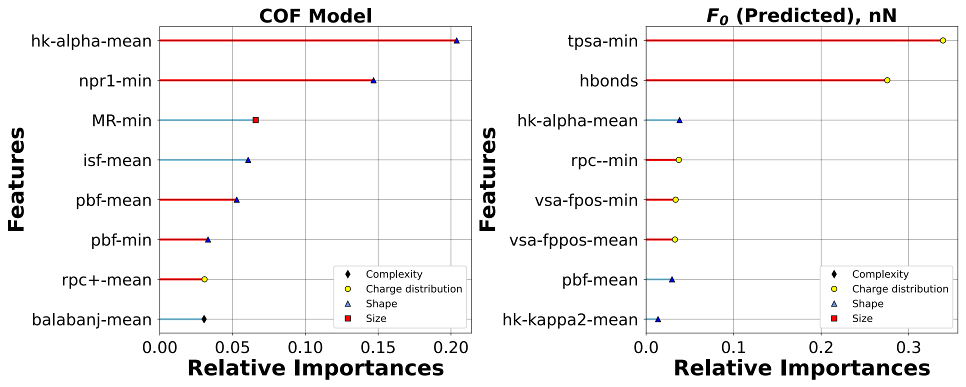


Figure D.14: Feature importance of models trained with all data points (7816) from the total-train, COF (left) and F_0 (right).

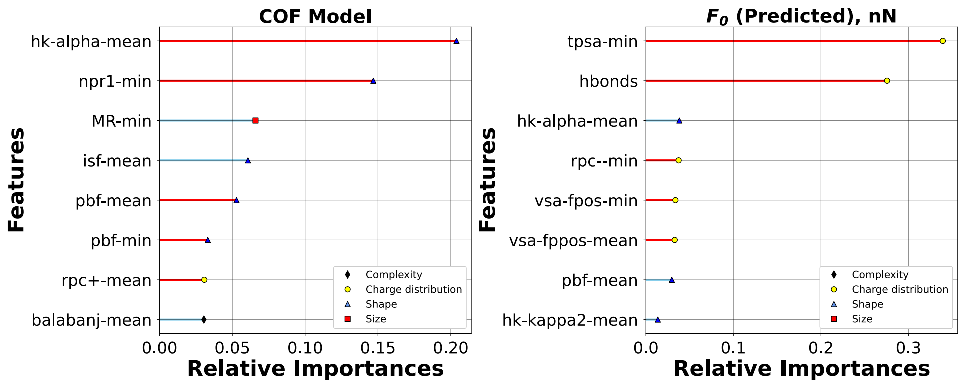


Figure D.15: Feature importance of models trained with 1000 data points from the *total-train* data, COF (left) and F_0 (right).

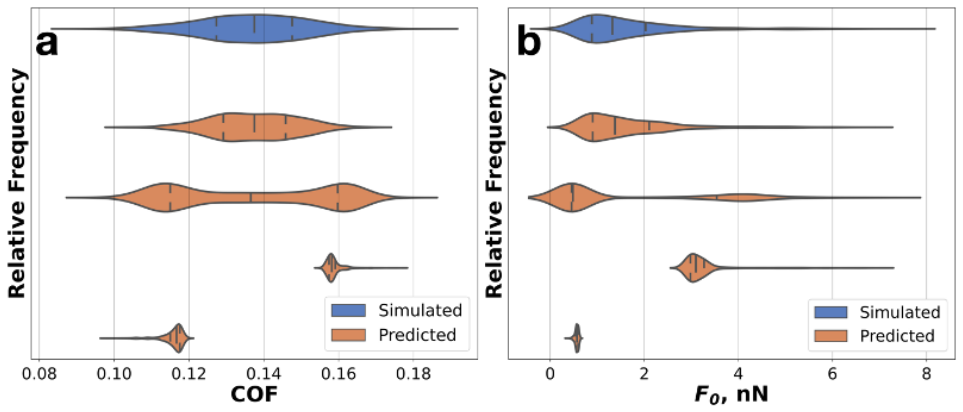


Figure D.16: Distribution of COF (a) and F_0 (b) properties calculated from MD simulation (blue) and estimated using ML models (orange). From top to bottom, distribution (1) calculated from MD simulations (reference), and estimated from ML models trained with (2) 1000 data points bootstrapping from the *total-train* set, (3) 500 highest values and 500 lowest values from the *total-train* set, (4) 1000 highest values from the *total-train* set and (5) 1000 lowest values from the *total-train* set.

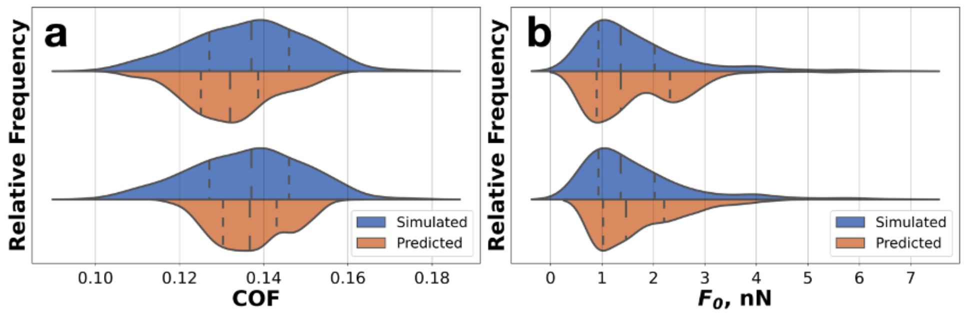


Figure D.17: Distribution of (a) COF and (b) F_0 for systems in the *5050-test* set from MD simulations (upper, blue) and from MD models prediction (lower, orange), trained with Summers *et al.* data (top) and with 100 data points from the *5050-train* set (bottom). In each distribution, the center line represents the median and the other two lines mark the two adjacent quantiles.

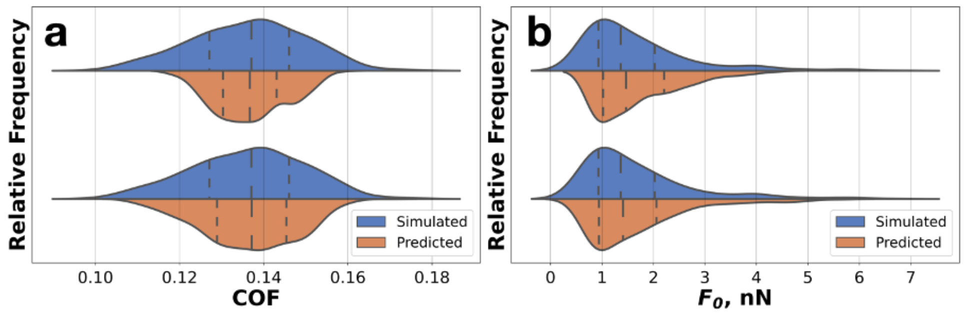


Figure D.18: Distribution of (a) COF and (b) F_0 for systems in the *5050-test* set from MD simulations (upper, blue) and from MD models prediction (lower, orange), trained with 100 (top) and with 1000 (bottom) data points from the *5050-train* set. In each distribution, the center line represents the median and the other two lines mark the two adjacent quantiles.

Bibliography

1. Summers, A. Z., Gilmer, J. B., Iacovella, C. R., Cummings, P. T. & McCabe, C. MoSDeF, a Python Framework Enabling Large-Scale Computational Screening of Soft Matter: Application to Chemistry-Property Relationships in Lubricating Monolayer Films. *Journal of Chemical Theory and Computation* **0.** PMID: 32004433, null. ISSN: 1549-9618. eprint: https://doi.org/10.1021/acs.jctc.9b01183 (Mar. 0).

2. Brewer, N. J., Beake, B. D. & Leggett, G. J. Friction Force Microscopy of Self-Assembled Monolayers: Influence of Adsorbate Alkyl Chain Length, Terminal Group Chemistry, and Scan Velocity. *Langmuir* **17,** 1970–1974. ISSN: 0743-7463 (Mar. 2001).

3. Frisbie, C. D., Rozsnyai, L. F., Noy, A., Wrighton, M. S. & Lieber, C. M. Functional Group Imaging by Chemical Force Microscopy. *Science (Washington, DC, U. S.)* **265,** 2071 (1994).

4. Noy, A., Frisbie, C. D., Rozsnyai, L. F., Wrighton, M. S. & Lieber, C. M. Chemical Force Microscopy: Exploiting Chemically-Modified Tips To Quantify Adhesion, Friction, and Functional Group Distributions in Molecular Assemblies. *Journal of the American Chemical Society* **117,** 7943–7951. ISSN: 0002-7863 (Aug. 1995).

5. Thompson, M. W., Gilmer, J. B., Matsumoto, R. A., Quach, C. D., Shamaprasad, P., Yang, A. H., Iacovella, C. R., McCabe, C. & Cummings, P. T. Towards Molecular Simulations That Are Transparent, Reproducible, Usable by Others, and Extensible (TRUE). *Molecular Physics* **118,** e1742938. ISSN: 0026-8976, 1362-3028 (June 2020).

6. Quach, C. D. & Gilmer, J. B. *High-Throughput Screening of Tribological Properties of Monolayer Films Using Molecular Dynamics and Machine Learning: Supplemental Repository* https://github.com/daico007/iMoDELS-supplements/tree/6054ccd91b19575c8d5ccbd5a672b9b55664f0c8. [Online; accessed 2022-03-16]. 2022.

7. Jorgensen, W. L., Maxwell, D. S. & Tirado-Rives, J. Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids. *J. Am. Chem. Soc.* **118,** 11225–11236. ISSN: 0002-7863 (Nov. 1996).

8. Lorenz, C., Webb, E., Stevens, M., Chandross, M. & Grest, G. Frictional Dynamics of Perfluorinated Self-Assembled Monolayers on Amorphous SiO2. *Tribology Letters* **19,** 93–98. ISSN: 1023-8883, 1573-2711 (June 2005).

9. Pavlov, Y. L. *Random Forests* ISBN: 978-3-11-094197-5 (2019).

10. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, É. d. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research* **12,** 2825–2830. ISSN: 1532-4435. arXiv: 1201.0490 (2012).

11. Vishwakarma, G., Sonpal, A. & Hachmann, J. Metrics for Benchmarking and Uncertainty Quantification: Quality, Applicability, and Best Practices for Machine Learning in Chemistry. *Trends in Chemistry* **3,** 146–156. ISSN: 2589-5974 (Feb. 2021).

12. Landrum, G. *et al.* RDKit: Open-source cheminformatics (2006).

13. Labute, P. A widely applicable set of descriptors. *Journal of Molecular Graphics and Modelling* **18,** 464–477. ISSN: 1093-3263. https://www.sciencedirect.com/science/article/pii/S1093326300000681 (2000).

14. Baumgärtner, A. Shapes of Flexible Vesicles at Constant Volume. *The Journal of Chemical Physics* **98,** 7496–7501. ISSN: 0021-9606, 1089-7690 (May 1993).

15. Balaban, A. T. Highly Discriminating Distance-Based Topological Index. *Chemical Physics Letters* **89,** 399–404. ISSN: 0009-2614 (1982).

16. Bertz, S. H. Convergence, Molecular Complexity, and Synthetic Analysis. *Journal of the American Chemical Society* **104,** 5801–5803. ISSN: 0002-7863 (Oct. 1982).

17. Hall, L. H. & Kier, L. B. in *Reviews in Computational Chemistry* 367–422 (John Wiley & Sons, Ltd, 1991). ISBN: 978-0-470-12579-3. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470125793.ch9.

18. Arteca, G. in *Reviews in Computational Chemistry, Vol 20* 191–253 (Jan. 2007). ISBN: 978-0-470-12586-1.

19. Todeschini, R. & Consonni, V. *Handbook of Molecular Descriptors* ISBN: 978-3-527-61310-6 978-3-527-61311-3 (2011).

20. Bonchev, D. & Trinajstić, N. Information Theory, Distance Matrix, and Molecular Branching. *The Journal of Chemical Physics* **67,** 4517–4533. eprint: https://doi.org/10.1063/1.434593 (1977).

21. Wildman, S. A. & Crippen, G. M. Prediction of Physicochemical Parameters by Atomic Contributions. *Journal of Chemical Information and Computer Sciences* **39,** 868–873. ISSN: 0095-2338 (Sept. 1999).

22. Sauer, W. H. B. & Schwarz, M. K. Molecular Shape Diversity of Combinatorial Libraries: A Prerequisite for Broad Bioactivity. *Journal of Chemical Information and Computer Sciences* **43,** 987–1003. ISSN: 0095-2338 (May 2003).

23. Firth, N. C., Brown, N. & Blagg, J. Plane of Best Fit: A Novel Method to Characterize the Three-Dimensionality of Molecules. *Journal of Chemical Information and Modeling* **52,** 2516–2525. ISSN: 1549-9596 (Oct. 2012).

24. Robinson, D. D., Barlow, T. W. & Richards, W. G. The Utilization of Reduced Dimensional Representations of Molecular Structure for Rapid Molecular Similarity Calculations. *Journal of Chemical Information and Computer Sciences* **37,** 943–950. ISSN: 0095-2338 (Sept. 1997).

25. Ertl, P., Rohde, B. & Selzer, P. Fast Calculation of Molecular Polar Surface Area as a Sum of Fragment-Based Contributions and Its Application to the Prediction of Drug Transport Properties. *Journal of Medicinal Chemistry* **43,** 3714–3717. ISSN: 0022-2623 (Oct. 2000).

Appendix E

Contributions to papers

In this appendix chapter, I describe my contributions to the publications I am listed as a co-author on using the CRediT contributor role taxonomy [1].

In Chapter 2 "Best Practices for Molecular Simulation", I contributed to the conceptualization, resource provisioning, manuscript original draft, manuscript review and editing, and visualization. I wrote a majority of the initial text for the thermostat (subsection 2.5.4) and barostat (subsection 2.5.5) sections, as well as Table 2.1. The figures, tables, and majority of the text for Chapter 2 are from [2].

In Chapter 3 "Towards Molecular Simulations that are Transparent, Reproducible, Usable By Others, and Extensible (TRUE)", I contributed to the conceptualization, resource provisioning, manuscript original draft, manuscript review and editing, and visualization. The figures, listings, tables and majority of the text for Chapter 3are from [3]. I also assisted with the software development and creation of the supplemental information, as well as a majority of the code listings.

In Chapter 4, I contributed to the methodology, software, validation, investigation, resources, data curation, manuscript original draft, manuscript review and editing, and visualization. I was the lead developer for the plugin system and its user documentation in both `mbuild` and `foyer` (Section 4.2). I was the lead developer for the support of crystal lattices and triclinic geometries in `mbuild` (Section 4.2). For Section 4.3, I contributed to the conceptualization, methodology, software, validation, investigation, manuscript original draft, and the manuscript review and editing. For Section 4.4, I contributed to the methodology, software, manuscript original draft, manuscript review and editing. Figures, tables, listings and text from [4].

For Chapter 6, I contributed to the methodology, software, validation, investigation, manuscript original draft, manuscript review and editing. This chapter is a combination of figures, tables, and text from [5, 6]. For [5], I included two additional test systems to validate the transferability of the model developed by Summers, and the subsequent scaling up of systems to study in [6]. I was involved in all aspects of the computational study in [6], but my co-first author, Co Quach, led the majority of the RF model development, extension, and training. I was the lead developer of the data space, extension of the project, COF and F_0 calculation, and organization and distribution of simulations for [6].

Bibliography

1. Brand, A., Allen, L., Altman, M., Hlava, M. & Scott, J. Beyond Authorship: Attribution, Contribution, Collaboration, and Credit. *Learned Publishing* **28,** 151–155. ISSN: 1741-4857 (2015).

2. Braun, E., Gilmer, J., Mayes, H. B., Mobley, D. L., Monroe, J. I., Prasad, S. & Zuckerman, D. M. Best Practices for Foundations in Molecular Simulations [Article v1.0]. *Living journal of computational molecular science* **1,** 5957. ISSN: 2575-6524 (2019).

3. Thompson, M. W., Gilmer, J. B., Matsumoto, R. A., Quach, C. D., Shamaprasad, P., Yang, A. H., Iacovella, C. R., McCabe, C. & Cummings, P. T. Towards Molecular Simulations That Are Transparent, Reproducible, Usable by Others, and Extensible (TRUE). *Molecular Physics* **118,** e1742938. ISSN: 0026-8976, 1362-3028 (June 2020).

4. Klein, C., Summers, A. Z., Thompson, M. W., Gilmer, J. B., McCabe, C., Cummings, P. T., Sallai, J. & Iacovella, C. R. Formalizing Atom-Typing and the Dissemination of Force Fields with Foyer. *Computational Materials Science* **167,** 215–227. ISSN: 0927-0256. http://www.sciencedirect.com/science/article/pii/S0927025619303040 (Sept. 2019).

5. Summers, A. Z., Gilmer, J. B., Iacovella, C. R., Cummings, P. T. & McCabe, C. MoSDeF, a Python Framework Enabling Large-Scale Computational Screening of Soft Matter: Application to Chemistry-Property Relationships in Lubricating Monolayer Films. *Journal of Chemical Theory and Computation* **0.** PMID: 32004433, null. ISSN: 1549-9618. eprint: https://doi.org/10.1021/acs.jctc.9b01183 (Mar. 0).

6. Quach, C. D., Gilmer, J. B., Pert, D. O., Mason-Hogans, A., Iacovella, C. R., Cummings, P. T. & McCabe, C. High-Throughput Screening of Tribological Properties of Monolayer Films Using Molecular Dynamics and Machine Learning. *The Journal of Chemical Physics,* 5.0080838. ISSN: 0021-9606, 1089-7690 (Feb. 2022).