ADVANCED AI-ML APPROACHES FOR CPS DESIGN AND FUNCTIONAL CORRECTNESS OF LEC IN CPS OPERATION

By

Harsh Vardhan

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

Computer Science

December 16, 2023

Nashville, Tennessee

Approved:

Dr. Janos Sztipanovits, Ph.D.

Dr. Gabor Karsai, Ph.D.

Dr. Xenofon Koutsoukos, Ph.D.

Dr. Daniel Work, Ph.D.

Dr. David Hyde, Ph.D.

To my late mother - in whose lap I was born, played, nurtured, and raised. Each time I remember her, I meet her with a smile and joy and finally have to let her go with a heavy heart. She would be safe in my memory and heart until I close my eyes.

ACKNOWLEDGMENTS

First and foremost, I would like to extend my heartfelt thanks to my advisor, Dr. Janos Sztipanovits, for his invaluable guidance, continuous support, and patience throughout my research journey. One thing that stood the test of time was his emphasis on learning rather than experimentation to broaden the knowledge horizon in the early years of my Ph.D. That philosophy helped to develop deep insight into research challenges and build a strong theoretical background. He guided me toward research challenges, answers, and solutions and always asked the question or assumption that I missed. I appreciate all his contributions of time, ideas, expertise, insightful feedback, unwavering commitment, and funding to make my Ph.D. experience productive and stimulating. I would also like to thank my committee members, Dr. Gabor Karsai, Dr. Xenofon Koutsoukos, Dr. Daniel Work, and Dr. David Hyde, for their valuable time, constructive criticism, and valuable suggestions that greatly enhanced the quality of this work. I am also grateful to the staff and faculty of Vanderbilt University for providing a conducive research environment and access to necessary resources. Their continuous support was crucial and instrumental in finishing this thesis. I would like to thank all my collaborators that I have had the chance to work with and learn from, namely Peter Volgyesi, Umesh Timalsina, Will Hedgecock, Xenofon Koutsoukos, Milkos Maroti, Mary Matelko, Ali Ozdagli, and David Hyde from the Symbiotic CPS project; it was an exhilarating experience, and Carlos Barreto, Himanshu Neema, Hongyang Sun, Aniruddha Gokhale, Yogesh D Barve, Zhuangwei Kang, and Neil Sarkar at ISIS. A special thanks to my colleague and friends Purboday, Shreyas, Bishnu, Ajitesh, Umesh and Brajesh for always being ready to listen to my waffles; that was quite therapeutic.

Furthermore, I would like to acknowledge my wife, Sonal, for her unwavering support, encouragement, and understanding throughout this journey. Her belief in my abilities and her unconditional love have been constant sources of motivation and strength. I would like to extend my appreciation to my big family back in India (my father, who supported me unconditionally through the good and bad times of life; my sisters, who raised me with love; my in-laws; my nieces and nephews; playing and talking with them is always fun and refreshing); and my family members in the USA (Jai, Neha, Babul, Sumit, and Sarthak) and the newest member of the family, my son, 'Ojas'. Although life became tougher after Ojas came into our lives due to sleep-deprived nights, morning headaches, and tireless and continuous work to take care of him, but his one smile compensates for everything.

All of the works related to my research were funded in part by the DARPA and NSF through various grants. The views and opinions of the authors expressed herein do not necessarily state or reflect those of the US government or any funding agency. I am really thankful to US government for funding my research efforts.

Finally, I want to express my gratitude to all the individuals who have directly or indirectly contributed to this thesis, even if not mentioned specifically. Each interaction, conversation, and exchange of ideas has played a role in shaping my understanding and growth as a researcher. Completing this thesis would not have been possible without the support and encouragement of all these individuals. I am truly grateful for their contributions and look forward to their continued guidance and support as I embark on future academic endeavors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AE Autoencoder

AI Artificial Intelligence

AL Active Learning

BNN Bayesian Neural Network

BO-EI Bayesian optimization-Expected Improvement

BO-LCB Bayesian optimization-Lower Confidence Bound

CAD Computer-Aided Design

CFD Computational Fluid Dynamics

CPS Cyber Physical System

DeepAL Deep Active Learning

DL Deep Learning

DNN Deep Neural Network

DOE Design of Experiment

EGL Expected Gradient Length

FEA Finite Element Analysis

GMM Gaussian Mixture Model

GP Gaussian Process

KL Kullback–Leibler

KNN K-Nearest Neighbour

LEC Learning Enabled Component

LES Large Eddy Simulation

LHC Latin Hyper Cube

LIDAR Light Detection and Ranging

ML Machine Learning

OOD Out of Distribution

QBC Query by Committee

RANS Reynolds-averaged Navier–Stokes

RL Reinforcement Learning

RRCF Robust Random Cut Forest

RRRCF Reduced Robust Random Cut Forest

SVDD Support Vector Data Description

UUV Unmanned Underwater Vehicle

VAE Variational Auto Encoder

VMC Vanill Monte Carlo

CHAPTER 1

Introduction

1.1 Motivation

The engineering of Cyber-Physical Systems (CPS) plays a crucial role in shaping our everyday physical environment. The considerable complexity inherent in their design process has rendered their engineering endeavors a formidable undertaking. The existing design process entails the iterative execution of sequential design selection, evaluation, and optimization, with the objective of meeting the criteria established or exhausting the allocated budget. One direct consequence of adopting this method is the emergence of computing complexity as a limiting factor, leading to extended design cycles with prolonged development duration. The observed limitations in simulation complexity encountered during the course of my research are attributed to the intricate nature of simulation physics involved in the design process, such as computational fluid dynamics and finite element analysis. These two fields possess widespread applications in designs that need the analysis of fluid movement and interactions, encompassing both liquids and gases, with solid materials. Addressing the challenges imposed by this bottleneck can benefit a wide range of design problems. In such a case, there are two major directions of research that I worked on:

1. Development and integration of sample efficient optimization methods and algorithms into design tools that can find an optimal design in very few evaluations.

2. Replacement of a computationally costly and complex process with a cheap, data-driven, trained surrogate approximation that is trained on a few design points and can generalize on others.

Both areas of research have the potential to utilize current AI-ML-based methodologies in order to augment innovation within the design process. Recent advancements in artificial intelligence (AI) have demonstrated the potential of AI-based optimization algorithms as very effective frameworks for optimization tasks, particularly in terms of sample efficiency. Bayesian optimization and its variants, Variation Auto-Encoder (VAE)-based Latent Space Optimisation (LSO), and Gaussian Process (GP)-based mixture models are prospective research avenues for such an endeavor. However, the utilization of surrogate modeling as a substitute for the original expensive function in the design optimization pipeline is increasingly being recognized as an another incredibly efficient alternative. Another unresolved concern in these fields pertains to the absence of open-source tools that can be utilized for research experiments without incurring substantial expenses on licensing fee. Additionally, there is a need for these tools to possess the capacity to include contemporary and sophisticated AI-ML algorithms.

After the finalization of a design, whether by direct optimization or surrogate-based optimization, the implementation of control action becomes necessary to alter the dynamic behavior of the design in order to achieve a specified and desirable behaviour. In recent years, there has been a growing utilization of learning-enabled components (LEC) in the field of engineering system control. When a LEC surrogate is implemented as a constituent of system operation and engages with a given environment, a pivotal inquiry arises: *to what extent are safety criteria upheld?* One common issue associated with the utilization of surrogate models is the potential for operational safety violations due to their inherent poorer fidelity. Therefore, it is necessary to assess surrogate models in safety cases in order to furnish probabilistic evidence that they do not generate safety breaches across various operational circumstances. The objective is to ascertain that safety remains uncompromised in all environmental circumstances. Given the impracticality of formally verifying the integrated system under all possible environmental circumstances in most practical scenarios, it becomes necessary to employ simulation-based methodologies for verification purposes.

1.2 Research challenges

Below, I consolidate the research challenges that serve as the motivation for my work. It focuses on certain issues related to the CPS (Cyber-Physical Systems) design process that are of particular significance:

1. **Sample efficient design optimization:** In order to achieve the objective of enhancing a design with a limited number of design analyses and, consequently, reducing design duration, it is crucial to investigate and advance contemporary artificial intelligence (AI) - driven optimization techniques and incorporate these optimization frameworks within complex engineering design domains. In pursuit of this objective, several interesting unresolved questions are:

   Can AI-based optimization approaches enhance sample efficiency and convergence behavior in comparison to conventional optimization techniques coupled with existing simulation and design tools? What is the way of integrating AI-based optimization algorithms with existing engineering design tools as well as how can their performance be empirically assessed in design problems?

2. **Surrogate modeling:** An alternate strategy to decrease the duration of the design optimization process entails the development of a low-fidelity surrogate model that is computationally inexpensive and data-driven, serving as a replacement for the computationally intensive simulation. In this particular field of research, my focus lies on discovering:

   Can an AI-based learning model or surrogate capture a complex property of two-way coupled solid-fluid dynamics physics or finite element-based numerical physics? Can AI-based surrogates be useful in solving inverse problems involved in these engineering domains?

3. **Challenges of data generation for surrogate modeling in complex engineering domains:** In complex engineering domains like CFD, FEA, etc., where data labeling is computationally costly, the interesting question is:

   What should be the sampling strategy for data generation in order to construct a surrogate model that effectively reduces the size of the training dataset?

Once a system is designed and deployed in an environment to operate, the LEC, acting as a surrogate for the embedded controller, is utilized to alter the system's behavior in order to achieve a predetermined and favorable outcome over a period of time. In this particular scenario, it is imperative to ensure the functional correctness of the AI-based surrogate for the embedded controller. In pursuit of this objective, I delineate two primary unresolved issues:

1. **Rare even failure detection:** For correct functional performance in all scenarios, it is imperative to rapidly discover situations where the surrogate is likely to fail even for in-distribution input data and may result in safety failures. For a well-trained LEC surrogate, these situations are rare, and their discovery is challenging even after extensive simulation-based trials. These rare event safety failures generally arise from various aleatory and epistemic uncertainties in the system, LECs, simulation tools, or its environment. Since detecting the presence and removal of these uncertainties is not directly known, the alternative approach is to detect these rare failures using statistical methods. An open problem in this case is:

   How to rapidly discover in-distribution rare event failures in a trained learning-based controller?

2. **Out-of-distribution detection:** Another aspect of validating the functional correctness of a trained surrogate controller is to ensure that the surrogate is exposed only to inputs that are in the training distribution. Any prediction made on out-of-distribution (OOD) input data may not be reliable and should not be used. OOD detection of such input samples would be a necessary component. OOD is classified as point OOD and contextual OOD. In the point OOD, a single input that is anomalous with respect to the collected training data should be detected. Research in the field of OOD is active, but good OOD detection is still an elusive task. An open problem in this case is:

   How to perform point OOD detection with good accuracy?

1.3 Summary of contributions

In this section, I will provide a brief introduction of my contributions to addressing the above-mentioned research challenges. Figure 1.1 represents an overview of the research contributions. A detailed exposi-

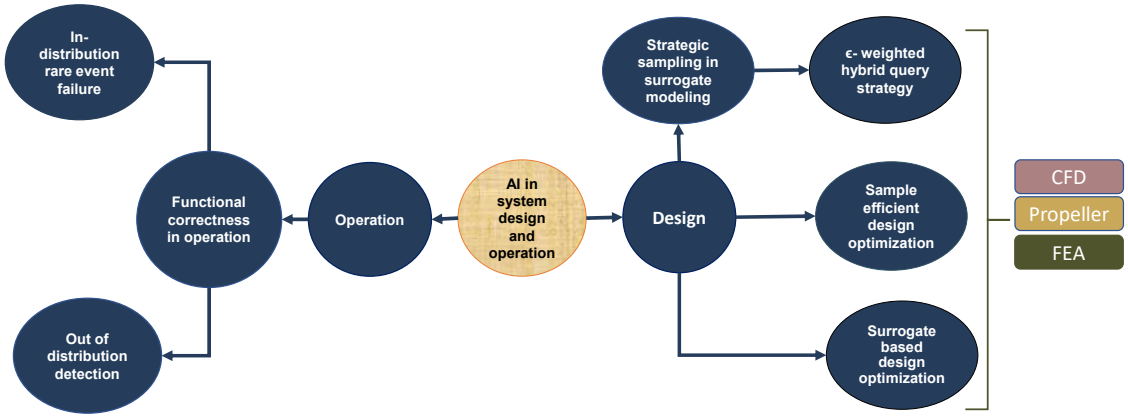


Figure 1.1: Overview of research contributions

tion of these contributions and the empirical results of the work will follow in later chapters. The research contributions are summarized here as follows:

- **Sample efficient and Surrogate-based design optimization in engineering design:**

  1. A comprehensive evaluation of a number of optimization methods on a real-world design problem involving complex engineering domain of CFD to find optimal design of an unmanned underwater vehicle (UUV) hull. This benchmark study may be useful for practitioners considering adopting these algorithms.
  2. A method for dealing with infeasible designs (designs with interference) as constraints during Bayesian optimization, along with a software package of implemented method for the UUV hull design optimization problem.
  3. A novel hybrid optimization strategy is proposed and evaluated to enhance the efficiency of design optimization issues characterized by computationally inexpensive yet high-dimensional design spaces. The approach leverages an artificial intelligence (AI) model to acquire an inverse mapping capability, enabling it to generate promising initial designs for subsequent optimization procedures.
  4. Conducted training on various deep neural network architectures to effectively capture the complex characteristics of two-way coupled solid-fluid dynamics, namely the physics related to drag force and stress analysis for structural integrity. Additionally when these trained models are utilized in the process of design optimization, as a substitute for physics-based simulations, it results in a significant acceleration in design optimization process.

5. Preliminary research was undertaken to explore the possibility of identifying a globally optimal design for the hull shape of an Unmanned Underwater Vehicle (UUV), which would be near optimal across various environmental and operating situations. The findings of the initial investigation indicate the potential presence of such a design.

6. A ready-to-use software package (called *Anvil*) is provided as a SciML tool for design optimization in a toolchain by integrating computer-aided design (CAD) and computational fluid dynamics (CFD) with Python and state-of-the-art AI optimization methods. This open-source software package can be readily used for shape optimization problems for both nautical and aeronautical design problems and to study the effect of solid-fluid dynamics for any subsonic flow conditions with very little work by the user.

- **Data-efficient surrogate modeling for engineering design: Ensemble-free batch mode deep active learning for regression:** A novel, scalable, and easy-to-train approach to batched deep active learning for regression problems is developed by selecting strategic samples during surrogate training to learn the input-output behavior of a function or simulator with a lesser number of labeled data than existing methods. I empirically evaluated the proposed method on different real-world engineering design domains (CFD, FEA, and propeller design domains) and compared it with other baseline methods to establish the performance gains from this novel approach in terms of sample efficiency or better accuracy.

- **In-distribution rare event failure test case generation in AI controllers:** An AI-based generative method is developed to generate scenarios that have a high likelihood of revealing potential failure. The empirical evaluation showed a speedup of a thousand times faster discovery of these failure cases than a standard Monte Carlo method.

- **Out-of-distribution detection (OOD) using RRCF:** Developed a white-box method that uses RRCF for OOD detection in a computationally efficient manner on large data sets. The empirical evaluation is conducted first on a reinforcement learning controller with a three-dimensional data stream and second on a high-dimensional image data stream generated by an open-source simulator CARLA [38].

The subsequent sections of this manuscript are structured in the following manner. In Chapter 2, I will present background & relevant literature survey on related work pertaining to the aforementioned challenges. In chapter 3, I will discuss sample efficient AI-based design optimization approaches with the use case of unmanned underwater vehicles in the CAD-CFD domain. I will also discuss deep learning-based surrogates in the CAD-CFD and FEA domains for sub-sea pressure vessels. Additionally, I will explore the optimiza-

tion process carried out through these surrogate-based designs, which offers a significant acceleration of two orders of magnitude compared to design optimization loops employing direct numerical simulation. Furthermore, I will delve into the lessons derived from this approach and the challenges associated with it. Additionally, I will explore the utilization of surrogate models in a hybrid optimization strategy for the resolution of a particular category of inverse problems. In chapter 4, I will elucidate the concept of noble active learning and adaptive sampling techniques that have been devised through the utilization of a hybrid query approach that can be used for deep active learning-based surrogate training. In chapter 5, the topic of rare event failures will be formulated and the contributions to solve this challenge will be discussed. In chapter 6, the issue of out-of-distribution (OOD) will be examined and the proposed methodology for addressing this issue will be explored. Finally, chapter 7 will serve as a platform to give the final analysis and draw conclusions regarding the potential future trajectory of my research contributions.

CHAPTER 2

Background & Literature Review

2.1 Design optimization methods in engineering

This section provides background on the different sampling and optimization techniques frequently used in engineering designs. Optimization frameworks are generally classified as gradient-based or gradient-free methods. Since the gradient information in complex engineering domains (like finite volume-based CFD or finite element-based FEA) are very costly and unreliable (even using adjoint solvers to compute gradients, and computing the adjoint sensitivities could be expensive and cannot be used for baseline design but only fine-tuning the design shape), we chose a gradient-free optimization methods for this study that is readily used in these domains. To this end, I selected the following frequently used optimization methods for our study:

1. Monte Carlo
2. Maximin Latin Hypercube
3. Genetic Algorithm
4. Nelder-Mead
5. Bayesian Optimization

The method called *Monte Carlo* sampling can be used to create a pseudo-random process from a seed for an input with the specific ranges and dimensions. A random series of numbers with a specified probability distribution can be used to carry out the procedure. Assuming equal likelihood for each experiment in design space, the prescribed distribution employed in this study was uniformly random, meaning samples were chosen from a uniform probability distribution function (pdf) between 0 and 1 and scaling in each dimension to the specified range of parameters.

The technique of Latin hypercube sampling [100] has been widely employed in several fields due to its extensive track record and very reliable performance. The primary characteristic of the standard LHC is its ability to project onto a grid consisting of n points. By applying design to any given factor, a more comprehensive representation of the design space can be achieved, resulting in n distinct levels for that design space. However, as a result of the complex nature of combinatorial permutations, obtaining high-quality Latin hypercube samples poses a significant challenge. The concept of LHC is expanded further to

include the pursuit of an optimal design through the optimization of a criterion that characterizes a desirable attribute of the design, rather than only relying on a permutation of the factors at the n-level. In order to achieve this objective, Morris and Mitchell [100] introduced a space-filling criterion to the vanilla Latin Hypercube Sampling (LHC) method. The primary aim of this optimization is to maximize the minimal distance between the sampled points. Consider a design denoted by X, which is represented by a matrix of size $n \times k$. Let s and t be two design points within this matrix. The distance between these two points may be determined using the formula $dist(s,t) = \{\sum_{j=1}^{k} |s_j - t_j|^p\}^{1/p}$. The parameter p specifies the specific distance metrics employed. An alternate conceptualization of the maximin criterion is put forth, as aim to optimize the lowest inter-sample distance, for which a list of distances $(d_1; d_2; ...; d_m)$ is generated in ascending order based on distance measurements. The distance list is accompanied by the creation of corresponding sample pairs, denoted as J_i, which are separated by a distance of d_i. A design X is referred to as a maximin design when it optimizes the variables d_i and J_i in a sequential manner, following the order of $d_1, J_1, d_2, J_2, ..., d_m, J_m$. The aforementioned method is accomplished by the utilization of a simplified scalar-valued function, which serves the purpose of evaluating and ranking competing designs.

The *Genetic Algorithm (GA)* is a computational approach for the optimization which mimic the gene evolution process in living bodies. It involves representing a design input as arrays of bits or character strings, known as chromosomes. These chromosomes are subject to manipulation operations performed by genetic operators. Selection of chromosomes is then carried out based on their fitness, with the objective of identifying a favorable or even optimal solution to the given problem [65, 160]. The aforementioned process typically involves the following main steps: the formulation of objectives or cost functions in a coded format; the establishment of a fitness function or selection criterion; the generation of a population comprising individual solutions; the execution of an iterative evolution cycle, which entails evaluating the fitness of all individuals in the population, generating a new population through operations such as crossover and mutation, and employing fitness-proportionate reproduction, etc. to replace the previous population; the decoding of the outcomes to derive the solution.

The *Nelder-Mead* algorithm, proposed by Nelder and Mead in 1965 [106], is a direct search technique that utilizes a simplex structure to solve unconstrained optimization problems in several dimensions. The simplex S in $\mathbb{R}^n$ can be defined as the convex hull of $n+1$ vertices, denoted as $S = \{v_i\}_{i=1,n+1}$. The algorithm utilizes an iterative process to update the size and form of the simplex by modifying a vertex in accordance with its corresponding function value. The equation $f_i = f(v_i)$ holds true for all values of i ranging from 1 to $n+1$. The centroid of a simplex, denoted as $c(j)$, refers to the central point of the vertices of the simplex, except the vertex v_j. The centroid $c(j)$ is defined as the average of the values of all v_i , where i ranges from 1 to $n+1$, excluding the value of v_j. Given that the problem at hand is a minimization problem, the vertices

are arranged in descending order based on their respective function values $f(v_i)$, with the vertex having the highest value assigned an index of 1, and the vertex with the lowest value assigned an index of $n+1$. By using a selected hyper-parameter, denoted as the reflection factor $\rho > 0$, the method aims to replace a vertex v_j with a new vertex $v_{\text{new}}(p, j)$ at each iteration. This replacement occurs along the line that connects vertex v_j to centroid $c(j)$. The formula for the new value function, denoted as v_{new}, is expressed as follows:

$$v_{new}(\rho, j) = (1+\rho)c(j) - \rho v(j)$$

The algorithm incorporates three additional parameters: the coefficient of expansion (χ), contraction factor (γ), and shrinkage factor (σ). These parameters play a crucial role in modifying the shape of the simplex during the expansion or contraction steps, allowing it to better conform to the local landscape. For more comprehensive information, please refer to Nelder and Mead's work on the simplex method [106].

Bayesian optimization [30, 77, 173] is a AI based method that starts with a probabilistic modeling of the function f that maps design space to observation space and use Bayes rule to iteratively update this probabilistic model on every new observations. The function f is generally modeled by probabilistic model called gaussian process (GP).

$$p(f) = GP(f; \mu, K)$$

Here μ is the mean vector and K is covariance matrix. On every new evaluation data $D = (X, f)$ the probabilistic model is updated on D as below:

$$p(f|D) = GP(f; \mu_{f|D}, K_{f|D})$$

The utilization of a probabilistic model of function f enables the building of an acquisition function ($a(x)$) that facilitates the identification of the optimal location for evaluating the function in next iteration. The acquisition function is commonly a cost-effective function that may be assessed at various locations in the domain using the probability model $p(f|D)$. This quantified assessment is the expected to improve the function f to the maximum at the chosen point x. For this purpose, the acquisition function is optimized in order to identify the most suitable place for the subsequent observation. In the context of Bayesian decision theory, it is possible to view $a()$ as the assessment of an anticipated regret linked to the assessment of f at a certain point x. There are two highly used acquisition functions: Expected Improvement and Lower Confidence Bound.

Expected Improvement : Given the current belief $p(f|D)$ on the function f, and assuming that f' represents the minimum value seen thus far, the Expected Improvement (EI) method, as described by Movckus

[98] and Jones et al. [73], determines the point at which the function f is expected to exhibit the greatest improvement above f'. In this particular scenario, the utility function can be expressed as:

$$u(x) = max(0, f' - f(x))$$

After conducting an evaluation, the reward can be determined as $f' - f(x)$, where f' represents the current minimum value and $f(x)$ represents the value of the function for the freshly evaluated sample x. If $f(x)$ is lower than f', the reward will be $f' - f(x)$; otherwise, the reward will be 0. The expected improvement, as a function of x, can be determined using this utility function:

$$\begin{aligned} a_{EI}(x) = \mathbb{E}[u(x)|x,D] &= \int_{-\infty}^{f'} (f' - f) N(f; \mu(x), K(x,x)) df \\ &= (f' - \mu(x)))\psi(f'; \mu(x), K(x,x)) + K(x,x) N(f'; \mu(x), K(x,x)) \end{aligned}$$

This expected improvement to the utility function makes an effort lower both the mean function ($mu(x)$) (which explicitly encodes exploitation by evaluating at locations with lower mean) and the variance $K(x,x)$ (which explores by evaluating in region of design space that have greater uncertainties). These improvements are both intended to improve the function's performance. The Bayesian decision-theory underpinnings of EI allow it to encompass both exploration and exploitation.

Another most used acquisition function is *Lower Confidence Bound* (LCB) [136]. The LCB aquisition function is defines as:

$$a_{LCB}(x; \beta) = \mu(x) - \beta\sigma(x)$$

$\beta \geq 0$ refers to a hyper-parameter that, during the optimization process, deals a balance between exploration and exploitation. The standard deviation (expressed as $\sigma(x)$) of the function $f(x)$ is defined as the square root of the product $K(x,x)$. Srinivas et al. [136] have provided an alternative way of looking at the acquisition function that was discussed previously in terms of regret. The regret is defined as:

$$r(x) = f' - f(x)$$

the optimization goal under LCB setting is:

$$min \sum_{t}^{T} r(x_t) = max \sum_{t}^{T} f(x_t)$$

where T represents the total cost of the evaluation for the process of optimization. Srinivas et al. [136] showed that with the value of the hyper-parameter β adjusted to $\sqrt{(v * \tau_t)}$ with $v = 1$ and $T_t = 2log(t^{d/2+2}\pi^2/3\delta)$ will have an no regret with very high probability, i.e., $lim_{T\to\infty}R_T/T = 0$, where R_T is cumulative regret.

2.2 Research works in underwater vehicle hull design

Gertler [50] carried out some of the earliest research that was done on the subject of discovering a UUV hull design with a low resistance to water flow. In order to achieve this goal, he created 24 unique bodies utilizing five parameters and a polynomial equation that describe the geometry of the UUV body. These bodies are collectively referred to as 'series 58'. To determine the amount of drag created by each hull design, these hulls were pulled through the water at varying speeds. In a later experiment, Carmichael [26] showed that the drag may be reduced for applications with a fixed frontal area or constant volume by lowering the fineness ratio or the surface area. When compared to traditional torpedo forms of comparable volume, Carmichael's tail-boomed body design, which he called 'Dolphin', achieved a 60% reduction in the amount of drag it generated in the Pacific Ocean during prolonged testing. This reduction is made possible due to the fact that the Dolphin shape maintains the laminar state of the boundary layer for a longer body length distance. One of the first people to develop a computer-based non-gradient algorithm as optimization procedure for an axisymmetric body of revolution optimization in finite constrained parameter space was Parsons et al. [109]. However, he used Young's formula for the drag estimation method that only applies to laminar flow conditions. Hertel examined the fuselage shape by considering fast swimming creatures in his research [62]. According to the results of his analysis, bodies with a revolution of profile that is identical to NACA's two-dimensional laminar forms do not give the appropriate pressure gradient for the boundary layer to remain laminar. On the other hand, bodies with a parabolic nose, such as Dolphin and Shark, provide a more laminar boundary layer. Myring [102] introduced an empirical methodology for predicting the drag of a body of revolution by utilizing the viscous-inviscid flow interaction method. Myring conducted stabilization of the transition site and conducted an investigation on the variance in drag at a Reynolds number of 10^7, while manipulating the body shape. He initiated the experimentation process by systematically modifying individual parameters, observing the subsequent impact on drag experienced by various body components such as the nose, tail, and others. Based on the his results, it has been observed that the alteration in body drag experiences a decrease when transitioning from a slender to stout nose or tail within a specific range. However, it undergoes a notable increase after the design surpasses this range.

Hess [63] formulated a simplified drag equation and employed it to conduct a comprehensive comparative analysis of drag efficiency across diverse body types. Zedan and Dalton [169] conducted a comprehensive analysis comparing the drag characteristics of various axisymmetric body shapes that were considered to be

among the most optimal designs. In the course of their investigation, the researchers made the assumption that turbulent boundary layer phenomena would prevail across a significant portion of the body surface. They subsequently arrived at the conclusion that, under conditions of a high Reynolds number, the presence of a laminar boundary layer capable of persisting over a certain distance would result in a significant reduction in drag. This study additionally posits that the unorthodox laminar-shaped structure exhibits the most favorable characteristics as a potential contender for achieving the lowest drag design at a high Reynolds number. The study direction of bionic-inspired hull shape for UUVs was gaining popularity as a means to enhance performance. In pursuit of this objective, Dong et al. [37] developed a gliding robotic fish that emulates the streamlined morphology of a whale shark. Additionally, the researchers demonstrated that the fish can effectively attain both a high degree of mobility and exceptional gliding capability through the utilization of adjustable fins and tails. In their study, Lutz et al. [92] devised a computational approach for shape optimization and incorporated a linear stability theory to assess the drag characteristics of axisymmetric structures. Utilizing this methodology, a low drag design was devised for a specified internal area. Alvarez et al. [6] have devised a first-order Rankine panel approach in their study to improve the hull shape of an UUV.

In the present era, the progress in computing power and the enhancement of mesh-based analysis tools have enabled the extensive application of CFD simulation in the evaluation of hydrodynamic performance of underwater unmanned vehicles (UUVs). The predominant approach employed in the field of study involves the utilization of either the Reynolds-averaged Navier-Stokes (RANS) formulation or the large eddy simulations (LES). Reynolds-Averaged Navier-Stokes (RANS) theory is frequently utilized in comparison to Large Eddy Simulation (LES) due to its enhanced treatment of viscous effects and lower computational demands. Stevenson et al. [137] developed a series of seven discrete revolution bodies, each of which was standardized to have the same volume. The major objective of their inquiry was to examine the drag characteristics displayed by these entities. The results suggest that a UUV body featuring laminar flow may demonstrate greater efficiency in comparison to a body shaped like a torpedo, given that the placement of external ancillary components is carefully done. Nevertheless, it is important to acknowledge that this particular design is more vulnerable to the potential consequences of manufacturing imperfections. Wei et al. [162] conducted a study whereby they undertook the design and evaluation of five commercially available UUVs. The primary objective of their investigation was to assess the performance of these UUVs in terms of two crucial metrics: speed and endurance. To accomplish this goal, the researchers performed an inquiry into five various forms at different velocities using CFD analysis. The research findings suggest that two of the participants display a higher degree of adaptability towards higher speeds, whereas three participants exhibit a larger level of adaptability towards lower speeds. In the year 2005, Yamamoto [168] created a long-range cruising autonomous vehicle, which subsequently established a global record for the longest duration of autonomous cruising. The

current vehicle is equipped with an X-shaped wing arrangement, which provides a larger wing surface area and improved control force in comparison to a conventional cross-shaped tail, and vice versa. Furthermore, he provided evidence to support the claim that the streamlined-shaped model displays a decrease in drag and an improvement in range when compared to the cylinder-shaped model. The integration of hydrofoils into the framework, together with the utilization of specific navigation strategies, led to a significant enhancement in the range of the UUV by approximately two times when compared to its initial design including a cylindrical body. In their study, Schweyher et al. [128] utilized an evolutionary strategy to minimize drag on the body, with a specific emphasis on the optimization technique adopted. Eismann et al. [39] conducted a study to examine the application of Bayesian optimization in the field of form optimization, specifically in the pursuit of minimizing drag in a two-dimensional flow field.

The application of surrogate modeling has been increasingly widespread in the field of engineering during the past few decades. The methodology employed in this study entails the replacement of complex engineering simulation techniques with Kriging/Gaussian Process-based machine learning models, as outlined in the works of Rasmussen and Williams [116] and Forrester et al. [45]. A multitude of recent studies have concentrated on the advancement of surrogate models for approximating the characteristics of flow fields, while also examining the influence of fluid flow on the morphology of designs. Bhatnagar et al. [16] devised and executed a study whereby they constructed and deployed a convolutional neural network (CNN) framework using an encode-decoder approach. The objective of this architectural design was to predict the flow characteristics of aerodynamic flow fields on two-dimensional airfoil configurations. Chen et al. [28] utilized a learning framework that incorporated U-net [121] to generate predictions of 2D velocity and pressure distributions in laminar flows, particularly in the vicinity of irregular geometries. Machine learning-based surrogate models are frequently utilized in various engineering disciplines to accelerate the design discovery process [150, 156]. There are various areas in which these investigations demonstrate shortcomings and present possibilities for improvement. The dynamics governing the development of fluid motion can be elucidated through the utilization of either a laminar or a simplified turbulence model. The usage of advanced modeling approaches is necessary to effectively represent the turbulent elements of flow characteristics in real-world UUV systems due to the design constraints connected with them. The k-ω shear stress transport (k-ω SST) turbulence model is extensively utilized in the field of UUV design owing to its proven accuracy in predicting outcomes, as evidenced by the research conducted by Jones et al. [72]. There is a lack of existing research that specifically examines the approximation of flow behavior and its consequences using the k-ω SST turbulence model. Previous research in the field of CFD has predominantly concentrated on surrogate modeling, with a particular emphasis on tiny airfoil and two-dimensional geometries. Although the aforementioned studies offered useful insights into the behavior of CFD, they failed to comprehensively

address the intricacies involved in developing substitutes for design scenarios seen in real-world settings. The research did not sufficiently address factors such as the failure of the CFD process caused by meshing issues, the time required for analyzing each sample, and the possibility of creating a satisfactory number of samples within a reasonable period.

2.3 Surrogate-based design optimization

A surrogate model is defined as a non-physics-based approximation typically involving interpolation or regression on a set of data generated from the original model [3]. We consider two fundamentally different motivating use cases of a surrogate model from an engineering perspective:

1. Model-based design process
2. Model-based system operation

The *model-based design process* incrementally synthesizes the model of the designed system, usually starting with component models. The design process typically follows the progressive refinement of the various design models while incorporating optimization and verification phases. The key characteristics of the models are evaluation complexity and fidelity in terms of some selected properties. Generally, computationally expensive high-fidelity models describe systems with high accuracy, while low-fidelity models are less accurate but computationally cheaper than high-fidelity models. One of the primary motivations of surrogate modeling is to speed up the evaluation of models that are otherwise prohibitively expensive to run in optimization processes.

In *model-based system operation*, embedded models are used for implementing various functions ranging from monitoring and control to diagnostics. Since the operation time evaluation of an embedded model is subject to real-time constraints, evaluation complexity is a driver for using a simplified surrogate model. The primary motivation is to reduce the operation time using the surrogate model. The surrogate of the embedded model is computationally cheaper. Another motivation for using a surrogate model is to design a human-level intelligent control, diagnostic, or monitoring system that cannot be designed without AI-ML methods. The goal is to learn from human intelligence or observations in simulated behavior to replace or assist humans in the loop process.

The general approach for creating a surrogate model is to fit a generic parametric model using data generated by simulation tools. Since AI and ML models are more flexible parametric models, these are the only models in consideration in the context of this manuscript. The abstraction used in a high-fidelity model mathematically describes the evolution of a set of continuous-time dynamical system equations or numerical simulations or optimization loops or evaluation of controlled behavior under consideration.

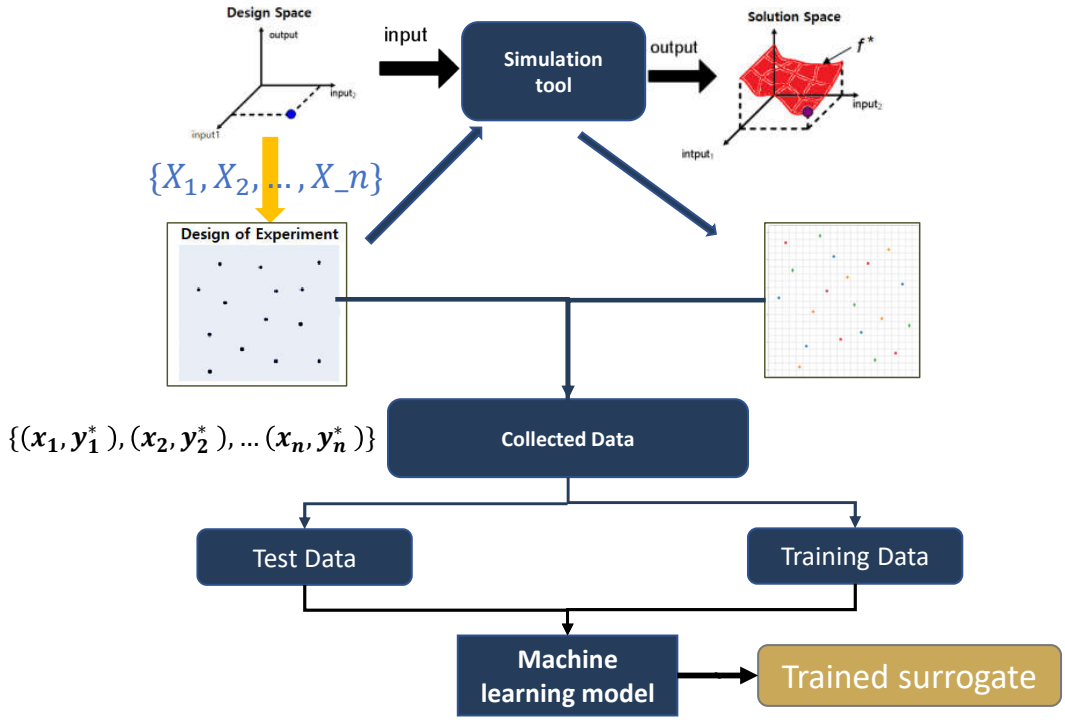


Figure 2.1: Overview of data-driven Surrogate Modeling

Figure 2.1 shows the high-level overview of a surrogate modeling process. The main elements of this process are *data generation*, *model selection*, *training*, and *model validation*.

- Data for surrogate modeling can be generated either by running a high-fidelity simulation tool, by gathering real-world data, or by using both sources. If we are using a simulation-based environment to generate the data, the domain of approximation/environment of operation needs to be defined apriori by a range of parameters. For a better approximation, we need to expose the original system with all possible combinations of inputs or environment scenarios which exposes all possible behaviors of the system. Design-of-experiment (DOE) is crucial for the data generation which affects the accuracy of the surrogate. The goal of the DOE/sampling plan is to expose the original system with those possible combinations of inputs that expose all possible behaviors. Designing a sampling plan is not trivial. Every sampling plan introduces some bias and results in different variances. If sample D corresponds to a random subset from the domain of interest, bias on a trained surrogate quantifies the extent to which the surrogate model outputs differ from the true values calculated as an average over all possible data sets D. However, variance measures the extent to which the surrogate model is sensitive to particular data set D. In principle, we can reduce both, the bias by choosing a flexible model and the variance by

increasing the number of evaluated points for training.

- The next crucial step is the selection of a learning model. Each machine learning model has its advantages and disadvantages. Some ML models are easy to train (e.g., random forests, decision trees, KNN), some can provide predictions uncertainty (e.g., Gaussian Processes (GP), Bayesian neural networks (BNN)), and some can extract important features from high dimensional data space (Deep Neural Network). Disadvantages include the need for large data sets for training (Deep Neural Networks), a very high computational complexity (GPs, BNNs) or not working well with high dimensional data (decision trees, random forests, GPs). It is a strategic decision to select a model and it is decided based on the surrogate modeling problem under consideration and the goal of the surrogate creation.

- Once the model's training is finished, some validation technique is used to test the model's ability to approximate the output (or some derived property of the output) generated by a high-fidelity model or a real system in the domain of interest. Testing machine learning models is an emerging and challenging research topic. There are various aspects against which a machine learning model should be tested, like correctness, robustness, security, or interpretability, but the general scope of testing is in the context of *correctness*. The empirical correctness of a machine learning model is the statistical quantification of producing the desired outcome during prediction. The goal of testing is to evaluate the correctness of a trained model and/or find a *good test scenario* that can expose the failure modes in the trained surrogate.

2.4 Major challenge in surrogate modeling of complex engineering process

During surrogate modeling, design space exploration is a process to understand the effect of design variables on the performance metrics and analyze the complex relationship between these variables on the performance metrics. The process usually involves successive sampling in the design space and analyzing the performance metric to find a promising design point or subspace which satisfies all the constraints and meets the necessary performance requirement. Learning these nonlinear hyperplanes about the relationship between input design variables and performance metrics is the role of a human design engineer. For this purpose, the design engineer evaluates/simulate the design space at various samples and tries to create a response surface in his mind. This response surface is further used in the future to find the design for a given performance metric. However the process has multiple issues: first, without any strategic sampling approach, the trivial randomized sampling approach to generate data becomes unavailing in high-dimension design variable space because the number of points needed to give reasonably uniform coverage rises exponentially- this phenomenon is infamously called the *curse of dimensionality*. Another issue is related to the simulation models and simulation

process, which has an iterative subroutine of optimization/convergence and consequently a long simulation time. Due to these reasons in most practical problems, design engineers always search for optimal design near an already explored region of design space. An alternative is to freeze many of the design variables at hopefully meaningful values and work with just a few at a time, iterating around to understand the impact of various variables progressively. Both of these issues do not resolve the problem of efficient global exploration. The utopian solution would be 'to learn the hyper-plane with few numbers of strategic simulation evaluations' and the learned model should have the 'capability to parallelize the evaluation process' on modern hardware (GPU). For a given design space with known boundaries, the goal of surrogate modeling is to create an alternative model for the response surface created by the simulation process. The benefit of creating a surrogate is parallel faster evaluation of design points, which can speed up the whole design decision process. However, the preliminary questions that need to answer before creating a data-driven surrogate in real design space are:

In the context of surrogate modeling, the practice of design space exploration is employed to get a thorough comprehension of the influence exerted by design variables on performance measurements. This investigation also entails examining the complex relationship that exists between these variables and the metrics of performance. The standard protocol involves a repetitive process of sampling from the design space and assessing the performance metric to locate a design point or subspace that shows potential, while also satisfying all constraints and meeting the necessary performance standards. The responsibility of understanding these non-linear hyperplanes, which illustrate the relationship between input design factors and performance metrics, lies within the domain of a human design engineer. To accomplish this goal, the design engineer evaluates and models the design space at multiple data points, with the intention of developing a cognitive representation of the response surface. The response surface indicated above can afterwards be employed for the goal of determining an ideal design based on a certain performance metric. However, the approach is afflicted by various challenges: In the absence of a well-defined strategic sampling methodology, the conventional randomized sampling technique for data production proves to be inadequate when applied to a design variable space characterized by a large number of dimensions. This phenomenon can be attributed to the exponential growth in the necessary quantity of points needed to attain a relatively even distribution. The widely recognized issue under discussion is frequently denoted as the 'curse of dimensionality'. Another worry arises in relation to the simulation models and simulation process, which involves an iterative subroutine that includes optimization and convergence, leading to an extended duration for the simulation. Design engineers continually strive to develop an ideal design within a previously explored region of the design space due to a variety of practical challenges. The produced design exhibits local optimality, indicating its effective performance within its immediate contextual framework. Nevertheless, it is crucial to acknowledge that this

particular design may not exhibit optimality when compared to all possible designs. One strategy that can be employed is the fixation of a considerable number of design variables by selectively changing a small selection of variables at any given moment. The utilization of an iterative method facilitates a systematic and incremental investigation of the impacts of many variables. Both of these problems fail to adequately tackle the issue of attaining efficient global exploration. The optimal strategy would entail gaining an understanding of the hyperplane through a restricted set of strategic simulation assessments. Furthermore, it is essential for the obtained model to demonstrate the capability of parallelizing the assessment process on modern hardware, particularly Graphics Processing Units (GPUs). The primary goal of surrogate modeling is to construct an alternative model that accurately represents the response surface produced by the simulation process inside a designated design space that has pre-established boundaries. The use of a surrogate provides the benefit of accelerating the analysis of design factors simultaneously, hence improving the efficiency of the whole process of making design decisions. However, it is important to consider a series of fundamental inquiries prior to constructing a data-oriented substitute within the framework of tangible design parameters.

1. In the context of sampling, the magnitude of the step size refers to the size or interval between consecutive data points or samples. The available approaches currently lack a means to determine an acceptable step size in the absence of hyperplane information. The accuracy of replicating outcomes through the use of a surrogate will be heavily influenced by the meticulous choice of the sampling step size.

2. What is the suitable process for sampling in order to acquire data? The utilization of planned and adaptive sampling strategies for data collecting is crucial due to the impracticality and probable exponential growth associated with randomized sampling.

The selection of the sampling method is of utmost importance in the creation of the surrogate model and has a direct influence on the precision of the acquired knowledge regarding the surrogate's behavior. In the next section, we will analyze the process of sampling, which includes both the traditional method of sampling and the active/adaptive sampling technique, within the context of surrogate modeling.

2.4.1 Sampling for data generation in design space

Sampling is an essential component of surrogate modeling as it facilitates the investigation of causal relationships between input parameters and their interdependencies. The sampling process can be classified into two primary components: the conventional Design of Experiments (DOE) and adaptive sampling methods. The conventional Design of Experiments (DOE) includes several sampling strategies, such as Latin hyper cube [94], factorial-based sampling (including full factorial and n-level factorial designs), response surface designs (such as Box-Behnken and central composite designs), and randomized sampling. Additionally, the

alternative approach referred to as active and adaptive sampling incorporates several sampling tactics like uncertainty sampling, predicted error reduction, and variance reduction [130]. The categorization is based on the underlying notion that machine learning algorithms in active and adaptive sampling have the ability to autonomously choose samples. This is in contrast to the traditional strategy in which samples are predetermined before the training phase, making the sampling process separate from the model. The standard sampling strategy lacks the advantage of incorporating any novel findings during the later modeling phase as a result of the fixed selection of sample locations. In contrast, adaptive sampling refers to an approach that utilizes previously learned information of sampled behavior to make informed recommendations regarding the optimal locations for sampling within the design space. The primary aim is to acquire the most valuable data that may be utilized to improve the training model being examined. It is commonly acknowledged that active and adaptive sampling demonstrates superior performance compared to standard sampling when suitably calibrated. In specific situations, the integration of these two techniques might result in synergistic effects, leading to optimal outcomes through the utilization of the respective strengths inherent in each paradigm.

2.4.1.1 Traditional Design of Experiments

Traditional DoE approaches have been developed mainly for costly physical experiments (like Finite Element Analysis, Computational Fluid Dynamics, etc) that have a long history. The most used classical sampling approaches include factorial designs (Full Factorial design, Fractional Factorial design), response surface designs (Box-Behnken design, central composite design), and Plackett-Burman design and other randomized designs (uniform random sampling, Latin hypercube) [94] etc. These sampling approaches determine the sample locations in order to make maximum coverage of design space based on a predetermined strategy. These strategies widely affect the performance of learned behavior. Each method has a notion of predetermined bias based on assumptions they made during sampling. For example, full factorial tends to cover the entire space uniformly spaced manner to gather the most information about the underlying function while central composite design samples more points around the boundary regions rather than the interior regions in the expectation that the most interesting things are happening in boundary regions. Similarly, the Latin hypercube which is an extension of stratified sampling, ensures each input parameter has all portions of the range represented [94]. All these traditional sampling methods introduce some predetermined bias in the sampling process. Among all of them, empirically it is observed that Latin hypercube is the most used traditional sampling method [158] as it covers the entire range of each input variable and has non-colliding in nature and extracts the most out from these samples in a more systematic way for discovering scientifically surprising behavior.

The vanilla Latin hypercube sampling/ simple Latin hypercube sampling was further extended in various other flavors like maximin Latin Hypercube [146], orthogonal arrays criterion based LHC [74, 85, 91], columnwise-pairwise [138], genetic algorithm [83], simulated annealing [111] etc. All these efforts can be perceived as attempts to include additional information while designing the LHC scheme to create an optimized DOE. However, the exact optimal solution to this DOE problem has been believed to be NP-hard [120]. The extensive study and comparison of different flavors of optimal LHC can be found in [36]. The further extension of LHC is done to optimize the sampling budget using translational propagation idea [108, 159]. It first creates small blocks containing a good seed design with a few points and then translates them over the entire hypercube. This approach proved to be much more efficient than the formal optimization-based approaches.

Another aspect of traditional DOE is it works in a single shot mode which implies that all the design points are generated in a single stage before evaluating the samples which is different from another strategy of DOE i.e. sequential DOE. In sequential DOE, we decompose the sampling problem in multiple iterative steps on a given computational budget. Directly converting single-shot traditional DOE to sequential DOE without a strategy can be counter-productive, due to space-collapsing properties of samples in less useful regions of Design Space. The more general approach to converting traditional single-shot DOE sampling to sequential DOE is to apply a space-filling approach with a distance-based threshold to ensure the separability of samples. This approach of constraining and generating sequential DOE is very widely studied and extended in various works [34, 166]. This section offered an introductory review of the traditional Design of experiment-based sampling approaches for surrogate modeling. In the next section, we will discuss the adaptive approach of sampling for surrogate modeling.

2.4.1.2 Adaptive/Active sampling for surrogate modeling

Active learning is an interesting sub-field of machine learning and has a long history in the statistics literature, generally referred to as optimal experimental design [41]. The active sampling approach uses the key hypothesis that if the learning algorithm is allowed to choose the data from which it learns—to be "curious," if you will—it will perform better with less training [130] and can achieve exponential acceleration in sample labeling/evaluation efficiency in training process [11]. This property of less training data is desirable, especially in high dimensional computationally expensive simulation-based design decision settings. The active learning sampling method is generally classified in the context of a scenario in which learners may be able to ask queries. These are classified as (1) stream-based selective sampling, and (2) pool-based sampling. In stream-based sampling, each unlabeled instance is typically drawn one at a time from the data source, and the learner decides whether to query or discard it. The main assumption is that obtaining an unlabeled

instance is free (or inexpensive), so it can first be sampled from the actual distribution, and then the learner makes a decision about whether to label it or not. In pool-based sampling [81], it is assumed that we have a large collection of unlabeled data, which is the case in many real-world machine learning problems. The goal of this sampling process is to select a small set of data and label it and this small labeled data should represent the whole pool of unlabeled data for the learning process. The difference between stream-based and pool-based active learning is that stream-based scans through the data sequentially and makes query decisions individually, whereas pool-based evaluates and ranks the entire collection before selecting the best query. Both methods have been widely applied to many real-world applications ranging from part-of-speech tagging [35], sensor scheduling [76], sentiment analysis [135], image classification and retrieval [143], video classification and retrieval [58], cancer diagnosis [90] to name a few.

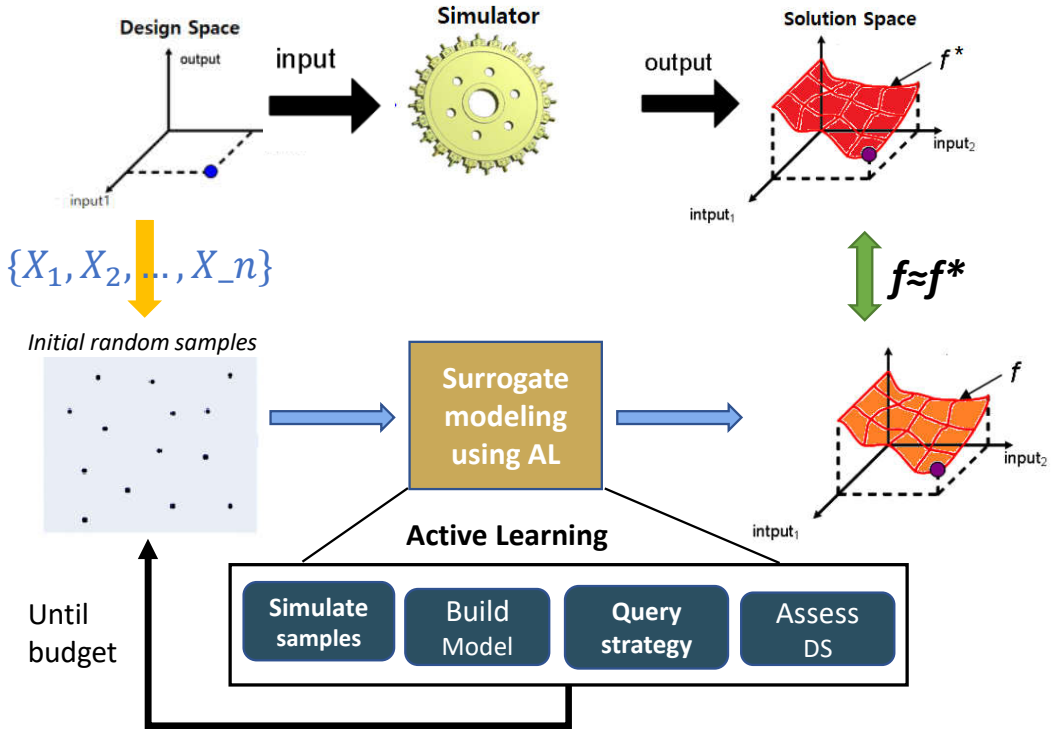


Figure 2.2: The general process of Active Learning (AL) approach

The general flow of the active learning approach in the context of surrogate modeling is shown in figure 2.2 (in other cases, the simulator is replaced by some other oracle). Active learning is an iterative process and involves various steps. The first step in the process is called the 'warm-up' stage, in which random samples are generated using the traditional sampling method and used to train the model, shown as the 'Build Model' stage. Once trained, either the entire sample is assessed (in the case of pool based scheme) or sequentially

samples are assessed using this trained model. The prediction made by the trained model on these candidate samples is used to synthesize a *query strategy*. Query strategy is a statistical measure of the informativeness of unlabelled samples. Both cases of active learning involve evaluating the informativeness of unlabeled instances. There are different query strategies has been proposed and used by researcher and these can be majorly classified in:

1. Uncertainty-based sampling: The most commonly used query strategy is uncertainty sampling [81]. In this approach, a query is made to the instances about which the ML model is least certain how to label. Using Shannon's entropy function [133], the informativeness about the certainty can be easily evaluated mathematically. This approach is tightly bonded in the probabilistic framework and work well with classification problem. Researchers have used this flavor of query sampling for regression problems in the Gaussian Process framework [33], where uncertainty is equivalent to variance. In such cases, the difference between uncertainty sampling and variance-based sampling blurs.

2. Query by committee: The query-by-committee (QBC) [46, 132] based adaptive sampling is ensemble-based, where an ensemble of models is trained on the same or different parts of the data. The response variances estimated by several metamodels are considered as ensemble uncertainty. Some major investigations in this field includes diversification of models [95] and boosting/bagging exploitation [2, 22].

3. Expected model change: It is a decision-theoretic approach, which selects the instance that would impart the greatest change to the current model if its label is known. An example of this query strategy would be the 'expected gradient length' (EGL) approach [131] for logistic regression class.

4. Variance reduction: The variance-based adaptive sampling [71] forces the samples in the regions with large prediction variances estimated by the machine learning model, with the underlying assumption that more variance results in a larger error. In this case, the ML model starts with prior probability distribution from all points in the input domain, and then the model is trained on the observed/evaluated sample through the Bayesian rule, which offers the posterior distribution on the prediction response on all the sample in the design space in form of $y(x) \sim N(\mu, \sigma^2)$. The successive construction and provided standard deviation (σ) are regarded as estimation of actual prediction error and employed to assist the sampling process for reducing model uncertainty.

The query strategies in the active learning paradigm are tightly interwoven with the underlying machine learning model and the deployment of a class of query strategies can be directly related to the class of ML model under consideration. From this perspective, the query strategies can be classified into two-part- *model dependent* and *model-independent* query strategies. For example, most uncertainty-based query strategies

are compatible with models which can predict probability as the output prediction. Similarly, the variance-based approach is tightly coupled with kriging, Gaussian process [116]. The query-based committee can be deployed only in the setting, where a population of models is deployed to learn a goal. Some model-independent approaches like the gradient-based approach and cross-validation approach can be useful for a large class of models but the expected benefits in each case are not guaranteed. A detailed study on various extensions and approaches can be found here [89].

There are various potential benefits of using AL in learning problems however vanilla AL finds it difficult to work in high dimensional data [142]. Additionally, AL often queries on already extracted features in advance and does not have the ability to extract features. On the other hand, a deep learning model/deep neural network can automatically extract features from the input data and work very well in high dimensional data has proven its role as a universal function approximator and strong learning capability due to its complex structure can benefit to the surrogate modeling process for a regression problem. Deep active learning (developing as a sub-field in machine learning by combining deep Neural network models with an active learning framework) has many promises for engineering design. Merging deep learning with active learning may achieve superior performance and it is demonstrated in various fields of work like image recognition [47, 67], text classification [126, 171], and object detection [4, 42] etc.

From the perspective of a learning task, design problems can be classified as classification and regression problems. The design decision which is a parametric search of a component/model is a regression problem. Although regression problems are long studied in the context of surrogate modeling [41, 45] but still it is challenging for most engineering problems due to input-output manifold complexity. DL has empirically proved to learn highly non-linear complex manifolds. Most of the work in the field of DeepAL is done in the context of classification problems, however, there is very little work for a regression problem. In the regression case, the most sought approach is to use an ensemble of neural network [69] or Bayesian neural network [18] to get an indirect measure of variance. These approaches are not scalable due to the training of multiple neural networks and have very little practical utility for learning a large class of behavior. There are few other works that try to incorporate the spirit of active learning into a single deep neural network. [144] used random drop-outs in the trained model to get the indirect measurement of variance for selecting a sample. [82] used the gradient-based approach for active learning in a single deep neural network. The active learning framework directly addresses the estimation of the utility of candidate data points. On the other hand, the deep Learning model has powerful learning capability and can scale to high dimension data and automatically extract features in large dimensional problems. By combining DL with AL, called DeepAL we can retain the strong learning capability of DL while getting benefited from strategic samples selected by AL.

Table 2.1: Models and algorithms implemented in SMT and Dakota

SMT	Dakota
Linear regression and Polynomial regression	Polynomial regression
Kriging	Gaussian process (Kriging interpolation)
Gradient enhanced neural network	artificial neural network
Kriging with partial least square (KPLS)	Multivariate Adaptive Regression Spline (MARS)
Inverse-distance weighting	Moving least squares
Radial basis functions	Radial basis function
Gradient enhanced Kriging with partial least square	Voronoi Piecewise surrogate(VPS)
Regularized minimal-energy tensor-product splines	

2.4.2 Software tools for surrogate modeling

In this section, we review two open-source surrogate modeling tools called SMT (Surrogate Modeling Toolbox) [21] and Dakota (Design Analysis Kit for Optimization and Terascale Applications) [3], developed by Aerospace Engineering laboratory at the University of Michigan and Sandia National Laboratory respectively. There is another commercial tool for surrogate modeling which is available for limited use free version called SUMO (SUrrogate MOdeling Toolbox) [52].

Both open-source tools have some common models and algorithms. A comparative list of machine learning models and algorithms implemented in both open-source tools is in table 2.1. Each implemented models have distinct advantages and disadvantages. Most models fall in the category of discriminative learning except for kriging and kriging-related methods (KPLS and gaussian process). VPS is an attractive model that helps to tackle the exponential growth in data with increasing dimensions. Although these tools are good starting points, they are still deficient in handling a larger class of simulation and system operations due to a lack of tight integration with active sampling approaches. This gives the scope for further work and extending these methods. Also, various new machine learning models are unexplored in these tools like bayesian neural networks, gaussian process for large data, gaussian mixture models, random forests, etc. AI is an evolving field and new methods and techniques are being developed at a rapid rate. The exhaustive list of freely available machine learning and DOE software tools is incomprehensible but we try to list down a few mostly used and important ones below:

1. For GP and its variant [53]
2. Bayesian Optimisation in PyTorch: BOTorch [10]; GPyOpt [54]
3. Graph algorithms in PyTorch: pytorch_geometric [43, 114]
4. PyMC [pyMC authors] ; TFP(Tensor Flow Probability) [probability]
5. Scikit -learn,gstat and other packages of Scikit [80]

6. Bezier(python) for Bezier fit [61]

7. Scipy.stats for statistical methods [79]

8. pyDOE for the design of experiment in Python [pyDOE authors]

Another aspect that is missing in these tools is the rigorous evaluation of the surrogate model. As machine learning models are prone to fail and sometimes they fail even after showing very high confidence in the prediction. Without a rigorous evaluation framework, it would be difficult to rely on the prediction made by these surrogate models. As there is no unified evaluation framework for ML models, they need to be studied diligently. The next section will provide a literature study about the evaluation of surrogate models.

2.5 Evaluation of trained surrogate model in the context of functional correctness

The evaluation of machine learning models is an emerging and challenging research topic. There are various aspects against which a machine learning model should be evaluated, like functional correctness, robustness, security, interpretability, etc. In this manuscript, the scope of the evaluation is in the context of *functional correctness*. Evaluation techniques for a surrogate model depend upon its use case. Evaluation of the surrogate model used for *optimization* in the design process/system operation is intended to establish two key requirements: (1) the amount of speed increase in the process of design optimization/search/prediction and (2) functional correctness. The first evaluation metric depends on the domain-specific use case of the surrogate model. The second evaluation metric is more general and applied to a general framework of machine learning testing. *Functional correctness* is defined as a statistical measure of equivalence between the surrogate model's behavior and desired behavior on training and validation data. Good functional performance ensures that the surrogate model will reproduce desirable behavior on data drawn from similar distribution as collected data. This Performance is generally measured on the testing and validation data to ensure the learned behavior is the desired one. However, a good performance guarantee on testing data does not always guarantee 100% functional correctness. This requirement becomes a very critical aspect when a surrogate model is used in system operation in a safety-critical system. This lead to the challenging problem to detect input signals on which the surrogate model is highly likely to fail. *Rare events* are those failures that have a very low probability to occur, and the model performs its desirable behavior most of the time. For the system operation use case of the surrogate model, even such rare failure can be catastrophic as it operates on a safety-critical real-world CPS system. The standard evaluation procedure for detecting a failure in a machine learning model is to run a simulation of a maximum up to training iteration or size of training data for detection of failure. [145] showed that this conventional Monte Carlo approach can miss failures entirely, leading to the deployment of unsafe embedded surrogate models. The attempt made by [145] is to screen

out situations that are unlikely to be problematic and focus evaluation on the most difficult situations (corner cases) in a probabilistic rigorous framework, to learn a failure probability predictor. For learning a probability predictor function, it is required to collect all these rare failures. But, a major impediment in doing this is since failures are rare, even after running very long simulations we may get very few failures to derive any conclusion. To address this problem, [145] introduces a *continuation approach* to learning a failure probability predictor, which estimates the probability the agent fails given some environmental or system conditions.

Another important aspect of evaluation is the detection of shifts in input data distribution or out-of-distribution detection. The training process of ML models involves the collection of data and then training on these collected data. During operation or prediction, we provide input to these trained ML models and get an output. As machine learning models are trained and validated against collected training data, their performance with new input that is not consistent with the statistical properties of the training data cannot be relied on. Therefore, it is imperative to have some mechanism to verify that a given input data is sampled from the training data distribution. An Out-of-training-distribution (OOD) data point is one that is significantly different from the training data i.e. a data point that is an anomaly relative to the training data so that it may stir speculation that it was generated by a different mechanism [59]. In the next section, we will see in detail the current state of the art for both evaluation metrics.

2.5.1 Rare event failure test case generation for in-distribution training data

The goal of testing an ML model is to evaluate the correctness of a trained model and find a *good test scenario*. We define a *good test scenario* as a test case that can expose the potential fault in the model behavior and provide reasoning about correctness. A general software testing paradigm can not be deployed for machine learning models as these are more statistically oriented data-driven programmable entities, where the logical decision boundary is outlined via the training process. The behavior evolves during the training process and the result of training cannot be outlined prior to empirical testing, which is contradictory with the traditional software testing paradigm, where the desired behavior is fixed first and then the underlying behavior of software is designed. This behavior does not dynamically alter with the amount of information in data. Contrary to traditional software testing, test case generation in machine learning is based on observed training data. Most machine learning models operate in high dimensional space, where sufficient testing by sampling the entire data space is not possible due to the enormous number of required samples. Another aspect that is challenging for ML model testing is the *oracle problem* [12] because the machine learning model is designed to find the solution to a problem for which the actual answer is not known. With any previous answer, it is difficult to do accurate testing [101]. Due to these challenges sometime ML models are

regarded as 'non-testable' software. With limited computational resources or available time, the alternative approach of testing is used by either separating collected data in train & test class or generating input test cases by vanilla Monte Carlo sampling up to the maximum size of training data for detection of failure. Empirical observation reflects that such testing may miss failures entirely, leading to the deployment of an unsafe trained model. *Rare event failure* is those failures that have a very low probability to occur, and the model performs its desirable behavior most of the time. For example, failures in Google's autonomous vehicle fleet, which was test-driven approximately 1.3 million miles in autonomous mode and was involved in 11 crash failures from 2009 to 2015 [19] are rare failure scenarios. These rare failures can be catastrophic in some cases. Finding these scenarios may help to retrain these models and increase the correctness of the model. A systematic and extensively automatable way to search for a good test scenario may increase the effectiveness and efficiency of the test and thus reduce the cost of testing as well as increase correctness in the system. Some early work for searching a good test case in an autonomous system uses a fitness function, that assigns numerical qualitative value to a test case [161]. By defining a fitness function, the search problem can be converted into an optimization problem, which is mathematically and computationally tractable. The fitness function framework is extended for generating test cases in the context of machine learning models. [57] addressed the methodological challenge of creating suitable fitness functions by formulating a fitness functions template for testing automated and autonomous driving systems. By attaching fitness functions to different test scenarios, recent works use either clustering algorithms or evolutionary algorithms to find a good test case for both single and multi-objective fitness function [1, 15]. Another approach to generating a failure test scenario is attempted through adversarial attack [170, 174]. [140] proposed several adversarial example generation methods for attacking deep neural networks. These methods majorly generated an adversarial example via adding calibrated perturbation to an image data that can confuse the DNN. [51, 99] have shown various techniques to generate adversarial examples in deep neural network (DNN). Although these works are done for image data we use the same DNN to learn a policy in a reinforcement learning setting and consequently they fail in a similar way. Some early work in finding failure in RL agent is done by [66, 86]. [66] proposes an adversarial attack tactic where the adversary attacks a deep RL agent at every time step in an episode. To create failure, it considers adversaries have the capability to introduce small perturbations to the raw input of the DNN. They added different perturbations based on whether the adversary has access to the deep policy network or not. [86] modified this approach in attempting to find the failure of the RL agent. For that, they designed two different attacks on the trained agent. In one case, they aimed to minimize the agent's reward by attacking the agent in some intermittent simulation time steps in an episode. In another form of attack, they lured the agent to some designated state rather than reaching the target state. In this kind of attack, they predicted the future state by using a generative model and generated a sequence of actions

that divert the agent from its target position. In all the above Adversarial input test cases are a perturbed version of original inputs so, they may or may not belongs to training data distribution. Although these inputs expose robustness or security flaws in the trained model, it is not justified to demand the ML model to work on a distribution that is not seen during training. The goal of a good testing process is to expeditiously find failures scenario in input training data distribution. There are only a few works in machine learning testing in the context of in-distribution rare event failure search. [145] implemented a guided search approach to find a rare event failure scenario by learning from training data and focusing on a region of input space that has a high probability to fail based on failures that occurred during training. Data from earlier failures were used as a driving signal for the segregation of input search space. Using these earlier failures, a failure predictor (called AVF) was trained, which produce the likelihood of failure on the given input sample. However, this guided search process has high variance, which depends on the training process. This underlying reason is the result of non-monotonic improvement in policies during training. As these policies are used for data generation by adding stochastic noise/variable to the action for exploration, the convergence to a good policy is arbitrary and earlier failure scenarios become ineffective to train a good failure predictor. Attempts were also made to control the step size for policy updates for making learning smooth, but it had a small effect on the performance of the predictor, and finding a good step size for monotonic improvement in policy is not a trivial problem.

2.5.2 Anomaly detection/Out-of-distribution detection

The OOD detection goal is to find whether input data X' given during prediction is sampled from $\mathbb{E}_t$ (environment, in which it is trained) or not. If $X' \sim \mathbb{E}_t$, then we call this observed state non-OOD, or else we call it OOD, i.e. the trained agent has not seen this kind of input during training and the trained agent may behave unexpectedly to this OOD input data. This kind of anomaly when single point data is anomalous with respect to the rest of data is called point anomaly [27]. Point anomaly detection is the most difficult to detect. Other anomalies like contextual anomaly [27] is looser version of point anomaly, where data instance is anomalous in a specific context. There are three major categories of approaches to detect out-of-training distribution data: *statistical detection techniques*, *deviation-based techniques*, *proximity-based techniques* [7]. Statistical detection techniques attempt to fit the training data in a parametric/non-parametric probability distribution. Parametric models like the Gaussian Mixture Model (GMM) or non-parametric models like kernel-density estimation can be used to define a probability distribution. The goal of learning is to find the training data distribution model and its parameters that can explain the training data. Using this learned densities distribution, inferences about new data points can be made. Most of these are based on the probability of it being generated from the trained densities. A data point is defined as an OOD if the probability of it being generated

is very low. The major limitation of this approach is the difficulty to find a good probability fit, even when the dimension of the problem increase beyond a hundred and consequently detecting anomalous data in high dimensions using learned densities is hard. [103] claim that state-of-the-art learned densities are not suitable for anomaly detection since they assign a higher probability to some out-of-distribution data than the data on which the models were trained and proposed performing an anomaly detection test based on the typicality of data under the learned density instead of the likelihood of data under the learned density.

Proximity-based techniques assume that anomalous data are isolated from the majority of the data and use various approaches to measure density or cluster representation and the relationship of data-point with the cluster. There are three ways to model proximity-based anomaly detection, clustering-based, density-based, and distance based. In the clustering-based approach, a clustering algorithm is applied to data, and the relationship of each data point with respect to a cluster is measured as an anomaly score. An example anomaly score can be distance from the centroid of the cluster. Density-based clustering approach tries to define anomaly as a data point that lies in a sparse region and the number of data points in a localized region can be measured as an anomaly score. Distance-based anomaly detection uses measurements that are related to the neighboring data points of a data point, and the distance value can be used as an anomaly score. In proximity-based approaches, the selection of algorithm and data structure should be such that relationships of the data points in metric space must be preserved in this data structure. The early work in this context is done by using randomized cut forests/isolation forests [88]. The isolation forest approach has several drawbacks, such as not being compatible with streaming data and missing crucial OODs in the presence of irrelevant dimensions. etc [55]. To address these challenges, [55] proposed Robust Random Cut Forest (RRCF) as a data structure, which is very promising in terms of a very small false alarm rate (high accuracy), and can work on streaming data, but it is not scaleable on large data [56].

Deviation-based approaches are based on one or other flavors of the encoder-decoder settings. The encoder is trained to embed the data in latent space and the decoder is trained to reconstruct the data from latent space. During the training of this model, the goal is to reconstruct the input data as the output of the decoder. Once trained, inference about a data point can be made by passing it through the encoder and decoder setting and estimating the reconstruction error/probability gap. The underlying assumption of deviation-based methods is data from out-of-distribution will have high reconstruction error as the encoder-decoder parameters are not trained for it. The encoder tries to find the lower dimensional embedding of the original data where anomalies and normal data are expected to be separated. Once the encoder-decoder is trained on training data there are three major approaches that are used to find the anomaly :

1. Reconstruction error in the encoder-decoder setting.

2. Reconstruction probability difference (KL divergence) in VAE

3. Latent space probability difference (KL divergence) in VAE

In comparison to other techniques for measuring deviation, the deep learning model has the most used method. The deep learning-based OOD detection algorithms can be classified into two of the following methods based on the kind of training data.

1. *Supervised methods*, which involves training a detector on a labeled dataset that contains both OOD and non-OOD samples.

2. *Semi-supervised methods*, which involves training a Deep learning-based detector on a dataset considered to only have non-OOD samples.

Supervised methods are really straight forward and the training problem is a binary classification problem. But the unavailability of labeled non-OOD data has resulted in the wide usage of semi-supervised and unsupervised detection techniques. Semi-supervised OOD detection is a complex process and involves automatic learning of intrinsic latent behavior hidden in data and does not require explicit labels. A deep learning-based encoder-decoder setting is majorly deployed in the deviation-based semi-supervised OOD detection process. A well-trained autoencoder would result in a low reconstruction error for samples similar to the training dataset and a high reconstruction error for samples that are not similar to the training dataset. For example, autoencoder [123] training is semi-supervised. A well-trained autoencoder would result in a low reconstruction error for samples similar to the training dataset and a high reconstruction error for samples that are not similar to the training dataset. Discriminative deep learning models like multilayer perceptron [32] and Deep-SVDD [122] have shown robust detection capability on high-dimensional data such as images. Among them, Deep-SVDD is the most prominent model. It trains a neural network to learn a minimum volume hypersphere that encloses the network's representation of the training data samples [122]. The results from Cai *et al.* [25] have shown that a well-trained Deep-SVDD can detect OOD images with high precision. However, improper selection of network hyperparameters results in a hyper-sphere collapse problem of the Deep-SVDD network [29].

CHAPTER 3

AI-ML approaches in Design optimization

3.1 Problem formulation

The area of design optimization in complex engineering domains presents significant challenges, and the utilization of artificial intelligence and machine learning models and algorithms holds promise for expediting engineering design problems. This chapter examines the various possibilities for implementing artificial intelligence and machine learning models and algorithms in practical engineering design optimization issues. It encompasses multiple scenarios, application cases, and experimental approaches. The challenges can be classified below according to the quantity of labeled data that can be generated as a consequence of complex computational complexity, the size of the design space, and the optimization objective. I am particularly interested in examining three distinct categories of problems and analyzing the corresponding methodologies employed in solving them.

I. Design optimization problem that involves computationally costly evaluation (labeling cost is very high, like in the CFD and FEA domains) and has a parametric design search space that is continuous (it may have infeasible regions in this continuous space), and the goal is to optimize for only one set of design requirements.

II. Design optimization problem that involves computationally cheap evaluation (due to the availability of a coarser approximate model that is very fast to evaluate) and has a parametric design search space that is high-dimensional and continuous. The challenge of design optimization in this case arises from a high-dimensional design space, called the curse of dimensionality. And the goal is to optimize for only one set of design requirements.

III. Problem of study and explore the capability of available AI-ML models to learn complex physical behavior encountered in engineering design and use these trained surrogates in the optimization process.

In the case of the first problem, the challenge is in the creation of a surrogate model that is not feasible due to computational complexity on a given limited time in hand, and the objective of the research is to identify a framework for sample-efficient optimization. In the second problem, too, the design of a cheap surrogate model is unwarranted (since we already have a cheap evaluation model), so the strategy is to formulate the design optimization problem as an inverse problem and use AI-ML to solve it. In the third scenario, the method investigates the capability of available AI-ML models to learn complex physical behavior and uses

the trained surrogate in the optimization process. By leveraging the generalization capability of AI-ML models much faster than conventional methods, it is anticipated that a large number of design optimization probelm can be solved. The disadvantage of this method is that the data generation and training processes are time-consuming and computationally costly.

Apart from the above-mentioned problems, I also present my research efforts in search of a universal hull shape that is optimal/near-optimal across a range of environmental and operating conditions. Finally, I present a tool called *Anvil*, which is a general-purpose SciML (Scientific Machine Learning) tool for CFD-based design evaluators integrated with AI-based optimization algorithms and can be used by researchers and engineers in the field of shape optimization involving solid-fluid coupling.

3.2 Approach to problem I: Search for a sample efficient optimization framework

The discipline of design optimization, encompassing both shape (geometry) and topology optimization, is widely studied in the field of computational physics. While the potential benefits of utilizing high-fidelity simulations in computer-aided design (CAD) are attractive, a significant obstacle that arises is the curse of dimensionality that arises when simulating intricate systems governed by partial differential equations (PDEs) with multiple physical phenomena (such as the $O(n^4)$ complexity of three-dimensional Eulerian fluid simulations in terms of space and time). One potential option is to develop cost-effective and approximate surrogate models for this intricate behavior. However, this approach necessitates a substantial investment of time and effort in activities such as data acquisition, training, validation, and so forth. In this particular scenario, it is essential to use a more prudent approach towards design optimization by formulating a highly efficient framework that minimizes the number of simulations required to identify an optimal design. In pursuit of this objective, this section aims to enhance sample efficiency in the context of a practical design optimization problem that necessitates the evaluation of CFD. In the subsequent part, the approach employed for the evaluation of CFD-based design is initially elaborated upon.

3.2.1 Methodology for CFD-based design evaluation

For design evaluation, the first step is to integrate different engineering design domains in a toolchain and design a CAD seed whose parameter defines the surface geometry.

3.2.1.1 Integrated tool chain

In Computer Aided Engineering (CAE), the first step in design automation and exploration is the integration and automation of simulation tools for end-to-end highly automated execution. The UUV hull design process involves two engineering design simulation tools and one optimization/sampling framework. The

engineering domain simulation tool consists of freeCAD [119] - a parametric CAD modeling tool built on concepts and algorithms of the Cascade Kernel, and OpenFoam [70] - a C++-based tool for fluid physics simulations that transforms continuum mechanics partial differential equations using finite volume discretization and solves their evolution. OpenFoam also consists of other auxiliary tools for meshing, parallelism, and different solvers integrated for seamless fluid simulation. For optimization and sampling purposes, I chose a python-based implementation of the optimization and sampling algorithms due to their easy deployment, extendibility, and availability as open-source software packages. All three tools are integrated in unison to create an integrated toolchain (refer to figure 3.1). The initial stage of design automation and exploration in Computer Aided Engineering (CAE) involves the integration and automation of simulation tools to provide a comprehensive and fully automated execution process. The process of designing the hull for an Unmanned Underwater Vehicle (UUV) entails the utilization of two engineering design simulation tools, along with an optimization/sampling framework. The engineering domain simulation tool comprises two main components: freeCAD and OpenFoam. freeCAD is a parametric CAD modeling tool that is constructed based on the Cascade Kernel. On the other hand, OpenFoam [70] is a C++-based software tool used for conducting fluid physics simulations. It employs finite volume discretization to transform partial differential equations derived from continuum mechanics and solves them to determine their evolution. In addition to its core functionalities, OpenFoam incorporates supplementary tools for mesh generation, parallel computing, and several solvers, which are seamlessly integrated to facilitate fluid simulation. In order to facilitate optimization and sample procedures, I have opted for a python-based implementation of the optimization and sampling algorithms. This decision is based on the advantages of quick deployment, extendibility, and the availability of these algorithms as open-source software packages. The three tools are combined together to form a cohesive toolchain (see figure 3.1). The utilization of Docker as a packaging and software delivery method was employed to establish a standardized software unit and ensure the maintenance of software version consistency, given that FreeCAD and OpenFOAM are self-contained entities encompassing numerous components. In the realm of parametric design optimization and data production, a parametric 3D CAD seed design is formulated. This seed design encompasses a spectrum of characteristics that govern the extent of flexibility and adaptability exhibited by a three-dimensional model. The CAD tool, namely freeCAD, has the capability to autonomously build a 3D geometry based on a specified parameter value. Subsequently, it employs a tessellation method to transform the generated geometry into the STL file format. This tessellation process involves the creation of a collection of 3D triangles that accurately represent the CAD model. The STL file that is produced is thereafter sent via a file exchange mechanism to another Docker environment in order to execute the fluid simulation on the provided STL design. Prior to executing OpenFOAM, it is imperative to do preliminary procedures such as mesh generation, flow initialization, and boundary condition specification,

among others. A comprehensive elucidation of these methods may be found in a subsequent section. All of these operations can be managed within a Python programming environment. After the simulation reaches convergence, the CFD docker provides the value of the drag force corresponding to the specified design parameter. To perform in-loop optimization for CFD, I employed the GPyOpt (Gaussian Processes for Optimization) [54], pyMOO (Multi-Objective Optimization) [20], and pyDOE (Design of Experiments) software packages. These programs were utilized to execute optimization algorithms and sampling methods. In the data collecting phase of surrogate modeling, a uniformly random selection of design points was employed within the design space to execute the simulation.

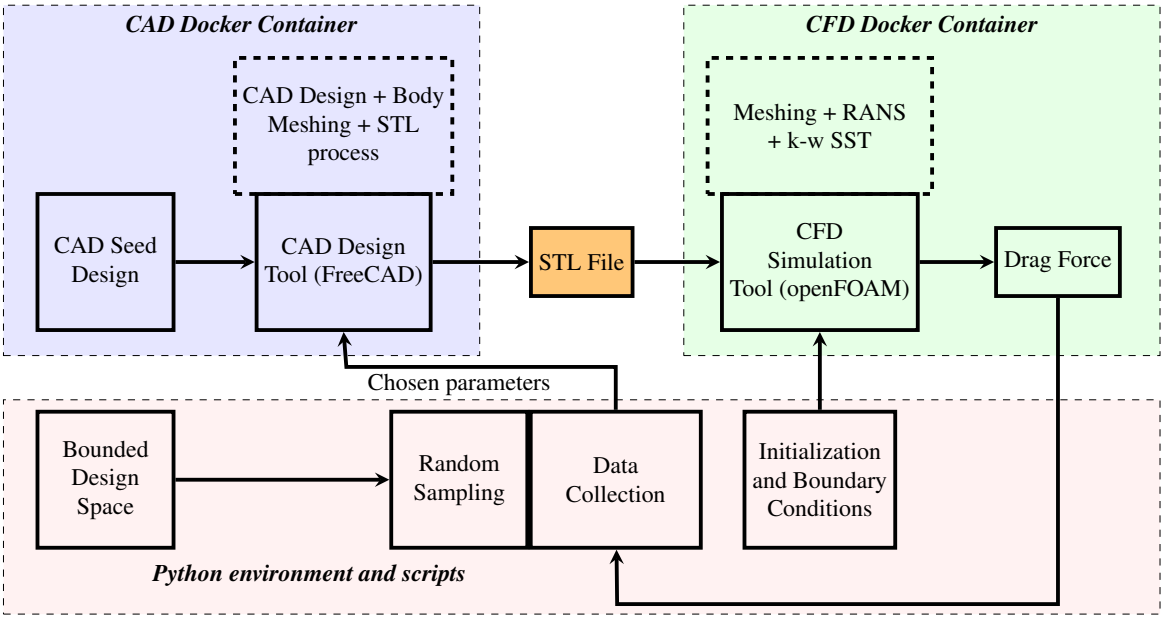


Figure 3.1: An integrated tool chain incorporating freeCAD (parametric CAD modeling tool), OpenFOAM (Computational Fluid Dynamics simulation tool) with Python environment (control the process flow and run optimizer and sampler) for CAD design generation, its drag evaluation and optimization in our workflow.

3.2.1.2 Shape generation/ Surface geometry

The drag of a vehicle is greatly affected by the hull geometry. The hull profile referred to as the 'Myring' [102] is extensively utilized among the several contemporary hull designs. The hull profile is a rotational symmetry around an axis, resembling a cylindrical structure with a specified ratio of length to diameter. The aforementioned design has numerous advantages, such as improved utilization of internal volume, less drag force, optimized flow characteristics, and a suitable geometry for dynamic and hydrostatic pressure.

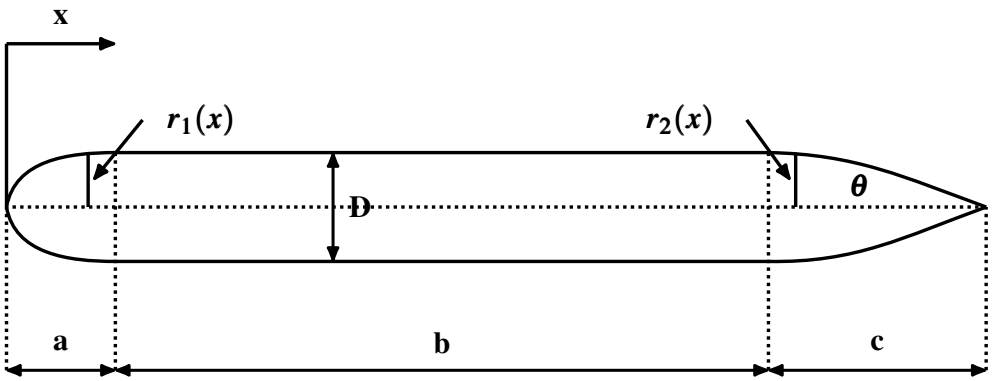


Figure 3.2: Myring hull profile

The nose shape and the tail shape for the Myring hull profile (refer to figure 3.2) are given by:

$$r_1(x) = \frac{1}{2}D\left[1 - \left(\frac{x-a}{a}\right)^2\right]^{\frac{1}{n}} \tag{3.1}$$

$$r_2(x) = \frac{1}{2}D - \left[\frac{3D}{2c^2} - \frac{tan\theta}{c}\right](x-a-b)^2 + \left[\frac{D}{c^3} - \frac{tan\theta}{c^2}\right](x-a-b)^3 \tag{3.2}$$

The variables r_1 and r_2 are used to describe the geometric properties of the nose/bow and the tail/stern of the hull, respectively. The variable x is defined as the distance measured from the tip of the nose. Let D be the diameter of the hull, and let a, b, and c denote the lengths of the nose, body, and tail accordingly (see to figure 3.2). The variables n and θ are employed to define the geometric properties of the nose and tail portions in different streamlined profiles. By manipulating the variables n and θ, one may effectively control the body profile of the hull. The influence of different values of n and θ on the structure of the hull is seen in figure 3.3. The analysis fails to account for the potential influence of nose and tail offset. The utilization of the six parameters, namely a, b, c, d, n, and θ, has the potential to yield a multitude of geometric configurations.

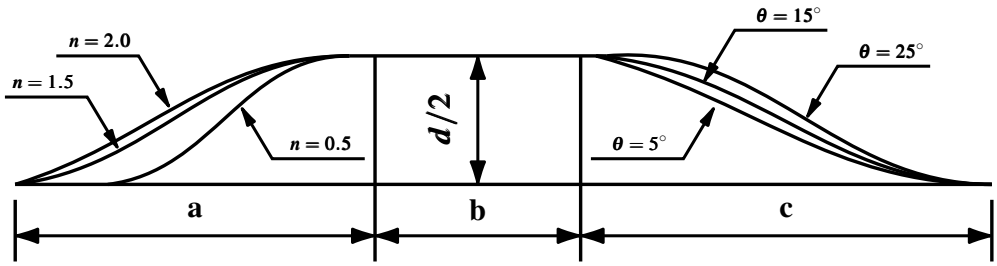


Figure 3.3: Myring hull profile change with n and θ

3.2.1.3 CAD design

With the goal of generating a surrogate for a large class of problems, it was imperative to design a parametric CAD model that should have flexibility and adaptability to design a 3D CAD model from a given parameter without manual intervention. The parametric mechanical CAD design should maintain the experimenter's assumption to generate a valid CAD design for the given parameter. Invalid designs can be generated due to the intrinsic nature of schema for parameterization or CAD tool issues. Ensuring this using a CAD design is hard [64]. I tackled this problem in two steps: first, by using stringent design methodology with completely constrained designs and second step storing a cross-section of the design in point-based schema during simulation to validate all schema by visual inspection.

3.2.1.4 Numerical equations and turbulence model for simulation

The utilization of CFD enables the numerical solution of the Navier-Stokes equation, enabling the study of the effects of fluid flow on a specific geometric design. There are two alternative methodologies that offer reasonable approximations to direct numerical simulation (DNS) of the Navier-Stokes equations, mostly because to the substantial processing expense involved. The two methodologies under discussion are Reynolds-averaged Navier-Stokes (RANS) simulation with closure-based turbulence models and large eddy simulation (LES) [124]. The Reynolds Averaged Navier-Stokes equation (RANS) was employed as the computational technique for solving the equations of mass and moment conservation over a temporal average. The Navier-Stokes equation that describe the dynamics of a Newtonian fluid, for incompressible and isothermal behaviour can be described as:

$$\rho \frac{d\vec{v}}{dt} = -\nabla p + \mu \nabla^2 v + \rho \vec{g} \tag{3.3}$$

$$\nabla \cdot \vec{v} = 0 \tag{3.4}$$

The symbol v represents the velocity field, μ denotes the viscosity of the fluid, p corresponds to the pressure, and g represents the gravitational acceleration vector. Given the assumption of the fluid being incompressible, the density ρ remains constant.

The turbulence physics utilized in this investigation involved the mathematical formulation of the k-ω shear stress transport (k-ω SST) model, which was originally presented by Menter [96]. According to it, this particular turbulence model has demonstrated greater reliability in forecasting flow separation when compared to alternative turbulence models. The closure model utilized in this investigation is the Shear Stress Transport (SST) methodology, which is rooted in the Boussinesq hypothesis on the Reynolds stress

tensor. Furthermore, the model integrates a modified formulation for the scalar eddy viscosity component. The underlying assumption posits the existence of a correlation between the Reynolds stress tensor and the mean rate-of-strain tensor, alongside the turbulent kinetic energy symbolized as k. In k-ω SST, two different turbulence models are utilized in different regions of the flow. The k-ω model [163] is used to capture the flow characteristics in the vicinity of the wall, while the k-ε model [78] is adopted to analyze the flow behavior in the region away from the boundary layer, commonly referred to as the free stream region. Furthermore, this model takes into account the inclusion of primary shear stress transmission inside boundary layers that are subjected to adverse pressure gradients. Furthermore, this methodology addresses the challenge posed by the k-ω model, which is known for its susceptibility to changes in the inlet free-stream turbulence parameter values. k-ω SST closure model has demonstrated considerable efficacy and has been empirically validated in several turbulent flow scenarios, owing to its multiple benefits. This turbulence model is commonly utilized in real world CFD simulations for the purpose of system design and has great success and validated against many real-world turbulence flow problems. The mathematical representation of the k-ω SST model utilized in OpenFOAM is described by Menter [96] and is stated as follows:

$$\frac{\partial(\rho k)}{\partial t} + \frac{\partial(\rho U_i k)}{\partial x_i} = P_k - \beta^* \rho k \omega + \frac{\partial}{\partial x_i}\left[(\mu + \sigma_k \mu_t)\frac{\partial k}{\partial x_i}\right] \tag{3.5}$$

$$\frac{\partial(\rho \omega)}{\partial t} + \frac{\partial(\rho U_i \omega)}{\partial x_i} = \alpha \rho S^2 - \beta \rho \omega^2 + \frac{\partial}{\partial x_i}\left[(\mu + \sigma_\omega \mu_t)\frac{\partial \omega}{\partial x_i}\right] + 2(1 - F_1)\rho \sigma_{w2}\frac{1}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i} \tag{3.6}$$

The blending function F_1 is defined by:

$$F_1 = tanh\left\{\left\{min\left[max\left(\frac{\sqrt{k}}{\beta^* \omega y}, \frac{500\nu}{y^2 \omega}\right), \frac{4\rho\sigma_{\omega 2} k}{CD_{k\omega} y^2}\right]\right\}^4\right\} \tag{3.7}$$

with $CD_{k\omega} = max\left(2\rho\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_i}\frac{\partial \omega}{\partial x_i}, 10^{-10}\right)$ and y is the distance to the nearest wall.

F_1 is equal to zero away from the surface (k-ε model) and switches over to one inside the boundary layer (k-ω model). The turbulent eddy viscosity is defined as follows:

$$\nu_t = \frac{a_1 k}{max(a_1 \omega, S F_2)} \tag{3.8}$$

where S is the invariant measure of the strain rate and F_2 is a second blending function defined by :

$$S = \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) \tag{3.9}$$

$$F_2 = tanh\left[\left[max\left(\frac{2\sqrt{k}}{\beta^*\omega y}, \frac{500v}{y^2\omega}\right)\right]^2\right] \tag{3.10}$$

In the k-ω SST model, a limiter is employed to mitigate the accumulation of turbulence in regions of stagnation and defined below:

$$P_k = \mu_t \frac{\partial U_i}{\partial x_j}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) \rightarrow \tilde{P_k} = min(P_k, 10.\beta^*\rho k\omega) \tag{3.11}$$

All the constants are computed by a blend funtion from the corresponding constants of k-ε and k-ω model via $\alpha = \alpha_1 F + \alpha_2(1-F)$ etc. The constants for this model are $\beta^* = 0.09$, $\alpha_1 = 0.31$, $\beta_1 = 0.075$, $\sigma_{k1} = 0.85$, $\sigma_{\omega 1} = 0.5$, $\alpha_2 = 0.44$, $\beta_2 = 0.0828$, $\sigma_{k2} = 1$, $\sigma_{w2} = 0.856$.

Initialization: The turbulent energy k is defined as:

$$k = \frac{3}{2}(UI)^2 \tag{3.12}$$

Where U is the initial flow velocity and I is the turbulence intensity. The turbulence intensity gives the level of turbulence and can be defined as follows:

$$I = \frac{u'}{U} \tag{3.13}$$

Where u' is the root–mean–square of the turbulent velocity fluctuations given as:

$$u' = \sqrt{\frac{1}{3}\left({u'_x}^2 + {u'_y}^2 + {u'_z}^2\right)} \tag{3.14}$$

The mean velocity U can be calculated as follows:

$$U = \sqrt{{U_x}^2 + {U_y}^2 + {U_z}^2} \tag{3.15}$$

The specific turbulent dissipation rate can be calculated using the following formula:

$$\omega = \frac{k^{\frac{1}{2}}}{l * {C_\mu}^{\frac{1}{4}}} \tag{3.16}$$

Where C_μ is the turbulence model constant which usually takes the value 0.09, k is the turbulent energy, and l is the turbulent length scale.

3.2.1.5 Mesh generation

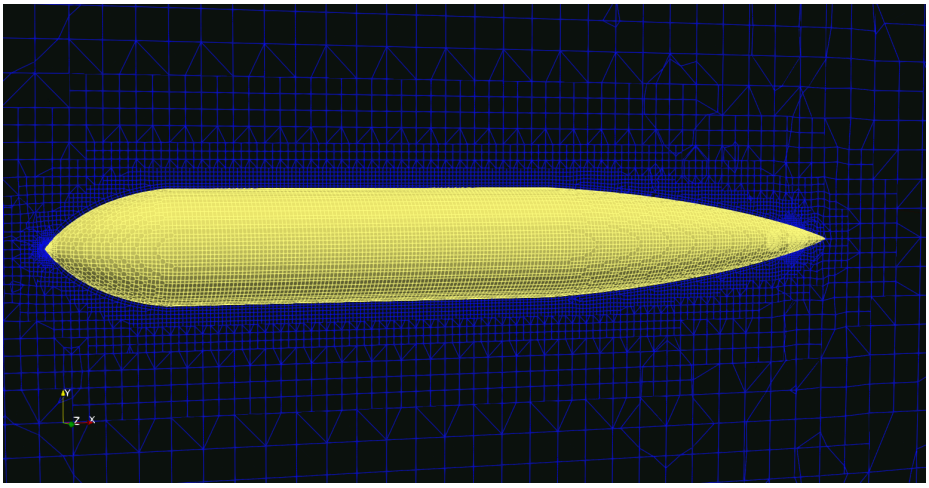
The meshing involved two steps: it starts with the creation of castellated 3D parametric volumetric mesh to fill the whole 3D volume using the OpenFOAM blockMesh utility. Our volume meshing process uses only hexahedral elements whose number is defined by the parameter in the blockMeshDict file. The volumetric mesh is further split and refined in the vicinity of the body surfaces. Once the mesh in the locality of the body is refined and split, cells inside the body shape are removed keeping the constraint that at least one volumetric region must be bounded inside the domain. This results in separating the body shape from the volume mesh and creating a volume mesh around the surface. The next step is refining these mesh near the boundary to generate three-dimensional unstructured or hybrid meshes consisting of hexahedra (hex) and split-hexahedra (split-hex) elements to remove cells that violate the mesh quality parameters in proximity volume of external walls of the hull using snappyHexMesh tool. Figure 3.4a and 3.4b provides a view of the mesh in the vicinity of the hull for one of the simulations.

3.2.1.6 Initial and boundary conditions

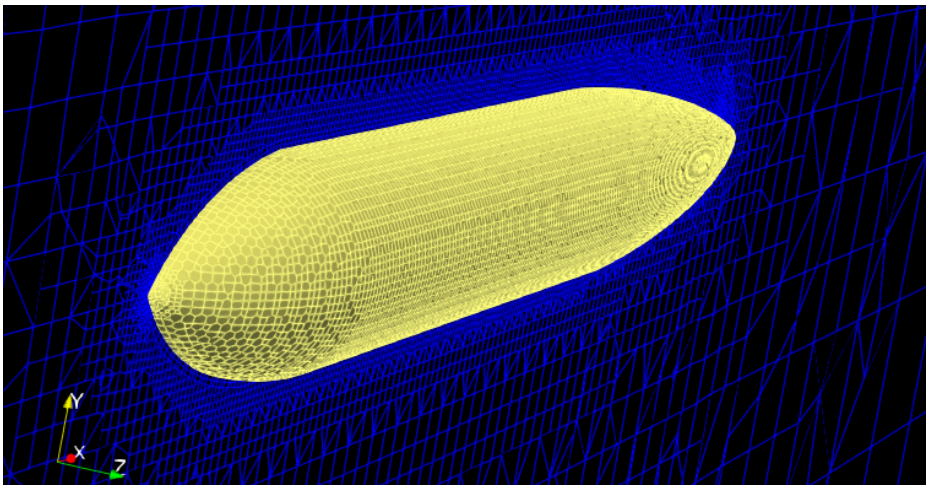
All simulations utilize the conventional Dirichlet boundary conditions that are appropriate for modeling incompressible flow around a solid object. At the inflow, the velocity inlet boundary condition was enforced with a magnitude of 5 knots, while at the outflow, a zero pressure outlet boundary condition was applied. The hull surface was imposed with a no-slip boundary condition, whilst the side wall, symmetry wall, and far field surfaces were defined as symmetry surfaces. The simulations were performed with a flow velocity of 2 m/s, assuming an inflow turbulence intensity level of 4%. The selection of medium turbulence was based on the flow's low-speed characteristics. The starting values for the variables k and ω were obtained using the same methods described in equations 3.12 and 3.16. The calculated values for the parameters k and ω are around $0.01 m^2/s^2$ and 57 per second, respectively. Assuming the Newtonian model, the kinematic viscosity is assigned a constant value of $1.7\ mm^2/sec$.

3.2.1.7 Solver setting

The fvSchemes, controlDict, and fvSolution files are set as part of the solution setting. The gradSchemes, laplacianSchemes, ddtSchemes, divSchemes (divergence scheme), interpolationSchemes, and snGradSchemes (surface normal gradient scheme) are controlled by the fvSchemes file. The gradSchemes are set to Gauss linear, the laplacianSchemes are set to Gauss linear uncorrected, and the div(phi,k) and div(phi, omega) terms are set to limited Gauss upwind. The controlDict sets the start time, end time, time step, etc. for the simulation. The maximum simulation time is set for up to 500 seconds, or until convergence, whichever comes first. The solver is set to SimpleFoam, which is a steady-state solver for non-compressible turbulent flow that uses



(a)



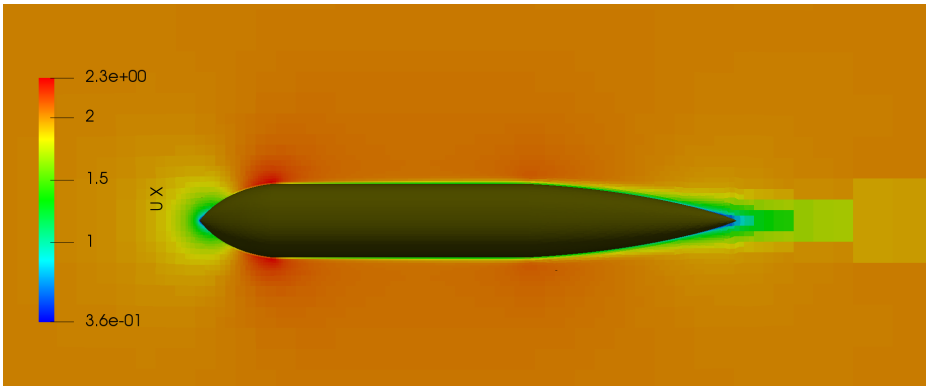
(b)

Figure 3.4: Meshing in the OpenFOAM after blockMesh based mesh generation and snappyHex based refinement (a) cross-sectional view, (b) oblique view.

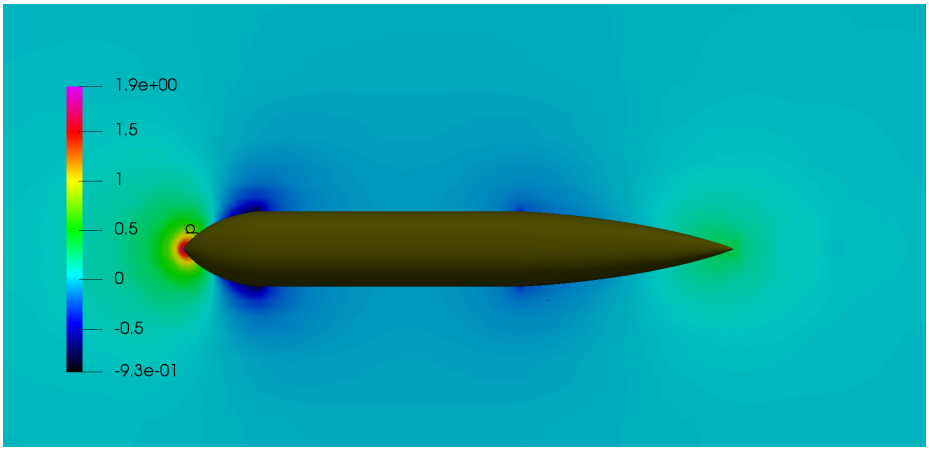
the SIMPLE (Semi-Implicit Method for Pressure Linked Equations) algorithm. In the fvSolution file, the pressure solver was GAMG (Geometric agglomerated algebraic multigrid preconditioner) with symGauss-Seidel smoother and a tolerance of $1.0 \times e^{-7}$ and a relative tolerance of 0.01. The GaussSeidel smoother was used with the smoothsolver for U, k, and ω. In each case, the range is set to $1.0e^{-07}$, with relTol = 0.1.

3.2.1.8 Example simulation results

The simulation output of one of the simulation experiments utilizing a UUV hull design in openFoam, with the aforementioned configuration and approach, is depicted in figure 3.5. The contour plot of the mean axial velocity and pressure field in the flow zone was generated using the Paraview post-processing software tool.



(a)



(b)

Figure 3.5: An example steady state flow properties (a) mean axial velocity, (b) pressure field.

3.2.2 Unconstrained sample efficient design optimization problem with UUV hull design use case

The objective of this section is to examine the performance of various optimization and sampling techniques when used in loop with a CFD simulator to find the optimal design parameters of an Unmanned Underwater Vehicle (UUV) hull. I undertake a comparative analysis of various numerical methods, which have been chosen based on our understanding of prevailing methodologies. In order to evaluate the efficacy of these various methods, two key metrics are of interest: the predicted sample efficiency in identifying the best design, and the variance in the search process, which reflects the convergence behavior. The following definitions are provided: Consider the set S comprising of labeled sample designs involved in the process of design optimization. Let f denote the mapping that associates each candidate hull design S_i with the steady-state drag force F experienced by the underwater unmanned vehicle (UUV) with design S_i as it moves through water. A sample-efficient design optimization process aims to decrease the size of the set S, while finds an optimal design. To address the inherent variability in optimization processes either due to intialization condition or randomness in the optimization process, I estimate the expected sample efficiency ($E(|S|)$).

Specifically, I aim to identify the method that, on a given budget, is expected to deliver the most optimal design. The measurement of convergence behavior in the search process pertains to the assessment of variance in the optimization procedure when initiated with various initializations conditons. Variance is quantified through the absolute difference between the maximum and minimum values of the drag coefficients, denoted as $|F^{\max} - F^{\min}|$, which represents the range observed over each iteration. In the context of design optimization, certain commercial CFD simulation software packages are equipped with a range of design exploration techniques, including response surface methods, sensitivity analysis methods, and classic optimization algorithms such as genetic algorithms. The recent advancements in artificial intelligence (AI)-based optimization techniques exhibit potential for surpassing the performance of current optimization methods. I am interested in doing a comparative analysis of a broad range of algorithms in order to ascertain the optimal combination of sample efficiency and convergence behavior for UUV hull design optimization. In order to achieve this objective, I have chosen six distinct design of experiment (DOE) and optimization algorithms, namely: Monte Carlo with sampling from a uniform random distribution, maximin Latin Hypercube-based optimization, Vanilla Genetic Algorithm, Nelder-Mead, Bayesian Optimization—Lower Confidence Bound (BO-LCB), and Bayesian Optimization—Expected Improvement (BO-EI). The specifics of each algorithm can be found in Chapter 2.

To formally define the optimization problem, I denote the 3D hull shape by Ω, which is parameterized on a multivariate parameter X, i.e. $\Omega = \Omega(X)$. Let $f : \Omega \mapsto F$ be a function that maps the 3D hull shape design Ω to the steady-state drag force F. Our optimization goal is then to minimize f on the domain of interest DS, $X \subset DS$:

$$\Omega^* = \underset{X \in DS}{\operatorname{argmin}} f(X) \tag{3.17}$$

The Remus100 class design, as discussed in the work of Winey et al. [164], serves as a relevant real-world case study that I utilize as the basis for our design space. The design of the Remus100 is characterized by a Myring hull-based shape, with a diameter of 0.191m and a total length of 1.33m. For the sake of optimization, I consider the diameter and total length of the Myring hull as constants, while allowing the remaining shape parameters (a, b, c, n, θ) to vary and be included in the optimization process. The search domain for design parameters (a, b, c, n, θ) is given in table 3.1.

The allocated budget for optimization has been established to be 50 evaluations as each evaluation is costly and takes approximately 10 minutes per evalution. The results of the design and its corresponding drag force, as obtained from the various methods under evaluation, are shown in figure 3.6. The top diagram illustrates that BO-LCB achieves the most optimal design with the fewest amount of samples. BO-EI exhibits similar expected sample efficiency to BO-LCB, however, it is important to note that the variance in finding the

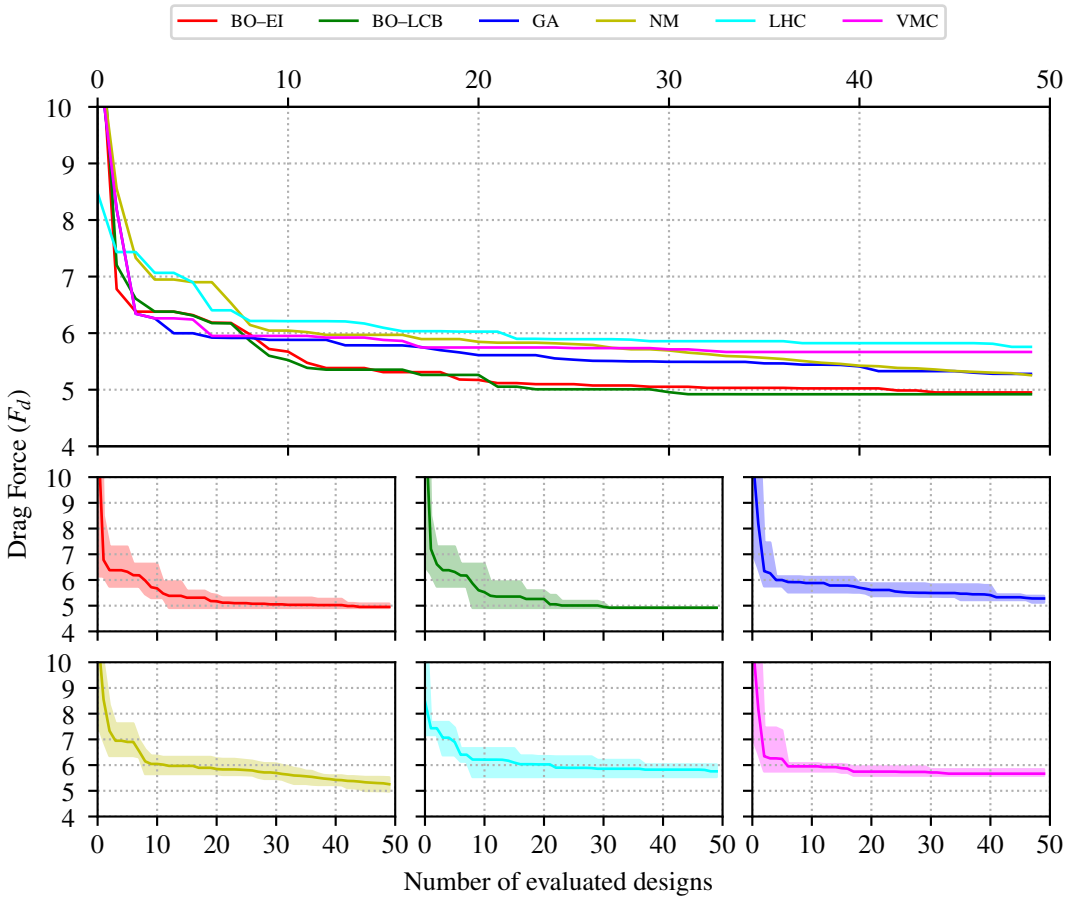


Figure 3.6: Optimal F vs. the number of evaluated designs. The top plot shows the expected optimal drag found by each optimization algorithm as the design space is explored. The expectation is calculated over five different optimization runs. The bottom six plots show the mean (the thick line) and the variance (shaded region) of drag forces found by using different optimization algorithms. Here BO-EI refers to Bayesian Optimization—Expected Improvement, BO-LCB refers to Bayesian Optimization—Lower Confidence Bound, GA refers to Genetic Algorithm, LHC refers to maximin Latin Hypercube, VMC refers to Vanilla Monte Carlo, and NM refers to Nelder-Mead method.

optimal design is significantly higher for BO-EI compared to BO-LCB. This observation can be corroborated by referring to the lower plots of figure 3.6, where it is evident that BO-LCB consistently converges to the same optimal design after approximately 30 iterations. Alternative approaches find optimal solutions that are significantly inferior within the specified budgetary constraints.

The design found by of one of five experiments are depicted in figure 3.7. This figure 3.7 displays the optimal hull form of an Unmanned Underwater Vehicle (UUV) that was found using above-mentioned approaches. The design achieved by BO-LCB exhibits a significantly streamlined body. The design obtained via BO-EI has the highest degree of proximity to the ideal design achieved from BO-LCB. The designs derived from the Genetic Algorithm (GA) and Nelder-Mead optimization methods exhibit degenerate nose and

Parameter	Symbol	Minimum	Maximum
Length of nose section	a	50 mm	573 mm
Length of tail section	c	50 mm	573 mm
Index of nose shape	n	1.0	50.0
Tail semi-angle	θ	0^o	50^o
$b = 1910 - (a + c)$			

Table 3.1: Design space: Range of design space parameters for optimization.

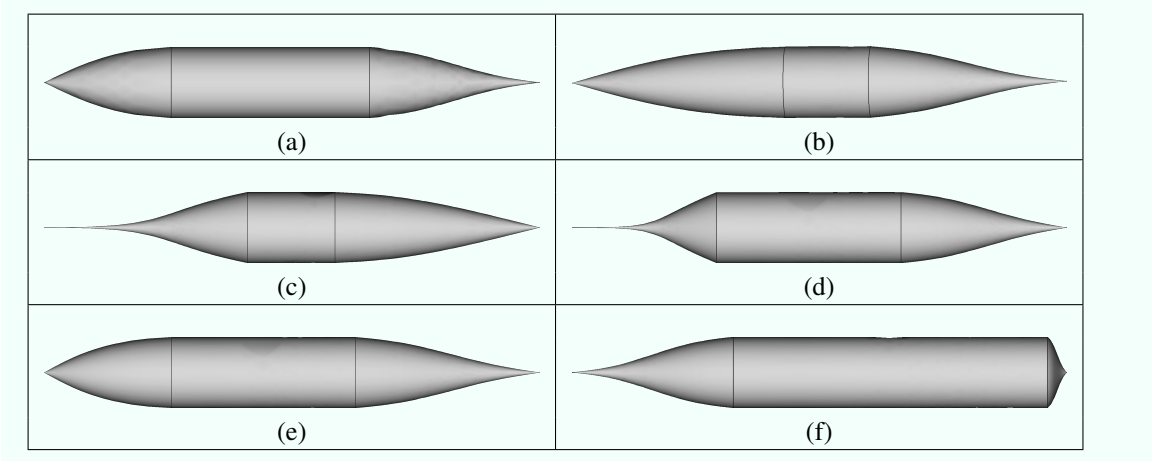


Figure 3.7: Results of one optimization run: optimal design discovered after exhausting the budget of simulation, using different optimization algorithms. (a) Bayesian Optimisation—Expected Improvement, (b) Bayesian Optimization—Lowest confidence bound, (c) Genetic Algorithm, (d) Nelder Mead, (e) maximin Latin Hypercube and (f) Vanilla Monte Carlo. The design produced by BO-LCB (design (b)) has lowest drag.

tail characteristics, which deviate significantly from the near-optimal design effectively identified by the BO-LCB approach. The design achieved by the maximin Latin Hypercube Sampling (LHC) approach exhibits good performance, akin to the design produced through Bayesian Optimization with Expected Improvement (BO-EI). However, it is important to note that both designs still not the optimal design.

3.2.3 Constrained Bayesian optimization with UUV hull design use case

In the previous experiment, I showed that Bayesian Optimization (BO) [30, 98, 173] has emerged as a promising paradigm for optimizing expensive-to-evaluate functions in a sample-efficient manner, and it has been successfully applied to other scientific domains. In some optimization problem, constraints may exist which make some of the designs and design space infeasible, and these must be handled appropriately during the optimization process. In this section, I formulate the UUV hull design problem as a constrained optimization problem. The UUV hull contains electronics, sensors, and other mechanical and electrical components. Packing them into the hull imposes a non-linear constraint on the optimization process. The hull design problem

can then be formulated as a constrained optimization problem defined as follows:

$$\Omega^* = \underset{x \in DS}{\operatorname{argmin}} f(X) \tag{3.18}$$

$$s.t.\ g(x) \leq 0 \tag{3.19}$$

Here, constraint function $g(x)$ ensures that all selected components can be packed inside the designed UUV hull. To solve this optimization problem, I utilize a constrained Bayesian Optimization framework as formulated by [48].

3.2.3.1 Constrained Bayesian optimization

Bayesian Optimization relies on a probabilistic model of the system of interest during optimization, and the fidelity of the model is the most decisive factor in the optimization process. I use the Gaussian process [116] defined below to model system behavior (f):

$$f \sim \mathscr{GP}(\mu(.), \kappa(.,.)) \tag{3.20}$$

Here $\mu(.)$ is the mean function and $\kappa(.,.)$ is the covariance kernel. For any given pair of input points $x, x' \in R^d$, these are defined as:

$$\mu(x) = \mathbb{E}[f(x)] \tag{3.21}$$

$$\kappa(x,x') = \mathbb{E}[(f(x) - \mu(x))(f(x') - \mu(x'))] \tag{3.22}$$

In the Bayesian sequential design optimization process, a crucial step at each iteration is to select the most promising candidate x^* for evaluation in the next iteration. In the BO setting, this is done by defining an acquisition function. The design of an acquisition function is a critical component in the performance efficiency of the BO. Let x^+ be the best-evaluated sample so far. To select a candidate point $\hat{x}$ in the next iteration, an improvement is defined according to Mockus et al. [98] as follows:

$$I(\hat{x}) = max\{0, f(\hat{x}) - f(x^+)\} \tag{3.23}$$

The expected improvement in such a case is defined as an EI acquisition function, which has a closed-form solution for estimating it from a new candidate point, as given by Mockus et al. [98] and Jones et al. [73]:

$$EI(\hat{x}) = \mathbf{E}\big[I(\hat{x})|\hat{x})|\big] \tag{3.24}$$

$$EI(x^+) = (f(x^*) - \mu^+)\Phi(\frac{f(x^*) - \mu^+)}{\sigma^+} + \sigma^+\phi(\frac{f(x^*) - \mu^+)}{\sigma^+}) \tag{3.25}$$

Here, ϕ is the standard normal cumulative distribution and Φ is the standard normal probability density function. Using this EI function, the most promising candidate sample is selected by choosing x^+ that has the maximum EI value.

$$x^* = \underset{x^+ \in DS}{\operatorname{argmax}}\, EI(x^+) \tag{3.26}$$

The newly selected sample x^* is evaluated and is included in the evaluated data set, called X. Accordingly, the posterior probability distribution is estimated by the conditioning rules for Gaussian random variables, as below:

$$\mu^* = \mu(x^*) + \kappa(x^*, X)\kappa(X, X)^{-1}(f(X) - \mu(X)) \tag{3.27}$$

$$(\sigma^*)^2 = \kappa(x^*, x^*) - \kappa(x^*, X)\kappa(X, X)^{-1}\kappa(X, x^*) \tag{3.28}$$

Constrained BO, which is an extension to standard BO meant to model infeasibility during the inequality-constrained optimization routine, is formulated and proposed by [48]. I use this formulation for our experimentation, and it models both function and constraint as Gaussian processes. Let g be the constraint function that is unknown *a priori*; the first step in this setting is to model f and g as Gaussian processes:

$$f \sim \mathscr{GP}(\mu_1(x), \kappa_1(x)) \tag{3.29}$$

$$g \sim \mathscr{GP}(\mu_2(x), \kappa_2(x)) \tag{3.30}$$

$$\mu_1(x) = \mathbf{E}[f(x)] \tag{3.31}$$

$$\kappa_1(x, x') = \mathbf{E}[\{f(x) - \mu_1(x)\}\{f(x') - \mu_1(x')\}] \tag{3.32}$$

$$\mu_2(x) = \mathbf{E}[g(x)] \tag{3.33}$$

$$\kappa_2(x, x') = \mathbf{E}[\{g(x) - \mu_2(x)\}\{g(x') - \mu_2(x')\}] \tag{3.34}$$

The improvement function in this case is modified as:

$$I_C(x^+) = \Delta(x^+) max\{0, f(x^*) - f(x^+)\} \tag{3.35}$$

$$\Delta(x^+) \in \{0,1\} \tag{3.36}$$

$\Delta(x^+)$ is a feasibility indicator function that is 1 if $g(x^+) \leq 0$, and 0 otherwise. It causes $\Delta(x^+)$ to be a Bernoulli random variable whose probability of getting a feasible design is:

$$PF(x^+) := Pr(g(x) \leq 0) = \int_{-\infty}^{0} P(g(x^+)|x^+, X) dg(x^+) \tag{3.37}$$

Due to the Gaussian behavior of $g(.)$, $\Delta(x^+)$ would be a univariate Gaussian random variable. The modified expected improvement to include the effect of infeasibility gives a joint acquisition function:

$$\begin{aligned} EI_C(x^+) &= \mathbf{E}[I_C(x^+)|x^+] && (3.38)\\ &= \mathbf{E}[\Delta(x^+) I(x^+)|x^+] && (3.39)\\ &= PF(x^+) EI(x^+) && (3.40) \end{aligned}$$

This joint acquisition function can be further optimized using standard optimization algorithms. Since our acquisition function has the property of being smooth and continuous, I used a two-step optimization to find x^*. The first step is Monte Carlo optimization and the second step is limited memory BFGS [44] (see Figure 3.8).

This integration of tools and capability to control the parameters and environmental conditions gives us the flexibility to run an optimization framework with design tools in the loop without human intervention (refer to figure 3.9).

3.2.3.2 Parametric CAD model and baseline packing geometry

For automatic design optimization, for the parametric CAD model is designed based on Myring hull [102] as the outer hull architecture (details are in section 3.2.1.2). The baseline design parameter values used for internal component packing and placement comes from another automated tool [93]. Component selection and packing are not within the scope of this paper; however, three-dimensional packing of components in an arbitrary shape is an NP-complete problem. Based on the capabilities of our external component selection and packing tool, I utilized a simple design with conical end caps (nose and tail) and a cylindrical body to determine the exact required hull dimensions. These conical-shaped parametric designs are optimized to

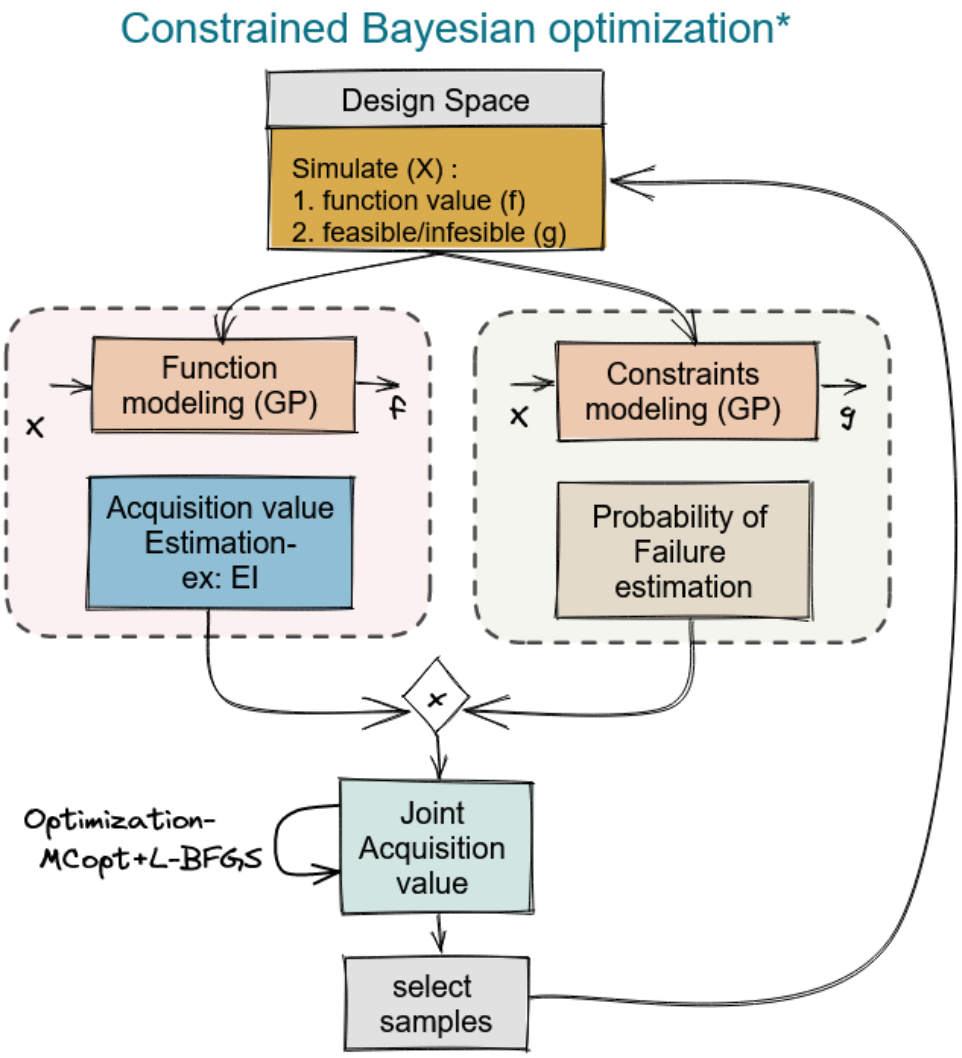


Figure 3.8: Constrained Bayesian optimization - o verview

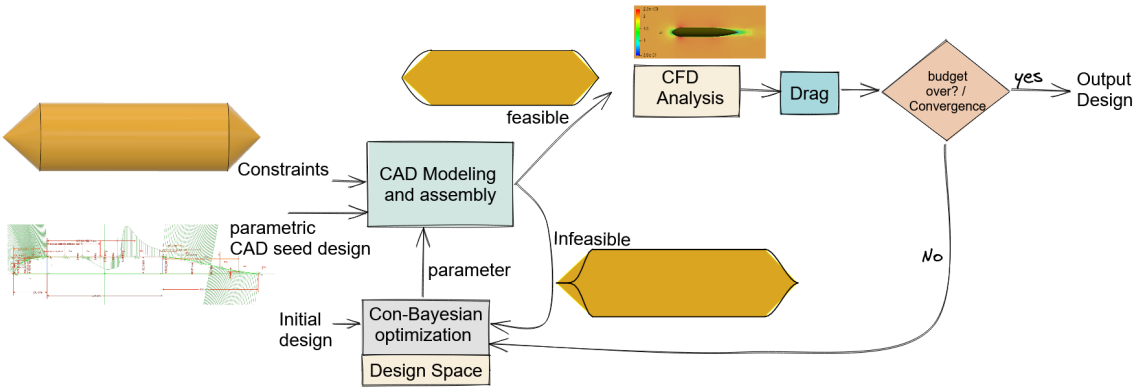


Figure 3.9: Optimization pipeline using integrated CAD and CFD tools with Bayesian Optimization

minimize internal hull volume while ensuring that packed components have no interferences. These baseline designs, however, are not optimal from the perspective of producing minimal-drag designs. Once components are packed and the parameters of a baseline design are found, this fixed geometry will act as a minimal constraint in the optimization of the hull design (refer to figures 3.10 and 3.11).

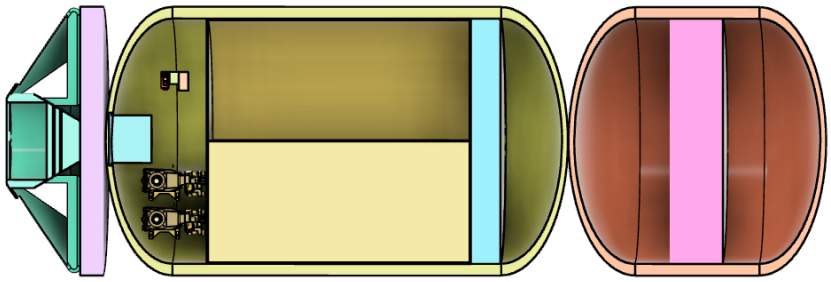


Figure 3.10: Selected components in the UUV in a specific packing configuration [93]

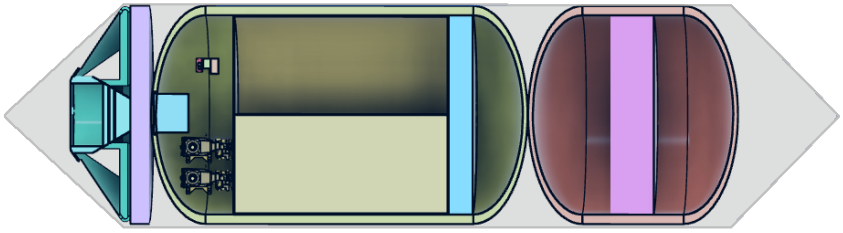


Figure 3.11: The components in the packing configuration inside a baseline packed geometry

3.2.3.3 Infeasible design heuristics

Since a parametric Myring hull can assume a wide range of shapes [102], the generated hull shape needs to be tested for interference with the baseline packed design. Any Myring hull parameters that cause interference with the baseline design are deemed to be infeasible. Since the computational cost of CAD assembly and running an interference test is much less than CFD simulation, the in-feasibility test on an optimized design is conducted during the CAD modeling and assembly stage (refer to figure 3.9). Running full CFD analysis on an infeasible design is a waste of computational time and resources, and it delays the optimization process. To address this situation, I implemented a heuristic that works as follows (for a minimization problem): for an infeasible design that is detected during CAD assembly, return the maximum drag value to the optimizer for all evaluated samples up to that point, instead of running a full CFD analysis. The opposite can be done for a maximization problem. However, for starting the experiment I need at least one drag value of in-feasible design. Accordingly, I run the first infeasible design and store its drag value.

Symbol	Minimum	Maximum
a	a_B	$a_B + 2500$ mm
c	c_B	$c_B + 2500$ mm
n	0.1	5.0
θ	0^o	50^o
$l = a + b + c$		

Table 3.2: Range of design parameters for optimization

3.2.3.4 Experimentation and results

In this section, I present two different experiments carried out using our optimization pipeline. In both cases, selected components are the same and consequently, the baseline packing geometry is identical. The operating conditions (i.e., the velocity of operation, initial and boundary conditions) and environmental conditions (e.g., turbulence intensity) are kept constant based on mission requirements. The baseline packing geometry is as shown in figure 3.12. The design space (*DS*) of the search process is selected as shown in Table 3.2.

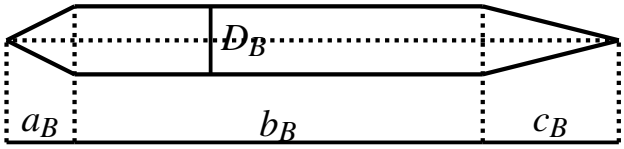


Figure 3.12: Baseline 3D hull design with $a_{baseline} = 555$ mm, $b_{baseline} = 2664$ mm, $c_{baseline} = 512$ mm, $D_{baseline} = 1026$ mm

3.2.3.4.1 Experiment 1

In this experiment, I only optimize the nose and tail shapes, defined by parameters n and θ. The range of the design space for optimization of parameters n and θ is given in Table 3.2. Due to it being a computationally costly process, I run 50 iterations of optimization using our optimization pipeline. The most optimal design (shown in figure 3.13) has a drag value of approx 69 Newtons.

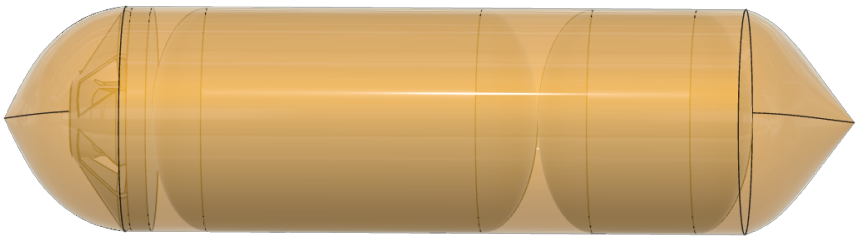


Figure 3.13: Optimal UUV hull shape with fixed nose and tail length. Optimal design parameters: $n = 1.0$; $\theta = 50.0$

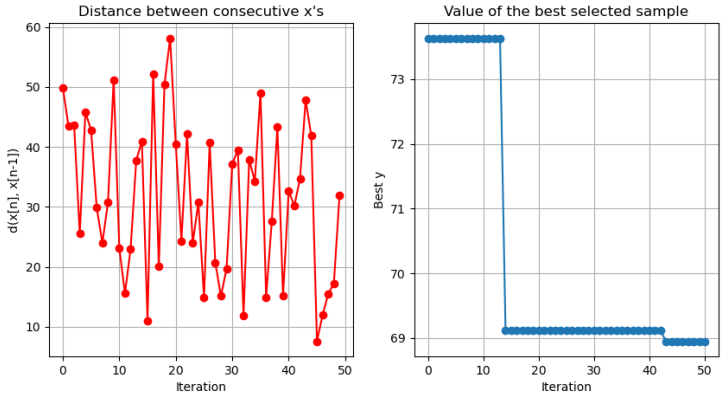


Figure 3.14: Optimization process vs number of evaluation/iteration: L2 distance between successive selected samples (left), drag value of best-selected sample in Newton (right)

3.2.3.4.2 Experiment 2

In this experiment, I also optimized the nose and tail length (parameters a and b) in addition to their shapes (parameters n and θ). The design space for optimization of all four variables is given in table 3.2. Again, I run 50 sequential optimization steps using our optimization pipeline. The most optimal design is shown in figure 3.15 and had a drag value of approximately 36 Newtons. This is a 50% reduction in drag due to the streamlined nose and tail shapes and would save a large amount of energy consumption during real-world operation of the vehicle.

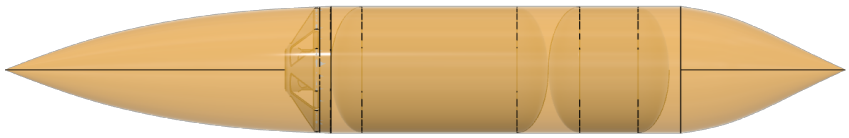


Figure 3.15: Optimal UUV hull shape with nose and tail length as free parameters. Optimal design parameters: $a = 2643.86$; $c = 1348.72$; $n = 1.144$; $\theta = 22.03$

3.2.3.5 Analysis of result

In both experiments, the allocated budget was 50 evaluations since the evaluation time was tens of minutes. But BO converges to optimal/near-optimal design in a few iterations. In exp1 (refer to right side plot in figure 3.14) even in 12 iterations a near-optimal design was found and no significant further improvement is observed. In exp2 (refer to right plot in figure 3.16) only in 10 iterations the optimal design was found and no further improvement was observed. This sample efficiency is due to the dynamic probabilistic modeling of the design space on labeled samples and state-of-the-art acquisition functions and accordingly costly optimization

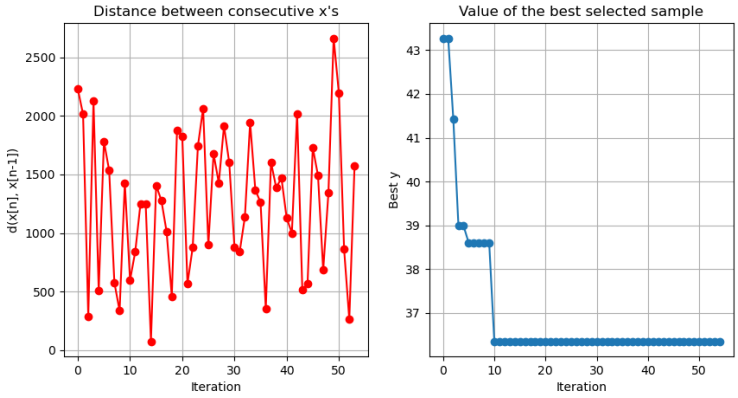


Figure 3.16: Optimization process vs number of evaluation/iteration: L2 distance between successive selected samples (left), drag value of the best-selected sample in Newton (right)

calculation. However, with the current multi-core implementation of BO, it takes milli-second to seconds for finding a new sample to evaluate and it is prudent to use BO in use cases where sample labeling and evaluation time can not be reduced beyond seconds.

3.3 Approach to problem II: Hybrid approach to optimization: Surrogate Assisted Optimization (SAO) with a use case in propeller design

In this section, I take a class of design optimization problem, that is computationally cheap to evaluate but has high dimensional design space. In such cases, traditional surrogate-based optimization does not offer any benefits. In this work, I propose an alternative way to use ML model to surrogate the design process that formulates the search problem as an inverse problem and can save time by finding the optimal design or at least a good initial seed design for optimization. By using this trained surrogate model with the traditional optimization method, I can get the best of both worlds. I call this as *Surrogate Assisted Optimization (SAO)*- a hybrid approach by mixing ML surrogate with the traditional optimization method. Empirical evaluations of propeller design problems show that a better efficient design can be found in fewer evaluations using SAO. In such cases, the trained surrogate acts as a memory of experience (similar to an expert human designer) and is used to find good design directly or at least provide a good seed design for further optimization. For this purpose, the surrogate uses both nonlinear interpolation and nonlinear mapping to provide a good baseline for further optimization. The challenge of creating a surrogate in this case arises due to modeling expectations in this case. The modeling expectation is to try to get a good design from the requirement directly. Due to the acausal relationship between the requirement on design and the design parameter, it must be modeled as an inverse problem. Due to the causality principle, the forward problem in engineering systems has a distinct solution. On the other hand, the inverse problem might have numerous solutions if various system parameters

predict the same effect. Generally, the Inverse modeling problem is formalized in a probabilistic framework which is complex and not very accurate for high dimensional input-output and design space. I attempt this problem from geometric data summarizing algorithms that can model inverse problems and are useful in these problems. To differentiate this approach from surrogate-based optimization (SBO), I call this approach *Surrogate-Assisted-Optimization (SAO)*. The main difference between SBO and SAO is that in SBO, we use a surrogate in the optimization loop while in SAO, a surrogate is external to the optimization loop and only used to get a good initial baseline, further design optimization starts with this initial seed design provided by surrogate. The other difference is, in SAO surrogate attempts to inverse modeling problem instead of forward modeling problem in SBO. In SAO, the role of a surrogate is to provide all possible good designs or seed designs. For surrogate modeling, our choice of models are random forest and decision tree. The random forest has empirically shown to work the best for inverse modeling problems [5]. I also selected to train one decision tree on the entire data to create a memory map of collected data. Empirically I observed adding one decision tree trained on the entire data set along with a random forest of decision trees trained on various sub-samples of the dataset and using averaging improves the predictive accuracy and control over-fitting.

For empirical evaluation, I take the use case problem of propeller design [156], and the design space after coarse discretization is of the order of approximately 10^{38}. Based on the collected data requirement and training, when the SAO approach is applied to multiple optimization problems sampled from the requirement space. In all cases, I found SAO that leverage on initial good seed design from surrogate can find a better design on a given budget in comparison to the traditional method.

3.3.1 Propeller and openProp

Propellers are mechanical devices that convert rotational energy to thrust by forcing incoming forward fluids axially toward the outgoing direction. On a given operating condition such as the advance ratio (J) rpm of the motor and desired thrust, the performance of a propeller is characterized by its physical parameters such as the number of blades (Z), diameter of the propeller (D), chord radial distribution (C/D), pitch radial distribution (P/R) and hub diameter(D_{hub}) [40, 156]. The goal of a propeller designer is to find the optimal geometric parameters that can meet this thrust requirement with maximum power efficiency (η) (refer to figure 3.17). I use openprop [40] as our numerical simulation tool in this work. The output of simulation informs about the quality of the design choice, and accordingly, a bad design choice may result in poor efficiency or infeasible design and vice versa. The biggest challenge in the design search process arises from the exponentially large design space of the geometric parameter.

Openprop is a propeller design tool based on the theory of the moderately loaded lifting line, with trailing vorticity oriented to the regional flow rate. Optimization processes in openprop involve solving the Lagrange

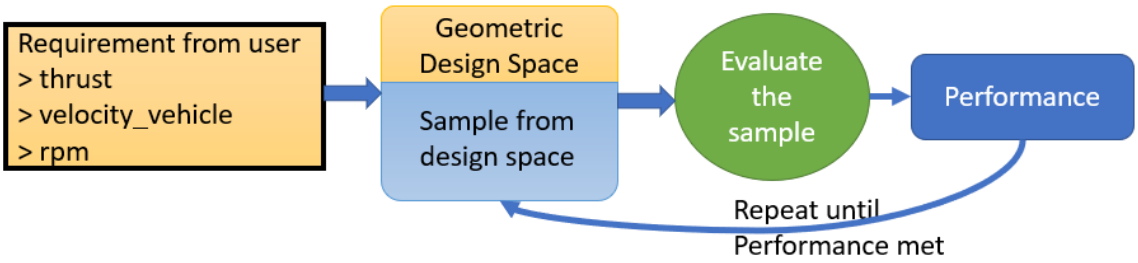


Figure 3.17: Propeller design optimization process in openProp. Sample evaluation is done in openProp simulator and performance is measured by the efficiency of the propeller.

multiplier (λ_1) for finding the ideal circulation distribution along the blade's span given the inflow conditions and blade $2D$ section parameters. The openprop applies Coney's formulation [31] to determine produced torque Q, thrust T, and circulation distribution *Gamma* for a given required thrust TS. For optimization purposes, an auxiliary function is defined as follows:

$$H = Q + \lambda_1 (T - T_s) \tag{3.41}$$

If $T = T_S$ then a minimum value of H coincides with a minimum value of Q. To find the minimum, the partial derivative with respect to unknowns is set to zero.

$$\frac{\partial H}{\partial \Gamma(i)} = 0 \; for \; i = 1, 2, ..., M \tag{3.42}$$

$$\frac{\partial H}{\partial \lambda_1} = 0 \tag{3.43}$$

By solving these M systems of non-linear equations using the iterative method -i.e. by thawing other variables and linearizing the equations with unknowns $\hat{\Gamma}, \hat{\lambda_1}$, an optimal circulation distribution and a physically realistic design can be found. For more details on numerical methods, refer to [31, 40]. Based on a given

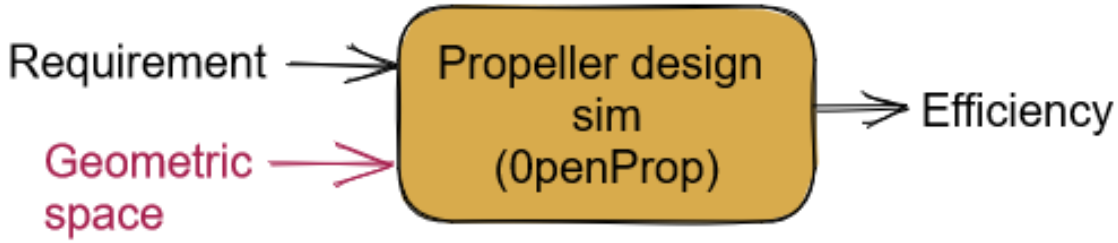


Figure 3.18: OpenProp numerical simulation

requirement imposed on a design in terms of operational and performance conditions, the goal of a designer

is to find an optimal geometric parameter of the propeller in minimum time. In OpenProp, the input design space can be split into two parts: (1) **Requirement space ($\mathscr{R}$)** that comprises of thrust, velocity of vehicle, rpm, and (2) **Geometric design space ($\mathscr{G}$)** comprises of chord profile radial distribution (C/D), diameter (D), hub diameter ($Dhub$), etc). The design space considered for this study is taken from [156]. Once samples were taken from this space, the requirement, and geometric design are put in the iterative numerical simulation algorithm to find the efficiency (η) of the design. The goal of design optimization is formalized as :

$$\underset{g \in \mathscr{G}}{\text{argmax}}\ \eta \quad for\ a\ given\ r \sim \mathscr{R} \tag{3.44}$$

Since this design optimization process for a given requirement involves running a sequential design selection from the input geometric space ($\mathscr{G}$), its evaluation and optimization until the requirements are satisfied. In such a case, another important aspect is to reduce the inception to design time ($\mathscr{T}_{design}$) i.e. design optimization time. Collectively, it can be written as:

$$\underset{g \in \mathscr{G}}{\text{argmax}}\ \eta \quad for\ a\ given\ r \sim \mathscr{R} \tag{3.45}$$

$$min\ \mathscr{T}_{design} \tag{3.46}$$

3.3.2 Formulation of design search as an inverse problem

In forward modeling and prediction problems, we use a physical theory or simulation model for predicting the outcome (η) of the parameter (g) defining a design behavior. The optimization process in the forward problem involves sampling from parameter space ($\mathscr{G}$) and striving to find the best parameter (g^*) that meets the requirement on the performance metrics (η). In the reciprocal situation, in inverse modeling and prediction problem, the values of the parameters representing a system are inferred from values of the desired output and the goal is to find the desired values of the parameters (g^*) that represent the output (η) directly.

In the propeller design use case, the objective of a designer is to determine the best geometric characteristics of the propeller in minimum time, based on a particular demand imposed on the design in terms of operational and performance conditions. The inverse setting in this case has some unique features:

1. One part of the input variables is known i.e. requirement. The other part of the input is unknown (geometry).

2. The effect or desired output is not fixed and the goal is to get the maximum possible efficiency that depends on requirements. (for example, it is not possible to produce a thrust with a small rpm motor

at some specific speed.)

To address these situations I formulate our inverse modeling problem as selecting and training a prediction model that can map a given requirement to the geometry and efficiency.

$$\mathscr{I}\mathscr{M} : \mathscr{R} \mapsto \{\mathscr{G}, \eta\}$$

Since it is not possible to find the maximum efficiency apriori, I filter all low-efficiency data sets (I treat these as infeasible designs) and keep only designs whose efficiency is higher. To model this inverse problem, I rely on geometric data summarizing techniques that learn the mapping between input and output space as sketches and the ability to regress between them. A sketch is a compressed mapping of output data set onto a data structure.

3.3.3 Why random forest and decision tree is our choice for modeling the inverse problem?

In the geometric data summarizing technique, the aim is to abstract data from the metric space to a compressed representation in a data structure that is quick to update with new information and supports queries. Let $D = \{d_1, d_2, ..., d_n\}$ are set of datapoints such that $d_i \in R^m$. For the purpose of representing data in sketches (S), the main requirement is the relationship (ψ) between the data points in metric space must be preserved in this data structure i.e $\psi\{T(d_k, d_l)\} \approx \psi\{S(d_k, d_l)\}$.

One of the selected relationships (ψ) between datapoints in metric space is L_p distance between datapoints. In such case, a distance-preserving embedding of this relationship in metric space is equivalent to tree distance between two data points d_k and d_l in a data structure (S). Tree distance is defined as the weight of the least common ancestor of d_k and d_l [55], then according to the Johnson-Lindenstrauss lemma [87] the tree distance can be bounded from at least $L_1(d_k, d_l)$ to maximum $O(d * log|k|/L_1(d_k, d_l))$. Accordingly, a point that is far from other points in the metric space will continue to be at least as far in a randomized decision tree.

$$L_1(d_k, d_l) \leq tree\ distance \leq O(d * log|k|/L_1(d_k, d_l))$$

Random Forest is a collection of specific kind of decision tree where each tree in a random forest depends on the values of a random vector that was sampled randomly and with the same distribution for all the trees in the forest. When the number of trees in a forest increases, the generalization error converges to a limit. The strength of each individual tree in the forest and the correlation between them determine the accuracy of a forest of tree. The error rates are better than Adaboost when each node is split using a random selection

of features [23]. To create a tree ($h(x,\theta_k)$ in the forest, θ_k is independent identically distributed random vectors independent of the past random vectors $\theta_1,...,\theta_{k-1}$ but from the same distribution. Due to ensembling and randomness in the forest generation process, the variance in *tree distance* also reduces to $L_1(d_k,d_l)$. Accordingly, geometric summarization of data from metric space to random forest can maintains the L_1 norm between data points in expectation. The decision tree trained on entire data-set has over-fitting issue and not suitable for generalization but due to space partitioning nature, it can map each observed requirement with multiple geometric designs and its efficiency when trained. By using both trained models in parallel, I can capitalize on both nonlinear mapping feature of decision tree as well as non linear regression/interpolation feature of random forest.

3.3.4 A hybrid optimization approach : Surrogate Assisted Optimization (SAO)

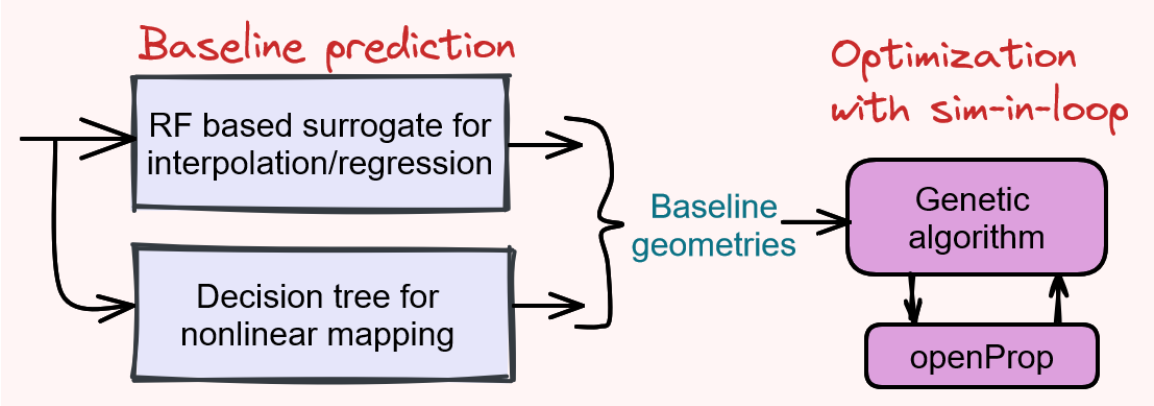


Figure 3.19: Surrogate assisted design optimization for propeller design

Figure 3.19 shows our approach to solving the propeller design optimization problem. It is a hybrid approach when the ML model is fused with a traditional algorithm with numerical physics in the loop of optimization. During training time, I train our random forest and decision tree. For training both models, I used requirement data ($r \sim R$) as an input and the corresponding geometric design values ($g \sim G$) and resulting efficiency (η) forming a tuple as output. The random forest is trained to learn the inverse regression and predict the design geometry along with efficiency on a given requirement. The decision tree on the other hand does inverse mapping from requirement space to design geometry and efficiency searched during data generation. The goal of random forest is to learn a function $f : \mathscr{R} \mapsto \mathscr{G}, \eta$ that is continuous so that I can regress for in between points however, the decision tree is a memory map and just does space partitioning on seen data. Using both gives up a good quality seed initial design. Since I do not know possible efficiency that can be achieved on a given requirement, I possibly take all possible predictions and sort on bases on efficiency to get the best design found yet. Direct prediction of random forest is an average of all geometric

designs and efficiency corresponding to the given requirement, which may or may not be a very good initial design. Here the role of random forest is generalization and regression on unseen data. The role of a decision tree is to do non-linear one-to-many inverse mapping. I selected all the designs that are on the leaf of the decision tree and include those as well to our baseline designs- this is called baseline prediction. Using both models, I get good quality initial seed designs. In the next stage, I take these baseline designs as the initial population and start the genetic algorithm search for the final optimized design. (refer to figure 3.19). In GA, chromosomes are represented by arrays of bits or character strings that have an optimization function encoded in them. Strings are then processed by genetic operators, and the fittest candidates are chosen. I run GA in a loop with the openProp numerical simulator until the budget.

3.3.5 Data generation & Training

For data generation, I took the design space used by [156]. The geometric design space is of the order of 10^{27} (diameter * nine alternative chord radial profiles), whereas the requirement space after coarse discretization is on the order of 10^{11} (thrust $\times$ velShip $\times$ RPM) with combined search space is 10^{38}. I take a single sample point from the physical design space and the requirement space and input it into the OpenProp optimizer. OpenProp internally optimizes this design using iterative numerical methods and computes the performance metric (η). I used this 0.205 million valid design data point for our training and testing. Using this design corpus, I trained both the random forest regression [23] model and the decision tree. Other hyperparameters of the random forest model are an ensemble of 100 decision trees with mean squared error as splitting criteria of the node. For the decision tree model, I chose squared error as the splitting criteria of the node, and nodes are expanded until all leaves are pure. Other hyperparameters are kept as default settings as in SKlearn [110].

3.3.6 Experiment results

For sharing the result, I have two things to share:

1. prediction accuracy of random forest on test data.
2. Empirical evaluation of SAO (on example design optimization problems and its comparison with baseline (Genetic Algorithm).

For testing the prediction accuracy of our trained model I selected 5% of data randomly from the dataset. To assess the quality of prediction, I used the following common statistics as evaluation metrics:

1. average **residual**, $\Delta Z = (\eta_{truth} - \eta_{predicted})/\eta_{truth}$ per sample
2. the **accuracy**, percentage of the number of samples whose residual is within acceptable error of 5% i.e $|\Delta Z| < 0.05$.

It measures the percentage of test data on which the prediction of efficiency is within 5% of error (since efficiency is a good metric and target of final prediction). I found percentage prediction accuracy on test data for the random forest is around 90%. For the decision tree, I fitted it with the entire data, since I just want space partitioning of collected data.

For the empirical evaluation of SAO, I chose Genetic Algorithm as our baseline optimization algorithm that is frequently deployed in such situations. Figure 3.20 shows the evaluation traces of the optimization process. It can be observed that due to the trained surrogate, I get a better initial seed design, and further optimization in the second step using GA provides better designs on the given budget in comparison to applying GA which starts with a random seed design.

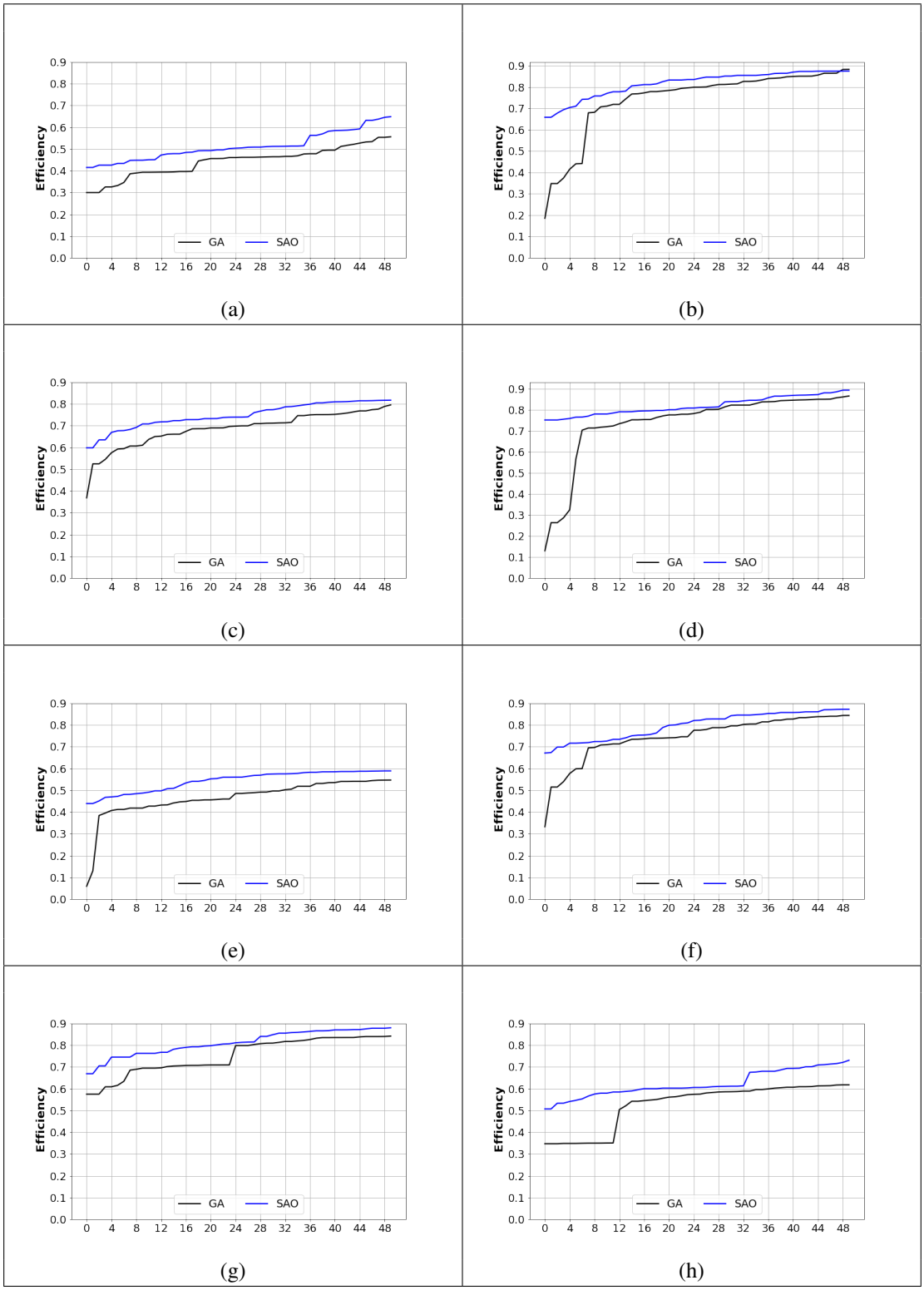


Figure 3.20: Results of sample optimization runs using GA and Surrogate Assisted Optimization (SAO): Due to learned manifold SAO provided better seed design for evolutionary optimization and get better performing design in given budget. Requirements for optimization are sampled randomly for Design space:{thrust (Newton) ,velocity of ship (m/s), RPM}(a) $\{51783, 7.5, 3551\}$, (b) $\{127769, 12.5, 699\}$,(c) $\{391825, 12.5, 719\}$,(d) $\{205328, 19.5, 1096\}$,(e) $\{301149, 7.5, 1215\}$,(f) $\{314350, 16.0, 777\}$, (g) $\{31669, 17.5, 2789\}$,(h $\{476713, 15.5, 2975\}$.

3.4 Approach to problem III: Surrogate modeling in CFD and FEA domains and Surrogate Based Optimization (SBO)

Surrogate model is a non-physics-based approximation that typically involves interpolation or regression on a set of data generated from the original model. Mathematically it can be seen as a non-linear inverse problem for which a function (f) needs to be discovered from a limited set of available data while minimizing the approximation error (e) on other points in the design domain. In the Engineering design field, the role of the surrogate model is to gain insight into the problem/process to make decisions based on the analysis. The most use of surrogate models is to augment the results coming from expensive simulation code to gain the key benefit of a significant increase in speed for the evaluation of a design. For designing a useful surrogate this benefit is possible only when acceptable accuracy is achieved by the surrogate model. The gain in run time for design evaluation becomes really important for some physical process that takes a very long time to evaluate, for example, computational fluid dynamics, Finite element-based analysis, etc. These computationally expensive high-fidelity models describe systems with high accuracy, while low-fidelity models are less accurate but computationally cheaper than high-fidelity models. The primary motivation of surrogate modeling is to speed up the evaluation of models that are otherwise prohibitively expensive to run in optimization and verification processes.

3.4.1 Deep learning-based FEA surrogate for sub-sea pressure vessel

The pressure vessel is a critical component in modern-day Unmanned Underwater Vehicles, Remotely operated Vehicles, etc. It is designed to withstand the sub-sea hydro-static pressure conditions while remaining watertight and contains power sources and electronic and other sensors that cannot be flooded. For this purpose, an FEA-based simulation is conducted to measure the maximum induced stress in a design subject to the subsea pressurized environment. The simulation consists of multiple steps - CAD modeling, body meshing, and numerical solution of FEA to get the stress distribution. The measured maximum static stress is compared with the yield strength of the material to check the integrity of the pressure vessel during operation. This process is repeated until an optimal design that can withstand the sub-sea pressure is found. In this work, our goal is to learn a surrogate model for this entire simulation process for a large class of problems. Since each simulation is computationally very costly ($\approx$ 202 seconds per simulation on a 20-core CPU), it is not possible to run many simulations and generate dense data. The motivation for learning a surrogate is the ability to interpolate or predict the numerical simulator's output at a very low cost using trainable models. Once trained the surrogate can replace the numerical simulator for outer loop optimization operations. The expected benefit is the ability to design the pressure vessel on a given requirement at a very small computational time for a range of problems.

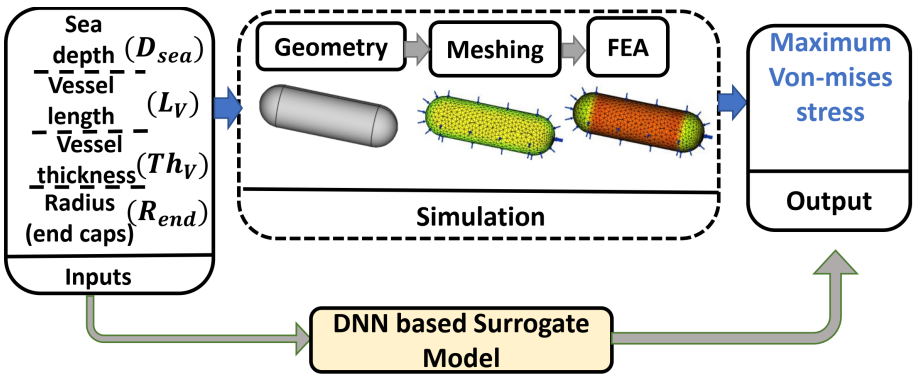


Figure 3.21: Finite element analysis and surrogate modeling process

The class of problem is defined by the design space (which are parameters that can be changed for changing the design and environmental conditions). For learning a surrogate, I first create a design space. Our design space consists of the maximum depth of the sea, which decides the maximum applied sub-sea pressure on the pressure vessel. Most pressure vessels have a circular cylindrical shell with two hemisphere end caps configuration, justified by its benefits such as balanced structural integrity, homeostatic balance, and better internal volume. This cylindrical pressure vessel is parameterized by 3 parameters (length of cylinder, radius of end-cap, thickness of vessel). Accordingly, the design space consists of 4 variable spaces ($D_{sea}, L_v, Th_v, R_{end}$) (refer to figure 3.21). In this paper, I am interested in designing a range of pressure vessels that can work between range 0-6000 meters of sea depth. The material used to design the vessel is Aluminium alloy (Al6061-T6) which is the most commonly used material for pressure vessel design.

The training process of the ML model starts with the collection of simulated data ($D_{simulated}$). The simulation involves the design of CAD geometry based on design parameters, the meshing of CAD designs, the application of desired hydrostatic pressure on an external surface of design, and then a numerical solution using finite element analysis. Once the FEA simulation converges, the maximum Von-mises stress is measured (refer to figure 3.21). After the collection of training data, I split it into train data (D_{train}) and test data (D_{test}). The NN architecture is a multilayered NN with 9 layers. Our network has four major constituents- a fully connected layer (linear+ReLU), a dropout layer, skip connections, and an output layer. The input layer is the concatenation of the normalized vector of design variables and sub-sea pressure ($[D_{sea}, L_V, Th_V, R_{end}]$). The fully connected layer (linear+ReLU) consists of linear weights and biases with the commonly used nonlinear activation function ReLU (Rectified Linear Unit [167]) defined by $f(x) = max(x, 0)$. All the neurons in the previous layer are connected to every neuron in the current layer. Two dropout layers are added in between these fully connected layers for regularisation and to prevent over-fitting with a dropout factor of 0.2. The output of the FEA simulation is a scalar value $s \in \mathbb{R}$, so our choice of the output layer is a linear fully con-

nected layer. The last important feature is the skip connection, which is a residual network added for smooth learning and to handle vanishing gradient or exploding gradient [60].

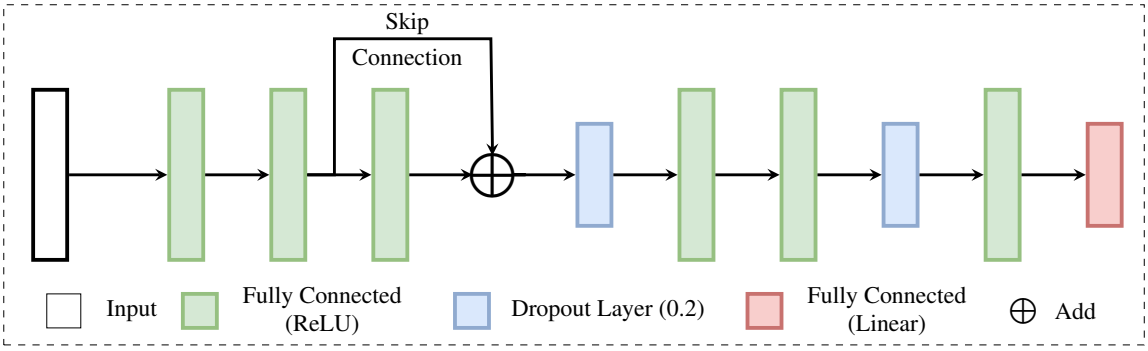


Figure 3.22: Our custom deep neural architecture

The overall architecture is represented in figure 3.22. The network has given the input in a batch of n with each sample vector size is 4. The number of parameters in our architecture is 22,993. A large network (22,993 parameters) in comparison with the small data size (8000 training data points) raises concerns about over-fitting. The drop-out layer and cross-validation-based early stopping are used to counter this in our regression model.

I divide the collected data (11311 data points) randomly into a training sample containing 8000 FEA evaluations and a testing sample containing the remaining 3311 evaluations. Due to the unavailability of large data, to increase the performance of prediction, I trained a family of deep learning networks as it has been seen to be more accurate than individual classifiers [125]. For this purpose, I used 5-fold cross-validation and trained one base deep neural network on each fold dataset. I further divided each fold dataset into the training dataset (90%) and the validation dataset (10%). The prediction of the individual trained networks was then averaged out to obtain the final values of the maximum Von-mises stress on a given design (refer to figure 3.23). After experimenting with different loss functions and learning rates, I found that L1 loss performed better than other losses with a learning rate of 0.001. I used Adam [75] optimizer with Xavier initialization for training the model. Apart from our deep learning-based learning model, I also trained the decision trees-based model to predict the maximum Vonmises stress. Most popular decision trees based predictive model random forest [23] and Gradient boost regressor [22] is trained which are considered the best classification and regression models and most popular predictive analytic techniques among practitioners, due to being relatively straightforward to build and understand, as well as handling both nominal and continuous inputs [24]. I trained 100 different decision trees and tuned their depth, number of trees, and split criteria to get the most accurate model. To assess the quality of prediction, I used the following common statistics:

1. average **residual**, $\Delta Z = (V_{truth} - V_{predicted})/V_{truth}$ per sample

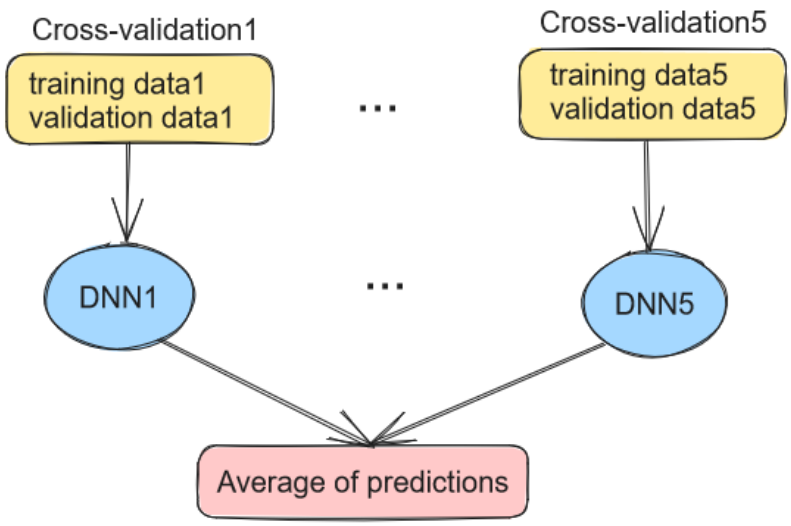


Figure 3.23: Ensemble of learning models

Table 3.3. Result on key performance metrics

Model	Accuracy	Residual	Outlier	Deviation
Deep ensemble	92.20%	0.045	48	118.67
Random Forest	72.27%	0.13	164	141.23
Gradient Boost	47.47%	0.21	157	132.60

2. the **accuracy**, percentage of the number of samples whose residual is within acceptable error of 10% i.e $|\Delta Z| < 0.10$,

3. η: the number of **outlier**, $|\Delta Z| > 0.50$, number of samples who which prediction has a high error.

4. the **standard deviation**,(σ) of prediction.

Here, V_{truth} and $V_{predicted}$ are the real label and the predicted outcome of the trained regression models. For comparison purposes, apart from our model I also trained other regression models i.e. Random forest regression model and the Gradient Boosting regression model. Models have been developed and trained using SKlearn, PyTorch, and XGBoost (extreme gradient boosting) Python libraries.

3.4.1.1 Results on model performance metrics

For regression modeling, the quantitative performance metrics are residual/mean absolute error (MAE) and root mean squared error (RMSE)/ standard deviation. I also have other evaluation metrics- accuracy and outlier which are defined earlier. The lower the MAE/residual, the more precise the model is. The accuracy metrics give us statistics about how many samples are with an acceptable level of accuracy. The standard deviation estimates the average proportion of the variation in the response around the mean. The outlier in

prediction provides insight into how many large error samples are in the prediction, which gives a better picture of the distribution of deviation. Table 3.5 shows the performance metrics on all three trained models on the test data set. All three models are trained on the same 8000 training data set and tested on the 3311 test data set. The DNN model performed at significantly higher accuracy (92%) in comparison to other models - Random forest (72%) and Gradient boost regressor (47 %). The DNN model also has one-third less number of outliers than the Random forest and Gradient Boost models. In all the metrics ensemble DNN model performs much better than Random forest and Gradient Boost-based regressors.

3.4.2 Deep learning-based surrogate for drag prediction on underwater vehicle hull

For evaluating the effect of hull shape on drag, I take CFD based simulation outcome as ground truth. The CFD is a great engineering process that revolutionized the field of aerodynamic and nautical design problems because it saves both cost and time in comparison to making a prototype, adjusting a real model, and quickly changing the computer-generated model to find a more refined design outcome before the design process. With many developments in numerical techniques over the years, CFD has become more and more accurate and in conformity with real-world behavior. The water tank/ wind tunnel testing is conducted in order to validate the results obtained with computer simulations.

For data generation using CFD simulation, I consider the space of solutions with a range shown in table 3.4. I tried to cover the design space for small UUV shapes. I used the Myring hull as our baseline parametric design shape which is the most frequently used UUV shape. The design of the hull is randomly sampled from the space from the range mentioned in the design space. For CFD physics evaluation, RANS simulation was used with k-ω turbulence model, which is inspired from real-world UUV design process [72] using open source tool OpenFOAM. For CAD modeling of each parametric design, I employ an open-source CAD design tool called freeCAD. The STL file generated by the CAD processing tool is used as the final design for the CFD. The pipeline of parametric CAD design and CFD process is automated and run in different Docker containers and coordinate through the file sharing system controlled by python scripts. The inflow horizontal velocity is kept fixed for all simulated designs at the nominal operating speed of small UUV Remus 100 class to approx 4 knots. The other flow characteristics are also kept constant for all different designs and obtained from empirical values. The flow characteristics which are kept constant for all designs are incompressible flow, with flow density of 1027 Kg/m^3, the kinematic viscosity of water is 1.73 *centiStoke*, the inlet turbulence intensity is assumed between medium and low due to moderate speed of flow and not-so-complex geometries at 1%. The turbulence model constant (C_μ) is also kept constant at 0.09. Data generation is done in multiple phases since each simulation is very costly and took an average evaluation time of approximately 10 minutes per simulation on a 20-core CPU (Intel i9-9820X 10-core CPU @ 3.30GHz and

Parameter	Symbol	Minimum	Maximum
Diameter of middle section	d	50 mm	200 mm
Length of nose section	a	50 mm	600 mm
Length of middle section	b	1 mm	1850 mm
Length of tail section	c	50 mm	600 mm
Index of nose shape	n	1.0	5.0
Tail semi-angle	θ	0^o	50^o
Total length	$l = a+b+c$		

Table 3.4: Design space: Range of design parameters for surrogate modeling

64GB of RAM). Due to the intermittent errors that occurred during meshing or flow filed calculation, the simulation is restarted multiple times. The number of the generated designs was 3333 and of which 312 were invalid designs and the rest 3021 were valid designs. For learning the larger spatial extent design space, it is typically preferable to generate a large number of data points. The choice of size of generated data was guided by the number of resources, time available, and accuracy of the trained model during experimentation. In the initial phase, I started with 500 data points and trained and tested different neural networks for desirable accuracy. Due to the failure to get desirable accuracy, I kept growing the data set. After every phase, I did testing and training on the collected data on different networks to get desirable accuracy. For maintaining sanity, I deleted everything done in the last stage of training and testing and do everything from scratch when moving to the next stage of data generation except the data.

Our neural network model is a custom neural architecture designed based on a fully connected network, which is widely used for a wide variety of tasks. Considering the size of the data, I started with a multi-layer feed-forward network because our model had 6 input features and 1 output. A neural network is an interconnected web of elementary computation units (neurons). When organized in multiple layers, it becomes a very powerful universal function approximator. Our NN architecture has a multilayered architecture with 9 layers. The main components in the network are: the fully connected nonlinear layer, dropout layer, skip connections, and output layer. The inputs are stacked in the following order $x_1 = a$, $x_2 = b$, $x_3 = c$,$x_4 = d$, $x_5 = n$, $x_6 = \theta$ and the concatenated vector $X = [x_1, x_2, x_3, x_4, x_5, x_6]$ of input data is normalized across each features using Min-Max Normalization.

$$\hat{X}[:,i] = \frac{X[:,i] - min(X[:,i])}{max(X[:,i] - min(X[:,i])}$$

The normalized vector $\hat{X}$ is fed into as the neural network input for prediction. The fully connected layer encompasses one linear matrix operation with a trainable weight parameter (W) and then is added with trainable bias (b). The outcome of this linear operation is passed through the nonlinear activation function ReLU

(Rectified Linear Unit [167]) defined by $f(x) = max(x,0)$. Each section of the fully connected layer does this information processing ($max(WX+b,0)$) and passes this information to the next layer. Two dropout layers were added in between these fully connected layers for regularisation and to prevent over-fitting and improve generalization. At the dropout layer, Hadamard product between last layer output vector size (let's assume Q) and generated random Bernoulli variable of the size Q ($Q_i \sim B(p)$ and $z = \{Q_1, Q_2, ..., Q_n\}$) is carried on. The dropout parameter p decides the probability of Bernoulli's output $B \in \{0,1\}$ is decided by the dropout parameter p. Here, p is the probability of outcome 1, and $(1-p)$ is the probability of outcome 0. The combined output of the fully connected layer and dropout layer can be written as $max(WX+b,0)\,o\,z$, where o is the Hadamard product. The number of parameters in our architecture is $25,664$. A large network ($25,664$ parameters) in comparison with the small data size ($2,400$ training data points) raises concerns about over-fitting. The drop-out layer and cross-validation-based early stopping are used to counter this in our regression model. The dropout parameter in our network is set to 0.2 for both drop-out layers. I also added a skip connection /residual layer at the inner hidden layer, which facilitates smooth learning and handles vanishing gradient or exploding gradient [60]. Our output layer is a linear layer, with only one linear unit.

The neural network is trained using a back-propagation algorithm with the evaluated cost function based on the error between the predicted output of the trained network and the ground truth generated from CAD-CFD simulation. I used Adam [75] optimizer built in PyTorch to train the parameters of our network. Initially, I used mean squared error (L2 loss) to train the network but I found that L1 loss (mean absolute error) gave a better result in comparison to L2 loss. I also deployed early stopping to prevent over-fitting and improve generalization on our model on training data apart from the dropout layer used in the network architecture. For training and testing purposes, I first split the total collected data points (3021) into two sets - the first set consists of 2400 data points, and the second set consists of 621 datapoints. For early stopping and cross-validation of the model, I divide the first set of data (2400 data points) further into a $90:10$ ratio and used the 90% part for training to find the network parameters that minimize the cost error function and used the rest 10% for validation and early stopping of training to get better generalization error from the trained network. The validation error is compared to the training error after each epoch of training and during the training process is model is saved only when the gap between the training error and validation error is reduced. The second set of data 621 data points are used for testing the performance of the trained surrogate.

To test the prediction accuracy and generalization capability of the trained surrogate, I used the following metrics:

1. **Residual**, $\Delta Z = \frac{(F_d^{gt} - F_d^p)}{F_d^{gt}}$
2. The **accuracy** (α), percentage number of test samples whose residual is within acceptable error of 5%

Table 3.5. Key performance metrics

Number of test data points	**Accuracy (α)**	**Number of Outliers**	**%age outlier (η)**
621	97.0%	9	1.4%

i.e $|\Delta Z| \leq 0.05$

3. The **outliers** (η), percentage number of test samples on which trained surrogate prediction error is more than acceptable accuracy,$|\Delta Z| > 0.10$.

Here, F_d^{gt} is the ground truth drag force estimated by running computational fluid dynamics simulations, and F_d^p is the predicted outcome using the trained DNN surrogate model. For each test sample in the test data, I calculate the residual error. Since, for regression problems whose output is a scalar value, the most effective way to measure the prediction outcome is whether the predicted output scalar is within an acceptable accuracy range.

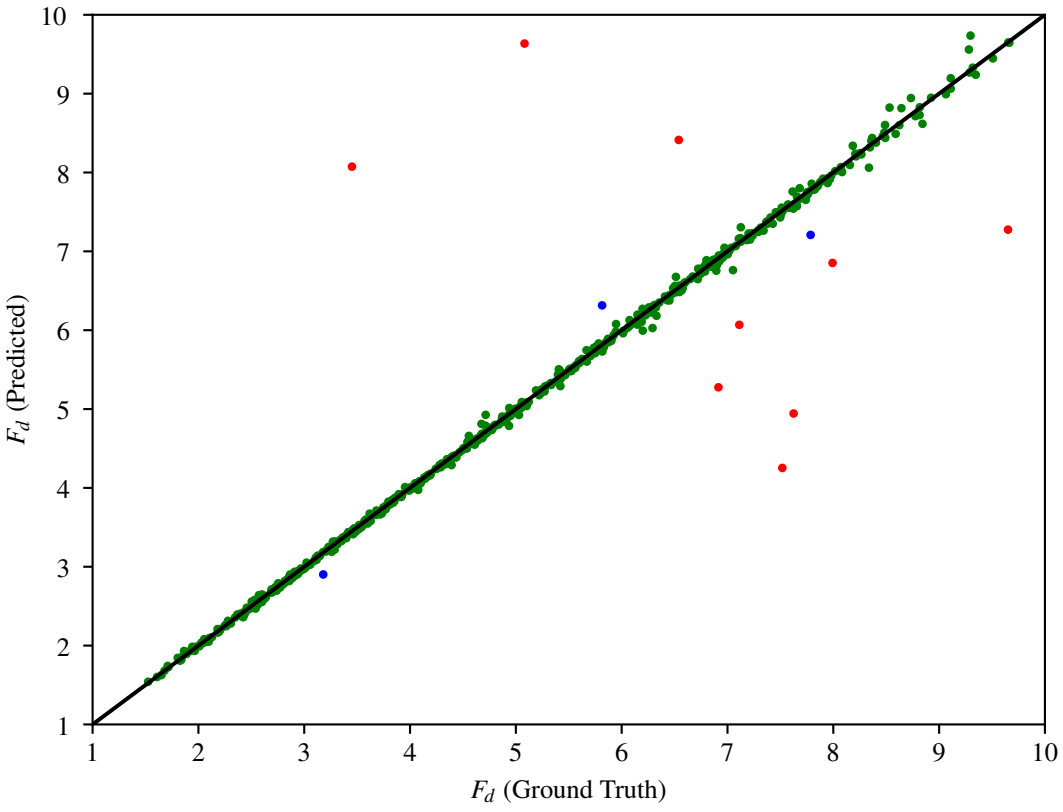


Figure 3.24: Ground truth vs. predicted values of drag forces (F_d) by the DNN surrogate. The green markers denote predictions with less than 5%, error; yellow markers denote predictions between 5%–10% error and the red markers denote predictions with errors greater than 10%

3.4.3 Surrogate-based design optimization for UUV hull design

In this section, I consider evaluating the performance of surrogate-based optimization. For this purpose, I consider three different design problems and run the surrogate-in-loop-optimization along with full OpenFOAM solver-in-loop-optimization using Bayesian optimization-Lower Confidence Bound (BO-LCB) as our optimizer (refer to figure 3.25). The goal of optimization is to find the UUV hull shape that produces the minimal drag force. The operating conditions like inlet velocity, turbulence intensity, boundary layer conditions, etc. are kept constant as what I used during the data generation process. The optimization objective in all

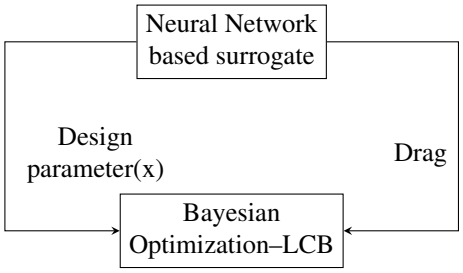


Figure 3.25: Surrogate based optimization

three cases is to minimize the drag force on the hull's body by finding the optimal shape parameters S (where $S := \Omega$ and $S \in DS$). The Reynolds number $\mathbb{R}_{\mathbb{L}}$ is based on the length of the UUV (L) and is different in each problem.

In order to validate to drag forces predicted by the DNN surrogate, I hypothesized that, on a given random seed (initial conditions), the optimal design obtained using the BO–LCB algorithm when used with our DNN surrogate in the loop should be similar to the running BO–LCB algorithm with OpenFOAM in the loop. To that end, I ran the following design problems with constraints on some of the variables:

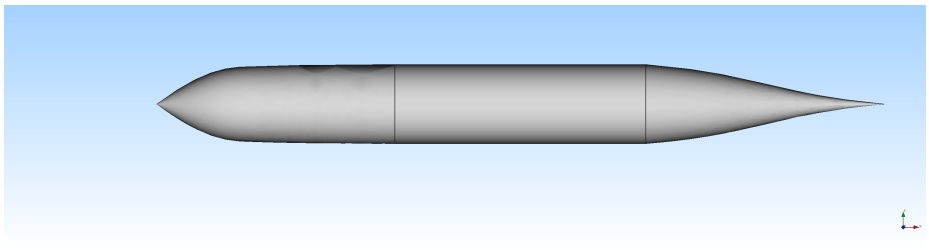
1. Design with constraints on total length (l), and the maximum diameter (d).
2. Design with constraints on nose length (a) , total length (l), and the maximum diameter (d).
3. Design with constraints on nose length (a), body length (b), tail length (c), and the maximum diameter (d).

These constraints are the conditions that must be met. In UUV design, the constraint is dictated by space taken by the components (battery, sensors, electronics, motors, etc.) and their placement for sensing and operation. During optimization, these constraints are imposed both on the CAD-CFD integrated simulation tool as well as our trained DNN surrogate.

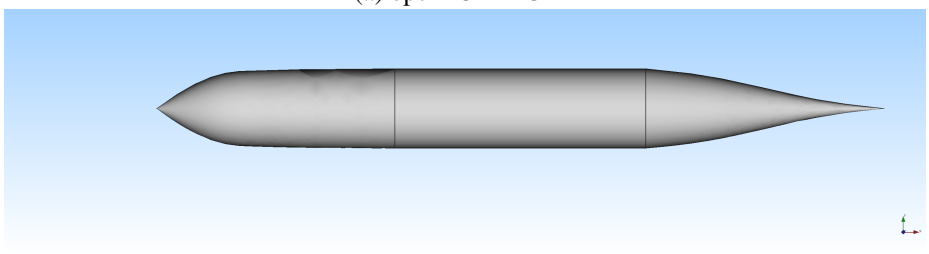
Table 3.6 shows the optimal design parameters when using both OpenFOAM and the trained DNN surrogate in the loop. When the optimization process converges or the budget gets over, I compared the obtained

Table 3.6: Optimal design parameters(a,b,c,d,n,θ) and drag force (F_d) found for selected design variable constraints using BO-LCB with both OpenFOAM and neural network surrogate in the loop. Here **D** is diameter , **L** is length and k denotes the number samples evaluated by optimizer before convergence

(D, L)	OpenFOAM in loop								DNN surrogate in loop							
	a	b	c	n	θ	F_d	k	**t(sec)**	a	b	c	n	θ	F_d	k	**t(sec)**
(**180.0**,**1750.0**)	573.0	604.0	573.0	10.0	1.0	6.089	46	**4181**	573.0	604.0	573.0	10.0	1.0	6.01	50	**0.14**
(**190.0**,**1330.0**)	50.0	834.61	445.38	4.467	2.544	7.89	50	**2982**	50.0	682.10	597.89	2.498	1.685	6.34	50	**0.115**
(**100.0**,**500.0**)	100.0	150.0	250.0	1.1	4.51	1.54	50	**1625**	100	150	250.0	1.0	19.95	1.86	50	**0.14**



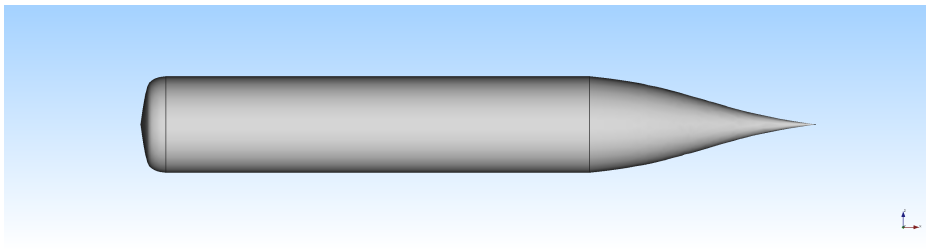
(a) openFOAM CFD



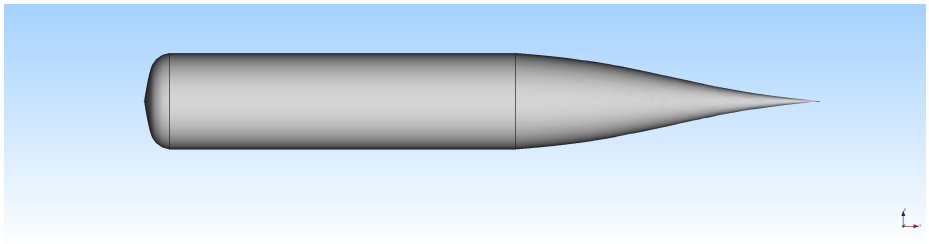
(b) DNN surrogate

Figure 3.26: Results: Optimal design with lowest drag force (F_d) founds using BO-LCB optimizer for design problem 1 ($\mathbf{D} = 180, \mathbf{L} = 1750$) (a) with OpenFOAM in the loop (b) with neural network surrogate in the loop.

optimal designs. In some cases, the optimal design obtained from the OpenFOAM and DNN surrogate are almost identical and their parameters are the same up to two decimal points ($\mathbf{D} = 180, \mathbf{L} = 1750$). In some cases, the designs are somewhat different but still, they are pretty close and need a careful visual inspection to find the differences ($\mathbf{D} = 190, \mathbf{L} = 1330$). In most cases, drag coefficients attached to each optimal design have good accuracy with what I obtain from OpenFOAM. Also the optimal design parameters for both yield very close drag forces and in some cases ($\mathbf{D} = 180, \mathbf{L} = 1750$), are identical while in other cases, it is similar. This can be easily addressed by the addition of some more data. However, our DNN surrogate in loop BO–LCB–the most sample-efficient optimization framework–requires more sample evaluations to converge than when using OpenFOAM in the loop. However, the time taken for finding the optimal design by our DNN surrogate is approximately $10,000$ times faster when compared with the time taken for finding the optimal design using OpenFOAM (refer to table 3.6).

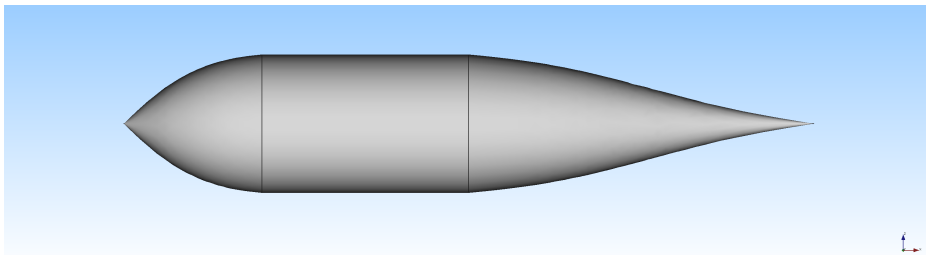


(a) openFOAM CFD

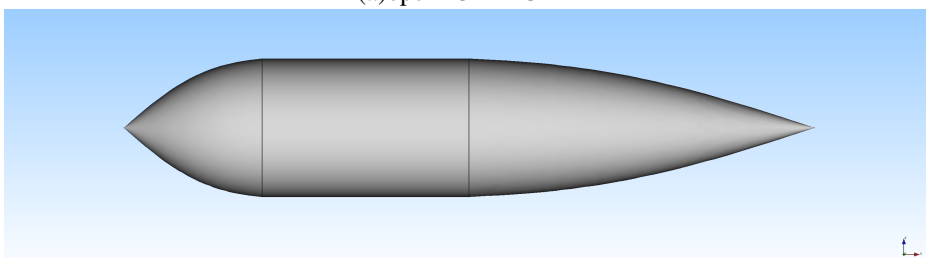


(b)DNN surrogate

Figure 3.27: Results: Optimal design with lowest drag force (F_d) founds using BO-LCB optimizer for design problem 2 ($\mathbf{D} = 190, \mathbf{L} = 1330$) (a) with OpenFOAM in the loop (b) with neural network surrogate in the loop.



(a)openFOAM CFD



(b) DNN surrogate

Figure 3.28: Results: Optimal design with lowest drag force (F_d) founds using BO-LCB optimizer for design problem 3 ($\mathbf{D} = 100, \mathbf{L} = 500$) (a) with OpenFOAM in the loop (b) with neural network surrogate in the loop.

3.5 Preliminary result in search of universally optimal UUV hull shape

In UUV design, hull resistance affects range of the vehicle, battery size, weight, and volume requirement of the design. By running the optimization at different operating velocities and turbulence intensity, I want to study/search for the possibility of a universal design that will provide the least resistance/near-optimal design

across all operating conditions (operating velocity) and environmental conditions (turbulence intensity) or to find what kind of different shapes that will emerges as the optimal design in different scenarios and is there any invariant feature in it. For creating the different environment and operating conditions for AUV, I consider five different environmental conditions (turbulence intensity of flow $\{0.1, 2, 5, 10, 20\}$ percent of mean flow velocity) and five different operating conditions (velocity of AUV $\{1, 2.5, 5, 7.5, 10\}$ in meters/second). The operating velocity and turbulence intensity values are taken from empirical ranges observed in real world underwater vehicle. Accordingly, the cartesian product of these two sets creates 25 different scenarios with different operating velocities and turbulence intensity. By running Bayesian optimization for each 25 scenarios with CFD in the loop, I want to find the optimal hull design in all scenarios. For running CFD, I chose our integrated tool chain and methodology explained above. For experimentation, I consider an axisymmetric body of revolution with a fixed hull length (1 meter) and fixed fineness ratio (5) that poses a constraint on the hull diameter, which is 0.2 meters. The rest of the shape of the hull can be changed by 6 control points across the body by changing its location in space. For each of the 25 experiments, the turbulence energy and dissipation rate are provided as initial conditions. The hull has two parts -nose and tail and their lengths can be controlled by the parameter nose_length and the tail_length (tail_length=1-nose_length (in meters)). By changing the length of the nose, I can move the fineness ratio constraint across the body and provide an optimizer capability to search for a wide variety of shapes. For the optimization algorithm, I chose Bayesian optimization, since it is a very sample-efficient optimization framework involving expensive functions.

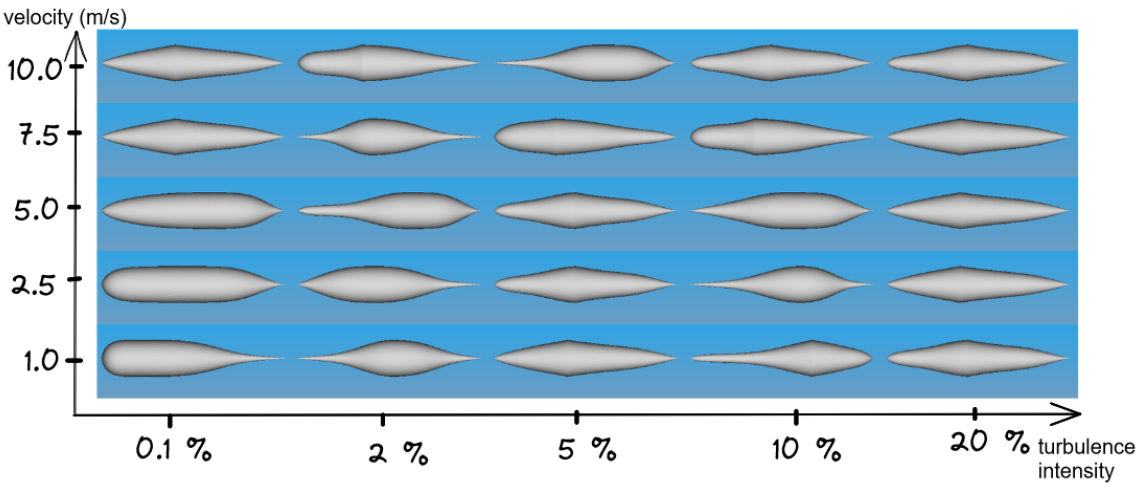


Figure 3.29: Optimal designs

At every 25 scenarios, I ran our optimization algorithm until convergence or up to the allocated budget of 100 iterations. At the end of the optimization process, the discovered optimal designs are shown in figure 3.29, and the related drag force is shown in figure 3.30. The preliminary results show that at high turbulence

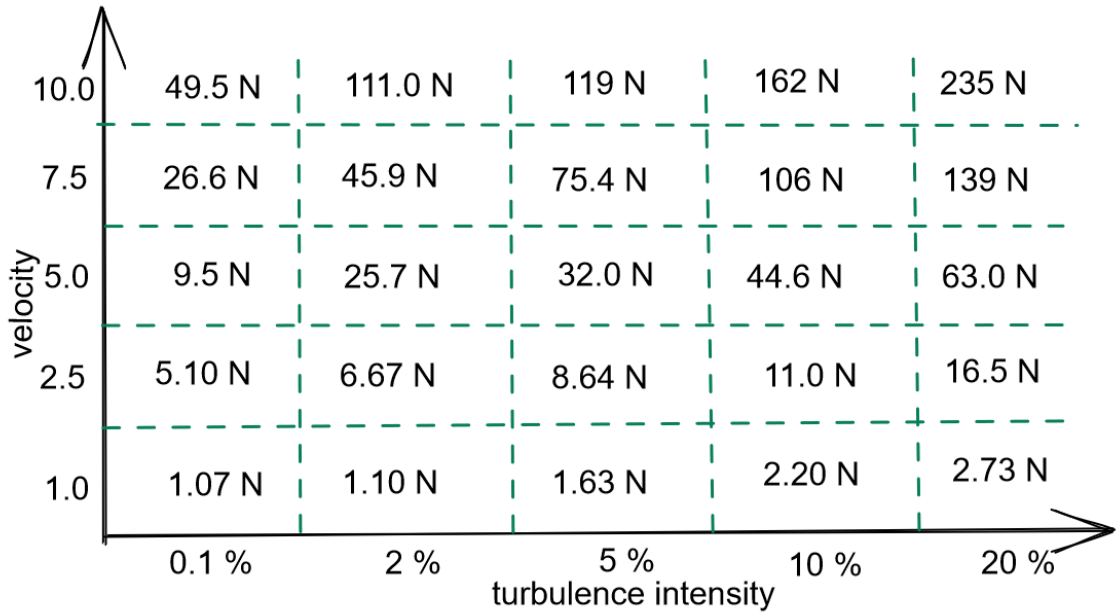


Figure 3.30: Ground truth drag values of UUV designs

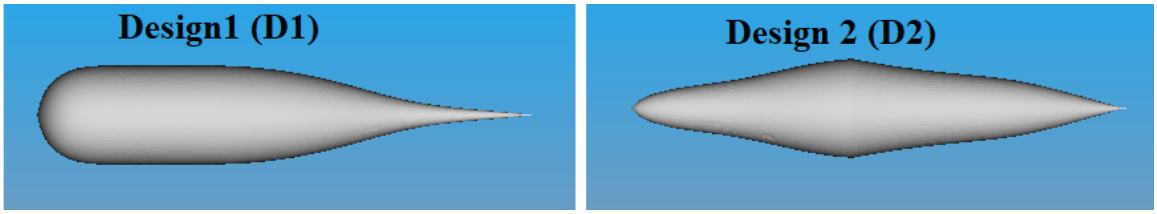


Figure 3.31: Two chosen extreme designs

intensity, the optimization converged to a single design (rightmost column of figure 3.29). For comparing the performance of optimal designs, in this work, I selected two optimal designs for study (refer to figure 3.31) first, the design obtained at lowest turbulence and lowest speed (D1: velocity:1m/s and turbulence 0.1%) and second, the design obtained at highest turbulence and highest speed (D2: vel:10m/s and turbulence 20%)). Once I tested both these designs' performance in all 25 scenarios, I observed, that design D1 produced more drag in all 24 scenarios while design D2 produced significantly lesser drag resistance in 18 out of 24 scenarios when compared to optimal design obtained by running BO (refer to figure 3.32 for drag resistance values). Also, the drag resistance of design D1 increases significantly at high velocity and high turbulence conditions in comparison to design D2. The future direction would be to conclude this research by analyzing all 25 optimal design and their performance across all considered environmental and operating conditions.

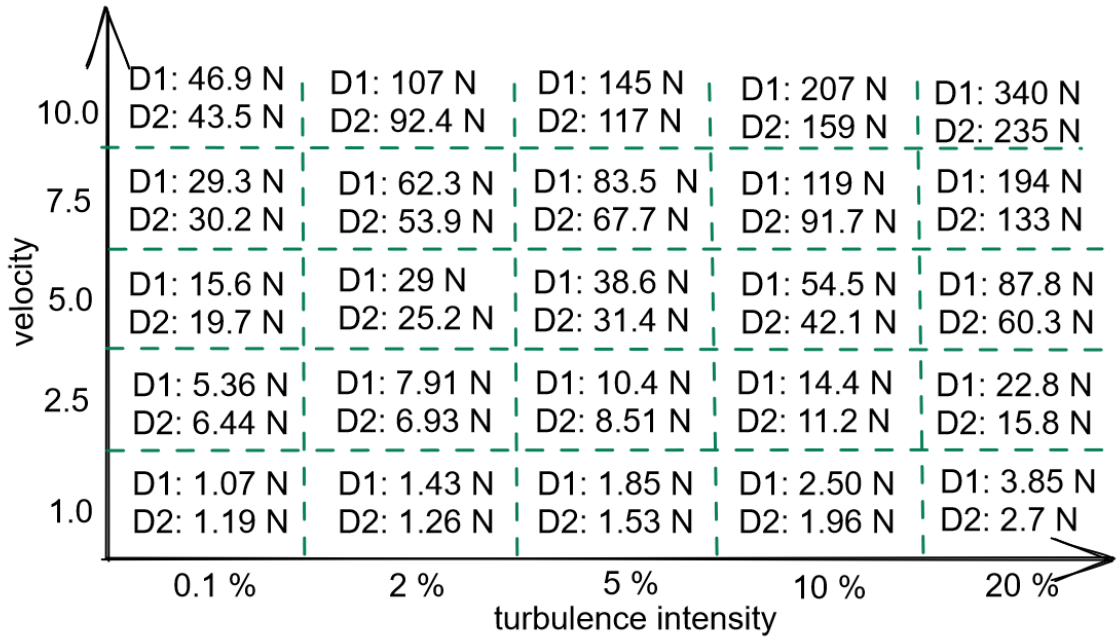


Figure 3.32: Ground truth values of designs D1 and D2 at all 25 different scenarios.

3.6 Tool outcome: *Anvil*- A SciML tool for CFD based shape optimization

In engineering design involving fluid mechanics, CFD offers numerical analysis to study and understand the effect of coupled solid-fluid dynamics of a shape. The shape optimization or prototyping process in this case involves iterative evaluation of different shapes using CFD until the design objectives are met. In this context, there are two main challenges: first, iterative evaluation without manual intervention requires integration of the CAD tools with the CFD tool, where CAD is used to create, modify and generate a design for CFD analysis. Some advanced commercial tools offer this integrated CAD-CFD to complete automatic CFD-based studies, however, there is no open-source version available to researchers in this field. The second challenge is the integration of state–of–the–art optimization algorithms developed in recent research or sampling process with this CAD-CFD tool that can make decisions about the design choices during the optimization or data generation process.

In this work, I offer an open-source freely available integrated CAD-CFD tool by integrating two open-source tools FreeCAD (for CAD modeling) and OpenFOAM (for CFD) that is integrated with AI-based based sample efficient optimization method (Bayesian optimization) and other sampling algorithms. I call this tool *Anvil* and present it as a scientific machine learning tool in the field of shape optimization that can be used in different modes: first, the data generation mode where on a given parametric CAD seed design, design space, computational resources, and budget the tool can automatically run CFD evaluations and generate data

for training a surrogate model, second in an optimization mode, where a given parametric CAD shape and objective, design space and budget, it searches to find the optimal design. *Anvil* is easy to set up and the experimenter only needs to provide a JSON file as a configuration file and a parametric CAD seed design. It can be used to study the effect of solid-fluid dynamics for any subsonic flow conditions (underwater, land, or air). Since it is integrated with Python, that also provides a means to easily extend its capabilities like integrating new and other optimization algorithms or sampling methods like GFlowNet, OpenMDAO, etc. I demonstrate the capability of *Anvil* in different simulation use cases. The open source code for the tool, installation process, artifacts (like CAD seed design, example STL models), experimentation results, detailed documentation, etc can be found at https://github.com/symbench/Anvil. The configurable parameters of JSON config file, its meaning, and options are given in table 3.7. *Anvil* can be used in three modes of operation. These modes are:

1. CFD evaluation: CFD evaluation is one of the primary building blocks of this tool. In this mode, the tool behaves as a CFD analysis tool on a given design and flow condition.

2. Data generation: Data generation on a given parametric design, design space, and budget.

3. Optimization: Optimization of a design using Bayesian optimization.

The selection of mesh size depends on the size and the shape of the design. Since there is no analytical equation to find the right mesh size, accordingly if an inappropriate mesh size is provided, the simulation fails. To address this, I provide an *auto-meshing* capability in *Anvil*, where the mesh size is selected automatically by capturing log errors from the meshing tool and by refining the mesh size accordingly to find the suitable mesh size. Some of the example simulations are shown below.

3.6.1 Example simulation experiments

I demonstrate the *Anvil*'s capability and highlight its features through different CFD evaluation experiments. The ease of doing CFD analysis is demonstrated by the work that needs to be done by the experimenter while using this tool. For a CFD analysis, the designer needs to provide the STL of the design and input configuration file for setting up the turbulence models and fluid parameters. An example configuration file is shown in figure 3.39.

1. **Unmanned Underwater Vehicle (UUV)**: To showcase the capability of the tool to simulate underwater designs just by proving a configuration file and the STL of the design, I ran our CFD evaluation tool with the following settings. The design is simulated at a speed of 2.25 miles/hr. The fluid density is of open sea density $1027.0 Kg/m^3$, the dynamic viscosity of the fluid is for open sea value $1.789 \times 10^{-5} N-s/m^2$, and the turbulence intensity is assumed to be 4%.

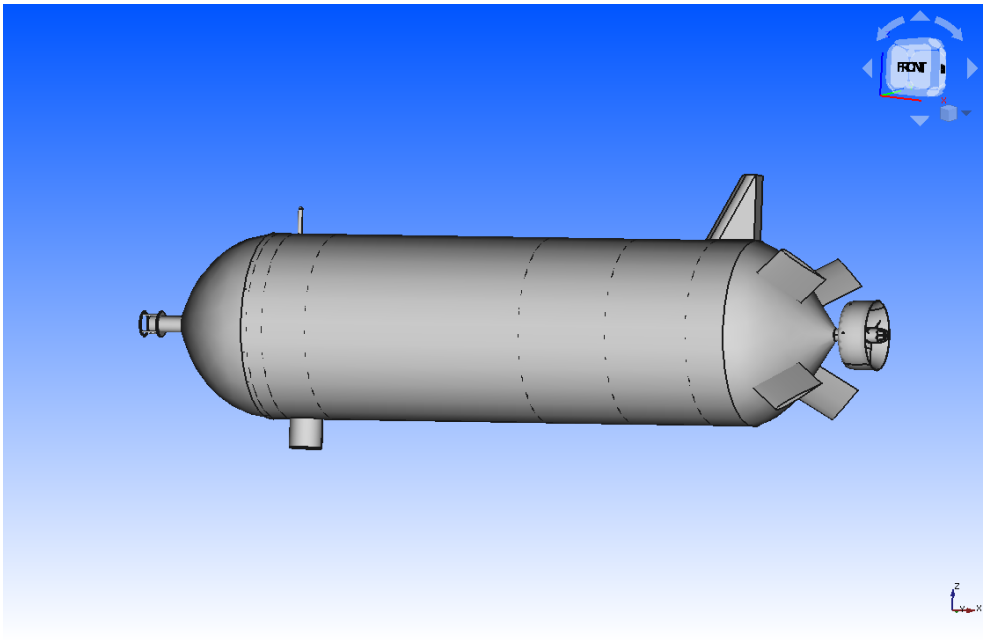
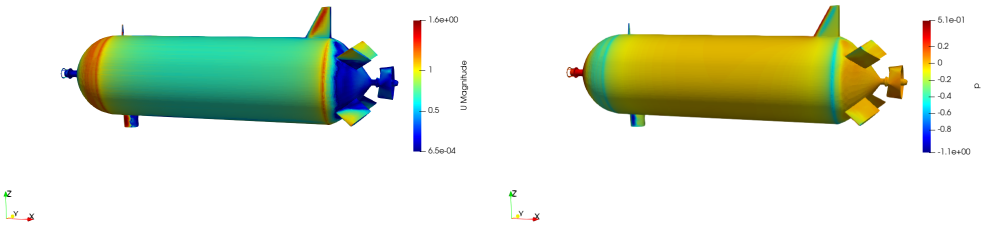


Figure 3.33: A concept UUV design in CAD environment



(a) (b)

Figure 3.34: Steady state flow at the surface layer of design, with the flow field colored by (a) magnitude of the velocity (meters/second), (b) pressure (Pascals).

2. **Land vehicle**: The CFD simulation of the land vehicle is carried on at a speed of 70 miles/hr. The fluid density is assigned sea-level air density $1.225 Kg/m^3$, with dynamic viscosity $1.789 \times 10^{-5} N-s/m^2$, and the empirical turbulence intensity to 1%.

3. **Unmanned Air Vehicle (UAV)**: UAV is Simulated at a speed of 50 meters/sec. The fluid density is the air density at 400 feet i.e. the flying height of most small UAVs $1.225 Kg/m^3$, dynamic viscosity at this altitude is $1.789 \times 10^{-5} N-s/m^2$, and the turbulence intensity is 1%.

3.7 Summary of contributions

In various works and experiments explained in this chapter, I have the following contributions:

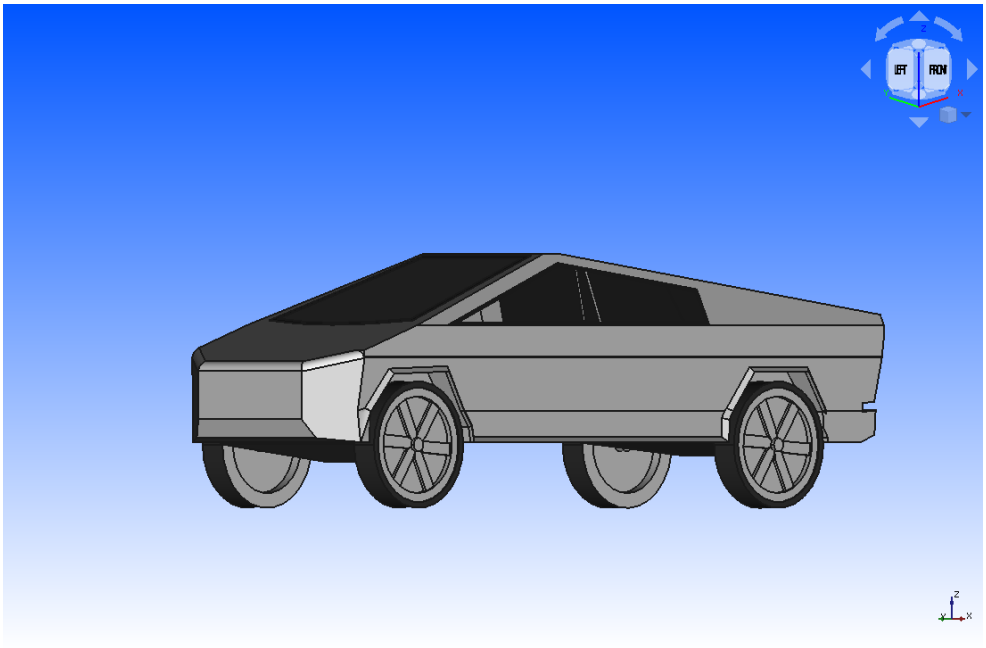
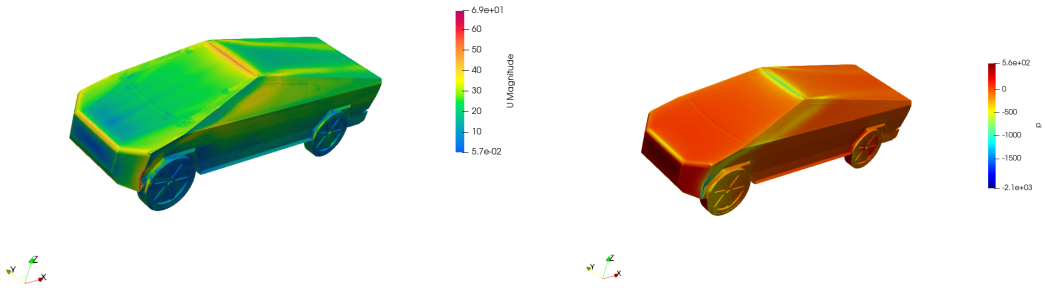


Figure 3.35: A concept model of a land vehicle.



(a) (b)

Figure 3.36: Steady state flow at the boundary layer of design, with the flow field colored by (a) magnitude of the velocity (meters/second), (b) pressure (Pascals).

1. I provide a comprehensive evaluation of a number of optimization methods on a real-world design optimization problem involving complex simulation domains, which is useful for practitioners considering adopting these algorithms. The empirical evaluation suggests that Bayesian optimization-Lower Confidence Bound (BO-LCB) has the best convergence guarantee and sample efficiency in comparison to other selected gradient-free optimization methods.

2. I also showed the way infeasibility constraint can be handled during Bayesian optimization and provide a software package to do it for UUV hull design optimization problem.

3. I proposed a hybrid optimization approach for speeding up the design optimization problems for computationally cheap but high-dimensional design space. I call it Surrogate Assisted Optimization (SAO),

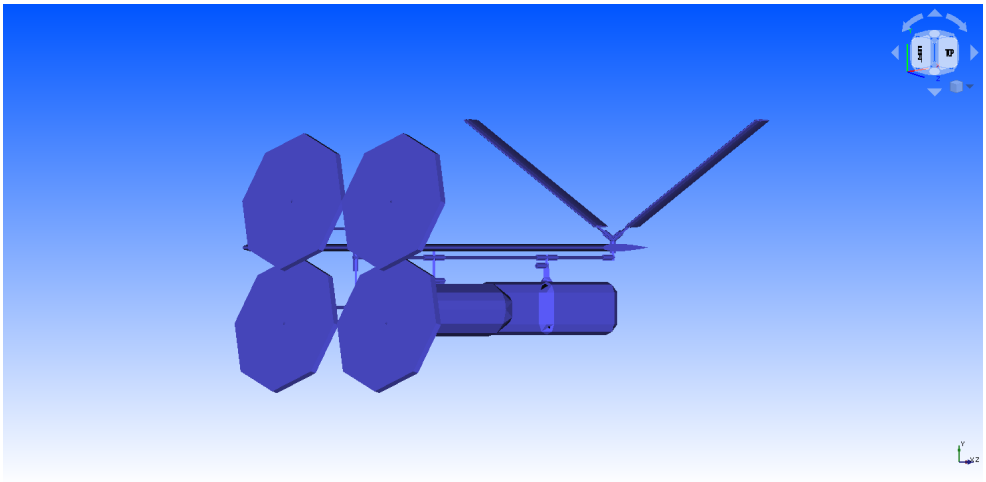
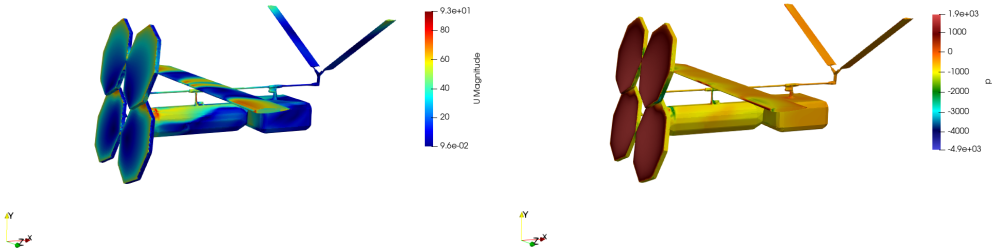


Figure 3.37: A concept unmanned air vehicle - Tiltie along with its cargo.



(a) (b)

Figure 3.38: Steady state flow at the boundary layer of design, with the flow field colored by (a) magnitude of the velocity (meters/second), (b) pressure (Pascals).

where AI model tries to learn the inverse mapping and at least provides good seed designs for further optimization.

4. I demonstrated that deep neural network architectures can accurately capture a complex property of two-way coupled solid-fluid dynamics (drag force/drag coefficient) as well as the effect of finite element analysis on pressure vessels to a very good agreement.

5. I showed that a surrogate-based design optimization algorithm can find optimal UUV hull designs in milliseconds, representing a two-orders-of-magnitude speedup over design optimization loops that use direct numerical simulation. Moreover, I am not aware of any previous studies that leverage an AI model for drag in the context of RANS with k-ω SST to optimize UUVs.

6. I provide a ready-to-use software package for drag-based design optimization for UUV design that can be readily generalized to other UUV designs.

```
{
  "seed_cad": "../usr_input/param_UUV_hull.FCStd",
  "cad_param": {
    "nose": 200,
    "first_y": 95,
    "second_y": 200,
    "third_y": 1000,
    ...
  },
  "foam_config": {
    "casefoldername": "UUVhull",
    "maxiter": 500,
    "infile": "./stl_repo/design.stl",
    "aoa": 0,
    "Uinlet": 20.0,
    "kinematic_viscosity": 0.000362,
    "kInlet": 0.06,
    "density": 1.225,
    "omegaInlet": 25.55,
    "meshing": "auto",
    "meshsize": 0.2
  },
  "design_space": {
    "nose": {
      "min": 100,
      "max": 800
    },
    "first_y": {
      "min": 5,
      "max": 100
    },
    "second_y": {
      "min": 5,
      "max": 100
    },
    "third_y": {
      "min": 5,
      "max": 100
    },
    ...
  },

  "mode": "data_generation",
  "sampling_method": "random",
  "budget": 8,
  "optimizer": {
    "method": "BayesOpt",
    "acquisition": "LCB"
  }
}
```

Figure 3.39: A configuration for a CFD evaluation of a design using *Anvil*. The parameters of the configuration parameters can be changed and the valid values of these fields are given in table 3.7

7. I also conducted preliminary research in search of a universally optimal UUV hull shape design that is near optimal for a range of environmental and operating ranges. Our preliminary study suggests the possibility of the existence of such a design.

8. Finally, I presented a SciML tool called *Anvil*, which is the integration of AI methods and CAD-CFD in unison that can be used for data generation or design optimization for engineering or research in shape optimization of designs and study the effect of solid-fluid dynamics for any subsonic flow conditions (underwater, land, or air). with very little work by the user.

Configurable parameter	sub-parameter	Options for parameter	*Meaning*
seed_cad			*Location of parametric CAD seed design along with CAD file name*
cad_param			*Name of the parameters that can be changed in CAD seed design and its default values*
foam_config	casefoldername		*Name of folder from where CFD runs*
	maxiter	$Z+$	*Maximum CFD simulation steps*
	subdomians	{2,4,8,16,32}	*Number of cores to use for CFD*
	aoa	R	*Angle of attack of vehicle*
	Uinlet	$R+$	*Velocity of vehicle*
	kinematic_viscosity	$R+$	*Fluid viscous property*
	kInlet	$R+$	*Inlet turbulent energy*
	density	$R+$	*Fluid's density*
	omegaInlet	$R+$	*Turbulence dissipation rate*
	meshing	{auto,self}	*Mesh size selection process*
	meshsize	$R+$	*Size of the mesh when meshing is "self"*
design_space			*Ranges (max and min) of design parameters that creates a design space*
mode		data_generation optimization cfd_eval	*Generate and store sim data* *Optimize design for the target* *Single CFD evaluation on given STL*
sampling_method		random lhc_minmaxcorr lhc_maximin	*Uniform random sampling* *Latin hypercube with minimize the maximum correlation coefficient* *Latin hypercube with maximize the minimum distance between points*
budget		positive integer ($Z+$)	*Number of maximum CFD evaluation*
optimizer	method	BayesOpt	*Bayesian optimization*
	acquisition	LCB EI	*Lowest Confidence Bound* *Expected Improvement*
	target	lift drag lift_drag_ratio	*Maximize lift force* *Minimize the drag force* *Maximize the lift to drag ratio*

Table 3.7: Details of configurable options in *Anvil*.

CHAPTER 4

Data efficient surrogate modeling for engineering design: DeepAL for regression problem

4.1 Problem formulation

Various works explained in chapter 2 show that it is possible to capture complex behavior using deep learning-based surrogate models, and once a data-driven surrogate is created, it speeds up the design optimization process by multiple folds. Still, there are many hurdles that hinder surrogate modeling, especially in the complex engineering design domain. The main challenge in creating an inexpensivedata-driven surrogate is the generation of a sheer number of data using these computationally expensivenumerical simulations. In such cases, Active Learning (AL) methods have been used to attempt to learnan input-output behavior while labeling the fewest samples possible. The current trend in AL for a regressionproblem is dominated by the Bayesian framework that needs training an ensemble of learning models thatmakes surrogate training computationally tedious if the underlying learning model is Deep Neural Networks(DNNs). However, DNNs have an excellent capability to learn highly nonlinear and complex relationshipseven for a very high-dimensional problem. Our interest is to leverage the excellent learning capability of deep networks while avoiding the computational complexity of the Bayesian paradigm so that a simple and scalable approach can be developed to create a surrogate model. Merging a deep learning model directly with AL outside of the Bayesian framework is difficult. The underlying reason is the lack of ways to design a query strategy for prediction. For a classification problem, researchers benefit from a sigmoid-based output probability distribution and can leverage Shannon's entropy function [133] to design a query strategy for active sampling (refer to figure 4.1), however, this aspect is missing in the case of the regression problem.

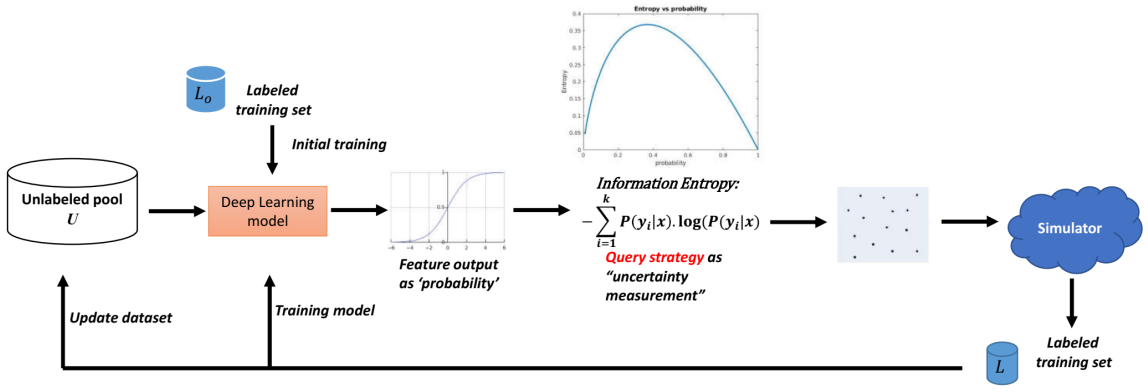


Figure 4.1: Deep active learning for logistic regression problem

For regression problems, due to the lack of any probability measure, the researchers resort to an ensemble

of models to measure uncertainty indirectly. For this purpose, they use models like the Bayesian neural network or Gaussian processes (GP). Bayesian neural network methods are expensive due to the training of a family of neural networks. In such a scenario, I lose the gain from active learning in terms of training time for a family of neural networks because ensemble methods require independent training of several models, which is very time-consuming and even intractable when I am dealing with DNNs. The Gaussian processes, on the other hand, have $O(n^3)$ time complexity during both training and prediction time. In such cases, using GP for even thousands of samples becomes computationally intractable. Another challenge arises from the active learning paradigm. AL framework works on single samples per iteration, while retraining a family of neural networks on just one added sample is not feasible. In this work, an attempt is made to solve this problem out of the Bayesian framework and ensemble models to make it scalable and practically useful for a large class of problems. To that end, I will first formalize the problem.

Let

$$y = f(x), x \in X \subset \mathbb{R}^n, y \in Y \subset \mathbb{R}^m \tag{4.1}$$

be an unknown function for which I want a surrogate approximation using the training data generated by a high-fidelity simulation process. Here X stands for the unlabeled data set in the design space DS and Y for the solution set that the simulation process evaluated. Accordingly, in the given DS, I view the simulator as a nonlinear function that maps the input space to the solution space ($simulator : X \mapsto Y$) and whose behavior I want to model using a k–parameter DNN (NN). For data-driven surrogate modeling, the first step is the generation of training data.

$$D^{train} = (x_j^{train}, y_j^{train}),\ j = 1, 2, ..., N_{train} \tag{4.2}$$

Since most engineering simulators have scalar outputs, in this work, I am interested in regression problems. The training goal is to minimize a loss function $l()$ by using D^{train} along with getting acceptable prediction accuracy on test data D^{test}. As some numerical simulation processes are computationally very costly, it is necessary to be strategic during the training data generation process to achieve acceptable accuracy with a minimal number of training data points. This problem can be formalized as:

$$\underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{(x,y)}[l(NN(x,\theta|D^{train})] : D^{train} \subseteq DS, (x,y) \in D^{train} \tag{4.3}$$

$$min\ N_{train} \tag{4.4}$$

$$subject\ to\ \ NN(x,\theta) = y, \forall x \in DS : y \approx y^* \tag{4.5}$$

Instead of a given training data set (X) and its label (Y), I assume that I don't have any data a priori and the designer has to select samples for labeling. To this end, I work in a pool-based setting, where unlabeled pool data (U) is given that contains a set of candidate points in DS for simulation. The heuristic-based research field called AL, if used, tries to choose the most informative sample for evaluation on which I want to run our simulations.

$$U = \{x_j, j = 1,2,...,N_{pool}\}\ , U \subset DS$$

As a subfield of machine learning, AL has been well-studied, and if deployed for training a learning model attempts to do it by evaluating the fewest samples possible. For such a purpose, AL methods rely on an acquisition function (A) which computes a scalar score ($s \in \mathbb{R}^+$) for a trained state of the model and unlabeled pool data (U).

$$A(NN, U) : U \mapsto \mathbb{R}^+ \tag{4.6}$$

The acquisition function ranks the points in U, which indirectly measures the utility of data points for training the surrogate. The unlabeled candidate data point with a maximum score is most appealing for maximizing the model's performance gain in the next iteration of model training, and vice versa. Therefore, the design of the acquisition function is crucial to the performance of AL methods. For a classification problem, the output of regression is a relative probability. In such a case, Shannon's entropy function [133] can be used to measure the uncertainty measurement of samples and can guide the sampling in the next iteration:

$$A = -\sum_{i=1}^{k} p(y_i|x).log(p(y_i|x)) : p(y_i|x) = NN(x)\ \forall\ x\ \in U \tag{4.7}$$

Here k is the number of possible output classes of an input sample. At each iteration after training, the output likelihood of each sample is estimated using a trained surrogate. The acquisition value is then estimated using these probability measurements and equation 4.7. Based on these acquisition function values, samples are selected from the unlabeled pool for the next iteration of training. However, this approach does not work for a regression problem, which is our interest. In a regression problem, output prediction on an unlabeled sample using a trained surrogate is a scalar value (in the case of a single output) or a vector of scalars (in the case of a multi-output). In such cases, researchers took the route of measuring uncertainty in prediction. A principal way to measure uncertainty in regression problems is to use Bayesian inference. For this purpose, I start with an informed or uninformed prior and compute the posterior distribution over the model parameters.

$$p(\theta|D^{train}) \propto p(\theta) \prod_{i=1}^{N_{train}} p(y_i|x_i, \theta)\ where\ (x,y) \in D^{train} \tag{4.8}$$

Then I compute the posterior predictive distribution $p(y|x,D^{train}) = \int p(y|x,\theta)p(\theta|D^{train})d\theta$ for each test point $x \in U$. The full Bayesian inference (integration) is not computationally tractable for high-dimensional parameter space on the entire distribution $p(\theta|D^{train})$. For computational reasons, the common approach is to approximate the posterior distribution by various point estimates and ensemble their outcomes to replace the integral to summation. Another alternative and indirect approach to measuring uncertainty is an ensemble of predictors trained on a separate subset of data. Both scenarios require the training of multiple networks. Given a family of neural networks NN_i for $i = \{1,2,...,n\}$, and the prediction made on an unlabeled sample (U) after an iteration of training is $p_i = NN_i(x)$ for $i = \{1,2,...,n\}$, an acquisition function is created using the prediction as a function of mean and variance measurement and given as:

$$A = f(\mu_i, \sigma_i) : \mu_i = \frac{1}{n}\sum_{i=1}^{n} p_i \;\&\; \sigma_i^2 = \frac{1}{n}\sum_{i=1}^{n}(x-\mu_i)^2 \;\forall x \in U \tag{4.9}$$

Some example functions are maximum variance, Expected Improvement, etc. By estimating the acquisition values on U, new samples for labeling are selected for the next iteration of training. The problem with this approach is to train multiple neural networks. In complex behavior modeling, training even one network takes a very long time. Training a family of neural networks in such a scenario is computationally tedious. I propose a different approach for deep active learning in a regression problem to avoid this situation. The proposed approach is simple, scalable, and called *student-teacher-based surrogate modeling*. I discuss it in the next section.

4.2 Proposed solution

In this section, I will present our proposed solution for the above-mentioned problem.

4.2.1 Student-teacher based surrogate modeling

For designing the acquisition function (A), I propose to use two networks (one regressor and another classifier). These networks are called student and teacher networks based on their abstract purposes. The goal of the student network is to learn the simulator's behavior ($student : X \mapsto Y$), while the goal of the teacher network is to guide the sampling and labeling for the next iteration of training based on the performance of the student network. Active learning is an iterative approach, and at each iteration, I train teacher and student networks. For training the student network, I first split the already labeled data ($X_{labeled}, Y_{labeled}$) into train and test data. After the completion of training the student network, I create a custom dataset based on the evaluation of the student network on already labeled training and test data. For evaluation purposes, I first

define a threshold for fractional error called acceptable accuracy ($aa : aa \mapsto [0,1]$).

$$fractional\ error = \frac{|Y_{labeled} - Y_{predicted}|}{|Y_{labeled}|} \tag{4.10}$$

The threshold value (aa) is a user choice that indicates the permissible maximum percentage error in predictions made by a trained student network. I declare predictions made by the student to be accurate when the fractional error is less than aa and vice versa. I chose the value of $aa = 0.05$ ($\pm 5\%$ error) for all experiments. By using this evaluation report, I label each sample with a binary failure indicator (F), where $F : X_{labeled} \mapsto \{0,1\}$, where $F = 0$, indicates the student network is able to predict the sample within an acceptable level of accuracy; otherwise, $F = 1$ (in the case of a multi-output regression problem, I can use logical and on all outputs to indicate a failure).

$$D_{labeled} = \{x_i, \mathbb{I}(y_i - y_i^*) \geq aa * y_i^*\}\ \forall\ x_i\ in\ X_{labeled}, where\ y_i = prediction, y_i^* : true\ label \tag{4.11}$$

Using the labeled binary failure outcome data ($D_{labeled}$), the teacher network is trained to learn a failure probability distribution of the student on the entire design space (DS). Due to the $0/1$ prediction, this problem is converted into a classification problem. The failure probability guides the sampling in the next iteration to the regions in DS on which the student network has poor performance or a high likelihood of failing. At each iteration of learning, I retrain the student and teacher networks on further evaluated samples at each iteration until I exhaust our budget for iterations (refer to figure 4.2).

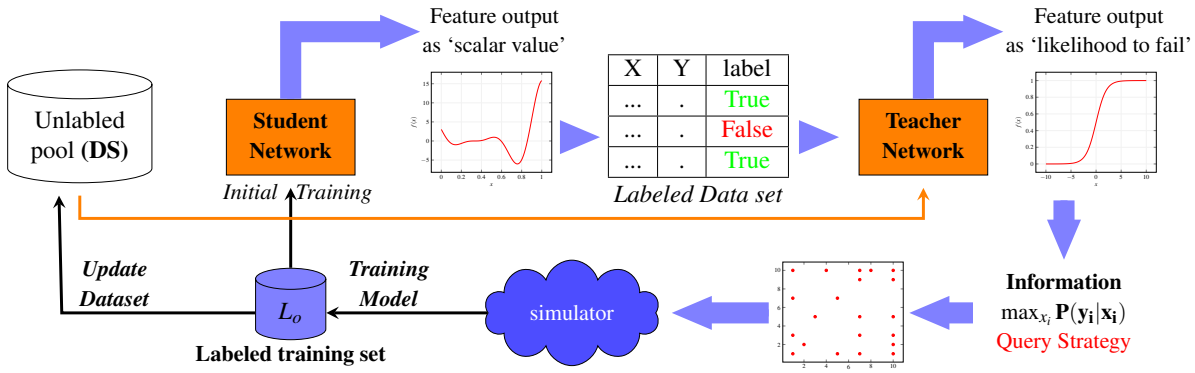


Figure 4.2: Student-teacher architecture: the process

Due to the classification job of a teacher, I used a smaller teacher neural network in comparison to a student neural network because creating a coarse approximation of the decision boundary is a simpler job than creating a nonlinear regression manifold in a given design domain. By leveraging the GPU implementation of a neural network, I can make inferences on thousands of samples at a cost negligible to running the costly

simulation. With the knowledge of the failure probability of the student network, it is possible to identify at each iteration, what are the samples $x \in DS$ on which the student network has the highest likelihood to fail.

$$S^* = \underset{x}{\operatorname{argmax}}\ f_t(x)\ \forall x \in U \tag{4.12}$$

4.2.2 Batching

As mentioned above, the AL-based approach uses one-by-one sample selection, which leads to the retraining of the neural network with a minimal change in the training data. There are two problems with this: first, retraining the entire network with one added sample will take a lot of training time and ruin the benefits gained from AL. Second, frequent retraining without adding much information to the data can also easily lead to overfitting [117]. To address this, at each active learning iteration, I want to select a batch of samples instead of only one sample. For this purpose, I score a batch of candidate unlabeled data samples $B = \{x_1, x_2, ..., x_b\} \subseteq U$. Based on the acquisition function A that is derived by a trained teacher, the goal is to select a batch of data samples $B^* = \{x_1^*, x_2^*, ..., x_b^*\}$ at each active learning iteration, which can be formulated as

$$B^* = \underset{B \subseteq U}{\operatorname{argmax}}\ P(B, f_t(x)) \tag{4.13}$$

Here P is a policy or algorithm or heuristic that chooses a batch of samples. In this work, I explored the following policies for batching drawn from current research in AL :

1. Top-b
2. Diverse Batched Active Learning (DBAL)
3. Epsilon(ε)-weighted Hybrid Query Strategy (ε-HQS)
4. Batched random

Now I will explain these policies of batching and its algorithms in detail.

4.2.2.1 Top-b

It is a greedy strategy, here b is the batch size at each iteration. In this approach, after evaluating the failure probability score on all unlabeled samples U, top-b samples are selected that have the maximum probability of failure. The selection policy, in this case, is defined as:

$$B^*(P = \text{top–b}) = \underset{B \subset U, |B| = b}{\operatorname{argmax}}\ \Sigma\ f_t(x)\ ;\ x \in U \tag{4.14}$$

This is a naive approach to selecting a batch of samples, and this method considers each sample independently. In such a situation, it may select highly information-rich but similar samples that collectively do not add much information to the learning process. Without considering the correlation between the samples, it may result in a waste of the evaluation budget and make the batch samples insufficiently optimized.The goal of the most optimal batch sampling should be to add the most collective information to the learning process. For this purpose, the goal is to select information-rich but diverse samples. The pseudo-code of the **top-b** algorithm is given in the algorithm 1.

Algorithm 1 Top-b in student-teacher setting

Require: student network $f_s(x,\theta)$, teacher network $f_t(x,w)$, unlabeled sample pool U, number of initial samples M, number of AL iterations T, acceptable accuracy aa, and batch size b
Create training data for surrogate: $S \leftarrow M$ examples drawn uniformly randomly from U and label it.
Train the student model to get θ_1 on S by minimizing $E_S[l_{MAE}(f_s(x;\theta),y)]$ ▷ mean absolute error loss
Create labeled dataset: $D_{labeled} = \{x_i, \mathbb{I}(y_i - y_i^*) \geq aa \times y_i\} \; \forall \; x_i \in S$
Train the teacher model to get w_1 on $D_{labeled}$ by minimizing $E_{D_{labeled}}[l_{CE}(f_t(x;w),y)]$ ▷ binary cross entropy error loss
for $t = 1,2,...,T$: **do**
    For all examples x in $U \setminus S$:
    1. Compute its failure probability $p(x) = f_t(x)$
    2. Select set S_t such that $\underset{S_t \subset U \setminus S, |S_t| = b}{\text{argmax}} \sum p(x)$ and label it.
    $S \leftarrow S \bigcup S_t$
    Train the student model to get θ_{t+1} on S by minimizing $E_S[l_{MAE}(f_s(x;\theta_t),y)]$
    Create labeled dataset: $D_{labeled} = \{x_i, \mathbb{I}(y_i - y_i^*) \geq aa \times y_i\} \; \forall \; x_i \in S$
    Train the teacher model to get w_{t+1} on S by minimizing $E_{D_{labeled}}[l_{CE}(f_s(x;w_t),y)]$
end for
return surrogate student model and its weights θ_{T+1}

4.2.2.2 Diverse Batched Active Learning (DBAL)

The general approach to imparting the maximum collective information in AL literature is to include diversity in the sample selection. Wei et al. [162] include diversity by formulating a submodular function on the distances between samples and selecting a batch of unlabeled samples, which optimizes the submodular function. Another approach to including diversity is attempted [129] as a core-set selection problem, i.e., choosing a set of points such that a model learned over the selected subset is competitive for the remaining data points. For this purpose, they defined a loss function called core-set loss, which is the difference between the average empirical loss over the labeled set of points and the average empirical loss over the entire data set, including unlabeled points. Since I do not have access to all the labels, the core-set loss reduction is not directly computable; consequently, [129] gave an upper bound for this objective function, which I can optimize. This upper bound for the loss function of the core-set selection problem is optimized by minimizing its equivalent to the k-Center problem (minimax facility location [165]). A similar approach is taken by

Zhdanov [172], who proposed the batch active learning using the clustering approach and proposed to solve it as a facility location problem. Since the direct solution to the facility location problem is NP-hard, the alternative formulation using the k-center/k-medoid algorithm is used in both [129, 172]. I took the Zhdanov [172] formulation to include diversity in the batch sample selection. K-mean cluster minimizes the below objective function to find the center of cluster μ_b and the cluster assignment $z_{i,b}$.

$$\sum_{x_i \in U} \sum_{b} z_{i,b} ||x_i - \mu_b||^2 \tag{4.15}$$

This clustering approach is further given informativeness about its probability to fail, i.e., failure probability estimation $p(x) : p(x) \in [0,1]$ for every unlabeled sample in U. This informativeness gives the weightage of different samples in clustering, and the modified objective for the minimization problem is defined as:

$$\sum_{x_i \in U} \sum_{b} z_{i,b} p_i(x) ||x_i - \mu_b||^2 \tag{4.16}$$

This is solved by a weighted K-means clustering algorithm. In order to improve the scalability of this approach, a pre-filtering process is adopted to select βb most informative samples before clustering. The good choice of β would depend upon data and batch size. For this experiment, I selected the same β values ($\beta = 10$ & 50) that are used in Zhdanov [172] and called it DBAL-10 and DBAL-50. The complete pseudo-algorithm is given in algorithm 2.

4.2.2.3 Epsilon-weighted Hybrid Query Strategy (ε-HQS)

One new batched sample selection policy is proposed in this work, which capitalizes on the solution proposed to solve the exploration versus exploitation dilemma in reinforcement learning literature [139, 151]; I call this ε-HQS. The role of a teacher in the student-teacher framework is to estimate the performance of the trained surrogate in the design space ($f_t : x \mapsto [0,1] \; \forall \; x \; \in DS$) by providing a failure estimate. At each iteration of training, I want to estimate the following:

$$f^*(x) = \mathbb{P}(f_s(x) \geq aa * y^*), \;\; x \in DS, y^* = true \; label \tag{4.17}$$

Here f^* is the true failure probability of the student surrogates in the design space (DS) at the end of training. However, since I do not have true labels (y^*) for all unlabeled points in U. In such a case, the goal of a teacher is to learn an approximation $f_t \approx f^*$. I want this approximation to be as accurate as possible. However, at each iteration of training, I have a limited amount of labeled data, consequently, $D_{labeled}$ has limited information

Algorithm 2 DBAL in student-teacher setting

Require: student; network $f_s(x,\theta)$, teacher network $f_t(x,w)$, unlabeled sample pool U, number of initial samples M, number of AL iterations T, batch size b, acceptable accuracy aa, and parameter β
Create training data for surrogate: $S \leftarrow M$ examples drawn uniformly randomly from U and label it.
Train the student model to get θ_1 on S by minimizing $E_S[l_{MAE}(f_s(x;\theta),y)]$ ▷ mean absolute error loss
Create labeled dataset: $D_{labeled} = \{x_i, \mathbb{I}(y_i - y_i^*) \geq aa \times y_i\} \; \forall \, x_i \in S$
Train the teacher model to get w_1 on $D_{labeled}$ by minimizing $E_{D_{labeled}}[l_{CE}(f_t(x;w),y)]$ ▷ binary cross entropy error loss
for $t = 1, 2, ..., T$: **do**
    For all examples x in $U \setminus S$:
    1. Compute its failure probability $p(x) = f_t(x)$
    2. pre-filter the top βb informative samples (I_s): $\underset{I_s \subset U \setminus S, |I_s| = \beta b}{\text{argmax}} \; \Sigma \, p(x)$
    3. Create b clusters on I_s using weighted K-Means: $\sum_{x_i \in U} \sum_{b} z_{i,b} p_i(x) ||x_i - \mu_b||^2$
    4. S_t =select b samples closest to the cluster centers and label it.
    $S \leftarrow S \bigcup S_t$
    Train the student model to get θ_{t+1} on S by minimizing $E_S[l_{MAE}(f_s(x;\theta_t),y)]$
    Create labeled dataset: $D_{labeled} = \{x_i, \mathbb{I}(y_i - y_i^*) \geq aa \times y_i\} \; \forall \, x_i \in S$
    Train the teacher model to get w_{t+1} on S by minimizing $E_{D_{labeled}}[l_{CE}(f_s(x;w_t),y)]$
end for
return surrogate student model and its weights θ_{T+1}

about the design space *DS* (a maximum up to training and testing data).

$$f_t(x) = \mathbb{P}(f_s(x,w)|D_{labeled}), \;\; \forall \, x \in DS \tag{4.18}$$

In such a case, I can only approximate f^* up to some relative accuracy (ρ approximation of f^*). It results in a biased estimation of the performance of the student network by the teacher network. The estimation or bound on ρ is an open research question, but I can increase the robustness of the teacher network and reduce the sensitivity of mismatches between the distributions of f_t and f^*. For this purpose, I introduce a two-step process: first, I filter all samples with the teacher's prediction (likelihood to fail) more than a threshold value (samples with a high probability of failure), and second, I introduce a belief weightage on the teacher network. For this end, I introduce two hyperparameters: threshold and ε. Since the output layer of the teacher network is a sigmoid function, I kept a standard threshold of 0.5 for all experiments (i.e. select all x if $f_t(x) \geq 0.5 \; \forall \, x \in DS$). ε is a belief factor that I assign to the teacher network. ε is a scalar real value ranging between 0 (indicates complete disbelief in the teacher network's estimate) and 1 (indicates a complete belief in the teacher's estimation) ($\varepsilon \in [0,1]$).

ε-greedy based policy is inspired by RL [139] literature, the ε factor controls the balance between exploration and exploitation. Since the teacher network does not know what it does not know - the explo-

ration versus exploitation dilemma exists in this situation similar to RL, i.e., the search for a balance between exploring the design space to find regions where I have not explored while exploiting the knowledge gained so far by the teacher network. The fixed value of ε expresses linear belief in the teacher's prediction. Another famous approach is to let ε go to one with a certain rate to increase our belief in the teacher with more and more labeled data available for its training in later iterations. It turns out that at a rate of $\frac{1}{T}$(T = number of training iterations) proved to have a logarithmic bound on the regret (maximum gain) [9]. ε-greedy rule [139] is a simple and well-known policy for the bandit problem.At each iteration of active sampling, this policy selects a $\varepsilon \times b$ number of samples from the filtered unlabeled data (samples that have a likelihood to fail more than the threshold), and the rest of the samples, $(1-\varepsilon) \times b$ are selected uniformly randomly from leftover unlabeled samples. The complete pseudo-code for this approach is given in the algorithm 3.

Algorithm 3 ε-HQS

Require: student network $f_s(,\theta)$; teacher network $f_t(x,w)$; unlabeled sample pool U; number of initial samples M; number of AL iterations T; batch size b, acceptable accuracy aa, hyper-parameter ε
Create labeled data: $S \leftarrow M$ examples drawn uniformly randomly from U and label it.
Train the student model to get θ_1 on S by minimizing $E_S[l_{MAE}(f_s(x;\theta),y)]$
Create labeled data: $D_{labeled} = \{x_i, \mathbb{I}(y_i - y_i^*) \geq aa \times y_i\} \; \forall \, x_i \in S$
Train the teacher model to get w_1 on $D_{labeled}$ by minimizing $E_{D_{labeled}}[l_{CE}(f_s(x;w),y)]$
for $t = 1,2,...,T$: **do**
    For all examples x in $U \setminus S$:
    1. Compute its failure probability $p(x) = f_t(x)$
    2. $X_f = \{x \mid p(x) \geq 0.5\}$ ▷ samples with high failure probability
    3. $X_{sel} = Choose\ (\varepsilon \times b)\ number\ of\ samples\ uniformly\ randomly\ from\ X_f$.
    4. $S \leftarrow S + X_{sel}$
    5. $X_{rest} = Choose\ ((1-\varepsilon) \times b)\ number\ of\ samples\ uniformly\ randomly\ from\ U \setminus X_{sel}$
    6. $S \leftarrow S + X_{rest}$
    $S \leftarrow S \bigcup S_t$
    Train the student model to get θ_{t+1} on S by minimizing $E_S[l_{MAE}(f_s(x;\theta_t),y)]$
    Create labeled dataset: $D_{labeled} = \{x_i, \mathbb{I}(y_i - y_i^*) \geq aa \times y_i\} \; \forall \, x_i \in S$
    Train the teacher model to get w_{t+1} on S by minimizing $E_{D_{labeled}}[l_{CE}(f_s(x;w_t),y)]$
end for
return surrogate student model and its weights θ_{T+1}

4.2.2.4 Batched random sampling

This algorithm samples a batch of random samples B from the unlabeled pool U and labels them at each iteration of training. The probability distribution for the random sample selection is assumed to be uniformly

distributed within the interval $[x_l, x_h)$ and given by:

$$p(x) = \frac{1}{x_h - x_l} \tag{4.19}$$

Where x_h and x_l are the highest and lowest values in each dimension of the design space. This is the simple batched sampling approach and has a computational complexity of $O(1)$.

In the next section, I empirically evaluate and compare all the above-mentioned methods to train neural network-based surrogate models in three different engineering domains.

4.3 Experiments and its evaluation

4.3.1 Experimental setup

In each experiment, I follow the same procedure as explained below. Since I am interested in a pool-based setting where I assume that I already have collected a pool of unlabeled input data (U) and during the selection process of data for training, I remove the budgeted data per iteration (b) from the pool and label it. For a given total number of iterations of the active learning (T), the entire procedure is outlined below:

1. Warm-up stage: Selection of initial data set (selected randomly from the pool) and getting its label to create a training data D_{train}. Initialize the training weights, bias, and learning hyperparameters. Initialize AL iteration counter t.

2. For each iteration of the AL:

   While $t \leq T$:

   (a) The student network is trained on D_{train}. After completion of training, the train and test data is evaluated on the trained student model.

   (b) The labeled data set is created and used to train the teacher network.

   (c) The score/ failure probability is computed using the trained teacher network on leftover pool data U.

   (d) The query policy is deployed on the computed failure probability to select the new batch of candidate samples ($S_{candidate}$) for training in the next iteration.

   (e) Exclude $S_{candidate}$ from corresponding $U : U := U \setminus S_{candidate}$

   (f) Get the label (L) for $S_{candidate}$ and create newly added data. $D_{new} =< S_{candidate}, L_{candidate} >$

   (g) Add D_{new} to D_{train}: $D_{train} := D_{train} \bigcup D_{new}$

For evaluating the performance of the trained student surrogate, I used the left-over data ($U \setminus D_{train}$) as our test data and evaluated the student surrogate after each iteration of active learning. To rigorously evaluate our approach and to ensure performance consistency, I ran each algorithm multiple times (every time with different random seeds) and averaged their results.

4.3.2 Metrics

For testing the performance of the trained surrogate model, I chose the evaluation metric as the fraction of test predictions that are within acceptable accuracy, I defined a regression output as acceptable if it is within 5% of error (+/- 5% error) i.e.,

$$Accuracy = \frac{\sum_{i=1}^{N_{test}} \mathbb{I}(|y_i - y_i^*| \leq 0.05 * |y_i^*|)}{N_{test}} \tag{4.20}$$

Here y_i is a prediction on i^{th} test sample with ground truth y_i^* and N_{test} is the total number of samples in the test data. Since other accuracy metrics like MSE/MAE do not explain whether the error value is due to a generalization error or outliers, our chosen metric gives a better understanding of prediction results. During the surrogate training, I choose $L1$ loss (MAE) as our optimization objective for the student network. I treated the optimization objective as a hyper-parameter, tested both $L1$ loss (MAE) and $L2$ loss (MSE), and observed better performance on test data using MAE. For training the teacher network, our loss function was binary cross-entropy, since the teacher's goal is to create a pass-fail (0-1) probabilistic map.

4.3.3 Intuitive explanation and validation of ε-HQS approach in 2D problem

To develop an intuitive understanding of this approach, I test our method on a low-dimensional example problem. In a low-dimensional problem, I can visualize the learning process and understand the roles of the teacher and student during surrogate training. For this purpose, I take a classic static regression problem as our example that comprises two input variables and one output scalar value. This function is called the bird function [97] and is defined as:

$$f(x,y) = sin(x)e^{(1-cos(y))^2} + cos(y)e^{(1-sin(x))^2} + (x-y)^2 \tag{4.21}$$

$$-10 \leq x \leq 0\,; -6.5 \leq y \leq 0 \tag{4.22}$$

The input ranges that make up the design space of interest are a subset of $\mathbb{R}^2$. Figure 4.3 shows the multi-modal non-linear manifold created by this function. The surrogate training goal is to learn this nonlinear manifold using a deep neural network. For this purpose, I apply the ε-HQS method as explained above and observe the evolution of the learning process after active learning iterations. Since the function's evaluation

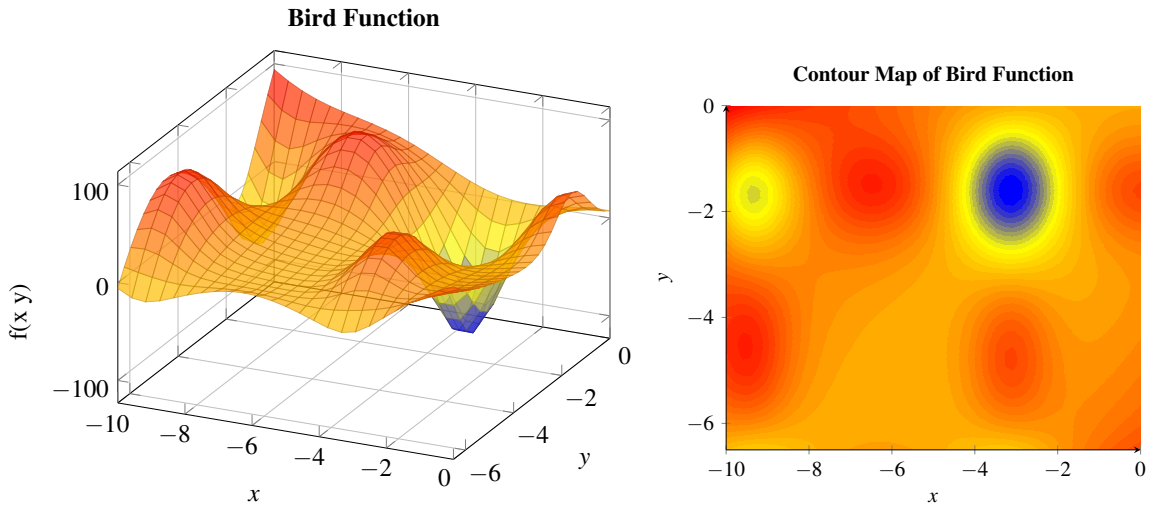


Figure 4.3: The two variables bird function (manifold-on left) and contour map (on right).

is very cheap, I first created dense mesh grid samples in design space with 10000 samples and evaluated these samples on the bird function to create a labeled mesh grid data {input samples, output label}(X_m, Y_m). I started the surrogate training using the ε-HQS process by sampling budgeted initial random samples from the input space, evaluating its output using the bird function, and training the student network. Then the labeled dataset ($D_{labeled}$) is created to train the teacher network (refer algorithm 3 for more information). Once the training phase (for both teacher and student networks) is finished for an iteration, I evaluate the mesh grid sample data (X_m) on the student network (f_s) and compare it with the ground truth (Y_m). If $|f_s(X_m) - Y_m|$ is within acceptable accuracy, I label it as 1 or else 0. This label indicates the mesh grid samples that the student network has failed to predict with acceptable accuracy. The goal of the teacher network (f_t) is to discover this failed design space, which guides the sampling process in the next iteration. To find out whether the teacher network is able to guide the student network in the right region of its weakness or failure, I make a prediction using the teacher network on the same (X_m) and observe the failure probability for each sample ($f_t(X_m)$). For evaluating teacher prediction, I keep the threshold at 0.5 which is generally the case in a binary classification problem. The failure probability greater than 0.5 indicates the teacher's belief in regions on *DS* on which I have a high probability to fail and vice versa. The performance measured on the mesh grid gives us a visual explanation and evaluates the teacher's ability to predict the regions where the student is not performing well, which can guide the sampling in the next iteration. I show the results of this learning and evaluation process in figure 4.4. The left column shows the performance of the student on the mesh grid data at the end of the labeled iteration, and the right column shows the performance of the teacher on the same mesh grid data in the corresponding iteration (red color in figure 4.4 indicates failure, and blue is a success).

As can be observed from the figure, using the student-teacher approach with the ϵ-HQS method, the teacher is able to estimate the design space region on which the student has a high probability of failure with good consistency. The difference in the exact failure prediction and the teacher's prediction is due to the limited labeled sampled information for training the teacher (up to the training and test data). But this approximated failure prediction by the teacher (red region) is good enough to guide sampling for student training in the next iteration.

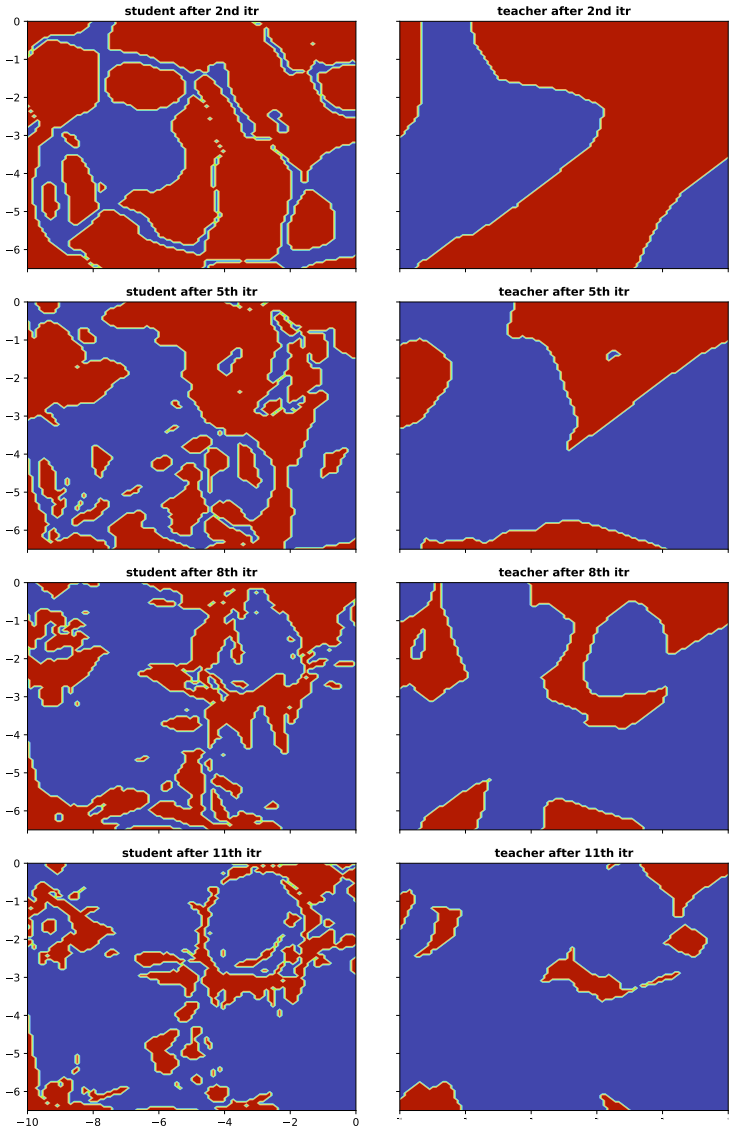


Figure 4.4: The ground truth of trained student network in the design space (shows prediction performance on mesh grid data - red: fail to predict within acceptable accuracy, blue: able to predict within acceptable accuracy) and its failure estimated by the teacher network at end of different AL iterations. left column: ground truth of student network's performance; right column: student network's performance estimated or approximated by teacher network on mesh grid data.

4.3.4 Empirical evaluation in different Engineering design domains

I take three different use cases of real-world engineering problems (multi-input, single-output regression problems) of surrogate modeling. The pooled data is accumulated either by generating data from the available open-source simulators or from already-collected baseline data from prior works. In the Finite Element Analysis (FEA) domain, I took the data generated by Vardhan et al. [150]. It uses FreeCAD [119] which is a toolbox for parametric 3D CAD designs and has numerical solvers like CalculiX for the FEA analysis. In the propeller design domain, I used the pool data generated by [156]. It uses the openProp numerical simulation tool for designing a propeller. Openprop [40] is a computational tool for designing and analyzing marine propellers and horizontal-axis turbines based on the vortex lattice lifting line methods. In the CFD domain, data generation is done using an integrated tool chain consisting of freeCAD [119] (for CAD modeling) and OpenFOAM [70] (for computational fluid dynamics solver). In all three cases, I want to replace a physics-based simulation process with a computationally cheap DNN-based surrogate model for a given design space (*DS*). For this end, I want to model the relationship between input (design parameters) and output (behavior of interest) using a neural network. By training a neural network in a given design space, I want to capitalize on the excellent generalization capability of the neural network and make predictions on other designs in the design space without running the actual simulation. Accordingly, it will result in huge savings in computational cost and can get the design's behavior of interest in almost negligible time in comparison to running simulation. In the FEA domain, the learning goal is to train a surrogate to predict the maximum Von-mises stress (which determines the hull's integrity when submerged at a depth) in the capsule-shaped pressure vessel used in an underwater vehicle [150]. In the propeller domain, the goal is to train a surrogate to predict the propeller's efficiency based on the given requirements and geometric design [156]. In the CFD domain, the surrogate learning goal is to predict the drag force on a given UUV (unmanned underwater vehicle) hull shape defined by hull parameters. The size of the design space (dimensionality and extension) dictates the amount of data required for training in each domain. Larger and higher-dimensional design spaces need more labeled samples for surrogate training, and vice versa. For all experiments, I chose a reasonable-sized design space and collected or labeled a reasonable amount of data for experimentation. I aim to evaluate the performance of all the above-mentioned sampling approaches in these real-world design domain problems.

4.3.4.1 Pressure vessel design problem using Finite Element Analysis (FEA)

4.3.4.1.1 Problem setting and numerical method

Static stress analysis is the most common type of structural integrity analysis of the sub-sea pressure vessel. A subsea pressure vessel contains dry components like electronics, batteries, sensors, and various underwater

equipment. During the design phase of the vessel, these pressure vessels are tested by subjecting them to external pressure to test the integrity of the design during subsea operation and to find an optimal vessel design that can withstand the operating pressurized environmental conditions (generally, optimal designs are the thinnest possible designs).For the parametric design of the vessel, [150] chose a cylindrical vessel with hemispherical end caps. This simplified geometry allows for an efficient distribution of the external pressure. Cylindrical pressure vessels have wide applications in thermal and nuclear power plants, process and chemical industries, space and ocean depths, and fluid supply systems in industries [68]. The material that is used in this research study is aluminum alloy ($Al-6061T6$) which is a widely used material for sub-sea pressure vessel design with material properties (density = 2700 Kg/m^3, Young's modulus = 69×10^6 Pa, Poisson ratio = 0.33) [150]. The accuracy of the FEA is highly dependent on the mesh employed. For an accurate result, an automated process of refining the mesh and evaluating the results is carried on until the result stabilizes and further refinement does not change the output [150].

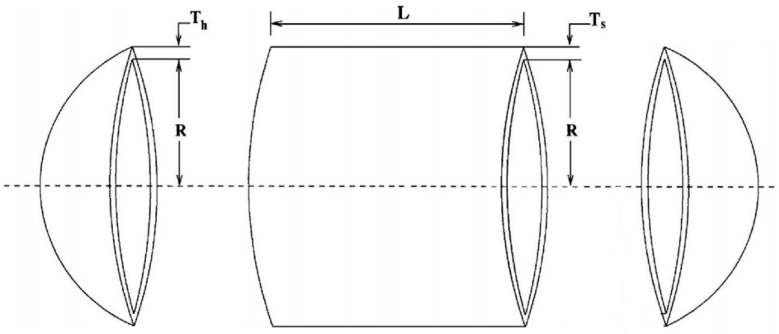


Figure 4.5: Parametric geometry of pressure vessel design

These design parameters and the depth of the sea (which determines the applied external pressure to the vessel) are parameterized, and a combination of these parameters creates a design space. For estimation of applied crush pressure, the following formulation is being used with water density as open sea density 1027 kg/m^3:

$$External\ pressure = \rho \times g \times depth \times safety\ factor \tag{4.23}$$

The value of the safety factor is taken to 1.5. For a given internal radius of the cylinder (a), uniform thickness of vessel (t) , the outside radius ($b = a+t$) , the internal pressure by p_i, and the external pressure by p_0 the tangential and radial stresses are given by [134]:

$$\sigma_t = \frac{p_i a^2 - p_o b^2 + a^2 b^2 (p_i - p_o)/r^2}{b^2 - a^2} \tag{4.24}$$

$$\sigma_r = \frac{p_i a^2 - p_o b^2 - a^2 b^2 (p_i - p_o)/r^2}{b^2 - a^2} \tag{4.25}$$

These equations of radial and tangential stress as a function of radius are known as Lamé's equations. Axial or longitudinal stress for this closed-end cylinder can be found by force equilibrium condition and is given by:

$$\sigma_l = \frac{p_i a^2 - p_o b^2}{b^2 - a^2} \tag{4.26}$$

In this experiment setting, it is assumed to have only external pressure i.e. $p_i = 0$. In such a case, the stress equations as a function of radius can be written as :

$$\sigma_t = \frac{-p_o b^2}{b^2 - a^2}\left[1 + \frac{a^2}{r^2}\right] \tag{4.27}$$

$$\sigma_r = \frac{-p_o b^2}{b^2 - a^2}\left[1 - \frac{a^2}{r^2}\right] \tag{4.28}$$

$$\sigma_l = \frac{-p_o b^2}{b^2 - a^2} \tag{4.29}$$

All these three stresses represent the three principal stresses acting on a cylinder. Hence the Von-mises stress (also known as equivalent stress σ_{eqv}) is estimated by using three principal stresses and given by:

$$\sigma_{eqv} = \sqrt{\frac{(\sigma_l - \sigma_t)^2 + (\sigma_t - \sigma_r)^2 + (\sigma_r - \sigma_l)^2}{2}} \tag{4.30}$$

Figure 4.6 shows the outcome of one of the simulation experiments using a UUV hull design simulation outcome in OpenFOAM using the above-mentioned setting and methodology.

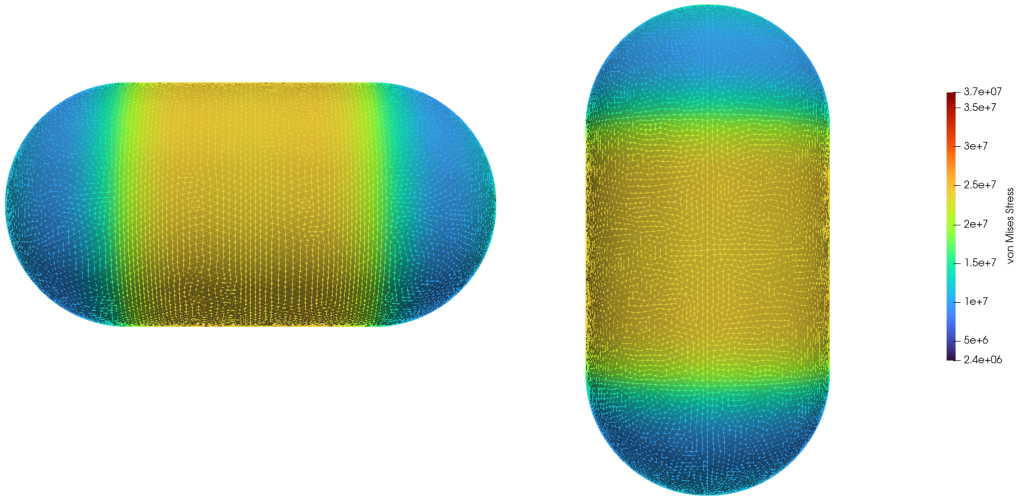


Figure 4.6: Example FEA simulation experiment result. (tested at sea depth =500 meters, the applied crushing pressure is calculated for open seawater density with safety factor 1.5 resulting in crushing pressure of $7.5MPa$. The estimated maximum von-mises stress was $37MPa$ which is much less than the yield stress of (Al-6061$T6$) which is $69MPa$). The hull shape parameter for this design: L: 100 mm, a : 40 mm, t: 10 mm, and D_{sea}:500 meters.

Parameter	Symbol	Minimum	Maximum
Depth of the sea	D_{sea}	0 m	6000 m
Length	L	50 mm	600 mm
Radius	a	1 mm	1850 mm
Thickness	t	50 mm	600 mm

Table 4.1: Design Space: Range of design parameters for surrogate modeling - FEA domain

4.3.4.1.2 Design space and data generation

The design space for this experiment is composed of 4 variables (D_{sea}, L, R, t) (refer table 4.1). From this design space, [150] randomly sampled 11311 data points and ran the FEA simulation to get the maximum Von-mises stress. I use this as our pool data. The sample labeling cost in terms of evaluation time is 200 $sec/sample$. At each iteration of training the surrogate model, I chose 50 samples ($b = 50$) from the pool of collected data and conducted 50 AL iterations on all the above-mentioned approaches, which resulted in 2500 evaluated samples used for training the surrogate.

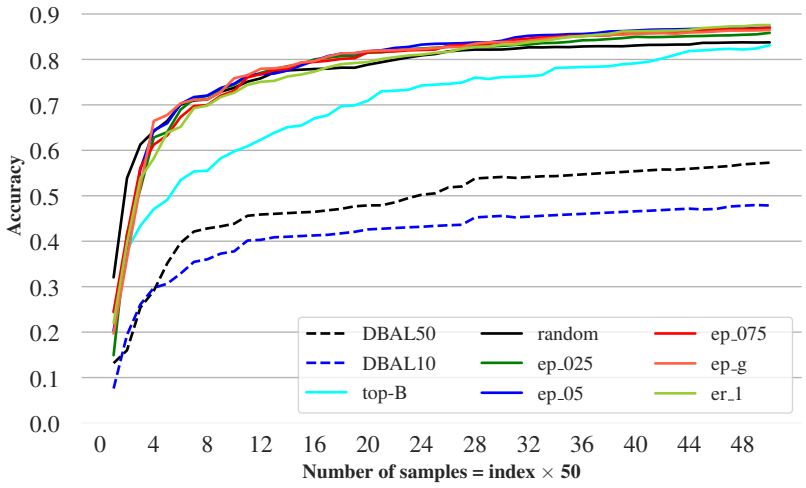


Figure 4.7: Surrogate prediction accuracy on test data using different proposed AL-based strategic sampling methods

4.3.4.1.3 Results

Figure 4.7 and table 4.2 show the accuracy result of trained surrogates on the test data using different strategic sampling methods. It can be seen that all ε-HQS methods (with different values of ε) perform better than other strategic sampling-based surrogate models. DBAL and Top-B methods perform worse than the batched random sampling method due to not handling bias and diversity properly during sample selection. In figure

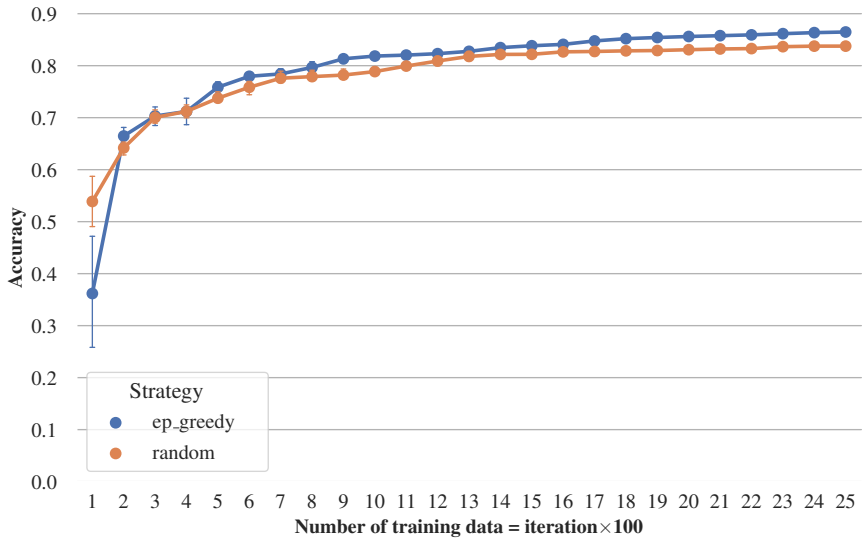


Figure 4.8: Comparison of accuracy between non-parametric ε-greedy method and best baseline methods non ε-HQS method

Sampling strategy	**Accuracy**
Random	83.76%
top-b	83.11%
DBAL-10	57.28%
DBAL-50	47.9%6
ε-greedy	86.47%
$\varepsilon = 0.25$	85.85%
$\varepsilon = 0.50$	87.01%
$\varepsilon = 0.75$	86.96%
$\varepsilon = 1.0$	87.56%

Table 4.2: Accuracy of a trained surrogate after exhausting the training budget in FEA domain: after iteration=50, budget/iteration=50 (total budget = 2500)

4.8, I show the comparison between batched random sampling strategy with the ε method with logarithmic varying ε (ε-greedy). It is empirically observed in this domain to get similar surrogate accuracy as the best-performing baseline (batched random) I need 42% fewer samples. Since the time for training the teacher network and making inferences on prediction is very cheap (approximately 120 seconds per iteration) in comparison to the simulation time (200 Sec), it saves days of sample labeling/simulation time. Since the time for training of teacher network and making inferences on prediction is very cheap (approximately 120 seconds per iteration) in comparison to the simulation time (approximately 200 seconds per sample), it saves days of sample labeling/simulation time.

4.3.4.2 Propeller design using lifting line method using OpenProp

4.3.4.2.1 Problem setting and numerical method

A propeller is a widely used mechanical design that converts rotational energy into thrust. The performance of a propeller depends on its geometrical characteristics and physical parameters. For example, [40] considers the number of blades (Z), propeller's diameter (D), chord radial distribution (C/D), pitch radial distribution (P/R), and hub diameter ($Dhub$) as geometrical characteristics and thrust coefficient (C_t), power coefficient (C_p) and advance ratio (J) as governing physical parameters. For a given thrust requirement at well-defined operational points (e.g., cruising speed and the actuator's spinning rate (RPM)), the goal of a designer is to search for the optimal geometric characteristics that can fulfill the thrust requirement at maximum efficiency (E).The numerical tool used for propeller design in this work is openprop [40]. Openprop is a moderately-loaded lifting line theory-based propeller blade design method with trailing vorticity aligned to the local flow velocity. The goal of the propeller design optimization procedures in openprop is to determine the optimum circulation distribution along the span of the blade under given inflow conditions and blade $2D$ section properties. For a given required thrust T_S, the openprop uses Coney's formulation [31] to determine produced torque Q, thrust T, and circulation distribution Γ. For optimization, an unknown Lagrange multiplier (λ_1) is introduced to perform circulation optimization to get an optimal design. For this purpose, an auxiliary function is defined as below:

$$H = Q + \lambda_1 (T - T_s) \tag{4.31}$$

If $T = T_S$ then a minimum value of H coincides with a minimum value of Q. To find the minimum, the partial derivative with respect to unknowns is set to zero.

$$\frac{\partial H}{\partial \Gamma(i)} = 0 \; for \; i = 1, 2, ..., M \tag{4.32}$$

$$\frac{\partial H}{\partial \lambda_1} = 0 \tag{4.33}$$

These M systems of equations are non-linear, and an iterative approach is used to solve them by freezing other variables and linearizing the system of equations with linearized unknowns $\hat{\Gamma}, \hat{\lambda_1}$. The system of equations 4.32 and 4.33 is solved for the linear $\hat{\Gamma}, \hat{\lambda_1}$, and the new linearized circulation distribution $\hat{\Gamma}$ is used to update the flow parameters. This process is repeated until convergence, which does yield an optimized circulation distribution and a physically realistic design. For more details on numerical methods, refer to [31, 40].

4.3.4.2.2 Design space and data generation

I used the design space used by Vardhan et al. [156]. Since the evaluation of each design is cheap and the dimensionality of the problem is high (14 dimensions), [156] generated approximately 200000 valid designs as pool data. For surrogate training, at each iteration of training, I chose 50 samples and trained the surrogate for 50 iterations, the same as the FEA domain.

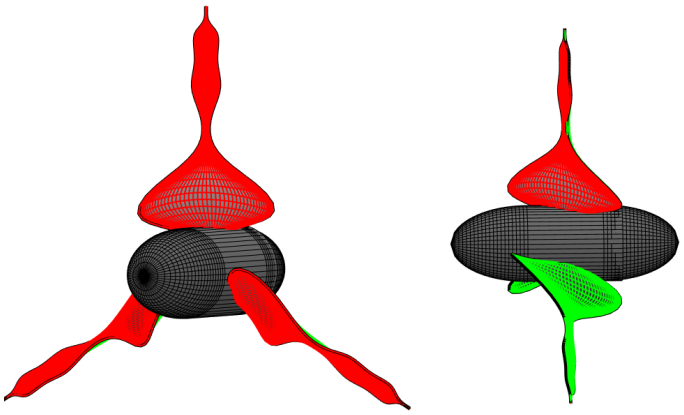


Figure 4.9: Result of a propeller design simulation [156] operating at cruising speed of 14 m/s with rotation rate 338 rpm with an efficiency of 90%.

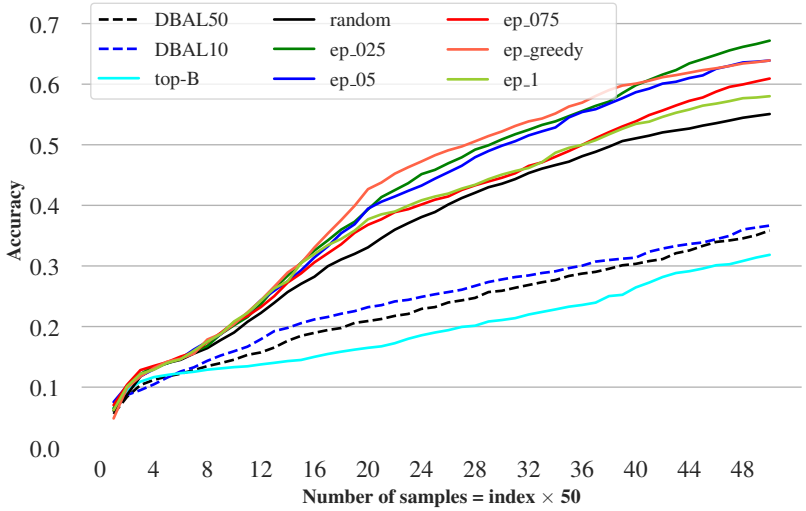


Figure 4.10: The comparison of expected/mean test accuracy of trained surrogate in propeller domain using all proposed approaches at the different iterations of training. DBAL50 (Diverse Batch Active Learning with β=50), DBAL10 (Diverse Batch Active Learning with β=10), random (batch uniformly random), ep_025 (ε-HQS with constant ε=0.25), ep_05 (ε-HQS with constant ε=0.5), ep_075 (ε-HQS with constant ε=0.75), ep_1 (ε-HQS with constant ε=1.0), ep_greedy (ε-HQS with logarithmic increasing ε).

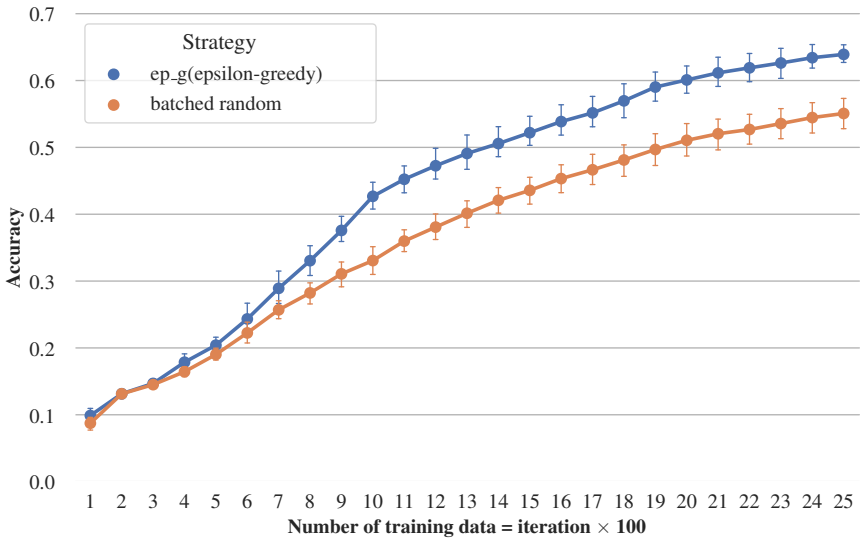


Figure 4.11: The mean and variance of test accuracy using ep_greedy (ε-HQS with logarithmic increasing ε) and DBAL50 (Diverse Batch Active Learning with β=50) strategy

Sampling strategy	**Accuracy**
Random	55.07%
Top-b	31.83%
DBAL-10	35.84%
DBAL-50	36.66%
ε-greedy	63.9%
$\varepsilon = 0.25$	67.18%
$\varepsilon = 0.50$	63.9%
$\varepsilon = 0.75$	60.92%
$\varepsilon = 1.0$	58.01%

Table 4.3: Accuracy table in propeller design domain: iteration=50,budget/iteration=50 (total budget = 2500)

4.3.4.2.3 Results

Figure 4.10 and table 4.3 show the accuracy result of trained surrogates on the test data using different strategic sampling methods. It can be seen that all ε weighted methods perform better than other strategic sampling-based surrogate modeling. DBAL50, DBAL10, and Top-B perform worse than the random batch uniform sampling method. Figure 4.11 shows the comparison between the batched random sampling strategy when compared with our proposed method with logarithmic varying ε (no parameter). It is empirically observed in this domain to get similar surrogate accuracy as the best-performing baseline (batched random) but with 32% fewer samples. It can also be seen that the prediction accuracy gap between the best ε-HQS method and the best baseline method (batched random in this case) is more than 12%. In this domain, the labeling cost is not much, but the goal of its inclusion was to analyze the performance of this approach in high-dimensional problems. A wider gap in this example domain reflects that in high-dimension problems

Sampling strategy	Accuracy
Random	94.27%
top-b	88.33%
DBAL-10	90.55%
DBAL-50	95.09%
ε–greedy	96.81%
$\varepsilon = 0.25$	95.93%
$\varepsilon = 0.50$	96.97%
$\varepsilon = 0.75$	96.67%
$\varepsilon = 1.0$	97.09%

Table 4.4: Accuracy table in CFD design domain: iteration=50, budget/iteration=10 (total budget = 500)

(14 dimensions in this case) the ε-HQS based ε-greedy is more sample efficient than sample labeling by other strategic sampling methods.

4.3.4.3 UUV hull design using CFD analysis

4.3.4.3.1 Problem setting and numerical method

In the design process of a UUV's hull, the drag resistance has the most dominant effect, and the goal of design optimization is to minimize the drag resistance for a given set of design requirements. For the pooled data set, I take the pool data from the experiment explained in section 3.4.2 that uses the process explained in section 3.2.1. Since, in this case, I have a small data pool, at each iteration of training, I chose 10 samples from the pool of collected data. I conducted this iterative training for 50 iterations, which resulted in 500 samples. While choosing samples at each iteration, I deployed all the policies mentioned above for batching.

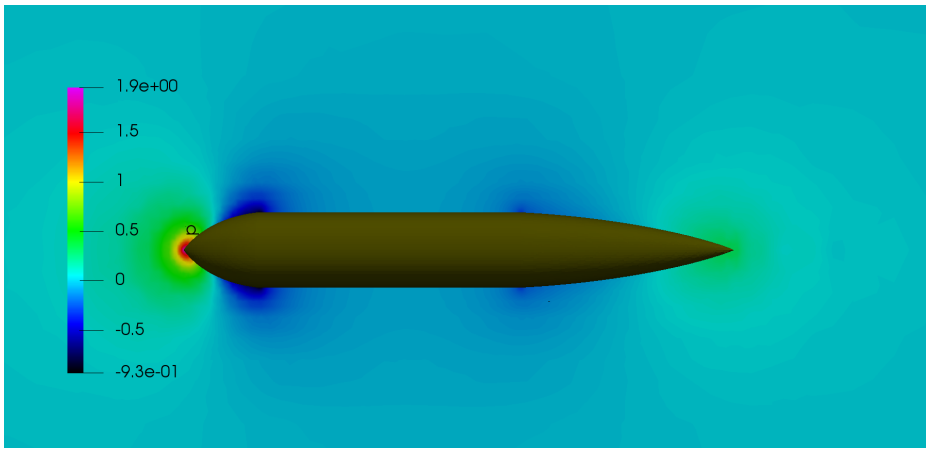


Figure 4.12: Output steady state pressure field of one of the simulations.

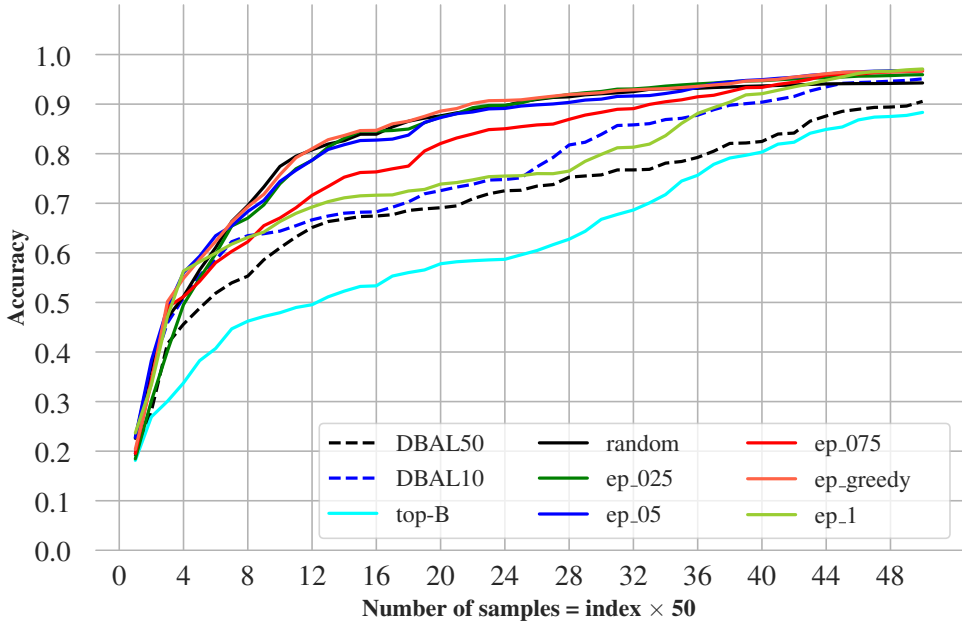


Figure 4.13: The comparison of expected / mean test accuracy of trained surrogate in CFD domain using all proposed approaches at the different iterations of training. DBAL50 (Diverse Batch Active Learning with β=50), DBAL10 (Diverse Batch Active Learning with β=10), random (batch uniformly random), ep_025 (ε-HQS with constant ε=0.25), ep_05 (ε-HQS with constant ε=0.5), ep_075 (ε-HQS with constant ε=0.75), ep_1 (ε-HQS with constant ε=1.0), ep_greedy (ε-HQS with logarithmic increasing ε).

4.3.4.3.2 Results

Figure 4.13 and table 4.4 show the accuracy result of trained surrogates on the test data using different sampling methods. It can be seen that all ε-HQS methods have better accuracy on test data than other sampling approaches for surrogate modeling. In this domain, DBAL-50 performed slightly better than the batched random sampling but still less than all ε-HQS methods. The Top-B approach performs worse than all other approaches due to a lack of consideration of diversification in sample selection. Figure 4.14 shows the accuracy of DBAL-50 sampling strategy-based trained surrogate and ε-HQS with logarithmic varying ε sampling-based trained surrogate on multiple runs across all iterations. It shows the mean accuracy as well as the variance in accuracy prediction. The predicted test accuracy performance suggests that empirically in this domain, to get similar surrogate accuracy as the alternative policy-based best-performing approach (DBAL-50), ε-HQS with non-parametric ε-greedy approach needs approximately 20% fewer samples. Since the time for training the teacher network and making inferences on prediction is very cheap (approximately 120 seconds per iteration) compared to the simulation time (approximately 300 seconds per sample), it saves days of sample labeling and simulation time.

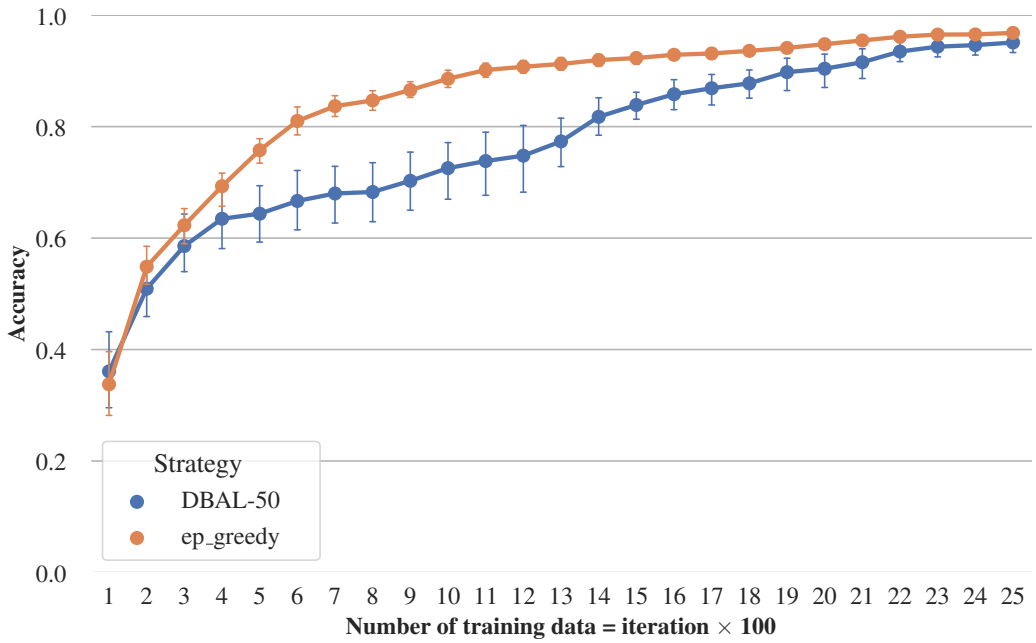


Figure 4.14: The mean and variance of test accuracy using ep_greedy (ε-HQS with logarithmic increasing ε) and DBAL50 (Diverse Batch Active Learning with β=50) strategy

4.4 Summary of contribution

In this work, I developed and evaluated a novel deep active learning approach for data-efficient surrogate modeling in regression problems. This method is computationally tractable and scalable since it does not use an ensemble of learning models for query design. I brought knowledge and approaches from the current research on active learning to increase collective information and diversity in a batch learning setting and designed algorithms based on them. When I compared these methods with different benchmarks and generated data sets, I found that ε-HQS outperforms all of them in all cases. I also observed that as the dimension of the problem increases, the performance gap between ε-HQS and other baselines becomes wider. This is a new paradigm for deep active learning in a regression problem.

CHAPTER 5

Discovering rare event failure in LEC

5.1 Problem formulation

When a surrogate model is used in system operation, ensuring functional correctness is critical for its reliable operation. For studying this problem, a reinforcement learning-based car braking controller is trained as a surrogate model. For designing a machine learning-based controller in a reinforcement learning setting, the training process involves running simulation experiments with the aim of learning a policy for the desired behavior through continuous interaction with the environment. These trained models are generally called an *agent*. Let's introduce some notation to formulate the problem. The state information X represents the agent's state in the environment, which the agent perceives from its environment. Z represents the environment's state (which may be the environment's private representation), which is sampled from some unknown distribution $\mathbb{P}_Z$. In our experimental setting, the experimenter neither observes nor controls Z. X is drawn from some distribution $\mathbb{P}_X$, which does not need to be known. The only requirement imposed on X is that the experimenter should be able to sample it quickly with approximately no cost in comparison to running the simulation. Our interest is the agent's performance on the environment distribution ($\mathbb{P}_Z$) over initial conditions $X \sim \mathbb{P}_X$. Once training is done, assessment of the trained agent is carried out by rolling out experiments with the trained agent given an initial condition $X \sim \mathbb{P}_X$. The outcome of the experiment can be failure or success, which is a random variable called *failure indicator* C. Here, C has a binary value, and $C = 1$ indicates a "failure". Our interest here is to find a good test case that can reveal the potential fault in the trained controller. The problem can be formulated as finding those $X \sim \mathbb{P}_X$ on which the trained controller fails.

In the naive approach to finding a failure, Vanilla Monte Carlo may not find a failure even after evaluating the same number of episodes as training episodes. It may give a sense of a safe controller, which may or may not be true. To make evaluation more rigorous, one approach is to use a priority replay (PR) adversary search, which tests the trained model on all earlier failures that happened during training. Generally, a well-trained agent does not fail during a priority replay search, as it has learned from all failures that occurred during training. After failing to find failure during PR adversary search, it is obvious to switch to randomized search (Monte Carlo sampling) from the input search space and run a simulation to observe the results. In a system with a large input search space, especially in the case of all CPS that work in a continuous domain, finding a failure using Monte Carlo is really expensive in terms of the number of simulation episodes required to find a failure case.

To speed up the search for failures in learned agents, [145] proposed AVF (failure probability predictor) guided search. The motivation behind AVF-guided search is to screen out situations that are unlikely to be problematic and focus evaluation on the most difficult situations (corner cases) in a probabilistic rigorous framework. AVF learns the failure pattern from the earlier agent's failure. During the testing of the agent, it assigns a probability of failure for each input search space. It is advantageous because if sampling from the search space and its prediction using AVF are computationally cheaper, it can narrow our search for failure scenarios to regions with a high probability of failure and save on the simulation's computation. Given input data, AVF predicts the probability of failure based on observed training data. Finding failure using AVF involves two phases of training: *controller training* and *AVF training* (refer to figure 5.1). In the controller training phase, an AI model is trained using reinforcement learning methods to achieve a learning goal. As training progresses, the learned controller becomes more robust and will have fewer and fewer failures. At each episode during training, the experiment starts with initial conditions $X \sim \mathbb{P}_X$, and its outcome is observed $C \rightarrow \{0,1\}$. For any episodic simulation, it is guaranteed to get a return value of C. During training, initial condition (X), simulation episodic outcome (C) and information about the agent (θ) are collected. Here, θ encodes all information that can tell about the agent state (like stochastic noise added during each episode or episode number). So, at each episode of training t, simulation starts with X_t and C_t and θ_t is observed. The training data in the form of $\{(X_1;\theta_1;C_1),(X_2;\theta_2;C_2),\cdots,(X_n;\theta_n;C_n)\}$ will be used to train AVF in the AVF training phase. Once training is finished, the controller is tested up to the maximum of the training episode. If testing does not give us any failures, it is assumed that our trained model is functionally robust, which may or may not be true. During the AVF training phase, training data is used to train a neural network in a supervised setting. In the initial phase of training, the agent is more likely to fail, so failure data from the initial phase of training is ignored, and only training data from the last phase of training is used for training AVF. X and θ form the input feature space, and C would be the output of the neural network. A neural network is trained to return the probability of failure given an initial condition and information about the agent. $\mathscr{NN} : (X,\theta) \rightarrow [0,1]$.

If f_* is the real failure probability function given the initial state X and θ, the goal of training AVF ($\mathscr{NN}$) is to approximate f_* up to a normalization constant.

$$f_*(x) = \mathbb{P}(C = 1|X,\theta); \quad X \sim P_X$$
$$\mathscr{NN} \approx f_*$$

In search of a failure condition, the traditional *Vanilla Monte Carlo (VMC)* sampling samples X from $\mathbb{P}_X$ and runs the experiment. If the experiment is episodic, then the result of the simulation will be a success

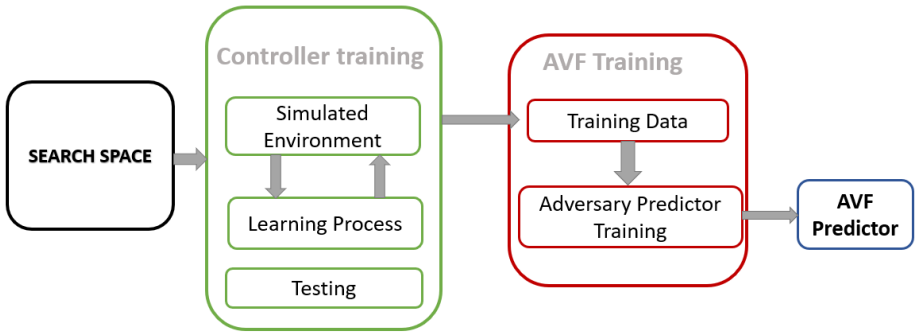


Figure 5.1: Phases of Training

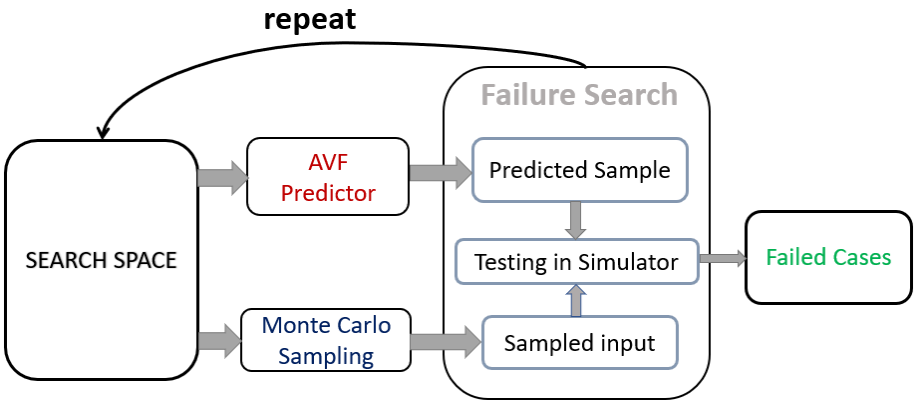


Figure 5.2: Failure search - AVF and Monte Carlo

($C = 0$) or failure ($C = 1$). For a well-trained agent, finding a failure case may take way more than training episodes. In contrast, when using AVF-guided search to find a failure, the obvious approach is to evaluate the agent on $argmax_{\mathcal{NN}}(x)$. However, maximization using a neural network is not feasible, and I also only have knowledge of approximate probability, so a feasible approach is to sample n initial conditions from $\mathbb{P}_X$, pick the initial condition from this set where $\mathcal{NN}$ is the largest, and run the experiment from the found initial condition. This process is repeated until a failure is found. The reasoning behind predicting first through $\mathcal{NN}$ is to discard all the search space samples that have a very low probability of failure and select the one that has a greater chance of failure. One assumption is implicit: if the cost of evaluating a sample on $\mathcal{NN}$ is negligible in comparison to running an experiment, then only AVF-guided search is advisable in comparison to VMC. The idea is to use $\mathcal{NN}$ to save on the cost of experimentation.

This guided search method has some limitations that are observed during empirical evaluation and also has scope for improvement. The first limitation of AVF-guided search is to rely completely on training data and the training processes for searching for failure. The training process in reinforcement learning is not guaranteed to be monotonic. Research efforts like constraining gradient updates of policy parameters

(TRPO) [127], stochastic weight averaging (SWA) [107] etc. attempt to learn a monotonically increasing control policy, but there is no guarantee for such convergence. The second limitation of AVF-guided search is it does not adapt to discoveries of new failed cases; in such cases, AVF does not improve this search process. In large state spaces, the adaptive search process is beneficial by saving cost and time by adapting to new discoveries. The third limitation is that there is no way to influence this guided search by including data derived using expert knowledge, statistics collected in the real-world or other simulators, similar experiments, etc. that may help to find a more realistic and speedier discovery of test scenarios.

5.2 Proposed work

To tackle all the above-mentioned issues, the following solution is proposed: Let's assume if $X_{f1}, X_{f2}, \cdots, X_{fn}$ are a collection of failure instances, the goal is to model $\mathbb{P}(X_f)$. This trained probability distribution tries to imitate and approximate the real failure probability distribution. Once trained, a sample can be generated from this distribution to generate a test scenario that has a high probability of failure. For this purpose, a mixture model-based clustering algorithm is used to model the $\mathbb{P}(X_f)$. A parametric probability distribution model $\mathbb{P}(X_f|w)$ fits our data distribution by finding the best parameter (w) which represents $\mathbb{P}(X_f)$, where w is the parameter of the model. Gaussian Mixture Model [118], which is a popular mixture model, is a reasonable model to work with. So, the problem of modeling the probability distribution of data is now training a GMM which can be written as:

$$\mathbb{P}(X_f|w) = \pi_1 * \mathscr{N}(X_f|\mu_1, \Sigma_1) + \cdots + \pi_n * \mathscr{N}(X_f|\mu_n, \Sigma_n)$$

$$w = \{\mu_1, \Sigma_1, \pi_1, \cdots, \mu_n, \Sigma_n\}$$

Here n is a hyper-parameter, which depends on the distribution structure of data and can be tuned using the Bayesian Information Criterion (BIC) and cross-validation. The training problem, in this case, would be

$$\max_{w} \prod_{i=1}^{N} P(X_f|w) = \prod_{i=1}^{N} (\pi_1 * \mathscr{N}(X_f|\mu_1, \Sigma_1) + \cdots)$$

$$subjected\ to\ \pi_1 + \cdots + \pi_n = 1; \pi_k \geqslant 0;\ k = 1, \cdots n$$

$$\Sigma_k > 0\ k = 1, \cdots n$$

The first hyperparameter to tune is the number of mixture models (n). This hyper-tuning can be done based on Bayesian information criteria and cross-validation. Expectation-Maximisation (EM) algorithm [17] with a full covariance matrix is used to train the GMM. EM is an iterative process to find maximum a posteriori (MAP) estimates of the parameter (w), where the data is modeled on unobserved latent variables. EM also

depends on the initialization value of w, so making different initializations and selecting one that represents the data the most This generative ML model may work stand-alone or in the augmentation of AVF. The preliminary result reflects that in augmentation with AVF, it can predict good test scenarios faster than AVF. It can be trained online, i.e., a new failure scenario data is obtained, the old posterior becomes the new prior, and then the condition on the prior using the Bayes rule is carried on to get the new posterior (i.e., a new probability distribution).

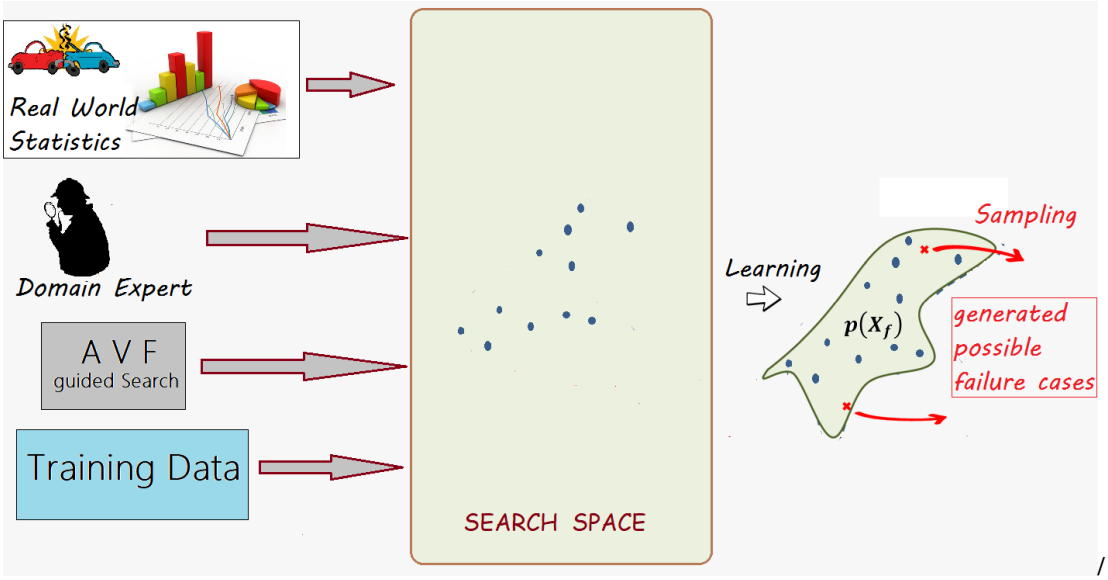


Figure 5.3: The motivation behind generative model-guided search

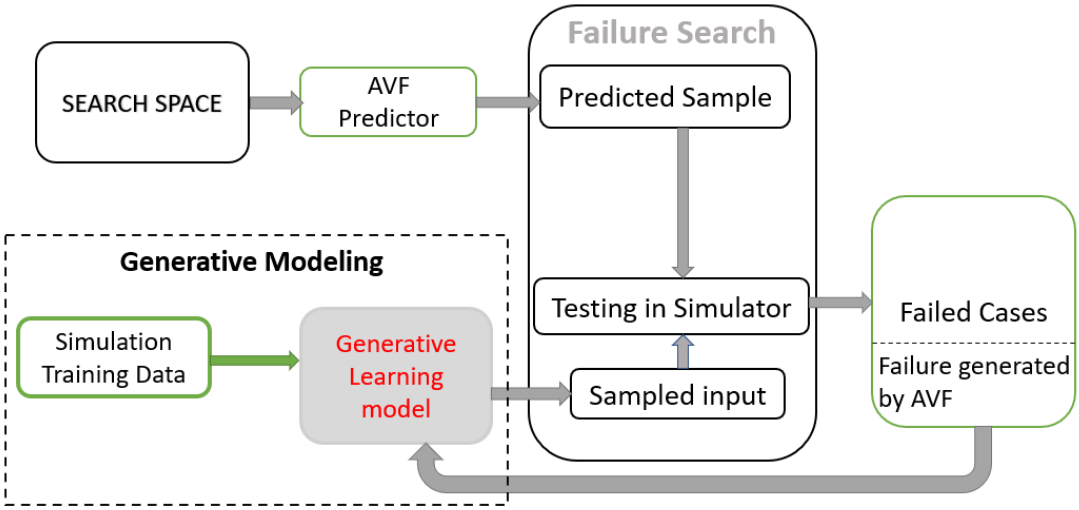


Figure 5.4: Failure search - AVF in augmentation with Generative model

Training data that was used for AVF training can be a reasonable starting point for modeling this distri-

bution. Once more and more data from various sources is available, the model can be retrained to make it better. After training, a sample is generated from this family of multivariate normal distributions for a probable test candidate. By running the simulation experiment with this sampled initial condition (X) the outcome is observed in terms of failure or success (C). Trained using earlier failure cases and other data sources, there is a high probability of generating samples that can fail and a very low probability of generating data in the region of search space that is less likely to fail. This generative model can also work in the augmentation of AVF-guided search, where failure cases generated by AVF-guided search also contribute to dynamically modifying the probability distribution model (refer to figure 5.4).

5.3 Empirical evaluation and results

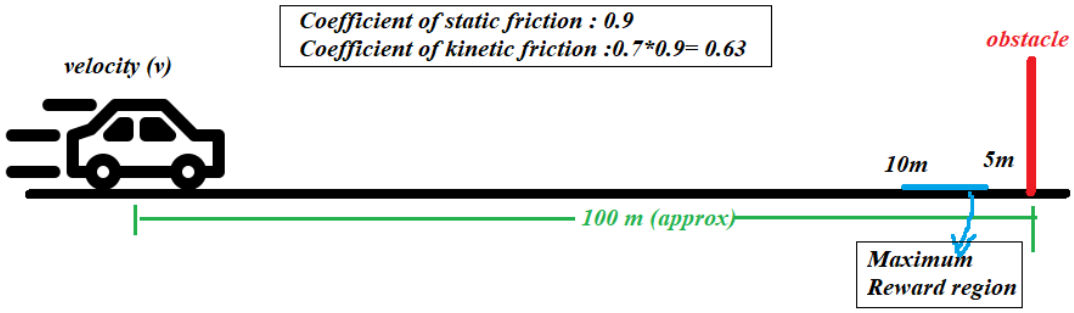


Figure 5.5: Scenario 1

To empirically evaluate the above-mentioned approaches, two different experiments were conducted (refer to figures 5.5, 5.6). In both cases, a reinforcement learning approach called DDPG (Deep Deterministic Policy Gradient) [84] based learning model was trained to design the braking system of a car. In *Scenario 1*, an approaching car detects an obstacle at a distance of 100 meters, and the learning goal is to stop without crashing (refer to figure 5.5). The friction coefficient of the road was constant (assuming a dry road) throughout the experiment, and it was 0.9. The random variable in this scenario was the speed of the car when it detects the obstacle, which was drawn from a normal distribution with a mean velocity of 38 miles per hour and a standard deviation of 11 miles per hour. $Initial\_Speed(v) \sim \mathcal{N}(38,11)$

In *Scenario 2*, I added a patch of road with different friction coefficients (from '0.6' for a dry surface to '0.1' for a wet surface) to make the environment more complicated. The size of this patch is drawn from a normal distribution with a mean of 15 meters and a standard deviation of 5 meters, and it can be anywhere between 0 and 100 meters (see figure 5.6). The speed of the car when it detects the obstacle in this scenario was drawn from a normal distribution with a mean velocity of 35 miles per hour and a standard deviation of 9 miles per hour. $Initial\_Speed \sim \mathcal{N}(35,9)$. The learning goal in both cases was to learn a controller for

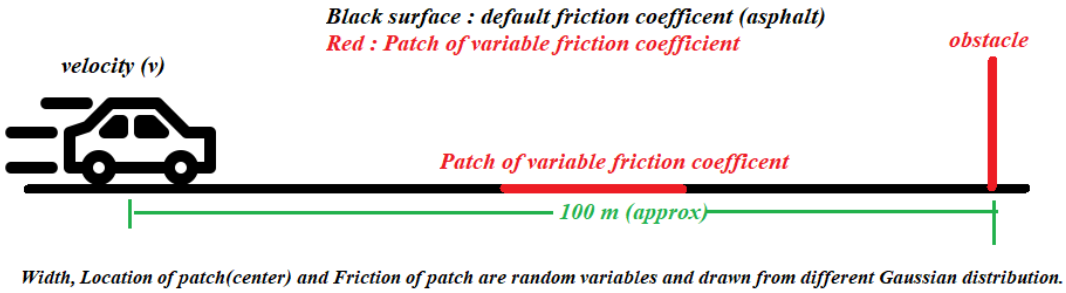


Figure 5.6: Scenario 2

Failure number	VMC	AVF	GMM
1	16349	13	1
2	518	1	1
3	131188	66	8
4	8217	2	13
5	28033	20	7
6	25481	11	1
7	21184	24	5
8	26979	41	3
9	8566	49	15
10	36466	16	2
Average	**30298**	**24**	**6**

Table 5.1: Episodes of simulation for 10 consecutive failure search in *Scenario2*. The agent is trained for 15000 episodes. The average number of episodes required in the case of VMC is almost double the number of training episodes and way higher than AVF and GMM guided search.

braking. The reward setting was such that the vehicle, when it brakes around a region of 5–10 meters, gets the maximum rewards. The output brake is $b \in [0,1]$, where $b = 0$ means no brake and $b = 1$ means maximum brake. I used a nonlinear slip-based braking equation to model the effect of slipping on braking friction.

For comparing vanilla Monte Carlo-based failure search with AVF-guided search and GMM-guided search, I chose the number of episodes required to find an initial condition whose trajectory resulted in a failure. Our purpose was to illustrate three key points. First, VMC-guided search, even when using the same number of episodes as training the agent, can lead to a false sense of safety by failing to detect any catastrophic failures. Second, AVF-guided search addresses this issue and reduces the cost of finding failures to a large extent. Third, generative model-guided search also does better than VMC-guided search, and in augmentation, with AVF, it was the most flexible and dynamic approach to finding a failure.

Table 5.1 shows the number of simulation episodes rolled out in search of ten different failure cases in *Scenario 2*. In an average case, the VMC took 30298 episodes, which was almost double the number of training episodes. In such a case, if an agent is tested only up to a maximum number of training episodes, it

Scenario	VMC	AVF	GMM	GMM +AVF
Scenario1	82/10738/47323	1/6/34	4/3991/21393	1/3/26
Scenario2	3/22252/154374	1/36/102	1/4/19	1/5/27

Table 5.2. The cost of various search strategies to find a failure case, expressed as the quantity of simulation episodes (for 100 failures). Each column reports the Min/Average/Max number of simulation episodes for 100 failures. In *Scenario1*, AVF+GMM guided search is two times faster than AVF guided search in the average case. However GMM guided search does worse in scenario 1 than in scenario 2, but it is still faster than VMC. **Training Episodes**: *Scenario1*(5000) ; *Scenario2*(15000)

has a high probability of giving a false sense of the correctness of this trained system. However, AVF took only six episodes to find a failure. So, VMC-based failure search is not only inefficient, but it may also lead to deploying an agent, which may result in catastrophic failure.

Table 5.2 shows the number of episodes rolled out required in a minimum, maximum, and average case in search of 100 different failures in both scenario 1 and scenario 2. I also tested GMM-guided search and GMM in augmentation with AVF-guided search. For GMM and GMM+AVF, I did not use data from domain expert knowledge and real-world statistics. However, including these data will surely increase the probability of getting better failure cases. For GMM, I used the same data that I used for AVF training. However, while using GMM+AVF, I first trained GMM, and during the failure search process, I sampled one test case from both GMM and AVF alternatively. If I encounter any failures in a sample that AVF generated, I update our probability distribution to account for these newly discovered data. For *scenario1*, in an average case, VMC took 10738 episodes, which is almost double the number of training episodes. AVF-guided search took only 6 episodes to find a failure in the average case. For *scenario2*, in an average case, the agent took 22252 episodes, which is almost 1.5 times the number of training episodes. AVF-guided search took only 36 episodes to find a failure in the average case. AVF-guided search does way better than the Vanilla Monte Carlo search.

In the average case, GMM-guided search did better than VMC in both scenarios. However, in *Scenario2*, GMM search was faster than the AVF-guided search. This was due to the nature of the data generated during training. If the training data used for AVF/GMM training is far from real failure data in the search space, then GMM performs poorer. However, it still did better than VMC. Our motivation for using the GMM model is to reduce dependency only on training data for generating failures and incorporate data from other sources like domain expert knowledge and real-world collected statistics, which is not possible in the case of AVF-guided search. GMM in augmentation of AVF performs twice faster than AVF only guided search in *scenario1* and five times faster than AVF only in *scenario2*. AVF with GMM was the most flexible and dynamic approach to finding failure in learning enable controller.

5.4 Summary of Contributions

The main contribution of this work was the development of a novel approach for detecting rare failure cases in an AI-based surrogate model. The need for this approach arises because standard approaches to evaluating failure in machine learning models are highly inefficient in detecting rare failure cases. In such a case, there is a possibility of deploying a trained model, which will give the illusion of completeness and safety. This approach can find such rare failures much faster than using traditional methods.

CHAPTER 6

Anomaly/Outlier detection using Robust Random Cut Forest

6.1 Problem formulation

As discussed in chapter 1, in modern CPS systems, LECs are being used to replace the traditional embedded controller. The expected gain is cheaper evaluation cost using LEC; however, it comes with challenges to ensure the input in these LEC is not coming from out-of-distribution input data. For problem formulation, let's introduce some notations. Let X represent an input data point given to a machine learning model during training. This information may be a result of observations made by sensors connected to the CPS system, such as image data from the camera or distance data from LIDAR, etc. During training, I collect all such Xs. From the ML model's perspective, this collection of all observed X represents the environment in which the model has been trained, I collectively call this distribution $\mathbb{E}_t$. Here, I am interested in point anomaly detection. The OOD detection goal is to find whether input data X' given during prediction is sampled from $\mathbb{E}_t$ or not. If $X' \sim \mathbb{E}_t$, then I call this observed state non-OOD or else I call it OOD, i.e., the trained agent has not seen this kind of input during training, and the trained agent may behave unexpectedly to this OOD input data. As we do not control or know $\mathbb{P}_T$, even after extensive training, it is possible to encounter X which was not part of $\mathbb{E}_t$. The detection of OOD may raise a flag about the reliability of predictions made by a trained model.

6.2 Proposed work

Our approach for anomaly detection relies on learning the data cluster (T) 's shape from metric space to a data structure (S). The motivation behind learning the cluster's shape into a data structure is to abstract the information from metric space in a structured manner such that computer and related algorithms can be efficiently deployed for inference. If $D = \{d_1, d_2, ..., d_n\}$ are set of data points such that $d_i \in R^m$. For the purpose of Out-of-Distribution detection, the following requirements are imposed on this data structure (S):

1. S should represent the cluster of data in a structured way.

2. Relationships (ψ) between the data points in metric space must be preserved in this data structure. i.e

$$\psi\{T(d_k, d_l)\} \approx \psi\{S(d_k, d_l)\}$$

3. Relationship (ϕ) of a data point with the cluster can be encoded in simple quantitative measure. i.e.

$\phi(T, d_k)$ can be measured as a scalar value in the data-structure (S).

I chose the RRCF [55] as our data structure to represent our cluster in a structured way. Robust Random Cut Forest can be formally defined as :

Definition 1 *Robust Random Cut Tree on set of data point* $D = \{d_1, d_2, ..., d_n\}$ *can be generated by following procedure:*

1. $r_i = max_{X \in D}(X_i) - min_{X \in D}(X_i)\ \forall i \in m$
2. $p_i = \frac{r_i}{\sum_{i=1}^{i=m} r_i}\ \forall i$
3. select a random dimension i with probability proportional to p_i
4. $choose\ x_i \mid x_i \sim Uniform(max(X_i) - min(X_i))$
5. $D_1 = \{X \mid X \in D, X_i \leq x_i\}$
6. $D_2 = D \setminus D_1$

recurse on D_1 *and* D_2 *until* $D_i \geq 1$

Robust Random Cut Forest is an ensemble of various RRCTs. The selected relationship (ψ) between data points in metric space is captured as the L_p distance between data points, then I require a distance-preserving embedding of this relationship in the data structure. For this purpose, the tree distance between two data points d_k and d_l in the data structure (S) is defined as the weight of the least common ancestor of d_k and d_l [55], then according to Johnson-Landatrauss lemma [87] the tree distance can be bounded from at least $L_1(d_k, d_l)$ to maximum $O(d * log|k|/L_1(d_k, d_l))$. Accordingly, a point that is far from other points in metric space will continue to be at least as far in a random-cut tree. The relationship (ϕ) of a data point with the cluster can be encoded in simple quantitative measure by displacement, which is an estimate of the change in model complexity (summation of the leaves' depth) before and after inserting a given point x in the tree data-structure.

RRCF data structure contains sufficient information about the given data set (Y) and approximately preserves distances in metric space, i.e., if a point is far from others, it will continue to be at least as far in a random cut tree in expectation and vice versa (proof can be found in Guha et al [55]). The anomaly score (also called DispValue) of the data point measures the change in model complexity incurred by inserting a given data point x in RRCF. The model complexity of a binary tree (RRCT) is defined as the sum of the depths of all data points in the tree. During the insertion of a data point that is far off the cluster in metric space, there is a high probability of being partitioned in the initial stage of RRCT construction. This will increase the depth

of all leaves below it and consequently increase the model complexity by a large number. On the other hand, if the inserted point is inside the cluster, it will be partitioned in the lower part of a tree, and consequently, it will have fewer leaves below it, which will reduce the model complexity by a small amount. Given a training dataset Y, I first create these random cut trees and find the maximum Disp value of all inserted data points. I define OOD as a data point that is significantly different from data used in training, i.e., it is away from the training data cluster (Y) in the normed vector space. These off-the-cluster data-point points have a high probability of being isolated in the initial stage of RRCT construction. Insertion of this point in the tree will significantly increase the model complexity.

Applying this approach to large, high-dimensional training data results in the creation of a large forest, and consequently, any inference will have high prediction inaccuracy, be computationally costly, and even sometimes be computationally infeasible. The underlying reason is as *current learning models and processes are not very sample efficient*, and a well-trained agent needs big training data. Research efforts are being made to make these processes more sample efficient [8] but they have few practical generalities. [49] showed that on large data sub-sampling may improve random forest performance, but these forests are sensitive to the extent of sub-sampling and become inconsistent with either no sub-sampling or too severe sub-sampling [141].

Algorithm 4 Reduced Robust Random Cut Forest(offline)

Input: training data (Y)
Output: reduced RRCF, threshold
Parameter: number of trees
$Initialization$: $randomly\ select\ Z \mid Z \subset Y$
$rrcf_{init} = createForest(Z)$
$DispVal_Z = DispValue(Z)$
$DispVal_{threshold} = mean(DispValue(Z))$
$L = Y \setminus Z$
for $i = 0;\ i < len(L);\ i = i+1$ **do**
    for $j = 0;\ j < numberoftrees;\ j = i+1$ **do**
        $rrcf_{new}(j) = insertpoint(rrcf_{init}(j), L(i))$
        $mci(j) = Dispvalue(L(i))$
    end for
    $point_{dispValue} = mean(mci)$
    if $point_{dispValue} \geq DispVal_{threshold}$ **then** ▷ include L(i) in featured Datapoint set
        $DispVal_Z.append(point_{dispValue})$
        $DispVal_{threshold} = mean(DispValue(Z))$
    else ▷ Do not include L(i) in featured Datapoint set
        for $j = 0;\ j < numberoftrees;\ j = i+1$ **do**
            $rrcf_{new}(j) = Deletepoint(rrcf_{init}(j), L(i))$
        end for
    end if
end for
threshold $= max(DispVal_Z)$
return $rrcf_{new}$, threshold

To address these issues, I attempted to find only those featured data points that will create an outline of the data cluster space and have significance in making a decision about OOD and dropping all other data points for the construction of RRCF. The underlying intuition is that in a dense data space, I need a few data points to gain information about the data space they represent, and I can drop most of the other data points. However, in the sparser regions, I select most of the data points to represent the data space they acquire. The sparsity and density in the dataspace are correlated with the average model complexity increment by the inclusion of datapoints in the data structure. Using this approach, I can significantly reduce the number of training data points that need to be stored for the construction of RRCF. RRCF created using these featured data points is called the Reduced Robust Random Cut Forest (RRRCF). For the reduction of the entire data to the featured data, I run a process of insertion and conditional deletion on each data point in training data. After one sweep of this process on the whole training data, I collect all featured data points which represent identical data subspace as training data space. For initialization, I first create an RRCF using a very small dataset (Z), where $Z \subset Y$, this will give us an initial small forest. Once I have created the RRCF using Z, I calculate the DispValue of all points in Z. For making a decision on whether a given point can be included in the RRCF or not, I choose the mean of all DispValue calculated over Z as the threshold. This threshold represents the average complexity of the data structure. For the rest of the data points ($Y \setminus Z$), I insert each point in the forest and I calculate the DispValue for this point. If DispValue is more than the threshold DispValue of the initial forest then I will keep this point in the forest or else I will reject this point and delete it from the RRCF. I recursively apply this on all left-over data points ($Y \setminus Z$). The final forest created from this process would be our reduced robust random-cut forest and can be used for making inferences during prediction for making decisions about OOD for a given data point. This is an offline method and needs to run only once. I can store the reduced RRCF (RRRCF) and threshold, which is the maximum model complexity from already included featured training data points of selected featured data points to be used for prediction. This reduced forest is a structured representation of our training data sub-space.

Algorithm 5 OOD detector(online)

Input: RRRCF(T) , Data-point(x), threshold
Output: Inference about x being OOD
$T' = Insertpoint(T, x)$
$mci = calculatemodelcomplexity$
$T = Deletepoint(T', x)$
$mcd = calculatemodelcomplexity$
$DispValue = mci - mcd$
if $DispValue \geq threshold$ **then**
    x is OOD
else
    x is not OOD
end if

During prediction, I insert a newly observed input data point (x) into the stored RRRCF model (obtained from the algorithm 4) and check whether the inclusion of this point increases the model complexity to an extent higher than a threshold value. Every new observation can be passed to this detector, and predictions made by the machine learning-based model can be accepted only if the scalar measure of the datapoint by the OOD detector is lower than the threshold generated by the algorithm 4. The setup for deploying this OOD detector is shown in figure 6.1.

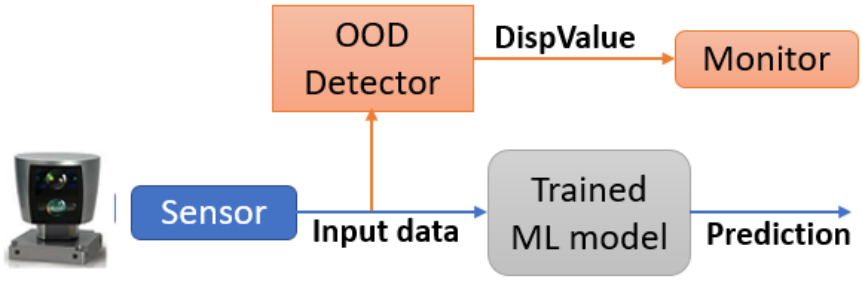


Figure 6.1: Deployment architecture of OOD detector

If the DispValue of the new point is greater than the threshold obtained from the algorithm 4, then I declare this datapoint as OOD and vice versa (refer to the algorithm 5). For making inferences on a data point, I do the insertion step, where I include a new data point in RRRCF and measure its increased complexity, and then I do the deletion step to remove the inserted point. So, during prediction, I do not extend our forest; I just do one insertion and one deletion step per prediction, and our reduced robust random-cut forest remains intact.

6.3 Empirical evaluation and results:

To empirically evaluate the above-mentioned approach, I set up two experiments. In both cases, the machine learning model is a reinforcement learning-based controller for a car braking system, wherein in the first experiment, observations are in low dimensions (3 dimensions), while in the second experiment, our observations are in high dimensions of image data.

In the *first experiment*, an approaching car detects a **stationary obstacle** at a distance of 100 meters, and the learning goal is to self-train a controller for the braking system to stop the car without crashing (refer to figure 6.2). I used a reinforcement learning algorithm called DDPG (Deep Deterministic Policy Gradient) [84] based learning model to design the braking system of a car. The static and kinetic friction coefficients of the road are constant throughout the experiment. The random variable in this scenario is the speed of the car when it detects the obstacle, which is drawn from a uniform distribution between 40 and 70 miles per hour. $Initial\_Speed(v) \sim \mathscr{U}(40,70)$. The reward setting is done in such a way that the vehicle, when it brakes around the region of 5–10 meters from the obstacle, gets the maximum reward. During training, I

observe three variables: d (distance from the obstacle), v (velocity of the car), and mu (friction coefficient). $X = \{d, v, mu\}$ provides state information about the vehicle in the environment. During training, X becomes the input of the neural network, and brake value b is the output. The braking system is trained and tested in

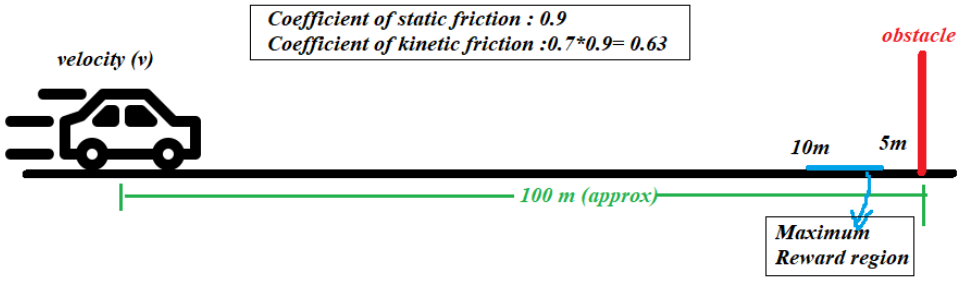


Figure 6.2: Training Scenario setup

this environmental setting. During *prediction*, I make some changes in the environment, which were never observed during the training process. I created two such scenarios: first, in place of a stationary obstacle, I used an obstacle that is moving toward the car at some small velocity(for this experiment, it is drawn from a uniform random distribution between 0.1-2 meters/second; refer figure 6.3, moving obstacle is represented by a walker). This situation was never observed during training, and the braking system does not respond to this changed scenario appropriately, which leads to a crash. In the second scenario, I kept the obstacle

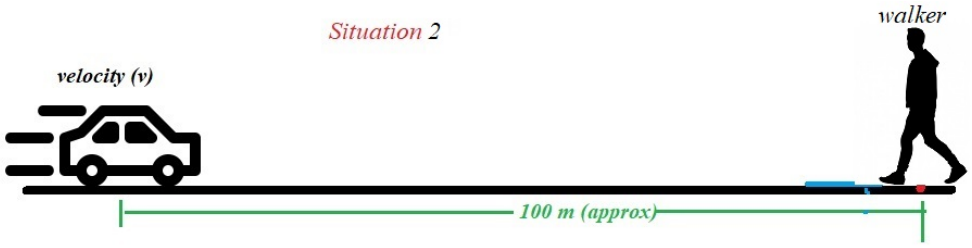


Figure 6.3: Prediction Scenario I

stationary but spawned the vehicle at a velocity of 75 miles per hour. It means the vehicle, when it detects the obstacle and invokes the braking system, has a velocity out of the training range (40–70 miles per hour). In this scenario also, the braking system fails to brake and leads to a crash because, during training, I never observed this speed.

In both cases, our RRRCF-based detection scheme detected these evolved situations and raised a flag for data being out of distribution. Figure 6.4 shows the results of the above three different test roll-outs. *Dispvalue* which is a scalar measure for each datapoint given to this RRRCF detector, provides an inference about the possibility of data being in or out of the training distribution. The threshold maximum *Dispvalue* derived for this set of experiments by running algorithm 4 is approximately 118.58, which is shown by the red horizontal

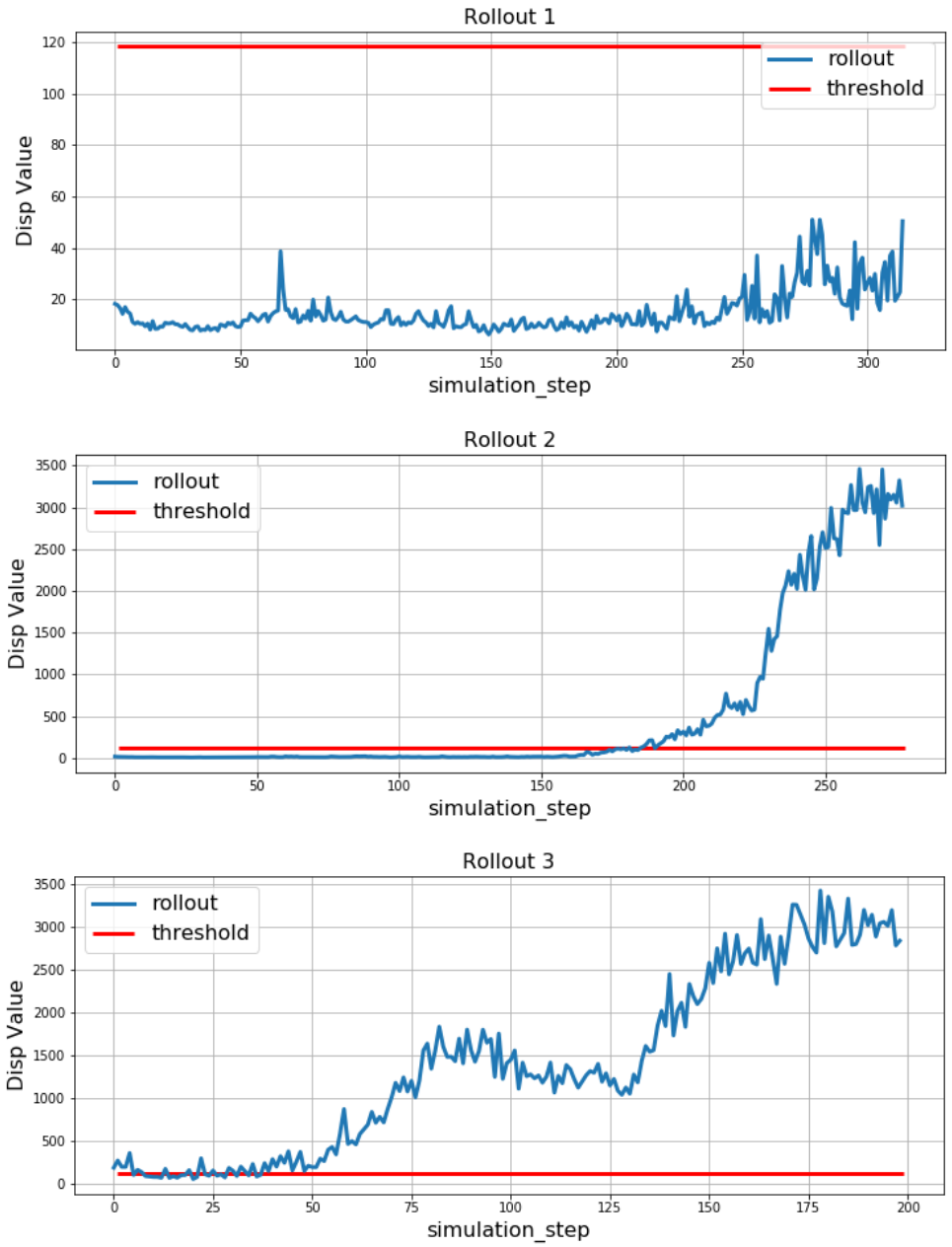


Figure 6.4: Three rollouts and respective output Dispvalue and threshold. Rollout1: Environment is same as training scenario (stationary obstacle with initial speed $\sim \mathscr{U}(40,70)$) ; Rollout2: Moving obstacle (walker) with initial speed $\sim \mathscr{U}(40,70)$; Rollout3: Stationary obstacle with initial speed (v) $\sim \mathscr{U}(40,70)$

line in figure 6.4. During test rollouts, at each step of the simulation, I estimated *DispValue* for the datapoint observed, i.e., $X = \{d, v, mu\}$ as the simulation progressed. In the *first* roll-out (Rollout 1), I did not make any changes in the environment, and the environment was similar to the training scenarios. The observed *Dispvalue* for each datapoint during rollout is always less than the threshold value, which implies there was

not any state observed during this rollout, which is OOD. In the case of the second roll-out (Rollout2), I made the obstacle move towards the car at a speed of 2 meters per second. It was observed that in the initial stage of simulation, the *Disp value* is less than the threshold value. It is because the obstacle is moving slowly and, at the initial stage, there is a very small change in the observed state. Once the car approaches the obstacle, the new state information X is slowly shifting from (i.e., getting far off) from the training data cluster, resulting in a higher and higher *Disp value* as the car reaches the end of the rollout. It is also observed that the change in Disp value is quite significant, which facilitates easy detection of such scenarios and fewer false alarms. In the case of the third roll-out (Rollout3), I kept the obstacle stationary, but I changed the initial spawning velocity of the car to 75 miles per hour. I observed *Dispvalue* higher than the threshold almost throughout the simulation step, the underlying reason the initial spawned velocity of the car is out of training range, and when the OOD detector observes this first initial state, it finds it as out-of-distribution. It can also be seen that the initially measured Dispvalue fluctuates around the threshold, which reflects very proximal OODs but with the progress of the simulation, this gap increases and results in a higher value of measured DispValue.

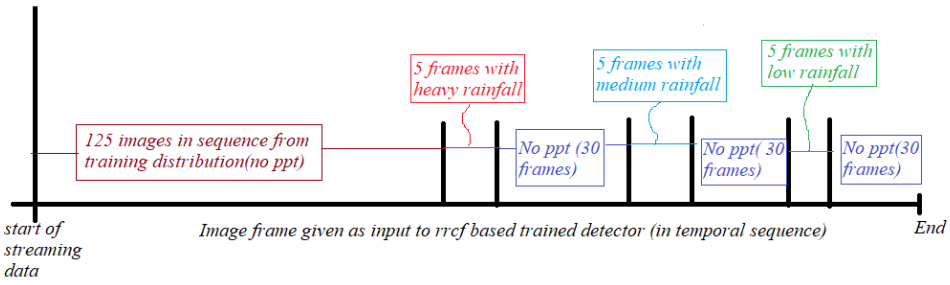


Figure 6.5: Image data stream given as an input to OOD detector (Stream I)

In the *second* experiment, I used the work done by Cai et al [25] for training a braking controller, but I trained it on a wider range of speeds (30–75 m/s) with different reward policy. Input in this case is the image scene from the car's camera, and the predicted output is the brake value. I trained and tested this model on image data generated with **no precipitation** condition in simulator CARLA. I collected all these images with no precipitation in several episodic rollouts and labeled them as non-OOD training scenario data. I used all these image data to build our RRRCF-based OOD detector by running algorithm 4. Threshold scalar *Disp Value* in this experiment was calculated and has a value of 23.26. For creating OOD scenarios, I changed the no-precipitation condition to three different participation conditions: *Case1:* Heavy precipitation, *Case2:* Medium precipitation, and *Case3:* Low precipitation (refer to figure 6.9) and collected image data captured by the camera. Our goal is to evaluate the performance of our OOD detector on the stream of image data. For this purpose, I created a stream of images called **stream I**, which has first 125 image frames with no precipitation scenario, then 5 frames of each of three different precipitation cases (heavy, medium, and low)

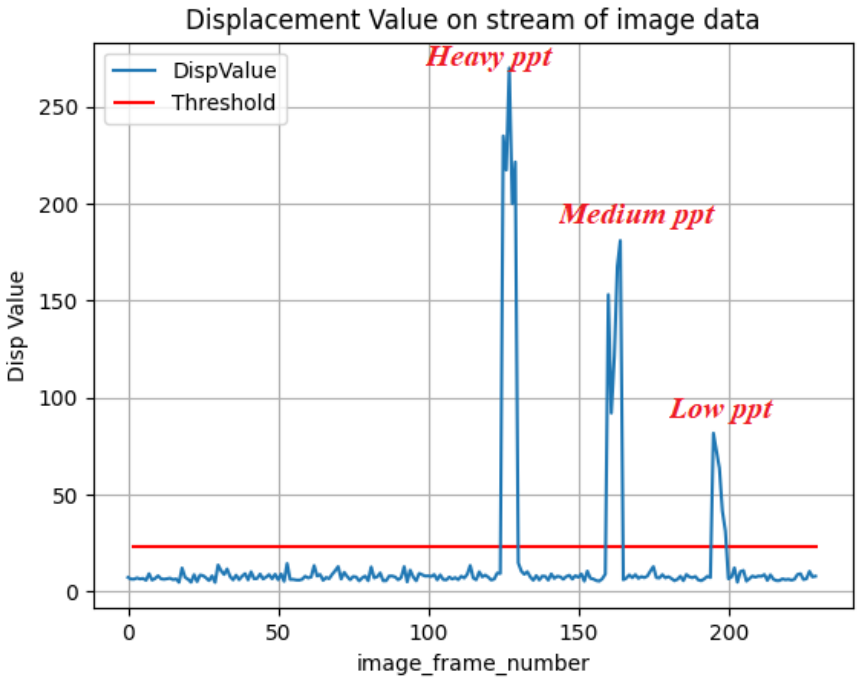


Figure 6.6: Output *Disp value* of stream I

with 30 intermittent frames of no precipitation in-between as shown in figure 6.5.

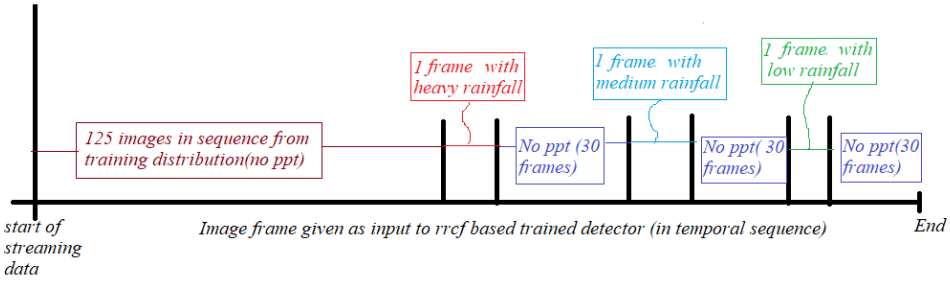


Figure 6.7: Image data stream given as input to OOD detector (Stream II)

With a given input image, if *Disp Value* is greater than the threshold value, then I declare it as OOD. Figure 6.6 shows the **Disp value** generated by the OOD detector for the data stream I. The red horizontal line represents the threshold value of the detector. It can be easily observed that the detector output is significantly higher than the threshold value in all three OOD cases (heavy, medium, and low precipitation) and lower for all non-OOD cases. I tried to find the performance of the detector in the context of point OOD, i.e., is the detector able to detect even a single OOD in the stream of non-OOD data? To find an answer to this question, I created another stream of image frames called **Stream II**. In stream II, I inserted a single frame of different

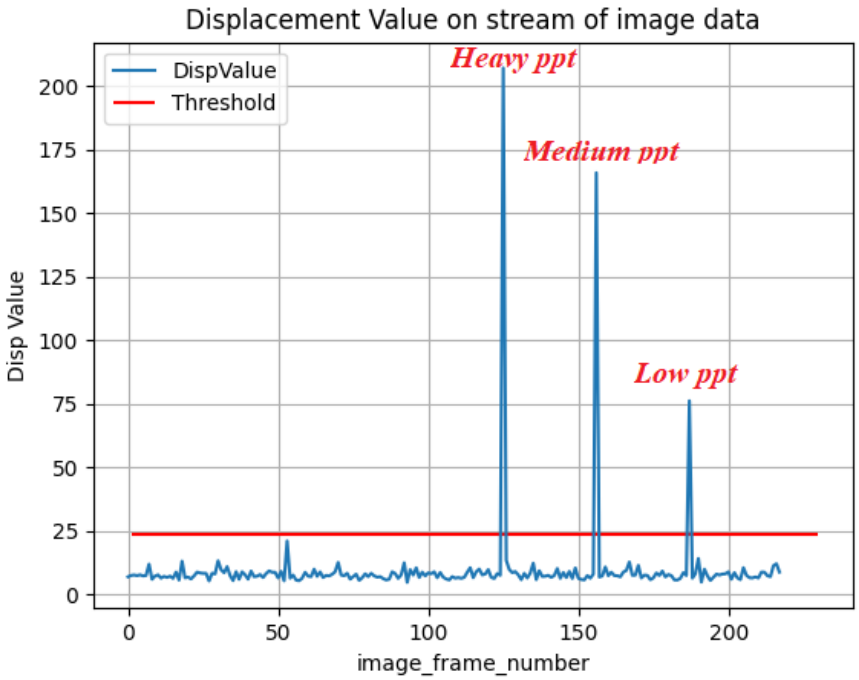


Figure 6.8: Output *Disp value* of stream II

precipitation cases (heavy, medium, and low) in a stream of image frames similar to training data, i.e., no precipitation (refer figure 6.7). I observed that the OOD detector can even detect a single OOD in a stream of no-OOD data (refer to figure 6.8).

6.4 Discussion

6.4.1 Time complexity analysis

The algorithmic complexity of the ODD detection process should be low during prediction time. Its time complexity depends upon the process required to make inferences on a data point by detector, i.e., 'Forest maintenance on stream'. Forest Maintenance on the stream of data involves two processes per data point: first, *insertion of a point in the tree data structure* and second, *deletion of a point from the tree data structure*. If I store each data point, i.e., the leaf of a tree, in a hash table, then the search process for locating a leaf will have a time complexity $O(1)$. Given a set of points Y and a point $x \in Y$, I construct a tree T on data Y. Consider if I want to delete a leaf x from the tree and produce tree $T(Y - x)$, I just need to remove the parent of x and make another child's parent pointer to its grandparent accordingly (refer to figure 6.10). This deletion process has a time complexity of $O(1)$. The total time complexity of the deletion process of a point (search and deletion) from a tree is $O(1)$. I can efficiently insert and delete data points into a random cut tree

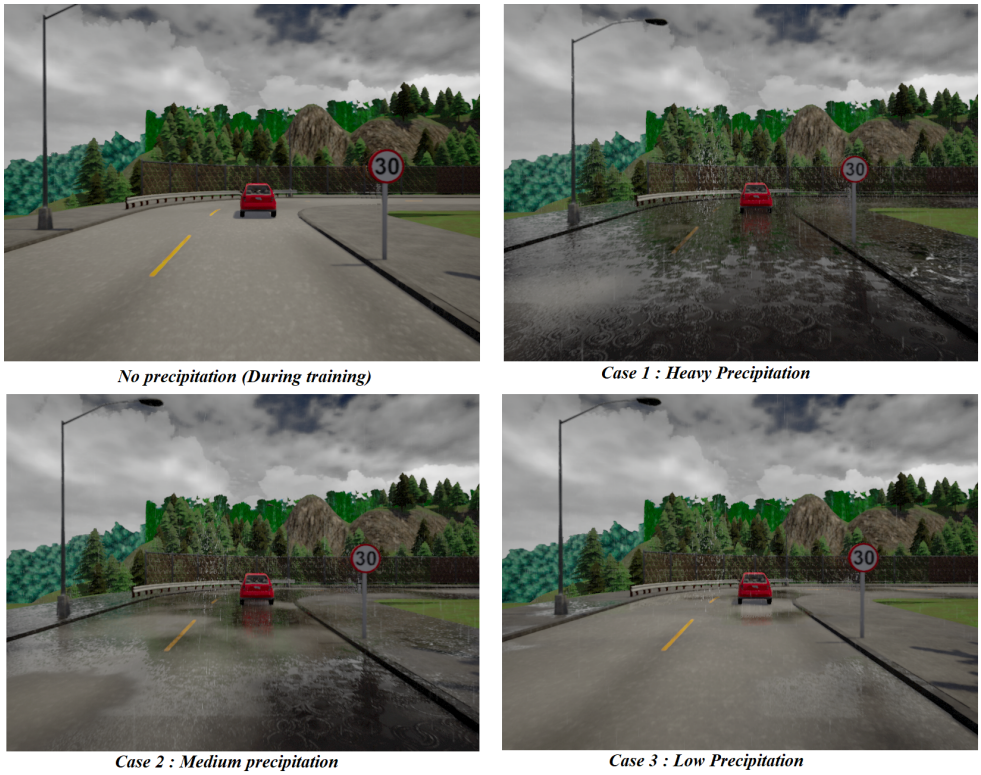


Figure 6.9: Different precipitation conditions generated during the training and prediction *(no precipitation, heavy precipitation, medium precipitation, low precipitation)*

data structure. Insertion of a point in a tree is a process when I have a tree T and I want to insert a leaf x to

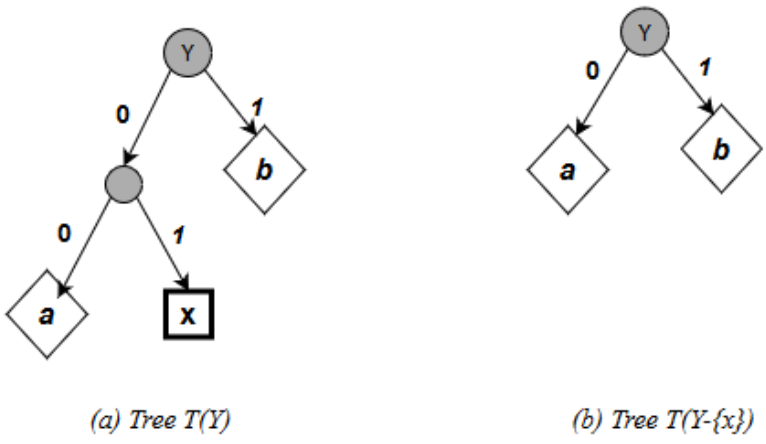


Figure 6.10: Deletion of a data point from a tree

the tree and produce tree $T(Y \cup x)$, where $x \notin Y$. In the worst case, the time complexity of this process would be $O(n)$, while in the average and best case it would be $O(logn)$ and $O(1)$ respectively, where n is the size of

the tree, i.e., the number of data-points used to create the tree. If I have m trees in a forest, then the total time complexity of the process is $O(mn), O(mlogn)$, and $O(m)$ in the worst, average, and best cases, respectively. The effect of m on the RRRCF creation, insertion, and deletion process can be made O(1) for time complexity analysis purposes as the process is completely parallelizable, i.e., I can do insertion and deletion on each tree on different threads or cores in parallel if a GPU- or multi-core-based parallel implementation of RRRCF is used. Consequently, the insertion process on the forest will have a time complexity of $O(n), O(logn)$, and $O(1)$ respectively for worst, average, and best-case scenarios. Here a reasonable question is how big the variable n can be. As the machine learning process is not very sample-efficient, a well-trained model results in large training data. For example, training a braking system perception LEC for a speed range variance of 10 km/hr (90 km/h to 100 km/h) in the Cai et al. [25] experimental setup took approximately 8160 images over a 100-meter distance. Similarly, in experiment set 1, our training data point was approximately 0.36 million. But for ODD detection, it is not required to use whole training data, and it is enough if I can make a sketch or outline of the data, i.e., I can ignore many data points that are densely clustered and use only those featured data points, which represent the outline of the data subspace.

I used our offline algorithm to select the number of data points that may represent the whole data. For the first set of experiments, our algorithm gave 11134 featured data points out of a total of 367035 data points. For the second set of experiments, our algorithm gave 1050 featured data points out of a total of 8197 image data points. Selection on n also depends upon the resources available. With the availability of a large number of parallel cores in modern GPU, it is also possible to increase m and reduce n and randomly choose a subset of elements out of n for creating a tree. With very large training data, it is possible to extract only a small segment of data to sketch the RRRCF outline.

6.4.2 Sensitivity on change in distribution

A good OOD detector should be able to detect even a very small deviation in input data from the training data distribution. For testing the sensitivity of the OOD detector, in experiment set 2, I used image frames generated from different weather conditions (different levels of precipitation: heavy, medium, and low. refer figure 6.9). The higher the precipitation level, the greater the deviation of the data from the training distribution. The empirical result suggests that this RRRCF-based detector can detect all three different levels of change in distribution. The empirical result also suggests that as test data points move away from the training data cluster in metric space, the anomaly score i.e. DispValue progressively increases in expectation.

6.4.3 Sensitivity on single OOD data in a stream of non-OOD data

Most work in the field of out-of-distribution has been done in the context of classification tasks in computer vision. OOD detection problems in machine learning can generally be classified in two parts- contextual OOD and point OOD. In the case of contextual OOD, the goal is to detect the anomaly in observed input data in the environmental context. These detection techniques do not take into consideration of dynamic nature of CPS and will not be suitable for its use. [25] proposed a method for OOD detection for CPS systems but this method needs a number of OOD data before making a conclusion about the OOD. It may fail in scenarios where I do not have a continuous stream of OOD data. rrrcf has tried to fill this gap, where it can detect even a single OOD data in the stream of non-OOD data.

6.4.4 Ease of training

Other OOD detectors, like GMM-based and VAE-based need to be tuned by selecting various hyper-parameters. These hyperparameter tuning and re-training is an iterative processes and the search for an effective hyperparameter is a slow process and time-consuming. In contrast, ***rrrcf*** has only two hyperparameters- first, *number of trees in the forest* and second, the *depth of each tree*. As rrrcf works on the expectation value of the anomaly score, any reasonable number of trees would work. I used 100 trees for both experimental setups. The second hyperparameter is actually derived from our algorithm 4 so, I do not need to tune it anyway while creating an rrrcf.

6.5 Training and experiment details

6.5.1 RRRCF training details

In the case of experiment set 1, I collected approximately 367035 training data points (Y), where each data point is a tuple of distance, velocity, and friction coefficient; $\{d, v, \mu\}$. For constructing the initial RRCF, I selected 1000 random data from a pool of total data, i.e., $Z = 1000$. After the construction of the initial rrcf data structure, the mean DispValue is calculated over all points in Z. It is the initial threshold value, and its value was approximately 6.4. This threshold was used for making decisions about the inclusion or discard of other points in training data ($Y \setminus Z$). After every 5000 data point, I recalculated the mean DispValue as our new threshold. The reason for repeatedly updating our threshold is to accommodate the changes made in the tree structure by the inclusion of data points. I recursively applied this process to the rest of the training data. After running algorithm 4, 10134 featured data points were selected, and the rest 355574 data points were rejected. These total $11134(10134 + 1000)$ data points were used to represent our whole data (367035) in the reduced RRCF data structure, which is approximately 3.03% percent of the whole data. In the case of experiment set 2, the same experimental setting as by Cai et al. [25] was used, and 8197 images were collected

Table 6.1. Total data points and featured data points for reduced robust random cut and percentage of total data points selected as featured datapoint

Scenario	Training points	Featured points	% reduction
1	367035	11134	96.97
2	8197	1050	87.2

for the training scenario, i.e., with no precipitation condition. For the creation of the initial RRCF, I selected 100 images i.e. $Z = 100$. Once the initial RRCF was created, the average threshold for Z was calculated and it was approximately 5.62. This threshold was used for making decisions about the inclusion/discard of other points in training data ($Y \setminus Z$). After every 100 data points, I recalculated the mean DispValue as our new threshold. After running algorithm 4, 950 datapoints were selected, and the rest 7147 datapoints were rejected. These featured 1050 datapoints were used to represent our whole data (8197 images) in the reduced RRCF data structure, which is approximately 12.8% percent of the whole data. For the construction of a random-cut tree, a modified version of the implementation of the tree written by Barto et al [13] was used.

6.5.2 Braking system training details

The braking system in experiment set 1 is trained using the actor-critic-based Deep Deterministic Policy Gradient algorithm with both actor and critic networks are three-layer neural networks, with 50 (layer1) and 30 (layer2) neurons in the hidden layer and 1 neuron in the output layer. For hidden layers in both cases, relu activation function was used and for the output layer, a sigmoid activation layer was used in actor and linear activation was used in critic. The model was trained using Adam [75] optimization method by tuning different learning rate for 5000 episodes. In the case of the experiment set 2, I used the same setting as done by Cai et al [25].

6.6 Summary of contributions

In this work, I looked into the problem of OOD from a geometric perspective, attempted to learn the Euclidean space occupied by the training data, and introduced a novel approach for this detection process using the Reduced Robust Random Cut Forest (RRRCF) data structure, which is also a white-box interpretable method for out-of-distribution detection. This method relies on a robust random-cut forest and derives a reduced robust random-cut forest data structure. From this data structure, I can calculate anomaly scores for every input data to the trained model. This anomaly score is used to make inferences about the new input data being in or out of the training data subspace. I also showed that this method can detect even a single OOD in a stream of non-OOD data. I demonstrated the effectiveness of this approach for both low- and high-dimensional input data spaces. I also discussed that a GPU-based parallel implementation of reduced RRCF

can significantly reduce the execution time. Parallel implementation of reduced rrcf may be used in real-time systems for real-time OOD detection.

CHAPTER 7

Conclusions

7.1 The future of AI-driven design optimization

System design has always been at the center of progress as well as challenges in human endeavors. To this end, for the last many decades, design processes have relied on using hand-crafted heuristics, human expert knowledge about the domain, running sequential decision-making processes that involve generating candidate design architectures, evaluating, selecting, and refining design options, and integrating the different components of designs until requirements are satisfied. This creates an exploration challenge to explore the high-dimensional design spaces that involve computationally costly design tools. Recent developments in AI hold potential promises in this sub-field. Concretely, it would be interesting to find the answer to the question: can AI-based optimization and search processes be faster than existing optimization approaches and consequently reduce inception to design time significantly? Based on the results of my research work, AI-based optimization algorithms showed promising results in finding optimal designs faster than traditional methods and having the better convergence properties. However, its scalability in handling large-scale, high-dimensional design spaces in these sample efficient optimization frameworks is the most exciting and still not known. There are many potential directions for research, like using AI for design space reconstruction by transforming these high-dimensional design spaces into low-dimensional spaces and using these low-dimensionally constructed design spaces for optimization, developing new algorithms that scale to high-dimensional problems, etc. On the other hand, there are many engineering domains that have very high computational complexity and act as bottlenecks in the optimization loop. In such cases, replacing these complex domains with a data-driven surrogate is desirable for speeding up the design exploration. During experimentation, superior performance of trained AI models, especially deep learning, is observed in learning these complex physics and solving them at negligible cost. The performance of surrogate-based optimization in complex engineering domains is exciting and can speed up the design search process. However, for training these data-driven surrogates, there is a need to generate a sheer amount of data that is not feasible, especially in complex engineering domains. In this scenario, research in data-efficient surrogate modeling would be a key player in removing the main bottleneck of data generation. I proposed and evaluated a new data-efficient surrogate training strategy called *student-teacher* with ε-HQS policy and when empirically evaluated, it performed better than all other baselines. This strategy showed that it can save days in the data generation and training processes for surrogate modeling. Our empirical result in this direction is phenomenal, and it opens

a multitude of future research directions, listed as follows:

1. To explore the possibility of design space reconstruction using modern AI methods to warp high dimensional design space to low dimensional design space and find possibilities for sample efficient design optimization for these high dimensional problems.

2. To find whether these AI-based sample efficient optimization methods can be parallelized and work on multi-objection constrained problems?

3. Can we provide a theoretical guarantee or bounds (upper and lower) on the performance of the ε-HQS approach?

4. Can a heuristic-based or AI-based method can be developed to design new batching policies or algorithms that can further improve sample efficiency?

Addressing these questions as future work can take this research sub-field in the forward direction. In all these experiments and research the toolchain *Anvil* can be really helpful for evaluating the design and testing new algorithms by simply extending the Python script to integrate new algorithms.

7.2 The future of AI-based system operation in the context of functional correctness

When an AI component is used during system operation, the functional correctness of this LEC behavior is critical. Since formal verification of most LECs is not possible, these models need to be evaluated probabilistically using simulation-based approaches. There are two aspects of functional correctness: in-distribution rare event failure and out-of-distribution detection. The standard approach to finding in-distribution rare event failure is Monte Carlo based and is highly inefficient in detecting rare event in-distribution failure cases. In such a case, there is a possibility of deploying a trained model that may give the illusion of correctness and safety. The guided search approach using the adversary failure indicator (AVF) is not robust and adaptive. Augmenting AVF with a generative model increases the robustness, flexibility, and adaptability of the search process and can include prior information, and information from other sources like domain expert knowledge, data from real-world collected statistics or other simulators, etc. The limitations of the generative approach are increased computational resources for training and inference of generative models and the difficult training process of these models in high dimensions. Research addressing these limitations would be an exciting direction.

Another aspect of functional correctness is OOD detection. OOD data point is significantly different from the training data, and predictions made by the trained model based on these data point can not be reliable. Despite various methods being developed for OOD detection, a robust and reliable OOD detector system is

still elusive. The proposed method of OOD relies on robust random-cut forest and derives a reduced robust random-cut forest data structure. From this data structure, we can calculate the anomaly score for every input data to the trained model that is used to make inferences about the new input data for being in or out of the training data subspace. Although our method can detect point OOD on streaming data, which is difficult to detect using a deviation-based detector, its computational complexity increases with the increasing dimensionality of the problem. A GPU-based parallel implementation of reduced RRCF can significantly reduce the execution time. A parallel implementation of reduced rrcf may be used in the real-time system for real-time OOD detection. Writing an open-source GPU and multi-core implementation of reduced RRCF and its evaluation for real-time performance can be an interesting direction for future work.

CHAPTER 8

List of Publications

Submitted Journals

- **Harsh Vardhan**, David Hyde, Umesh Timalsina, Peter Volgyesi, Janos Sztipanovits. "Sample-Efficient and Surrogate-Based Design Optimization of Underwater Vehicle Hulls", submitted to Journal of Computational Physics,April 2023 [147]

- **Harsh Vardhan**, Umesh Timalsina, Peter Volgyesi, Janos Sztipanovits. "Data efficient surrogate modeling for engineering design:Ensemble-free batch mode deep active learning for regression" submitted to Engineering Application of Artificial Intelligence (EAAI), Nov 2022. [154]

Published Journal

- **Harsh Vardhan**, Janos Sztipanovits. "Reduced Robust Random Cut Forest for Out-Of-Distribution detection in machine learning models" accepted at 2021 8th International Conference on Soft Computing and Machine Intelligence (ISCMI 2021), Egypt, Nov 26-28, 2021- in proceeding at International Journal of Machine Learning and Computing (IJMLC 2022).[152]

- Yogesh D Barve, Himanshu Neema, Zhuangwei Kang, **Harsh Vardhan**, Hongyang Sun, Aniruddha Gokhale. "EXPPO: EXecution Performance Profiling and Optimization for CPS Co-simulation-as-a-Service",Journal of Systems Architecture 118 (2021) : 102189[14]

Conference and Workshop Papers

- **Harsh Vardhan**, Janos Sztipanovits, "Search for universal minimum drag resistance underwater vehicle hull using CFD", in proceeding at The 29th International Conference on Computational & Experimental Engineering and Sciences (ICCES2023), May, 2023. [153]

- **Harsh Vardhan**, Janos Sztipanovits, "Constrained bayesian optimization for automatic underwater vehicle hull design", in proceeding at The ACM/IEEE DESTION 2023 at CPS-IoT week 2023, May, 2023. [155]

- **Harsh Vardhan**, Janos Sztipanovits, "Fusion of ML with numerical simulation for optimized propeller design", preprint at arXiv. (yet to submit) [157]

- **Harsh Vardhan**, Janos Sztipanovits, “Deep Active Learning for Regression Using ε-weighted Hybrid Query Strategy”, in proceeding at ReALML, International Conference on Machine Learning (ICML 2022), Baltimore, July, 2022. [151]

- **Harsh Vardhan**, Janos Sztipanovits, “Deep Learning-based FEA surrogate for sub-sea pressure vessel”, in proceeding at 2022 6th International Conference on Computer, Software and Modeling, Rome, Italy, July 21-23, 2022. [150]

- **Harsh Vardhan**, Janos Sztipanovits, “Rare event failure test case generation in Learning-Enabled-Controllers”, published at ACM 6th International Conference on Machine Learning Technologies (ICMLT 2021). South Korea, April 23-25, 2021. [149]

- **Harsh Vardhan**, Peter Volgyesi, Janos Sztipanovits.“Machine Learning Assisted Propeller Design”, published at 12th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS 2021). Nashville, TN, USA, May 19-21,2021. [156]

- Himanshu Neema,**Harsh Vardhan**, Carlos Barreto, Xenofon Koutsoukos. “Web-Based Platform for Evaluation of Resilient and Transactive Smart-Grids”, published at 7th IEEE Workshop on Modelling and Simulation of Cyber-Physical Energy Systems (MSCPES), 15 - 18 April 2019, Montreal, Canada.[105]

- **Harsh Vardhan**, Neal M Sarkar, Himanshu Neema, ”Modelling and Optimization of a longitudinally distributed Global Solar Grid”, published on Dec 22, 2019, at IEEE International Conference on Power Systems.[148]

Posters

- Himanshu Neema, **Harsh Vardhan**, Carlos Barreto, Xenofon Koutsoukos, “Design and Simulation Platform for Evaluation of Grid Distribution System and Transactive Energy”, published at 6th Annual Hot Topics in the Science of Security (HoTSoS), 2019.[104]

References

[1] Abdessalem, R. B., Nejati, S., Briand, L. C., and Stifter, T. (2018). Testing vision-based control systems using learnable evolutionary algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 1016–1026. IEEE.

[2] Abe, N. (1998). Query learning strategies using boosting and bagging. *Proc. of 15 Int. Conference on Machine Learning (ICML98)*, pages 1–9.

[3] Adams, B. M., Dalbey, K. R., Eldred, M. S., Gay, D. M., Swiler, L. P., Bohnhoff, W. J., Eddy, J. P., Haskell, K., et al. (2010). DAKOTA design analysis kit for optimization and terascale. Technical report, Sandia National Laboratories.

[4] Aghdam, H. H., Gonzalez-Garcia, A., Weijer, J. v. d., and López, A. M. (2019). Active learning for deep detection neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3672–3680.

[5] Aller, M., Mera, D., Cotos, J. M., and Villaroya, S. (2023). Study and comparison of different Machine learning-based approaches to solve the inverse problem in Electrical Impedance Tomographies. *Neural Computing and Applications*, 35(7):5465–5477.

[6] Alvarez, A., Bertram, V., and Gualdesi, L. (2009). Hull hydrodynamic optimization of autonomous underwater vehicles operating at snorkeling depth. *Ocean Engineering*, 36(1):105–112.

[7] An, J. and Cho, S. (2015). Variational autoencoder-based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18.

[8] Asadi, K. and Williams, J. D. (2016). Sample-efficient deep reinforcement learning for dialog control. *arXiv preprint arXiv:1612.06000*.

[9] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256.

[10] Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. (2020). BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*.

[11] Balcan, M.-F., Beygelzimer, A., and Langford, J. (2009). Agnostic active learning. *Journal of Computer and System Sciences*, 75(1):78–89.

[12] Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., and Yoo, S. (2014). The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, 41(5):507–525.

[13] Bartos, M. D., Mullapudi, A., and Troutman, S. C. (2019). RRCF: Implementation of the robust random cut forest algorithm for anomaly detection on streams. *Journal of Open Source Software*, 4(35):1336.

[14] Barve, Y. D., Neema, H., Kang, Z., Vardhan, H., Sun, H., and Gokhale, A. (2021). Exppo:Execution performance profiling and optimization for cps co-simulation-as-a-service. *Journal of Systems Architecture*, 118:102189.

[15] Ben Abdessalem, R., Nejati, S., Briand, L. C., and Stifter, T. (2016). Testing advanced driver assistance systems using multi-objective search and neural networks. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 63–74.

[16] Bhatnagar, S., Afshar, Y., Pan, S., Duraisamy, K., and Kaushik, S. (2019). Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545.

[17] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[18] Bishop, C. M. and Tipping, M. E. (2003). Bayesian regression and classification. *Nato Science Series sub-Series III Computer And Systems Sciences*, 190:267–288.

[19] Blanco, M., Atwood, J., Russell, S. M., Trimble, T., McClafferty, J. A., and Perez, M. A. (2016). Automated vehicle crash rate comparison using naturalistic data. Technical report, Virginia Tech Transportation Institute.

[20] Blank, J. and Deb, K. (2020). Pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509.

[21] Bouhlel, M. A., Hwang, J. T., Bartoli, N., Lafage, R., Morlier, J., and Martins, J. R. R. A. (2019). A python surrogate modeling framework with derivatives. *Advances in Engineering Software*, page 102662.

[22] Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

[23] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[24] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (2017). *Classification and regression trees*. Routledge.

[25] Cai, F. and Koutsoukos, X. (2020). Real-time out-of-distribution detection in learning-enabled cyber-physical systems. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 174–183, Los Alamitos, CA, USA. IEEE Computer Society.

[26] Carmichael, B. H. (1966). Underwater vehicle drag reduction through choice of shape. In *AIAA second propulsion joint specialist conference*.

[27] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58.

[28] Chen, J., Viquerat, J., and Hachem, E. (2019). U-net architectures for fast prediction of incompressible laminar flows. *arXiv preprint arXiv:1910.13532*.

[29] Chong, P., Ruff, L., Kloft, M., and Binder, A. (2020). Simple and effective prevention of mode collapse in deep one-class classification. *arXiv preprint arXiv:2001.08873*.

[30] Clark, C. E. (1961). The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162.

[31] Coney, W. B. (1989). *A method for the design of a class of optimum marine propulsors*. PhD thesis, Massachusetts Institute of Technology.

[32] Cordella, L. P., De Stefano, C., Tortorella, F., and Vento, M. (1995). A method for improving classification reliability of multilayer perceptrons. *IEEE Transactions on Neural Networks*, 6(5):1140–1147.

[33] Cressie, N. (2015). *Statistics for spatial data*. John Wiley & Sons.

[34] Crombecq, K., Laermans, E., and Dhaene, T. (2011). Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling. *European Journal of Operational Research*, 214(3):683–696.

[35] Dagan, I. and Engelson, S. P. (1995). Committee-based sampling for training probabilistic classifiers. In *Machine Learning Proceedings 1995*, pages 150–157. Elsevier.

[36] Damblin, G., Couplet, M., and Iooss, B. (2013). Numerical studies of space-filling designs: optimization of Latin hypercube samples and subprojection properties. *Journal of Simulation*, 7(4):276–289.

[37] Dong, H., Wu, Z., Chen, D., Tan, M., and Yu, J. (2020). Development of a whale-shark-inspired gliding robotic fish with high maneuverability. *IEEE/ASME Transactions on Mechatronics*, 25(6):2824–2834.

[38] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR.

[39] Eismann, S., Bartzsch, S., and Ermon, S. (2017). Shape optimization in laminar flow with a label-guided variational autoencoder. *arXiv preprint arXiv:1712.03599*.

[40] Epps, B., Chalfant, J., Kimball, R., Techet, A., Flood, K., and Chryssostomidis, C. (2009). Openprop: An open-source parametric design and analysis tool for propellers. In *Proceedings of the 2009 grand challenges in modeling & simulation conference*, pages 104–111.

[41] Fedorov, V. V. (2013). *Theory of optimal experiments*. Elsevier.

[42] Feng, D., Wei, X., Rosenbaum, L., Maki, A., and Dietmayer, K. (2019). Deep active learning for efficient training of a LIDAR 3d object detector. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 667–674. IEEE.

[43] Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[44] Fletcher, R. (2013). *Practical methods of optimization*. John Wiley & Sons.

[45] Forrester, A., Sobester, A., and Keane, A. (2008). *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons.

[46] Freund, Y., Seung, H. S., Shamir, E., and Tishby, N. (1992). Information, prediction, and query by committee. *Advances in neural information processing systems*, 5.

[47] Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep Bayesian active learning with image data. In *International Conference on Machine Learning*, pages 1183–1192. PMLR.

[48] Gardner, J. R., Kusner, M. J., Xu, Z. E., Weinberger, K. Q., and Cunningham, J. P. (2014). Bayesian optimization with inequality constraints. In *ICML*, volume 2014, pages 937–945.

[49] Genuer, R., Poggi, J.-M., Tuleau-Malot, C., and Villa-Vialaneix, N. (2017). Random forests for big data. *Big Data Research*, 9:28–46.

[50] Gertler, M. (1950). *Resistance experiments on a systematic series of streamlined bodies of revolution: for application to the design of high-speed submarines*. Navy Department, David W. Taylor Model Basin.

[51] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[52] Gorissen, D., Couckuyt, I., Demeester, P., Dhaene, T., and Crombecq, K. (2010). A surrogate modeling and adaptive sampling toolbox for computer-based design. *Journal of Machine Learning Research*, 11(Jul):2051–2055.

[53] GPy (since 2012). GPy: A Gaussian process framework in Python. http://github.com/SheffieldML/GPy.

[54] GPyOpt Authors, T. (2016). GPyOpt: A Bayesian optimization framework in python. http://github.com/SheffieldML/GPyOpt.

[55] Guha, S., Mishra, N., Roy, G., and Schrijvers, O. (2016). Robust random cut forest-based anomaly detection on streams. In *International conference on machine learning*, pages 2712–2721.

[56] Habeeb, R. A. A., Nasaruddin, F., Gani, A., Hashem, I. A. T., Ahmed, E., and Imran, M. (2019). Real-time big data processing for anomaly detection: A survey. *International Journal of Information Management*, 45:289–307.

[57] Hauer, F., Pretschner, A., and Holzmüller, B. (2019). Fitness functions for testing automated and autonomous driving systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 69–84. Springer.

[58] Hauptmann, A. G., Lin, W.-H., Yan, R., Yang, J., and Chen, M.-Y. (2006). Extreme video retrieval: joint maximization of human and computer performance. In *Proceedings of the 14th ACM international conference on Multimedia*, pages 385–394.

[59] Hawkins, D. M. (1980). *Identification of outliers*, volume 11. Springer.

[60] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[61] Hermes, D. (2017). Helper for bézier curves, triangles, and higher order objects. *The Journal of Open Source Software*, 2(16):267.

[62] Hertel, H. (1966). Full integration of vtol power plants in the aircraft fuselage(Full integration of VTOL lifting fans into aircraft fuselage). *1966.*, pages 65–96.

[63] Hess, J. L. (1976). On the problem of shaping an axisymmetric body to obtain low drag at large Reynolds numbers. *Journal of Ship Research*, 20(01):51–60.

[64] Hoffmann, C. M. and Kim, K.-J. (2001). Towards valid parametric CAD models. *Computer-Aided Design*, 33(1):81–90.

[65] Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[66] Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. (2017). Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*.

[67] Huang, Y., Liu, Z., Jiang, M., Yu, X., and Ding, X. (2019). Cost-effective vehicle type recognition in surveillance images with deep active learning and web data. *IEEE Transactions on Intelligent Transportation Systems*, 21(1):79–86.

[68] Hyder, M. J. and Asif, M. (2008). Optimization of location and size of opening in a pressure vessel cylinder using ANSYS. *Engineering Failure Analysis*, 15(1-2):1–19.

[69] Ïpek, E., McKee, S. A., Caruana, R., de Supinski, B. R., and Schulz, M. (2006). Efficiently exploring architectural design spaces via predictive modeling. *ACM SIGOPS Operating Systems Review*, 40(5):195–206.

[70] Jasak, H., Jemcov, A., Tukovic, Z., et al. (2007). OpenFOAM: A c++ library for complex physics simulations. In *International workshop on coupled methods in numerical dynamics*, volume 1000, pages 1–20. IUC Dubrovnik Croatia.

[71] Jin, R., Chen, W., and Sudjianto, A. (2002). On sequential sampling for global metamodeling in Engineering design. In *International design engineering technical conferences and computers and information in engineering conference*, volume 36223, pages 539–548.

[72] Jones, D. A., Chapuis, M., Liefvendahl, M., Norrison, D., and Widjaja, R. (2016). RANS simulations using OpenFOAM software. Technical report, Defence Science and Technology Group Fishermans Bend Victoria Australia.

[73] Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492.

[74] Joseph, V. R. and Hung, Y. (2008). Orthogonal-maximin Latin hypercube designs. *Statistica Sinica*, pages 171–186.

[75] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[76] Krishnamurthy, V. (2002). Algorithms for optimal scheduling and management of hidden markov model sensors. *IEEE Transactions on Signal Processing*, 50(6):1382–1397.

[77] Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise.

[78] Launder, B. E. and Spalding, D. B. (1983). The numerical computation of turbulent flows. In *Numerical prediction of flow, heat transfer, turbulence and combustion*, pages 96–116. Elsevier.

[79] Learn, S. SKL: Fundamental algorithms for scientific computing in python. https://scipy.org/.

[80] Learn, S. SKL: Machine learning in Python. https://scikit-learn.org/stable/.

[81] Lewis, D. D. and Gale, W. A. (1994). A sequential algorithm for training text classifiers. In *SIGIR'94*, pages 3–12. Springer.

[82] Li, X., Kesidis, G., Miller, D. J., Bergeron, M., Ferguson, R., and Lucic, V. (2021). Robust and Active Learning for Deep Neural Network Regression. *arXiv preprint arXiv:2107.13124*.

[83] Liefvendahl, M. and Stocki, R. (2006). A study on algorithms for optimization of Latin hypercubes. *Journal of statistical planning and inference*, 136(9):3231–3247.

[84] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[85] Lin, C. D., Mukerjee, R., and Tang, B. (2009). Construction of orthogonal and nearly orthogonal Latin hypercubes. *Biometrika*, 96(1):243–247.

[86] Lin, Y.-C., Hong, Z.-W., Liao, Y.-H., Shih, M.-L., Liu, M.-Y., and Sun, M. (2017). Tactics of adversarial attack on deep reinforcement learning agents. *arXiv preprint arXiv:1703.06748*.

[87] Lindenstrauss, W. J. J. (1984). Extensions of Lipschitz maps into a Hilbert space. *Contemp. Math*, 26:189–206.

[88] Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE.

[89] Liu, H., Ong, Y.-S., and Cai, J. (2018). A survey of adaptive sampling for global metamodeling in support of simulation-based complex engineering design. *Structural and Multidisciplinary Optimization*, 57(1):393–416.

[90] Liu, Y. (2004). Active learning with support vector machine applied to gene expression data for cancer classification. *Journal of chemical information and computer sciences*, 44(6):1936–1941.

[91] Loeppky, J. L., Moore, L. M., and Williams, B. J. (2012). Projection array based designs for computer experiments. *Journal of Statistical Planning and Inference*, 142(6):1493–1505.

[92] Lutz, T. and Wagner, S. (1998). Drag reduction and shape optimization of airship bodies. *Journal of Aircraft*, 35(3):345–351.

[93] Maroti, M., Hedgecock, W., and Volgyesi, P. (2022). Rapid Design Space Exploration with Constraint Programming. In *2022 IEEE Workshop on Design Automation for CPS and IoT (DESTION)*, pages 27–33. IEEE.

[94] McKay, M. D., Beckman, R. J., and Conover, W. J. (2000). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61.

[95] Melville, P. and Mooney, R. J. (2004). Diverse ensembles for active learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 74.

[96] Menter, F. R. (1992). Improved two-equation k-omega turbulence models for aerodynamic flows. Technical report.

[97] Mishra, S. K. (2006). Some new test functions for global optimization and performance of repulsive particle swarm method. *Available at SSRN 926132*.

[98] Močkus, J. (1975). On Bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer.

[99] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582.

[100] Morris, M. D. and Mitchell, T. J. (1995). Exploratory designs for computational experiments. *Journal of statistical planning and inference*, 43(3):381–402.

[101] Murphy, C., Kaiser, G. E., and Arias, M. (2007). An approach to software testing of machine learning applications.

[102] Myring, D. (1976). A theoretical study of body drag in subcritical axisymmetric flow. *Aeronautical quarterly*, 27(3):186–194.

[103] Nalisnick, E., Matsukawa, A., Teh, Y. W., Gorur, D., and Lakshminarayanan, B. (2018). Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136*.

[104] Neema, H., Vardhan, H., Barreto, C., and Koutsoukos, X. (2019a). Design and simulation platform for evaluation of grid distribution system and transactive energy. In *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, pages 1–2.

[105] Neema, H., Vardhan, H., Barreto, C., and Koutsoukos, X. (2019b). Web-based platform for evaluation of resilient and transactive smart-grids. In *2019 7th Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pages 1–6. IEEE.

[106] Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The computer journal*, 7(4):308–313.

[107] Nikishin, E., Izmailov, P., Athiwaratkun, B., Podoprikhin, D., Garipov, T., Shvechikov, P., Vetrov, D., and Wilson, A. G. (2018). Improving stability in deep reinforcement learning with weight averaging. In *Uncertainty in artificial intelligence workshop on uncertainty in Deep learning*.

[108] Pan, G., Ye, P., and Wang, P. (2014). A novel Latin hypercube algorithm via translational propagation. *The Scientific World Journal*, 2014.

[109] Parsons, J. S., Goodson, R. E., and Goldschmied, F. R. (1974). Shaping of axisymmetric bodies for minimum drag in incompressible flow. *Journal of Hydronautics*, 8(3):100–107.

[110] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12:2825–2830.

[111] Pholdee, N. and Bureerat, S. (2015). An efficient optimum Latin hypercube sampling technique based on sequencing optimisation using simulated annealing. *International Journal of Systems Science*, 46(10):1780–1789.

[probability] probability, T. F. TFP: Probabilistic models and deep learning on modern hardware (TPU, GPU). https://www.tensorflow.org/probability.

[pyDOE authors] pyDOE authors, T. pyDOE: Design of Experiments in Python. https://pythonhosted.org/pyDOE/.

[114] pygeo authors, T. (2016). GPyOpt: A graph algorithms in python. https://github.com/pyg-team/pytorch_geometric.

[pyMC authors] pyMC authors, T. pyMC: Python package for Bayesian statistical modeling focusing on advanced markov chain monte carlo (MCMC) and variational inference (VI) algorithms. https://github.com/pymc-devs/pymc.

[116] Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer.

[117] Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Gupta, B. B., Chen, X., and Wang, X. (2021). A survey of deep active learning. *ACM computing surveys (CSUR)*, 54(9):1–40.

[118] Reynolds, D. A. (2009). Gaussian Mixture Models. *Encyclopedia of biometrics*, 741.

[119] Riegel, J., Mayer, W., and van Havre, Y. (2016). FreeCAD.

[120] Rimmel, A. and Teytaud, F. (2014). A survey of meta-heuristics used for computing maximin Latin hypercube. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 25–36. Springer.

[121] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

[122] Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., Müller, E., and Kloft, M. (2018). Deep one-class classification. In *International conference on machine learning*, pages 4393–4402.

[123] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

[124] Sagaut, P., Terracol, M., and Deck, S. (2013). *Multiscale and multiresolution approaches in turbulence-LES, DES and Hybrid RANS/LES Methods: Applications and Guidelines*. World Scientific.

[125] Sagi, O. and Rokach, L. (2018). Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1249.

[126] Schröder, C. and Niekler, A. (2020). A survey of active learning for text classification using deep neural networks. *arXiv preprint arXiv:2008.07267*.

[127] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

[128] Schweyher, H., Lutz, T., and Wagner, S. (1996). An optimization tool for axisymmetric bodies of minimum drag. In *2nd international airship conference, Stuttgart/Friedrichshafen*, pages 3–4.

[129] Sener, O. and Savarese, S. (2017). Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*.

[130] Settles, B. (2009). Active learning literature survey.

[131] Settles, B., Craven, M., and Ray, S. (2007). Multiple-instance active learning. *Advances in neural information processing systems*, 20.

[132] Seung, H. S., Opper, M., and Sompolinsky, H. (1992). Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294.

[133] Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55.

[134] Shigley, J. E. (1972). *Mechanical engineering design*. McGraw-Hill Companies.

[135] Smailović, J., Grčar, M., Lavrač, N., and Žnidaršič, M. (2014). Stream-based active learning for sentiment analysis in the financial domain. *Information sciences*, 285:181–203.

[136] Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. W. (2012). Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE transactions on information theory*, 58(5):3250–3265.

[137] Stevenson, P., Furlong, M., and Dormer, D. (2007). AUV shapes-combining the practical and hydrodynamic considerations. In *Oceans 2007-Europe*, pages 1–6. IEEE.

[138] Stocki, R. (2005). A method to improve design reliability using optimal Latin hypercube sampling. *Computer Assisted Mechanics and Engineering Sciences*, 12(4):393.

[139] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[140] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

[141] Tang, C., Garreau, D., and von Luxburg, U. (2018). When do random forests fail? In *Advances in neural information processing systems*, pages 2983–2993.

[142] Tong, S. (2001). *Active learning: theory and applications*. Stanford University.

[143] Tong, S. and Chang, E. (2001). Support vector machine active learning for image retrieval. In *Proceedings of the ninth ACM international conference on Multimedia*, pages 107–118.

[144] Tsymbalov, E., Panov, M., and Shapeev, A. (2018). Dropout-based active learning for regression. In *International conference on analysis of images, social networks and texts*, pages 247–258. Springer.

[145] Uesato, J., Kumar, A., Szepesvari, C., Erez, T., Ruderman, A., Anderson, K., Heess, N., Kohli, P., et al. (2018). Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures. *arXiv preprint arXiv:1812.01647*.

[146] Van Dam, E. R., Husslage, B., and Den Hertog, D. (2010). One-dimensional nested maximin designs. *Journal of Global Optimization*, 46(2):287–306.

[147] Vardhan, H., Hyde, D., Timalsina, U., Volgyesi, P., and Sztipanovits, J. (2023a). Sample-Efficient and Surrogate-based Design Optimization of Underwater Vehicle Hulls. *arXiv preprint arXiv:2304.12420*.

[148] Vardhan, H., Sarkar, N. M., and Neema, H. (2019). Modeling and optimization of a longitudinally-distributed global solar grid. In *2019 8th International Conference on Power Systems (ICPS)*, pages 1–6. IEEE.

[149] Vardhan, H. and Sztipanovits, J. (2021). Rare event failure test case generation in Learning-Enabled-Controllers. In *2021 6th International Conference on Machine Learning Technologies*, pages 34–40.

[150] Vardhan, H. and Sztipanovits, J. (2022a). Deep learning-based FEA surrogate for sub-sea pressure vessel. *arXiv preprint arXiv:2206.03322*.

[151] Vardhan, H. and Sztipanovits, J. (2022b). DeepAL for regression using epsilon-weighted Hybrid query strategy. *arXiv preprint arXiv:2206.13298*.

[152] Vardhan, H. and Sztipanovits, J. (2022c). Reduced Robust Random Cut Forest for out-of-distribution detection in machine learning models. *arXiv preprint arXiv:2206.09247*.

[153] Vardhan, H. and Sztipanovits, J. (2023). Search for universal minimum drag resistance underwater vehicle hull using CFD. *arXiv preprint arXiv:2302.09441*.

[154] Vardhan, H., Timalsina, U., Volgyesi, P., and Sztipanovits, J. (2022). Data efficient surrogate modeling for engineering design: Ensemble-free batch mode deep active learning for regression. *arXiv preprint arXiv:2211.10360*.

[155] Vardhan, H., Volgyesi, P., Hedgecock, W., and Sztipanovits, J. (2023b). Constrained Bayesian optimization for automatic underwater vehicle hull design. In *Proceedings of Cyber-Physical Systems and Internet of Things Week 2023*, pages 116–121.

[156] Vardhan, H., Volgyesi, P., and Sztipanovits, J. (2021). Machine learning assisted propeller design. In *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, pages 227–228.

[157] Vardhan, H., Volgyesi, P., and Sztipanovits, J. (2023c). Fusion of ML with numerical simulation for optimized propeller design. *arXiv preprint arXiv:2302.14740*.

[158] Viana, F. A., Haftka, R. T., and Steffen, V. (2009). Multiple surrogates: how cross-validation errors can help us to obtain the best predictor. *Structural and Multidisciplinary Optimization*, 39(4):439–457.

[159] Viana, F. A., Venter, G., and Balabanov, V. (2010). An algorithm for fast optimal Latin hypercube design of experiments. *International journal for numerical methods in engineering*, 82(2):135–156.

[160] Vose, M. D. (1999). *The simple genetic algorithm: foundations and theory*. MIT press.

[161] Wegener, J. and Bühler, O. (2004). Evaluation of different fitness functions for the evolutionary testing of an autonomous parking system. In *Genetic and Evolutionary Computation Conference*, pages 1400–1412. Springer.

[162] Wei, K., Iyer, R., and Bilmes, J. (2015). Submodularity in data subset selection and active learning. In *International conference on machine learning*, pages 1954–1963. PMLR.

[163] Wilcox, D. C. et al. (1998). *Turbulence modeling for CFD*, volume 2. DCW industries La Canada, CA.

[164] Winey, N. E. (2020). *Modifiable stability and maneuverability of high speed unmanned underwater vehicles (UUVs) through bioinspired control fins*. PhD thesis, Massachusetts Institute of Technology.

[165] Wolf, G. W. (2011). Facility location: concepts, models, algorithms and case studies. Series: Contributions to Management science: edited by Zanjirani Farahani, Reza and Hekmatfar, Masoud, Heidelberg, Germany, Physica-Verlag, 2009, 549 pp.,€ 171.15, $219.00,£ 144.00, isbn 978-3-7908-2150-5 (hardprint), 978-3-7908-2151-2 (electronic).

[166] Xiong, F., Xiong, Y., Chen, W., and Yang, S. (2009). Optimizing Latin hypercube design for sequential sampling of computer experiments. *Engineering Optimization*, 41(8):793–810.

[167] Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

[168] Yamamoto, I. (2015). Research on next autonomous underwater vehicle for longer distance cruising. *IFAC-PapersOnLine*, 48(2):173–176.

[169] Zedan, M. F. and Dalton, C. (1979). Viscious Drag computation for Axisymmetric bodies at high Reynolds numbers. *Journal of Hydronautics*, 13(2):52–60.

[170] Zhang, M., Zhang, Y., Zhang, L., Liu, C., and Khurshid, S. (2018). DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 132–142. IEEE.

[171] Zhang, Y., Lease, M., and Wallace, B. (2017). Active discriminative text representation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

[172] Zhdanov, F. (2019). Diverse mini-batch active learning. *arXiv preprint arXiv:1901.05954*.

[173] Zhilinskas, A. (1975). Single-step Bayesian search method for an extremum of functions of a single variable. *Cybernetics*, 11(1):160–166.

[174] Zhou, H., Li, W., Kong, Z., Guo, J., Zhang, Y., Yu, B., Zhang, L., and Liu, C. (2020). Deepbillboard: Systematic physical-world testing of autonomous driving systems. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 347–358. IEEE.