Data-Driven Algorithms for Smart Transportation Systems

By

Michael Wilbur

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:

Abhishek Dubey, Ph.D.

Ayan Mukhopadhyay, Ph.D.

Samitha Samaranayake, Ph.D.

Aniruddha Gokhale, Ph.D.

Daniel Work, Ph.D.

Jonathan Sprinkle, Ph.D.

Acknowledgements

This research was supported in part by the National Science Foundation under grant CNS-1952011 and award numbers 1818901 and 1647015, as well as the Department of Energy under award numbers DE-EE0008467 and DE-EE0009212.

I am grateful to my advisor Dr. Abhishek Dubey for his mentorship, support and encouragement during my graduate studies. His guidance has been instrumental not only to my work, but also in my development as a researcher and person. I am also thankful for my committee members Dr. Ayan Mukhopadhyay, Dr. Samitha Samaranayake, Dr. Aniruddha Gokhale, Dr. Daniel Work and Dr. Jonathan Sprinkle. I would also like to thank Philip Pugliese for facilitating our collaboration with the Chattanooga Area Regional Transportation Agency (CARTA). Philip's guidance and collaboration has been invaluable throughout the course of this work. Thank you to Lisa Suttles as well for her important insights regarding transit operations at CARTA and for her patience as we developed our software solutions. I would also like to thank Max Coursey for his help with the software implementation and Sophie Pavia for her help with the software, as well as the preparation, documentation and training materials for the CARTA pilot.

I would like to give a special thank you to my friends and collaborators in the SCOPE-lab including Dr. Ava Pettet, Dr. Scott Eisele and Baiting Luo for their support over the past several years. Thank you to all of my other collaborators throughout this work including Dr. Aron Laszka. I would especially like to thank Dr. Ying Chen and Dr. Michael Hyland for their mentorship during my masters studies.

I would like to thank my parents Mike and Linda, my brothers Mark and Matt, and my close friends - Anthony Pennella, Brian Gidcumb, Tom Hansel and Nick Slocum for all of their moral support and guidance through the years.

Above all, I would like to thank my wife Dr. Catherine Wilbur for her love and constant support throughout this journey. Thank you for being my best friend. This endeavor would not have been possible without you. And to our son Patrick - may your blessings outnumber the shamrocks that grow and may trouble avoid you wherever you go. We love you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

There are more than 7,000 public transit agencies in the U.S. (and many more private agencies), and together, they are responsible for serving 60 billion passenger miles each year. A well-functioning transit system fosters the growth and expansion of businesses, distributes social and economic benefits, and links the capabilities of community members, thereby enhancing what they can accomplish as a society [3, 4, 5]. Furthermore, large-scale adoption of smart phones and sensing technologies have revolutionized urban mobility in recent years. Led by companies such as Uber, Lyft and Via, new on-demand transportation options such as ride-share, e-scooters, and bike-share have been introduced alongside existing transportation services provided by public transit agencies. The explosion in transportation options, and the complicated relationship between public and private offerings present myriad new challenges in the design and operation of these systems. As transit agencies try to adapt to their new environment it has become critically important to identify new ways to address the complex, and often competing, operational objectives that complicate the implementation of efficient services. How these entities address these challenges will have an outsized impact on our ability to strengthen urban communities, address the climate challenge, and foster equitable growth going forward.

Urban mobility is a broad field with many stakeholders. It encompasses public and private transportation operators, the residents and users who rely on these services, as well as regional and local municipalities. Given the complex, and often competing, operational objectives of these various stakeholders, it is impossible to identify an exact definition of what constitutes a ”well-functioning” transportation system. Therefore, in this work we narrow the focus to the view-point of transit authorities and public transit in general. There are multiple reasons to focus on this view. First, affordable public transit is the backbone of many communities. A well designed public transit system provides critical service to communities most in need where residents are least likely to own a personal vehicle. Additionally, services such as paratransit provide mobility options for handicapped and elderly residents that may not be able to access regular fixed-line public transit or operate their own vehicle. Second, as the number of private transportation modalities increases, the transportation landscape has become more fragmented [6]. As public entities, transit agencies are tasked with providing service in a way that maximizes social benefits to residents and the city as a whole. Additionally, in many urban areas transit agencies also have some degree of authority over the various operators in the region. This allows transit agencies to view the global transportation system holistically and presents opportunities to optimize services at the system-level.

While there are many ways to assess the performance of transportation systems, we largely focus on evaluating these systems in the context of optimizing three system-level objectives - *utilization* (i.e. ridership), *efficiency* (i.e. reducing operational costs) and *coverage* (expanding service geographically). Increasing utilization requires learning mobility patterns over wide geographical areas and adapting systems to better meet the demand for mobility. It also requires flexible, demand-responsive mobility options that can adapt to demand in real-time and thus better serve potential passengers. Additionally, more efficient systems alleviate the impact on the environment by reducing emissions and can free resources by reducing costs. Lastly, it is important to discuss optimization in the context of ridership versus coverage, the latter of which is an important consideration in the design of equitable and fair transportation. While the problem of ridership versus coverage is open-ended, there are ways in which these dimensions can be modelled such that transit agencies can better optimize over the objectives that matter most to them.

Fundamentally, the design of a well-functioning transit system requires solving complex combinatorial optimization problems related to planning and real-time operations. These problems span many well studied fields, from classical line planning to offline vehicle routing problems (VRPs) and online, dynamic vehicle routing problems (DVRPs). This work focuses on the design and implementation of *real-time* transportation systems. In particular, we focus on *demand-responsive* transportation. Demand-responsive transportation can be thought of as any transit solution that includes a component that adapts to changes in demand, the environment and available resources in real-time. Demand-responsive transportation has great potential to improve utilization and coverage by expanding public transit service through flexible door-to-door Mobility-on-Demand (MoD), as well as improve efficiency by making existing public transit more flexible and responsive [7, 8, 9, 10, 11].

Demand-responsive applications can be framed as resource allocation problems in which demand (users, trip requests) must be matched to available resources (vehicles, e-scooters, e-bikes) over time. Therefore, these systems require making decisions in real-time over large geographical areas and computationally intractable state-action spaces. Due to the intractability of these problems, traditional analytical methods fall short. Therefore, transit agencies have turned to computational approaches that enable large-scale, data-driven optimization. Artificial intelligence (AI)-based methods can address this problem by learning complex abstractions of vast volumes of data to aid the decision-making process in a computationally efficient manner at scale. In this way we aim to derive methods for which transit agencies can utilize the vast sums of data generated from the explosion in network-connected devices throughout urban areas to create more flexible and efficient services.

We propose a computational framework for demand-responsive transportation that tackles problems related to data processing and integrity, prediction, decision-making under uncertainty, deployment and software design. We model the problem as a real-time resource allocation problem that can be decomposed into four concrete atomic sub-problems - 1) Planning, 2) Prediction, 3) Deployment and 4) Software Design. To highlight our solution and approach we tackle various challenges in the design and operation of demand-responsive paratransit, microtransit and fixed-line transit faced by our partner agency, the Chattanooga Area Regional Transportation Authority (CARTA). Finally, we integrated the components of our framework into a software ecosystem for real-time MoD applications called SmartTransit-AI that is currently in use by CARTA.

1.1 Planning

The focal point of our pipeline for demand-responsive transportation is the algorithms for *planning*. Recall that we can frame demand-responsive transportation as a large-scale resource allocation problem where resources (vehicles, e-scooters, bike-share ect.) must be matched to service demand (user trip requests). In this context, planning refers to the algorithms used to assign these resources to service demand. Take for example ride-pooling systems such as microtransit, paratransit, UberPool and Lyft Line. Ride-pooling is a form of MoD that provides direct transportation between origin and destination for a user that requests a trip through a smart phone application, similar to ride-hailing. However, where ride-hailing only allows for a one-to-one mapping between resources (drivers) and demand (passengers), ride-pooling allows for users to share trips. The key idea is that by grouping riders with similar trips together, ride-pooling can increase efficiency compared to single-rider services [6, 12].

Ride-pooling can be framed as a dynamic vehicle routing problem (DVRP), which extends the offline VRP to real-time settings. All DVRPs share a basic set of constraints related to vehicle capacities and pickup and dropoffs. Depending on the domain, serviceability requirements can be included as constraints as well by setting hard limits on maximum waiting time or passenger detour time. The goal is then to design a planning algorithm that will match trip requests to vehicles to optimize a system-level utility function that is evaluated over the course of a day while meeting the constraints at each decision point. These decisions are made sequentially, in real-time. Therefore, it is important to design efficient decision procedures that are societal-scale and robust to changing demand and environment conditions.

Due to the scale of these systems, current state-of-the-art sequential planners are typically myopic in that they aim to maximize immediate reward [13, 14]. Myopic controllers therefore only consider the impact of decisions at time t until time $t+1$. However, there are numerous factors which can change in the near and long term that can lead to significant inefficiency by not accounting for future demand and future system states. This includes changing traffic conditions, weather, and the distribution of future requests. All of these factors can be predicted and incorporated in non-myopic planners but are ignored in myopic planning.

In the cases where non-myopic solutions are presented, they typically rely on value functions learned offline through simulation which can become stale in non-stationary environments as they evolve [15, 16]. Non-myopic methods can be classified as offline, online or hybrid. In the case of offline non-myopic planning a value function is learned directly from data through simulation which is then accessed at inference time to evaluate actions in the context of future demand. While these methods are highly scalable, the models rely on learned representations of expected demand and environment dynamics which can become stale in highly dynamic and non-stationary environments. This can cause sub-optimal planning precisely when decision-making is of most importance: for instance during spikes in demand from large-scale sport events or when normal operations are disrupted due to extreme weather.

Therefore, we aim to study what is required for *adaptive* planning for DVRPs. We focus on adaptive planning for paratransit services, which are a socially important MoD service that extends public transportation access for users with disabilities or handicaps that can not reliabilly use fixed-line public transit. We propose a fully online approach that is robust to changing environmental dynamics. Our approach leverages the structure of the paratransit DVRP to design heuristics that can sample promising actions that are then evaluated non-myopically through online search and a generative model. Our contributions are presented in Chapter 3.

1.2 Prediction

Generative models are just one data-driven method which can aid the decision-making process during planning. Similarly, *predictive* models have great potential to improve optimization procedures by learning efficient representations from vast volumes of data. There are primarily two ways in which predictive models can aid the planning process. First, predictive models the system dynamics in a way that can be used directly in the optimization algorithms to model future scenarios or state. For example, an online DVRP algorithm might require estimates of future traffic conditions or travel times which can be learned through supervised learning. Second, these methods can be used to directly learn policy or utility functions through offline simulation. The learned functions can evaluate current actions or state in the context of expected future reward, and therefore are non-myopic. These methods collectively fall under the field of reinforcement learning (RL).

Predictive models require large amounts of high quality data for training offline, which is not always available. An interesting case study is the problem of designing predictive models for energy and emissions for mixed-fleet transit. A mixed-fleet occurs when a transit agency is tasked with managing a variety of classes including internal combustion vehicles (ICEVs), electric vehicles (EVs), and hybrid vehicles (HVs). This is a common situation as many transit agencies, including CARTA, gradually introduce new fuel-efficient EVs and HVs to their existing fleet of ICEVs. In managing mixed-fleets, agencies require accurate predictions of energy use for optimizing the assignment of vehicles to transit routes, scheduling charging, and ensuring that emission standards are met. However, the ad-hoc nature of vehicle acquisition results in not only a mix of vehicle classes (ICEV, HV, and EV), but also different vehicle models within each class. For example, CARTA manages a total of six ICEV models, two HV models and two EV models. Current state-of-the-art is to train separate predictive models for each vehicle model [7, 17]. However, as datasets can be highly imbalanced between vehicle classes and models, predictive performance can vary greatly and ultimately negatively affect downstream optimization applications. Therefore, in Chapter 4 we present our work on utilizing multi-task and inductive transfer learning overcome these challenges in the context of energy and emissions prediction for mixed-fleets. Our work shows that we can leverage broader generalizable patterns that govern the consumption of energy and vehicle emission to improve performance in this setting.

1.3 Deployment

Its also important to note that current research typically frames decision-making as an algorithmic or artificial intelligence (AI) problem, which ignores the computation frameworks and resources on which these systems will be deployed. In this context, decentralized operations has the potential to reduce latency by utilizing compute at the network edge, can remain operation when access to the cloud is removed, and can improve user privacy and security related to large-scale systems [18, 19, 20].

In this work we argue that to fully utilize recent advances in edge computing for the transportation domain, algorithms and learning should be jointly designed and adapted for these new environments in a way that captures the benefits of these systems. In particular, we focus on the potential of roadside-unit (RSU) networks for real-time transportation systems. We address two key challenges in this setting. In Chapter 5 we present a decentralized, congestion-aware route planning algorithm for private RSU-edge networks. We leverage recent advances in federated learning to collaboratively learn shared prediction models online and investigate our approach with a simulated case study using data in Nashville, Tennessee. Second, in Chapter 6 we address key challenges related to anomaly detection and security in RSU deployments. We propose a multi-tiered anomaly detection framework for fast, real-time detection of orchestrated data-integrity attacks in which an adversary attempts to compromise a subset of sensors with the goal of maximizing the effect of the attack on the transportation system.

1.4 Software

Throughout this work we present a variety of solutions to critically important technical challenges related to demand-responsive and MoD systems. Ultimately, the goal of all research is to create real-world impact. In this work, we set forth with the goal of improving operations for transit agencies in a way that maximizes social benefits. Indeed, we identified three ways to achieve these aims - by optimizing 1) utilization, 2) efficiency and 3) coverage. We hypothesized that demand-responsive systems could achieve these aims by allowing public transit agencies to provide more flexible and adaptive real-time services. To do this we created a framework, and resolved a variety of real-world challenges facing transit agencies through 1) planning, 2) prediction and 3) deployment. To facilitate the use of these contributions, we developed SmartTransit-AI which is a software framework for MoD services.

This software framework addresses an important limitation in real-time transportation research. As it currently stands, the operations software used by transit agencies is provided by various private companies. These offerings are closed in that the agencies themselves have limited access to the underlying algorithms and don't have an easy way to apply agency-specific constraints. These one-size-fit-all solutions severely limit the ability of transit agencies to customize their services to their unique populations as well as implement state-of-the-art algorithms from the academic and open-source communities.

SmartTransit-AI fits within the broader category of Mobility-as-a-Service (MaaS) software - however with two important distinctions. First, SmartTransit-AI is more than a wrapper joining existing mobility options. It allows direct access to modular optimization components which can be extended or adapted as new algorithms are developed over time. Second, we provide an easy to use configuration framework that gives transit agencies full control over system constraints and objectives.

SmartTransit-AI was designed with critical input from CARTA and was used to pilot our own set of algorithms in CARTA's paratransit operations. The SmartTransit-AI framework for MoD and the results of the paratransit pilot in Chattanooga, Tennessee is presented in Chapter 7.

Chapter 2

Background

In this Chapter, we present the relevant background required to study real-time, data-driven transportation systems. First, we discuss the data collected from our partner agencies CARTA (Chattanooga, Tennessee) and WeGo (Nashville, Tennessee) in Section 2.1. Second, we discuss our cloud-based data management platform for ingesting, processing and storing the data in Section 2.2. Finally, in Section 2.3 we provide a high-level description and formulation of real-time decision-making in the context of demand-responsive, data-driven transportation.

The work comprising this Chapter has been published in Proceedings of the Workshop on AI for Urban Mobility at the 35th AAAI Conference on Artificial Intelligence (AAAI-21), 2021 [1] and book chapter to appear in Vorobeychik, Yevgeniy., and Mukhopadhyay, Ayan., (Eds.). (2023). Artificial Intelligence and Society. ACM Press (pre-print [21]).

- Michael Wilbur, Philip Pugliese, Aron Laszka, and Abhishek Dubey. Efficient data management for intelligent urban mobility systems. In *Proceedings of the Workshop on AI for Urban Mobility at the 35th AAAI Conference on Artificial Intelligence (AAAI-21)*, 2021

- Michael Wilbur, Amutheezan Sivagnanam, Afiya Ayman, Samitha Samaranayeke, Abhishek Dubey, and Aron Laszka. Artificial intelligence for smart transportation. *arXiv preprint arXiv:2308.07457*, 2023

2.1 Data Sources

The "data" component refers to the need for high-quality data sources that can be used to generate datasets for training predictive models and to generate data online for real-time

Table 2.1: Data sources.

Data	Source	Frequency	Scope	Features	Schema/Format
Diesel vehicles	ViriCiti and Clever Devices	1 Hz	50 vehicles	GPS, fuel-level, fuel rate, odometer, trip ID, driver ID	Viriciti SDK and Clever API
Electric vehicles	ViriCiti and Clever Devices	1 Hz	3 vehicles	GPS, charging status, battery current, voltage, state of charge, odometer	Viriciti SDK and Clever API
Hybrid vehicles	Viriciti and Clever Devices	1 Hz	7 vehicles	GPS, fuel-level, fuel rate, odometer, trip ID, driver ID	Viriciti SDK and Clever API
Traffic	HERE and INRIX	1 Hz	Chattanooga and Nashville region	TMC ID, free-flow speed, current speed, jam factor, confidence	Traffic Message Channel (TMC)
Road network	OpenStreetMap	Static	Chattanooga and Nashville region	Road network map, network graph	OpenStreetMap (OSM)
Weather	DarkSky	0.1 Hz	Chattanooga and Nashville region	Temperature, wind speed, precipitation, humidity, visibility	Darksky API
Elevation	Tennessee GIC	Static	Chattanooga region	Location, elevation	GIS - Digital Elevation Models
Fixed-line transit schedules	CARTA, WeGO	Static	Chattanooga and Nashville region	Scheduled trips and trip times, routes, stops	General Transit Feed Specification (GTFS)
Video Feeds	CARTA	30 Frames/Second	All fixed-line vehicles	Video frames	Image
APC Ridership	CARTA , WeGO	Every Stop	All fixed-line vehicles	Passenger boarding count per stop	Transit authority specific

applications. In the transportation domain, this data is mostly streamed from various sensors and is spatio-temporal in nature. Therefore, sensor data typically includes a timestamp indicating when the reading was taken, the value of the reading, as well as a spatial component representing where this sensor is located. Sensors can be statically located along roadways or can be dynamic in that they travel over time, e.g. GPS receiver within a vehicle or bus. The spatio-temporal nature of these data sources presents challenges in efficient storage, synthesis, and data retrieval [22, 23, 1]. Synthesis requires processing data in a variety of domain-specific formats and at irregular intervals. Raw sensor streams must often be enriched by joining them with infrastructure data, such as the identifier of the road along which a vehicle is travelling.

Data sources can be classified as static or real-time. Static data are datasets related to the road networks, GIS layout, and schedules for the operating regions, while real-time data consists of data streaming from sensors and collected during real-time operations. In this sense, the static data is not fully "static." For instance, both roadway information and fixed-line schedules can change over time. However, the static data in this context is not required to be updated daily in the way that real-time data is. A summary of data sources collected is provided in Table 2.1.

2.1.1 Static Data Sources

Static data are datasets related to the road networks, GIS layout and schedules for the operating regions of Nashville, TN and Chattanooga, TN. We use the term static to differentiate these sources from real-time data that is streaming from sensors and collected during real-time operations. In this sense, the static data is not fully ”static”. For instance, roadway information can get updates over time and fixed-line schedules can change. However, the static data in this work is not required to be updated daily in the way that real-time data does. The key static datasets are as follows:

- **OpenStreetMaps (OSM)** [24]: provides road infrastructure modeled as a graph. OSM is community-driven and open-source with updates provided by the OSM community. The open-source nature of OSM data allows us to use it for mapping and routing. Therefore, in many ways OSM data is the backbone of our work in urban mobility. OSM data is provided in XML format that can be downloaded at a variety of mirrors online. We periodically download updated OSM data from the Geofabrik mirror [25] for the Nashville, TN and Chattanooga, TN regions respectively. We utilize multiple tools built on top of OSM data including Open Source Routing Machine (OSRM) [26] which is a highly optimized open-source router that can be deployed as a server and queried for shortest paths, travel times and distances.

- **Static GTFS** [27]: the General Transit Feed Specification (GTFS) is a specification for static fixed-line transit schedules which includes buses and light rail. It includes routes, trips, stops and scheduled stop arrival times. Static GTFS also includes geospatial shape objects representing routes so that fixed-line transit can be mapped to the operating region. Current GTFS schedules for Chattanooga, TN can be acquired from CARTA’s developer portal [28] and for the Nashville region from WeGo [29].

- **Elevation data**: additionally we collected static GIS elevation data from the Tennessee Geographic Information Council [30]. From this source, we download high-resolution digital elevation models (DEMs), derived from LIDAR elevation imaging, with a vertical accuracy of approximately 10 cm. We can incorporate the elevation data in the OSM network by adding the elevation from the GIS data to each node in the OSM network.

2.1.2 Real-time Data Sources

Real-time data is either collected directly from web-based API's or historical readings are downloaded in batches depending on the data source. The key data sources are as follows:

- **Weather**: Weather data is acquired from multiple data sources. We collect weather data from multiple weather stations in Chattanooga, TN and Nashville, TN in real-time at 5-minute intervals using the DarkSky API [31]. This data includes real-time temperature, humidity, air pressure, wind speed, wind direction, and precipitation. For large-scale historical weather data collection we use Meteostat [32]. For example, Meteostat was used to create a unique dataset for tracking wildfires that was made available to further research in data-driven wildfire management [33]. In select works, we aquire historical data directly from NOAA [34].

- **Real-time fixed-line**: we collected real-time GTFS data which contains service alerts, trip updates and vehicle positions for fixed-line buses [35]. Additionally, CARTA has a partnership with Clever Devices, Inc [36] ViriCiti [37] which both installed telemetry kits on buses in Chattanooga. The Clever Devices data provides information similar to real-time GTFS. ViriCiti provides energy consumption and fuel consumption for their mixed-fleet of electric, hybrid and diesel vehicles. Additionally, we have Automated Passenger Counter (APC) data for both Nashville, TN and Chattanooga, TN which provides occupancy information including the number of passengers that board and leave a bus at each bus stop.

- **MoD**: we have paratransit and microtransit trip data for Chattanooga, TN and paratransit data for Nashville, TN. Trip data includes pickup locations, dropoff locations and requested pickup time trip requests.

- **Traffic**: we collect traffic data at 1-minute intervals using the HERE API [38], which provides speed recordings for segments of major roads, which provides data in the form of timestamped speed recordings from selected roads. Every road segment is identified by a unique Traffic Message Channel identifier (TMC ID). Each TMC ID is also associated with a list of latitude and longitude coordinates, which describe the geometry of the road segment. Additionally, we collect historical data from INRIX [39] which includes traffic data at 1-minute intervals as well. INRIX data also includes TMC IDs as well as direct mappings to OSM segments.

2.2 Data Management

There are numerous challenges in storing and processing data for urban mobility systems. First, these sources present data in domain-specific formats and at irregular intervals that can vary by provider and source, making it challenging to join data streams to be used by downstream applications [40]. Second, the spatiotemporal nature of these data sources presents challenges in efficient storage, synthesis and data retrieval [22], [23]. A third challenge is efficiently representing and presenting the data in ways that can be used by domain experts [41]. In addition, there are the typical challenges of working with high-velocity, high-volume streaming data.

To address these challenges we presented a cloud-based data management architecture for urban mobility [1]. This data management framework has been used to collect and process data urban mobility data from our partner agencies WeGo and CARTA for Nashville, TN and Chattanooga, TN respectively. In addition to storing, processing and

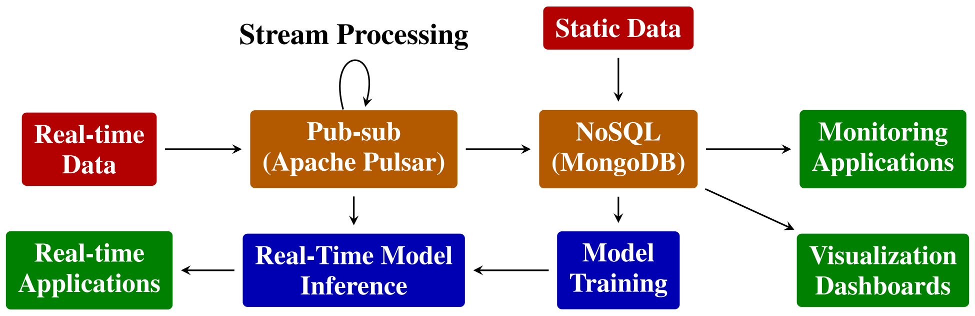


Figure 2.1: An overview of our cloud-based data management architecture for urban mobility [1].

sharing data between corroborators, the framework is the backbone for visualization tools used by WeGo and CARTA as well as for model training and application development.

An overview of the framework is provided in Figure 2.1. The various downstream applications such as monitoring systems, visualization dashboards and energy and ridership prediction models require data from various streams to be merged. Typical implementations of stream processing architectures require external processing frameworks such as Apache Spark and Storm. For our implementation we decided to join the data streams within Apache Pulsar. This process involves designing functions that read from a set of data stream topics, merge the streams in a series of time windows, and output the joined data on a new Pulsar topic. Running our stream processing applications within Pulsar has two benefits. First it provides real-time access to consumers that subscribe to the output topic of these applications. Second, we include a subscriber that continuously adds geospatial indexing the the streams and writes to MongoDB. The Apache Pulsar and MongoDB components make up the core of the data ingestion and storage components. This cloud-centric architecture has been used for various research activities as well as dashboards, applications and software for our partner agencies. Additionally, Chapter 5 and Chapter 6 we expand on these ideas to discuss the potential of, and challenges related to, decentralized edge deployments.

2.3 Demand-Responsive Transportation as a Sequential Decision-Making Problem

In this work we largely focus on real-time transportation systems, in particular demand-responsive transportation. Demand-responsive transportation is fundamentally a resource allocation problem in which a set of resources, such as vehicles, must be matched to meet demand (passengers, trip requests) in a way that 1) satisfies a set of hard constraints at each decision point and 2) optimizes various objects such as ridership, costs or emissions over time.

Demand-responsive transportation can be modelled as a sequential decision-making problem. Sequential decision-making is commonly formalized as a Markov Decision Process (MDP) [42]. An MDP can be specified by a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, p(s_o), \mathcal{C}\}$:

- Set of states $\mathcal{S}$ and distribution over the starting state $p(s_0)$.
- Set of actions $\mathcal{A}$.
- Transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ takes as input the current state $s_t \in \mathcal{S}$ and an action $a_t \in \mathcal{A}$ at timestep t and returns the next state $s_{t+1} \in \mathcal{S}$.
- Reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$.
- The policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ controls the agent's behaviour by selecting an action given the current state.
- The set of hard constraints C must be satisfied at every decision point. Examples include capacity constraints where a vehicle can not have more than a set number of passengers on-board at any given point or time window constraints where a passenger must be picked up within some time frame. C can vary depending on form of transportation and domain setting.

The *environment* consists of the transition function and reward function. At each timestep t the environment takes s_t, a_t and returns the next state from $\mathcal{T}(s_{t+1}|s_t, a_t)$ and a scalar reward from $\mathcal{R}(s_t, a_t)$. The objective of sequential decision-making is to select actions given a policy $\pi(a_t|s_t)$ to optimize *cumulative* reward over time while satisfying C at each decision point.

2.4 Examples of Demand-Responsive Transportation Applications

Demand-responsive transportation can be thought of as any transportation mode that adapts to new demand in real-time. While there are many interesting modalities that can be labelled as demand-responsive, we focus on two high-impact modalities for public transit agencies - ride-pooling and fixed-line transit. As discussed, ride-pooling is a form of MoD that provides direct transportation between origin and destination for a user that requests a trip through a smart phone application or by calling into a centralized dispatching service. Ride-pooling extends the traditional ride-hailing model (Uber, Lyft) to allow users to share trips and thus has the potential to increase efficiency by reducing vehicle miles and utilization by reducing the number of vehicles required to service passengers [13, 43, 2]. There are two separate ride-pooling services often offered by transit agencies - microtransit and paratransit. Microtransit and paratransit both rely on high-capacity vehicles, however they differ in the set of constraints C set on the system. In this context, paratransit additionally requires hard quality-of-service (QoS) constraints for each passenger in accordance with the Americans with Disabilities Act (ADA). We discuss the unique aspects of the paratransit setting further in Chapter 3 and Chapter 7.

We also discuss demand-responsive transportation in the context of fixed-line public transit. Traditional fixed-line public transit involves scheduling routes and vehicle schedules to service the routes ahead of time. In recent years, there has been increased interest in making these systems demand-responsive by either incorporating algorithms for automated dispatching to address demand in real-time or to automate the vehicle-to-trip assignment to minimize energy usage and emissions [11, 7, 17, 44]. We further discuss vehicle-to-trip assignment problems in Chapter 4.

Chapter 3

Non-Myopic Planning for Dynamic Vehicle Routing Problems

3.1 Overview

Urban mobility has been transformed by new demand-responsive modes of travel including ride-hailing, ride-share, e-scooters and e-bikes. Demand-responsive, Mobility-on-Demand (MoD) transportation consists of users requesting rides through a mobile app. In ride-hailing systems such as Uber and Lyft, the trip is then assigned to a driver who provides direct service between the origin and destination. Ride-hailing has proved popular with users due to its ease-of-use, however ride-hailing can increase total vehicle miles travelled (VMT) compared to private vehicle ownership [6]. This has led to increased interest in ride-share, or ride-pooling services which also provide direct transport between origin and destination while allowing for users to share trips. The key idea is that by grouping riders with similar trips together, ride-pooling can increase efficiency compared to single-rider services.

Public transit agencies already often run various ride-pooling services including microtransit and paratransit. Both of these settings utilize a fleet of high-capacity vehicles to service trips that can be requested beforehand or on the day-of, in real-time. The offline problem, where all requests are known ahead of time, involves solving a large-scale vehicle routing problem (VRP). The vehicle routing problem (VRP) is a well-studied combinatorial optimization problem which aims to find an optimal set of routes for a fleet of vehicles to traverse. There are many flavors of VRPs but all include a set of vehicles that start and end at a depot. In this work, we are primarily concerned with high-capacity ride-share as a dynamic VRP (DVRP), which extends the traditonal VRP to online settings with real-time, stochastic trip requests. The goal here is to assign vehicles to routes in real-time as requests arrive to optimize a long-term utility function over the course of the day. DVRPs are modelled as a sequential decision-making problem.

A particularly interesting application is paratransit services which offer door-to-door service through flexible scheduled and on-demand trips in a way that extends existing transit to residents with handicap or limited access to public transit. It is considered a ”social good” and is a required service in many urban areas in the United States. The design of algorithms for assigning trips to routes for paratransit is important since any improvements to these systems can allow better service for residents in the most need of reliable transportation services from a city. As we will discuss in this chapter, there are numerous challenges in designing algorithms in the paratransit setting. This chapter presents a fully online approach to solve the dynamic vehicle routing problem (DVRP) with time windows and stochastic trip requests that is robust to changing environmental dynamics by construction. We focus on scenarios where requests are relatively sparse—our problem is motivated by applications to paratransit services. We formulate DVRP as a Markov decision process and use Monte Carlo tree search to evaluate actions for any given state. Accounting for stochastic requests while optimizing a non-myopic utility function is computationally challenging; indeed, the action space for such a problem is intractably large in practice. To tackle the large action space, we leverage the structure of the problem to design heuristics that can sample promising actions for the tree search. Our experiments using real-world data from our partner agency show that the proposed approach outperforms existing state-of-the-art approaches both in terms of performance and robustness.

The work comprising this chapter has been published in the Proceedings of the 13TH IEEE International Conference on Cyber-Physical Systems (ICCPS) [43]:

- Michael Wilbur, Salah Kadir, Youngseo Kim, Geoffrey Pettet, Ayan Mukhopadhyay, Philip Pugliese, Samitha Samaranayake, Aron Laszka, and Abhishek Dubey. An online approach to solve the dynamic vehicle routing problem with stochastic trip requests for paratransit services. In *ACM/IEEE 13th International Conference on*

Cyber-Physical Systems (ICCPS). IEEE, April 2022

3.2 Introduction

The vehicle routing problem (VRP) is a well-known combinatorial optimization problem that seeks to assign a fleet of vehicles to routes to serve a set of customers/requests [45]. Many real-world use cases of transportation agencies are modeled by the dynamic version of the problem (DVRP) with stochastic trip requests. In such settings, some customer requests may be known at the time of planning while others are unknown, and some stochastic information may be available about potential future requests [46]. Although the dynamic and the stochastic versions have traditionally been tackled separately, Bent and Van Hentenryck [46] and Hvattum et al. [47], among others, showed that dynamic planning could use the stochastic information to improve performance. There are three broad approaches for solving DVRPs with stochastic requests. First, as requests arrive, a group of requests can be batched together, and routes can be optimized myopically for the particular batch in an *online* manner [13]. Second, the routing problem can be solved to maximize a non-myopic utility function by learning a policy in an *offline* manner that maps any given state of the problem to an action (i.e., a route plan for the vehicles) [48]. Third, a combination of offline computation and online heuristics can be used for non-myopic planning [46, 15].

A fundamental challenge in solving the DVRP non-myopically is computational tractability; indeed, real-world applications of DVRP are often intractable since they entail solving a hard combinatorial optimization problem with consideration for future requests. Prior work based on the combination of offline learning and online heuristics has addressed this bottleneck to some extent. Bent and Van Hentenryck [46] continuously generate and store a pool of promising plans. Then, at the time of execution, they use a least-commitment strategy to select a key plan from the pool. Shah et al. [16] use approximate dynamic programming and leverage a neural network-based approximation of the value function to handle the complexity from combinations of passenger requests. Joe and Lau [15] ensure near real-time response by combining online routing-based heuristics (e.g., simulated annealing) with offline approaches (e.g., value function approximation) [15]. Myopic approaches, on the other hand, focus on pooling requests together to optimally allocate a specific batch of requests to routes [13]. A major challenge in such a setting is computing solutions fast enough to assign vehicles to requests in real-time (in practice, some delay is acceptable after a request is made). Hence, myopic approaches that operate in an online manner must be *anytime* or able to compute a feasible action quickly; requests are stochastic and a decision must be computed before future requests arrive. Alonso-Mora et al. [13] scale a myopic approach to assigning routes to pooled requests for real-world use cases by using an anytime algorithm that starts from a greedy solution and then improves it through constrained optimization.

We focus on *paratransit* services [49] in this paper, which are an important real-world example of DVRPs. Paratransit service is a socially beneficial curb-to-curb transportation service provided by public transit agencies for passengers who are unable to use fixed-route transit (e.g., passengers with disabilities). Our partner agency, the Chattanooga Area Regional Transportation Authority (CARTA), operates paratransit services in a mid-sized metropolitan area in the USA. While paratransit services resemble traditional on-demand ride-pooling services in some ways (e.g., the arrival of real-time requests and ride-sharing), our collaboration revealed some crucial differences between canonical examples of DVRPs in prior literature (e.g., ride-pooling or cargo delivery) and paratransit services. First, the frequency of requests is usually lower than services like taxis. For example, CARTA operates paratransit services in a metropolitan area with about 1.8 million people and usually serves about 200 requests per day (with 10 hours of operation each day). This constraint makes it difficult to batch requests together for myopic algorithms. Note that while the requests are temporally sparse, the decision for each request must be computed quickly. Second, paratransit services operate under the Americans with Disabilities Act (ADA), which enforces time windows as a hard constraint, unlike on-demand taxi services, thereby requiring strict adherence to such constraints. Our discussion with CARTA also revealed a potential issue with offline approaches for solving DVRPs. In practice, the environment in which CARTA operates is highly dynamic; traffic conditions in a city can change due to construction, events, or accidents, and the number of available vehicles or drivers can vary. In such cases, offline approximations can potentially lead to decisions that are far from optimal.

This paper introduces a *fully online* approach for use-cases such as paratransit services that is *anytime*, *non-myopic*, *robust* to dynamic changes in the environment, and also *scalable* to real-world applications. Designing a completely online approach to solve DVRPs in a non-myopic setting is extremely challenging — the action space for such a problem is intractably large for real-world applications. For example, for our partner agency in a mid-size metropolitan city in US with five vehicles, each with a capacity of eight passengers, the action space is of the order of 10^{22}. Also, note that while requests are relatively sparse in our problem setting (one request every few minutes on average), computation for each decision needs to be fairly quick; naturally, it is infeasible to keep customers waiting for more than some exogenously defined duration.

The summary of contributions are as follows: **(1)** We design a fully online and non-myopic solver for DVRP with stochastic requests that scales to real-world problems by leveraging the structure of the problem instance. Our approach, MC-VRP (**M**onte **C**arlo tree search based solution for **v**ehicle **r**outing **p**roblem) is robust to environmental dynamics by construction. **(2)** We model the DVRP as a route-based Markov decision process (MDP) [50]. Given an arbitrary state of the MDP, we use generative models over customer requests and travel time to simulate the environment under consideration, which in turn enables us to use Monte Carlo tree search [51] to find promising actions for the state. Our approach does not require offline training, the only requirement is a generative model that can be sampled at run-time. **(3)** To tackle the intractably large action space, we leverage the structure of the problem to find promising actions. Specifically, given a set of routes and

Table 3.1: Symbols

Notation	**Description**
V	A set of vehicles each of capacity c
θ^t	The set of route plans for all the vehicles with θ_t^i denoting the plan vehicle $v_i \in V$
Θ_t	The set of all possible route plans at time t
$vloc_t$	A set consisting of locations of all vehicles at time t with $vloc_t^i$ denoting the location of $v_i \in V$
R_t	A new trip request at time t
$\mathcal{R}$	Ordered set of trip requests for a day
$[e_t, p_t]$	Service time window for request R_t
$C(\theta_i^m)$	Cost of vehicle m servicing at route plan
M	Set of vehicles in the paratransit fleet
S_i	State tuple $< T_i, vloc_i, \theta_i >$ for timestep i
E	Generative demand model
K_{max}	Maximum number of feasible actions to consider at each time epoch

a new request, we create a weighted graph based on a budget-based heuristic whose edges represent: (a) which vehicles can serve the new request, and (b) which vehicles can swap unpicked requests from their routes to maximize utility. Then, we sample independent sets from the graph that correspond to feasible actions for the given state of the MDP.

To evaluate the proposed approach, we consider three baselines. First, we look at a greedy strategy, which assigns a new request to the vehicle that provides the highest myopic utility. Second, we look at recent work by Joe and Lau [15], which outperformed well-known state-of-the-art approaches such as the multiple scenario-based approach [46] and the approximate value iteration [52]. Third, we compare our approach with a batched myopic setting by using the work by Alonso-Mora et al. [13]. Our experimental evaluations show that MC-VRP approach outperforms the baselines in terms of performance and robustness to changing environmental conditions.

3.3 Problem Description and Model

Table 3.1 provides a notation lookup table used throughout this work. Typically, in paratransit services, passengers request pick-up times in advance of the trip, even when requesting on the same day. This is in contrast of on-demand ride services that are often requested with a short lead time. For example, customers can call in the morning to request for a ride during early afternoon. However, in some cases customers request rides with a short lead time. We assume that requests are i.i.d. according to a distribution D. We denote the request at a given time t by R_t, which consists of a pick-up location, a drop-off location, a pick-up time, and a drop-off time.

In practice, it is common for paratransit services to use a pick-up window, e.g., they commit to picking passengers up at most 15 minutes before the requested pickup time and drop them off at most 15 minutes after the requested drop-off time. For a request R_t, we refer to such time points as the earliest pick-up time (denoted by e_t) and the latest drop-off time (denoted by p_t). Naturally, the latest drop-off time must be greater than the sum of the earliest pick-up time and the minimum time it takes to travel to the drop-off location from the pick-up location. We assume that requests follow such constraints. In practice, the application used for making requests or a human operator can enforce such constraints. Note that the time windows specified by the customers are treated as strict guidelines making this a pickup and delivery problem with time-windows (PDPTW) [53]. In case constraints cannot be met, the system *rejects* requests. Given this setting, the goal of the decision-maker is to maximize the total number of requests that can be served in a day while ensuring that the pick-up and drop-off constraints are met.

3.3.1 Route-based Markov Decision Process

Our problem consists of a set of identical vehicles, denoted by V, each with a capacity of c passengers. The set of all possible locations in the area is represented by the graph

$G = (L, E)$, where L denotes the set of vertices (or locations) in the graph, and E denotes the set of edges weighted by travel time. A route plan for vehicle $v^i \in V$ at time t is denoted by θ_t^i, which is an ordered sequence of pick-up/drop-off locations that the vehicle needs to visit in its route. Therefore, a route plan θ_t^i can be represented as an ordered set $\{l^1, l^2, \ldots\}$, where each $l^i \in L$ is a vertex of graph G. At any given time t, the set of all feasible route plans is denoted by Θ_t (we define feasibility below). We assume that vehicles start operations at a depot and return to the depot at the end of the day ($depot \in L$).

To solve the problem of identifying the best routes for all vehicles, we model the dynamic vehicle routing problem (DVRP) as a Markov decision process (MDP) based on prior work done by Ulmer et al. [50]. An MDP is defined by a 4-tuple $\{S, A, P, \gamma\}$, where S is a set of states that capture relevant information for decision-making, A is a set of control choices or actions, P is a state-action transition function, and γ is a reward function that defines the utility of taking an action at a given state [54]. The route-based MDP formulation is beneficial in DVRP settings where actions involve assigning the current request to a vehicle and optimizing the entire route under consideration. Since our goal is to maximize the number of requests that can be served in expectation (with respect to the distribution D), optimizing the route as a whole towards this goal is a natural choice.

Decision Epoch We define decision epochs as in prior work by Joe and Lau [15]. A decision epoch occurs at the time a request is received. Between requests, the environment evolves in continuous time, e.g., the vehicles move continuously, and requests can arrive at any point in time. At each decision-epoch, the decision-maker takes an action (states and actions are defined below). The effect of the action results in a state transition, which consists of two parts — a transition from a pre-decision state to a post-decision state, and from the post-decision state to the next pre-decision state [50, 55].

State We denote the set of states by S. We use s_t to denote the pre-decision state at time t, which includes the vehicle locations, current route plans for all vehicles, and details about passengers aboard the vehicles. Note that while it suffices to keep track of the route plan for some formulations, we must track drop-off times for passengers already on board as actions may dynamically change routes. Formally, we represent the state s_t by the tuple $(R_t, R, vloc_t, \theta_t)$, where R_t is the new trip request at time epoch t, $R = \{R^1, \dots, R^{|V|}\}$ is the set of requests assigned to each of the vehicles, $vloc_t$ is a vector of location of all the vehicles at time t, and $\theta_t = \{\theta_t^1, \dots, \theta_t^{|V|}\}$ is the current route plan for each of the vehicles. The post-decision state is denoted by s_x^t, which denotes the effect of an action x on s_t, and includes the updated route plan [50].

We associate some additional information with each route plan. Consider a route plan θ_t^i for vehicle $v^i \in V$. For each location l^j in the route plan, we associate four pieces of information. First, let $a(\theta_t^i, l_j)$ denote the planned arrival time of the vehicle v_i at location l^j. We can calculate the arrival time from a pre-computed travel time matrix (or a router). Second, let $e(\theta_t^i, l^j)$ be the earliest time service may begin at this location. If l^j is a pick-up location, $e(\theta_t^i, l^j)$ is equal to the earliest pick-up time for the customer; otherwise, we set it to some default value. Third, $p(\theta_t^i, l^j)$ denotes the latest time at which service is desired at this location. If l^j is a dropoff location, $p(\theta_t^i, l^j)$ is set to the latest drop-off time for the customer associated with the location; otherwise, we set it to some default value. We also maintain the number of passengers on-board the vehicle at each location as $w(\theta_t^i, l^j)$. Lastly, let y^j be a binary variable set to 0 if l^j is a pickup location and set to 1 otherwise.

Actions We denote the set of all feasible actions at time t by X_t. An arbitrary action in X_t involves assigning the new trip request R_t to a vehicle $v^i \in V$ and subsequently updating the route plan of *all* the vehicles. Note that in our problem setting, X_t simply reduces to Θ_t, the set of all feasible route plans at time t. Updating the route plan can entail changing the order in which existing requests are picked up/dropped off in a vehicle's route plan or swapping requests that have not been picked up between vehicles. While allowing swapping between vehicles increases the complexity of the problem significantly, we include such actions nonetheless to maximize utility. The action space is, therefore, the set of *all* feasible route plans. A feasible route must meet three requirements: first,

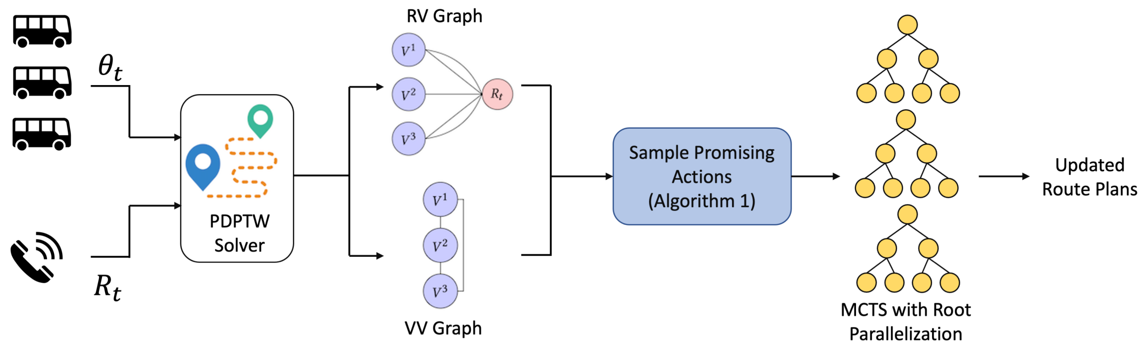


Figure 3.1: An overview of MC-VRP. A decision epoch corresponds with a new trip request R_t . We generate request-vehicle (RV) and vehicle-vehicle (VV) graphs by combining a heuristic based PDPTW solver with metrics to quickly estimate the utility of route plans. We then select promising actions from the graphs by sampling independent sets of high weights to be evaluated by an online approach based on MCTS.

the pick-up location for each trip requests must appear before the corresponding drop-off location in the route plan. Second, the pick-up time for a request must not occur before its earliest pick-up time, and finally, the drop-off time must not occur after its latest possible drop-off time.

State Transitions and Rewards At decision-epoch t, the decision-maker can take an action $x^j \in X_t$ (say), which results in a transition from the pre-decision state s_t to the post-decision state s_t^x. Then, the system transitions from s_t^x to the next pre-decision state s_{t+1}, within which a new request might arrive. We assume that requests are drawn from a distribution D. We refrain from defining the exact state-action transition function since we use a simulator (or a generative model) for online planning. The reward for an action $x^j \in X_t$, denoted by $\gamma(x^j)$, is simply the number of additional requests served by the action. Since only one request arrives at any decision epoch, the reward simplifies to 1 if the request can be accommodated and 0 otherwise. Since paratransit services require strict adherence to time windows, note that some requests can be rejected. In practice, human operators can suggest alternative choices to customers in such situations.

3.4 Approach

We provide an overview of MC-VRP in Figure 3.1. Our approach consists of the following three broad components: **1)** For a given state of the MDP, we sample feasible and promising actions by exploiting the structure of the problem. To compute feasibility, we use a heuristic-based solution approach to the PDPTW. To compute the potential utility of a feasible action, we introduce two heuristics, one based on passenger travel times and another based on *budget* (or slack) in route plans. **2)** We compute weighted graphs based on the feasible actions and their potential utilities. Specifically, we create vehicle-vehicle (VV) and request-vehicle (RV) graphs (we describe the graphs below). Then, we generate promising actions by sampling independent sets from the graphs (based on the weights of the sets). **3)** The sampled actions are then used by our online non-myopic planner based on Monte Carlo tree search. To build the search tree into the future, we sample future requests from a data-driven generative model. Finally, to lower computation time, we utilize pre-computed samples of requests and *root parallelization* [56] to efficiently explore the search space and recommend an action for the given state. We describe each component of our approach in detail below.

3.4.1 PDPTW Solver

At each decision epoch, our goal is to optimize existing vehicles' routes to accommodate a new request. First, we check whether a vehicle can accommodate the request given its current route plan. Recall that accommodating a request in our setting requires strict adherence to time windows. We use a heuristic-based solver for the pickup and delivery problem with time-windows (PDPTW). The solver enables us to check if the current request can be accommodated in a feasible route plan. The PDPTW is NP-hard [53, 57]; as a result, we use a heuristic subroutine to solve it. Using a heuristic approach is critical in our setting; as we show below, the PDPTW solver needs to be invoked for the given state of the MDP as well as for states we sample as we look into the future. While any heuristic designed for solving PDPTW can be incorporated in our framework, we use the *insertion heuristic* [13], which seeks to insert the pickup and dropoff locations of the new request within the existing route plan. We introduce some additional notation to describe the PDPTW solver. Consider a vehicle $v^i \in V$ that has an assigned route θ_t^i at time t. For a new request R_t, we use $PDPTW.feasiblePlans(\theta_t^i, R_t)$ to denote a module that returns the set of all feasible route plans for the vehicle v^i that include the new request. Also, let $PDPTW.bestFeasiblePlan(\theta_t^i, R_t) = \text{argmax}_\theta U_\omega(PDPTW.feasiblePlans(\theta_t^i, R_t))$, which denotes a function that computes the utility of each feasible route plan generated by the PDPTW solver based on a specific utility function U and a metric ω, and returns the one with the highest utility. For example, a metric could be the total travel time of the route, and the utility function, in the simplest case, can be the identity function.

3.4.2 Handling Exponential Action Space

A feasible action in our problem corresponds to a set of route plans for all vehicles, given that one of them can accommodate the new request in consideration. In case no feasible action is found, the request is rejected. Additionally, we design our action space to let vehicles swap requests from their assigned routes that have not been picked up (we describe how we use the PDPTW solver to this end below). As a result, the number of possible actions for a given state is combinatorially large; on average, an arbitrary state in our MDP has 10^{22} possible actions. Such an action space is infeasible to explore in an online setting. To address this challenge, we introduce an approach that enables us to sample promising actions from the set of feasible actions. We start by introducing two heuristic metrics that can be used to gauge the long-term utility of a route plan quickly.

1.) Maximizing the budget to serve future requests: Our goal is to maximize the number of requests the vehicles serve on a given day while following the specified time constraints. Intuitively, a vehicle can accommodate future requests in an existing route plan if there is sufficient *room* (time) in the route. To capture this idea formally, we build upon prior work by Ulmer et al. [58] to extend the idea of a budget-based heuristic to DVRPs with capacity and time window constraints. Our budget-based heuristic captures the idea that maximizing the time a vehicle has no passengers on board also maximizes the slack to serve future requests. We define the budget-based utility for a route plan for vehicle $v^i \in V$ at time t as

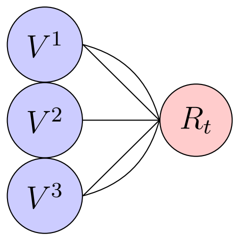
$$\bar{b}(\theta_t^i) = t_{max} - t - \sum_{j \in \{1,\ldots,|\theta_t^i|-1\}} \mathbb{1}(w(\theta_t^i, l^j) > 0)\big\{a(\theta_t^i, l^{j+1}) - a(\theta_t^i, l_j)\big\} \tag{3.1}$$

where t_{max} denotes the maximum time up to which the vehicle is available to serve requests (e.g., end of a day), $(w(\theta_t^i, l_j))$ denotes the number of passengers on board vehicle v^i from location l^j in its route plan to the next location, $\mathbb{1}()$ denotes the indicator function, and $a(\theta, l^k)$ denotes the time a vehicle operating under a route plan θ reaches location l^k. The summation in the equation 3.1 represents the total time in the route plan for which at least one passenger is on-board.

2.) Minimizing passenger travel time: An alternative to the budget-based heuristic is to minimize passenger travel time (PTT). Intuitively, by minimizing passenger travel time, we maximize the available capacity in each vehicle over the time horizon. We define the utility of the PTT-based heuristic for a route plan θ_t^i as $\overline{PTT}(\theta_t^i)$ for vehicle $v^i \in V$ at time t as shown in equation:

$$\overline{PTT}(\theta_t^i) = \sum_{j \in \{1,\ldots,|\theta_t^i|-1\}} w(\theta_t^i, l_j) * (a(\theta_t^i, l_{j+1}) - a(\theta_t^i, l_j)) \tag{3.2}$$

In this case $\overline{PTT}(\theta_t^i)$ is the summation of the number of passengers on board after picking up the passenger at location j, represented by $w(\theta_t^i, l_j)$, multiplied by the time to reach the next location, $j+1$. Therefore, by minimizing $\overline{PTT}(\theta_t^i)$ we maximize the number of seats available to incorporate future requests.



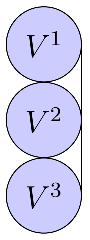


Figure 3.2: (left) RV-graph (G_{RV}): there is an edge between a request and a vehicle for every feasible route plan in which a vehicle can service the new request. (right) VV-graph (G_{VV}): an edge between vehicle v^i and v^j represents the swap with the highest utility between the two vehicles.

Having described metrics to assess the potential utility of a specific vehicle route for a given vehicle, we now introduce an approach to sample feasible route plans. We begin by describing two graphs we construct based on the new request that arrives at a decision epoch and the existing route plans of the vehicles.

RV graph: At each decision epoch, we first generate a graph that incorporates which vehicles can service the new request. Our idea is based on prior work by Alonso-Mora et al. [13]. We denote the RV graph by $G_{RV} = (L_{RV}, E_{RV})$, where L_{RV} denotes a set of vertices and E_{RV} denotes a set of edges. To create the RV graph, we add a node for each vehicle $v_i \in V$. We add an additional node to denote the request R_t. Then, for each feasible vehicle route that can accommodate the request, we add an edge between the node denoting the request and each node representing a vehicle. We denote a specific edge in E_{RV} by $\mathrm{e}_{\mathrm{RV}}(i, j)$, which denotes the jth feasible route plan for vehicle v^i that can accommodate the request under consideration. The feasible route plans are generated by the module $PDPTW.feasiblePlans(\theta^i, R_t)$ (for vehicle $v^i \in V$ and request R_t). We associate two pieces of information with each edge. First, for an edge $\mathrm{e}_{\mathrm{RV}}(i, j)$, we use $U_\omega(\mathrm{e}_{\mathrm{RV}}(i, j))$ to denote the utility of the edge based on a utility function U and metric ω. Also, we use $\theta(\mathrm{e}_{\mathrm{RV}}(i, j))$ to denote the updated route plan corresponding to the edge $\mathrm{e}_{\mathrm{RV}}(i, j)$. We show an example of an RV graph in figure 3.2.

VV-graph: While an RV graph is useful to represent which vehicles can accommodate the new request in their existing route plans, it is possible that vehicles might want to swap requests that have not been picked up to maximize utility. Note than in a non-myopic setting, the route plans are optimized at each decision-epoch to maximize the expected utility with respect to D (the request arrival distribution). However, a specific realization of R_t presents an opportunity for vehicles to re-plan and potentially swap requests. In order to represent such swapping actions, we create an undirected vehicle-vehicle (VV) graph at each decision epoch. We denote a VV graph by $G_{VV} = (L_{VV}, E_{VV})$.

To construct the graph, we add a node for each vehicle $v^i \in V$. Edges between two vehicles denote potential swaps of requests that have not been picked up. An edge $\mathrm{e}_{\mathrm{VV}}(i, j, k) \in E_{VV}$ denotes the potential swap of request R_k from vehicle v^i to vehicle v^j. Note that a swap action creates two new route plans, one for each vehicle. We use $\theta(\mathrm{e}_{\mathrm{VV}}(i, j, k), v^i)$ to denote the route plan for vehicle v^i *after* the swap (similarly, we use $\theta(\mathrm{e}_{\mathrm{VV}}(i, j, k), v^j)$ to denote the route plan for vehicle v^j). The utility of a swapping action, denoted by $U_\omega(\mathrm{e}_{\mathrm{VV}}(i, j, k))$ is denoted as the difference in utility of the updated route plans and the original route plans, i.e., the plans without the swapping action. As before, U_ω denotes utility computed according to a function U and metric ω. For example, using an identity utility function and a metric based on the budget heuristic introduced in equation 3.1, $U_{budget}(\mathrm{e}_{\mathrm{VV}}(i, j, k)) = \bar{b}(\theta(\mathrm{e}_{\mathrm{VV}}(i, j, k), v^j)) + \bar{b}(\theta(\mathrm{e}_{\mathrm{VV}}(i, j, k), v^i)) - \bar{b}(\theta^i_t) - \bar{b}(\theta^j_t)$. Note that once a request is swapped, its pickup and dropoff can be inserted at multiple places within the existing route of the vehicle that receives it. For computational tractability, we choose the best insertion point using the module $PDPTW.bestFeasiblePlan$ (introduced in section 3.4.1). We show an example of an VV graph in figure 3.2.

Generating Feasible Actions: A feasible action is an updated set of route plans for all vehicles that do not violate the time window or capacity constraints. We seek to sample a feasible route plan from the constructed graph. Our approach is motivated by prior work by Zalesak and Samaranayake [59]. Each edge in the RV graph represents an operation that creates an updated route plan for a vehicle that includes the new request. Additionally, each edge in the VV-graph represents a swapping operation which creates a new route plan for the two vehicles involved in the swap. Therefore, any selected edge in $G_{RV} \cup G_{VV}$ represents a feasible action. Additionally, multiple edges can be selected to generate a new action, *if and only if* the set of selected edges includes only one of the edges from the RV graph. The rationale for such a condition is straightforward; a new request can only be assigned to one vehicle, and consequently, only one edge from the RV graph can be selected for a particular feasible action at each decision epoch. To generate potential feasible actions quickly, we sample independent sets (a set of edges that have no vertices in common) from $G_{RV} \cup G_{VV}$ that includes one edge from G_{RV}. Such an independent set guarantees feasibility, as we show below:

Theorem 1. *Consider graphs G_{RV} and G_{VV} generated at decision epoch t. An independent set of edges from $G_{RV} \cup G_{VV}$ that includes one and only one edge from G_{RV} must be a feasible action for the current state s_t of the MDP.*

Proof. We first list the conditions for feasibility: 1) An action is feasible if it services the current request with adherence to time constraints, and 2) time constraints for existing requests are met (including potential swaps). An independent set with one edge from G_{RV} meets condition 1 by construction—all edges from G_{RV} service the current request and meet time constraints (recall that edges are checked for feasibility through the PDPTW solver). As the vehicle that services the request cannot swap requests (by the property of independence), all swaps sampled from G_{VV} are also feasible—all edges in G_{VV} are checked for feasibility through the PDPTW solver. Hence the set of independent edges must correspond to a feasible action for the given state. □

Notice that the only condition in which a chosen set of edges from $G_{RV} \cup G_{VV}$ might result in an infeasible action is the following: consider a situation in which the vehicle (say v^i) that is assigned the request (denoted by an edge from the RV graph) also engages in swapping and receives an additional request from a different vehicle (say v^j). While both

Algorithm 1 Generating feasible actions

```
Require: $\theta_t, R_t, K_{max}, M, G_{VV}, G_{RV}$
1: $\Theta_t \leftarrow \{\}$                                   ▷ empty set of feasible actions
2: $\mathbb{U} \leftarrow \{\}$                                  ▷ empty set of utilities for each action
3: for $\mathrm{e}_{\mathrm{RV}}(i, j) \in E_{RV}$ do
4:     $\theta \leftarrow \theta_t$
5:     $\theta[i] = \theta(\mathrm{e}_{\mathrm{RV}}(i, j))$
6:     $\Theta_t.append(\theta)$
7:     $U_x \leftarrow U_\omega(\mathrm{e}_{\mathrm{RV}}(i, j))$
8:     $\mathbb{U}.append(U_x)$
9:     $G'_{VV} = G_{VV}[\{v_k \in L_{VV} : k \neq i\}]$
10:    while $|E'_{VV}| > 0$ do
11:        $\mathrm{e}_{\mathrm{VV}}(m, n, k) = \arg\max_{(m,n,k)}(U_\omega(\mathrm{e}_{\mathrm{VV}}(m, n, k))$
12:        $\theta[m] = \theta(\mathrm{e}_{\mathrm{VV}}(m, n, k), v^m)$
13:        $\theta[n] = \theta(\mathrm{e}_{\mathrm{VV}}(m, n, k), v^n)$
14:        $\Theta_t.append(\theta)$
15:        $U_x = U_x + U_\omega(\mathrm{e}_{\mathrm{VV}}(m, n, k))$
16:        $\mathbb{U}.append(U_x)$
17:        $G'_{VV} = G'_{VV}[\{v_k \in L_{VV} : k \neq m \;\; \& \;\; k \neq n\}]$
18:    end while
19: end for
20: // Filter: $X_t$ is the $K_{max}$ actions with highest utility
21: $X_t = \Theta_t[argSort(U)[1 : K_{max}]]$
22: Return: $X_t$
```

the actions, namely the swapping action and the servicing of the new request, are checked for feasibility in isolation, vehicle v^i might violate time constraints if it seeks to service both. In theory, it is possible to enumerate over each possible set of edges in $G_{RV} \cup G_{VV}$ and check for feasibility; however, enumerating over all such sets is intractable in practice. Therefore, we point out that the vehicle that services the request *can* engage in swapping and still maintain feasibility; however, ensuring that such a vehicle does not participate in swapping *guarantees* feasibility. As a result, we use the heuristic of sampling independent sets (with one edge from the RV graph) that guarantee feasibility by construction.

Recall that our goal is to sample promising actions from the set of feasible actions since evaluating all feasible actions in an online setting is not tractable. We present our approach to selecting promising actions in algorithm 1. Our approach is based on sampling independent sets based on the cumulative sum of edge weights of the sets. First, we initialize an empty set of feasible actions Θ_t and an empty set of utilities $\mathbb{U}$. Then, we begin selecting an independent set by selecting an edge from G_{RV} (step 3). Next, we initialize the utility for the route plan (that the independent set corresponds to) with the utility of the chosen edge (step 7). Then, we drop the corresponding vehicle node from G_{VV} and consider swaps iteratively (step 9-10). The utility of the resulting action is calculated by adding the utility of serving the request and the swapping action (step 15). Finally, we return a subset of K_{max} actions with the highest total utility (step 21-22) to be evaluated by the tree search, where K_{max} is an exogenous parameter.

3.4.3 MCTS Evaluation

Non-myopic approaches to DVRP rely on hybrid offline-online solutions in which an offline component is trained on historical data and embedded in an online search [15, 60]. Offline components typically require long training periods and must be re-trained each time the environment changes, making them unsuitable for highly dynamic environments. This motivates us to use MCTS, an online probabilistic search algorithm, to evaluate the long-term utility of potential actions. MCTS is an anytime algorithm, and any changing environmental conditions that are detected can immediately be incorporated into its underlying generative models for making decisions [61].

MCTS represents the planning problem as a "game tree," where states are represented by nodes in the tree. The current state is treated as the root node, and actions represent edges that mark transitions from one state to another. The fundamental idea behind MCTS is that the search tree can be explored asymmetrically, biasing the search toward actions that appear promising. To estimate the value of an action, MCTS simulates a "rollout" to the end of the planning horizon using a *default policy*. In practice, the rollout policy only needs to be computationally cheap; a common method involves selecting actions randomly during rollout. As the tree is explored and nodes are revisited, each node's utility is estimated. As the search progresses, the estimates converge towards the true value of the node. This asymmetric tree exploration allows MCTS to search very large action spaces quickly. MCTS typically requires a few domain specific components: a generative model of the environment, the *tree policy* used to navigate the search tree, and the *default policy* used to estimate the value of a node. We describe each component below.

Generative demand model: The generative demand model (denoted by E) provides a method for sampling new requests as the tree is built into the future. We use a hierarchical modeling approach to sample trip requests based on historical data. We optimize model parameters based on maximum likelihood estimation. We model two processes as part of the generative demand model. First, we model the distribution of the number of requests per day as a Gaussian distribution. We learn the parameters of the distribution by maximizing the likelihood of historical paratransit data. Second, to model individual trip requests, we aggregate historical trip requests and weigh each trip request by the number of times it is observed (often, some passengers in paratransit services request trips that have the same source, destination, and time every week). To sample a sequence of trip requests for a day, we perform the following steps: 1) we sample the number of requests (say m) from the learned Gaussian distribution over trip requests. 2) We sample m requests from the weighted aggregation of trip requests.This process is repeated a number of times to generate multiple sampled *chains* offline. During inference at decision epoch t, we provide a method $E.sample(t, n)$ which samples n chains uniformly at random from E and returns the requests in each chain that occur after the current time t.

Search policy: We use the standard Upper Confidence bound for Trees (UCT) [51] to navigate the search tree and decide which nodes to expand. When expanding a node, we use algorithm 1 to sample feasible actions for the given state. When working outside the MCTS tree to estimate the value of an action during rollout, we rely on a default policy. This is a lightweight policy which is simulated up to a time horizon and the utility of the simulation is propagated up the tree. Our default policy is a greedy assignment—for a given state, we

Algorithm 2 MCTS evaluation

```
Require: $X_t, S_t, E, n_{chains}$
1: $eventChains = E.sample(S_t, n_{chains})$              ▷ sample chains
2: $A = MCTS(X_t, eventChains)$                            ▷ action scores
3: $\bar{A} = \{\}$
4: for $a \in A$ do parallel
5:     $\bar{A}.append(mean(a))$                           ▷ aggregate across chains
6: end for
7: // Return action with highest action score
8: Return: $argmax_{x_i \in X}(\bar{A}[i])$
```

choose the edge with the highest myopic utility from the RV-graph. To ensure tractability, we do not incorporate swapping requests between vehicles into our rollout policy; this saves time during MCTS evaluation as the VV graph is not generated during rollout.

Root parallelization: Given that the sampled paratransit requests can be both sparse and highly uncertain in time and space, sampling one chain of requests might not adequately represent future demand. To handle this uncertainty, we use *root parallelization*, which involves sampling many chains, and instantiating a separate MCTS tree for each with the current request as the root node. Crucially, each tree is explored in parallel. After execution, the score for each of the actions from the common root node is averaged across trees. Then, the action with the highest average score across the trees is returned as the selected action.

The process for evaluating and selecting an action is provided in algorithm 2. The algorithm takes X_t from the feasible actions component as well as the current state S_t, the generative model E and the number of chains to sample n_{chains}. First, the set of $eventChains$ is sampled from the generative model (line 1). Second, a tree is instantiated and MCTS is performed in parallel on each of the trees. This processed is represented by $A = MCTS(X_t, eventChains)$, where A is a two dimensional tensor of size $|X_t| \times n_{chains}$ (line 2). In this sense, each row in A represents a feasible action and the columns represent the chains. The mean of each row in A represents the action score and is stored in $\bar{A}$ (line 5). Finally, the action with the maximum action score in $\bar{A}$ is returned (line 8).

3.5 Experiments and Results

In this section we describe the data, experimental setup, system parameters, baselines and results.

3.5.1 Data Description and Pre-Processing

Paratransit Data: We acquired six months of paratransit trip requests between January 1, 2021 and July 1, 2021 from CARTA. Our dataset consists of a total of 25,843 trip requests. Each request in our dataset consists of a geo-coordinate (longitude, latitude) for the pickup location and dropoff location, and also the requested pickup time. As an example, we show the temporal distribution of one of the days and the requested travel duration for each request in figure 3.3. For pre-processing, each pickup and dropoff location was assigned to the nearest node in the road network graph. We used the haversine distance to calculate the distance between two geo-coordinates. We filtered out any requests that had a pickup or dropoff location that was farther than 200 meters from the nearest node in the road network; this process filtered out approximately 5% of the requests. Of the 25843 trips in the original dataset, 24543 remained after the distance filter. As paratransit services are significantly more sparse on weekends, we included only weekday trips for our analysis that spanned 129 days (94% of the trips in our data occur on weekdays).We randomly selected 15 days from the dataset for the evaluation (test) set. The remaining 114 days were use to compute 100 synthetic days worth of trip requests (i.e., chains) for the generative model using the procedure outlined in section 3.4.3. Three more chains were generated as a calibration set for parameter tuning.

Road Network and Travel Time Matrix: We use OpenStreetMap (OSM) [62] for the road network for the area under consideration and OSMNX [63] to generate a routing graph of the road network with travel time for edge weights. Unless otherwise noted, the experiments used the free flow speed as edge weight. Then, we calculated the shortest paths

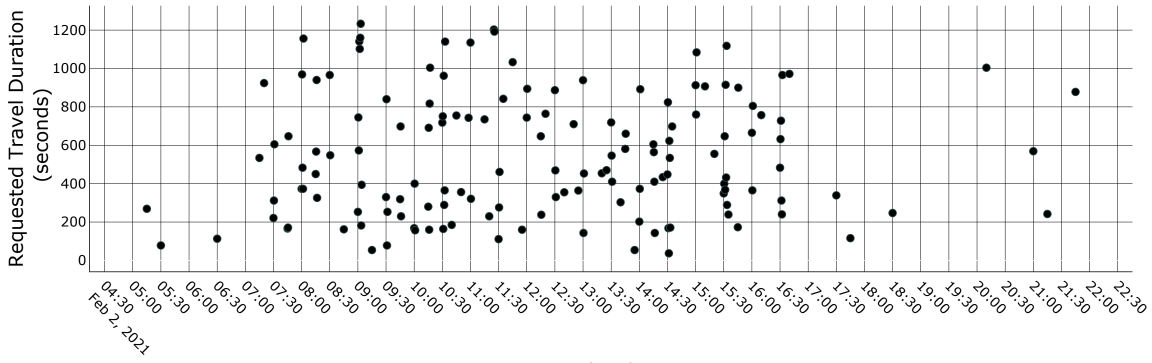


Figure 3.3: The temporal distribution of requests and the requested travel duration for the requests from one day.

between all pairs of network to generate a matrix of travel times. The travel time matrix is generated offline and therefore, provides constant lookup time for querying travel times between arbitrary locations in the area under consideration. Additionally, we collected historical traffic data from INRIX [39] to estimate typical travel times during times of high congestion. We use congestion data for evaluating the robustness of the proposed approach.

Road network and travel time matrix: We use OpenStreetMap (OSM) [62] for the road network for the area under consideration and OSMNX [63] to generate a routing graph of the road network with travel time for edge weights. To evaluate robustness we use INRIX traffic data to generate new travel time matrices that represent typical congested patterns in the city. The INRIX dataset included average roadway speeds per hour for each day in the week and included an OSM ID which was mapped to the OSM road network.

3.5.2 System Parameters

Our experimental parameters are as follows: we vary the number of vehicles from 3 to 5 as we find that 5 vehicles can serve 100% of the trip requests in our setting using MC-VRP. The vehicle capacity is set to 8, which is in accordance with the capacity of typical paratransit vans. The request arrival time, defined as the time before the requested pickup time that the request is available to the system, is set to 60 minutes. In practice, such requests can also be made before 60 minutes. The time window, i.e., the amount of time before the requested pickup time that a request can be picked up is set to 15 minutes (in accordance with settings used by CARTA). Additionally, when a customer requests a trip we provide an estimated dropoff time which is the sum of the requested pickup time and the minimum travel time to the dropoff location. The late time window is 15 minutes after the estimated dropoff time which is again, in accordance with settings used by our partner agency. Additionally, we reiterate that as the time windows are hard constraints, a trip is only feasible if the passenger is picked up and dropped off between the early pickup window and late dropoff window.

Since our online approach is anytime, we vary the amount of time that is allocated to the algorithm for making a decision (referred to as runtime cutoff). Practitioners can vary this parameter to account for the maximum time they can afford to assign a request to a vehicle. Note that the passenger making the request need not wait for this duration to receive a feedback; whether a request can be serviced or not can be computed in less than a second using our approach (we only need to construct the RV graph to find at least one feasible action). We consider two variants of our proposed MC-VRP approach. ***MC-VRP (budget)*** uses our budget-based utility for scoring feasible actions in algorithm 1, while ***MC-VRP (PTT)*** uses the PTT-based utility for scoring feasible actions. The MCTS evaluation as outlined in section 3.4.3 remains the same for both variants.

3.5.3 Baselines

We evaluate the performance of MC-VRP against the following baselines. For all baselines, we use the same number of vehicles (we present results by varying the number of vehicles) and vehicle capacity as MC-VRP. We compare our approach to two myopic online approaches (greedy and MA-RTV) and one non-myopic hybrid approach (DRLSA).

- **Greedy assignment (greedy-PTT, greedy-budget):** Greedy assignment consists of first generating feasible actions according to algorithm 1 MA- We refer to greedy assignment with the PTT utility as greedy-PTT and greedy assignment with the budget utility as greedy-budget.

- **Deep reinforcement learning-based vehicle routing [15] (DRLSA):** Joe and Lau combine deep reinforcement learning, which approximates a state-value function for routing, with a simulated-annealing based routing heuristic to solve the dynamic vehicle routing problem [64]. The state representation of DRLSA is based on the total cost of the planned routes of the vehicles. While the original approach is designed for cargo-delivery problems, we add pickup and dropoff constraints given our problem setting. Also, the problem formulation by Joe and Lau [15] seeks to minimize the sum of the travel and waiting times of all vehicles. It serves all requests and incurs a penalty cost for time-window violations. To apply DRLSA to our problem formulation, where time-windows constraints are strict, we take the route plans output by DRLSA and iteratively remove requests that violate time windows until a maximal feasible set of requests is left. Further, for a fair comparison, we run the DRLSA algorithm itself with shorter time windows but remove requests that violate the original time windows, which we found significantly improves the service rate of DRLSA in our setting. Finally, we point out that since there was no open-source implementation available for this approach, we implemented DRLSA for this paper.

- **Myopic and Anytime trip-vehicle assignment-based on RTV graphs [13] (MA-RTV):** We also compare our approach with an anytime algorithm that batches requests together and then maximizes the myopic utility for the specific batch of requests. For each batch, the algorithm creates an RTV graph by checking *shareability* between requests as well as requests and vehicles. Since the best passengers-vehicle pair is found in each batch, the approach works well in practice despite being myopic (our experiments confirm this). We refer to this baseline as *MA-RTV*. To adapt MA-RTV for our problem setting, we set the parameters as follows. First, as MA-RTV is not able to handle requests in advance,

Table 3.2: Parameter settings

Parameter	**Values**
Number of vehicles (M)	3, 4, 5
Vehicle capacity	8
Request arrival time	60m
Time window	15m
Runtime cutoff	Inf, 30s
K_{max}	10
MCTS depth	20
MCTS iterations	1000
n_{chains}	25

we set the request arrival time to the early time window. Second, we set the latest dropoff time to 30 minutes (15 minutes for early time window and 15 minutes for late time window) plus the shortest path between the pickup and dropoff locations. Therefore, the time window requirements are equivalent to our problem setup. Lastly, MA-RTV is a batching approach which waits for a set period of time before grouping requests together for assignment. In the paratransit setting, requests rarely arrive very close together and are expected to be handled one at a time, typically over the phone. Therefore we set the batch interval to 20 seconds, which closely mimics the observed paratransit data.

3.5.4 Parameter Tuning

We used 3 days of paratransit data for hyperparameter tuning. MC-VRP has two parameters for tuning—the depth of the search tree and the number of feasible actions. MCTS tree depth, which is the number of future requests to consider from the generative model, was varied between $\{10, 20, 30\}$. The number of feasible actions to explore, denoted by K_{max}, effectively sets the maximum branching factor for MCTS. We varied this parameter between $\{10, 15, 20, 25, 30\}$. We performed a grid search using the calibration set and found the best parameters in terms of service rate (MCTS depth of 20 and $K_{max} = 10$). The final parameters for MC-VRP are provided in Table 3.2.

The calibration set was also used to select the parameters for DRLSA. The neural network of the DRLSA baseline approach consists of 1 hidden layer with 64 neurons. The activation function of the hidden layer is rectified linear (ReLu) and linear for the output layer. We trained the network with a batch size of 32, the discount factor for future reward is 0.99, and the learning rate is 0.01. We found the best result with $K_{max} = 500$ for Simulated Annealing. Similar to MC-VRP, we used grid search to find the best parameters for DRSLA.

3.5.5 Reproducibility

Our source code, implementation of the baseline approaches, and samples of our datasets are available online (https://github.com/smarttransit-ai/iccps-2022-paratransit-public). We implemented MC-VRP in the Julia programming language using the pomdps.jl framework [65]. Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation [66].

3.5.6 Results

Service Rates: Our primary objective is to maximize the number of requests serviced each day. The service rate per day for the 15 days in the evaluation (test) set is provided in figure 3.4 with fleet sizes of 3, 4, and 5 vehicles. We first present results in a setting where MC-VRP (budget) and MC-VRP (PTT) are allowed to run for 1000 iterations without early termination. First, we note that 5 vehicles is enough to service all of the requests on most days as MC-VRP (budget), MC-VRP (PTT), and MA-RTV all have a median service rate of 100%. We observe that MC-VRP (budget) outperforms all baselines for fleet sizes of 3 and 4 with with a median service rate of 87.0% and 97.6% respectively. MC-VRP (PTT) had a service rate of 84.7% and MA-RTV had a service rate of 81.7% for 3 vehicles. MA-RTV outperformed MC-VRP (PTT) in the case of 4 vehicles with a median service rate of 96.2% compared to 95.8%.

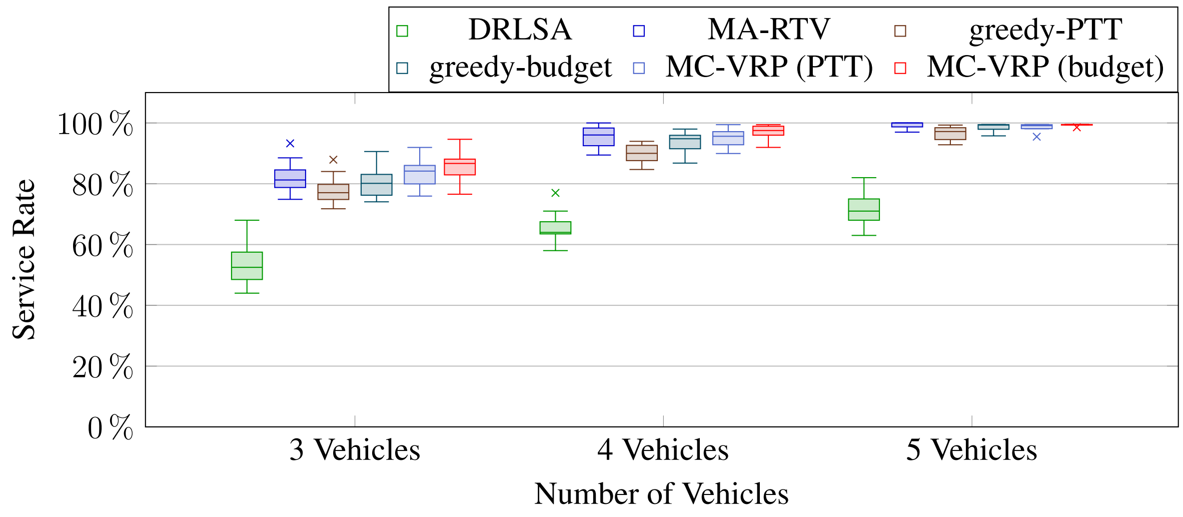


Figure 3.4: Service rate, defined as percentage of trips served, per day on 15 day test set for 3, 4 and 5 vehicles. 5 vehicles was enough to service all request on most days for MC-VRP (budget), MC-VRP (PTT) and MA-RTV. MC-VRP (budget) had the highest median service rate for 3 and 4 vehicles. The budget-based heuristic improves service rate for MC-VRP by 2.2% and 2.1% and greedy assignment by 3.4% and 4.9% for 3 and 4 vehicles respectively.

We also observe that the budget-based heuristic works better than the travel time-based heuristic for both Monte Carlo tree search as well as greedy assignment. Indeed, MC-VRP (budget) results in a higher service rate than MC-VRP (PTT) and greedy-budget outperforms greedy-PTT for all fleet sizes. This indicates that the budget utility, which aims to maximize time for which a vehicle has no passengers on-board, can be used to quickly compute promising actions to explore. It is important to note that while we focus on paratransit services, the budget-based heuristic can be applied for other DVRPs with capacity and time window constraints. We also observe that DRLSA had the lowest service rate across all fleet sizes. An influencing factor, as discussed in section 3.5.3, is that time windows are soft constraints in DRSLA which is not particularly suited to paratransit settings.

Computation Time: The computation time per request for MC-VRP and the baselines is shown in figure 3.5. MC-VRP (budget) takes slightly longer than MC-VRP (PTT) to compute a decision, with a median computation time of 38 seconds, 49 seconds, and 40 seconds as compared to 36 seconds, 44 seconds, and 37 seconds for 3, 4, and 5 vehicles

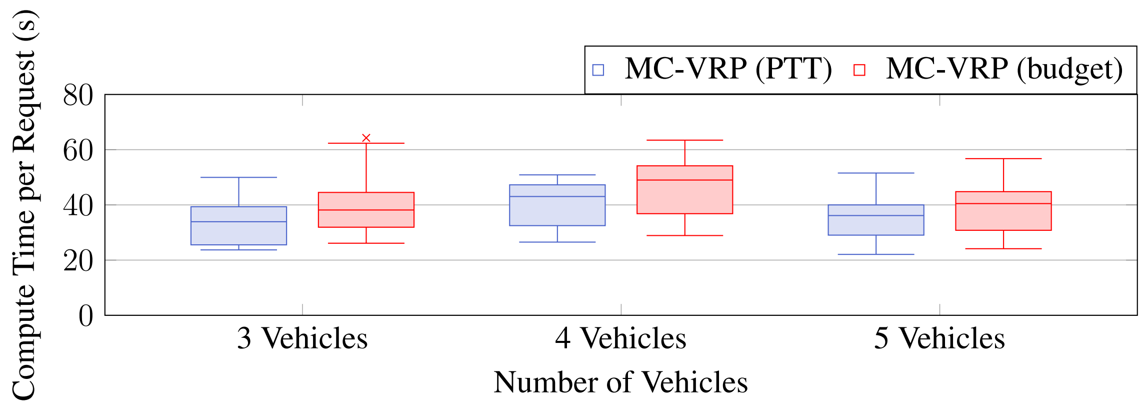


Figure 3.5: Computation time per request in seconds for each day in the test set for 3, 4 and 5 vehicles. Median computation time per request for MC-VRP (budget) is 38 seconds, 49 seconds and 40 seconds for 3, 4 and 5 vehicles respectively. DRLSA, MA-RTV and greedy all had median computation times less than 3 seconds per request.

respectively. DRLSA, MA-RTV, and greedy all had median computation times less than 3 seconds. In this context, the fact that MC-VRP is both non-myopic and fully online is reflected in the higher runtimes; while the observed runtimes are acceptable in our setting, i.e., paratransit services, the application of online and non-myopic methods remains an open question in general VRP settings. Note that if the MCTS evaluation of our approach is not used (i.e., the early stopping time approaches 0), our approach simplifies to the greedy baseline, which takes less than a second on average. This observation means that the majority of computation time in our approach is spent on MCTS.

As discussed, the MCTS evaluation can be run in the background between calls. Therefore, a potentially limiting factor for our approach is the rate at which requests arrive; even when running in background between requests, MC-VRP must be stopped on the arrival of a new request. Since MC-VRP is an *anytime* algorithm, it is possible to set a maximum computation time per request. In such a setting, the algorithm is stopped early (in our case, before the default number of iterations is reached), but outputs a feasible solution. In our dataset, 99% of requests arrive more than 30 seconds after the previous request and the median time between requests was approximately 5 minutes. Therefore, we evaluate the performance of MC-VRP with a 30 second cut-off time in figure 3.6. We observe only

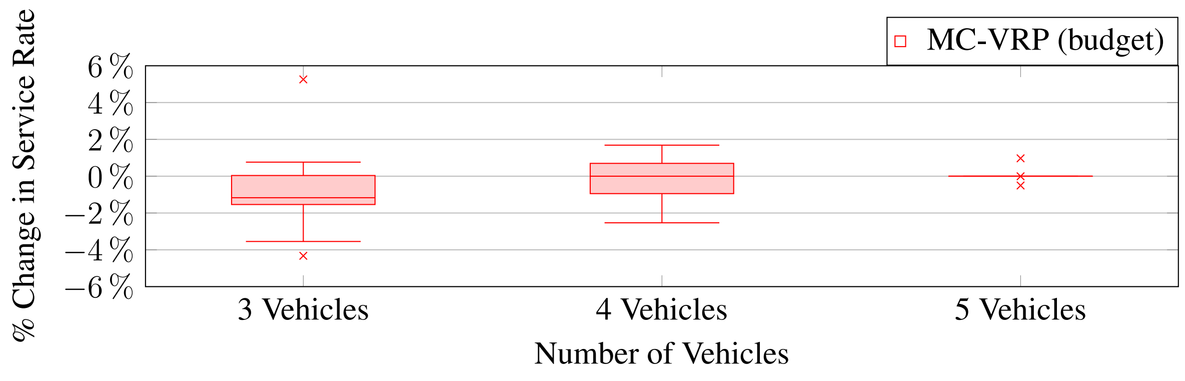


Figure 3.6: Runtime analysis: Percent change in service rate for MC-VRP (budget) with a cutoff of 30 seconds per request compared to no cutoff. For three vehicles there was a median of 1.2 percent decrease in service rate on test set while the computation cutoff had negligible affect for four and five vehicles.

a negligible decrease in service rates; the median service rate decreases by 1.2% for 3 vehicles and 0% for 4 and 5 vehicles.

Robustness: To evaluate the robustness of the proposed approach with respect to changing environmental conditions, we change the travel time distribution in the city. For experiments, we assume the existence of a service that uses short term observations about an environmental variable (e.g., travel times) and provides an updated model. In practice, transit agencies can use services like Google maps for this purpose. In this case, when a user requests a ride, we can use the updated travel time matrix to give an accurate estimate of the dropoff time. We assume that all approaches have access to the updated travel time matrix at inference time, i.e. when a user requests a ride. However, approaches that rely on models trained offline do not have the ability to update their model.

We generate a modified travel time matrix that resembles a congested road network. The distribution of the free-flow travel time matrix and the congested travel time matrix is shown in Figure 3.7. The original travel time matrix was calculated by finding shortest paths between all nodes in an OSM graph where the edge weights were travel time between the nodes using free flow speed. Each edge in the OSM graph had a unique OSM ID which mapped to roadway speeds in the INRIX dataset. The INRIX dataset included average

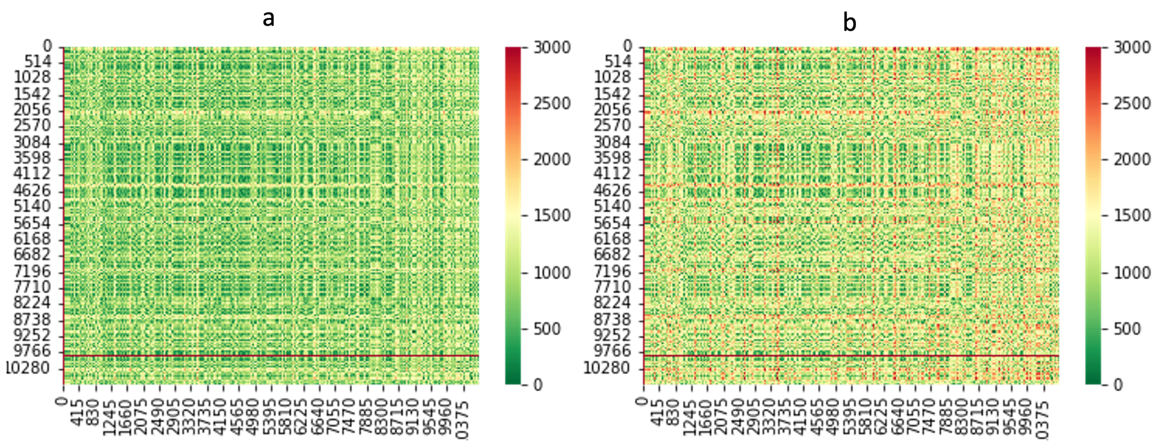


Figure 3.7: (a) Free-flow speed vs (b) Irregular travel times in seconds. Irregular travel times represent a congested roadway network. Each cell shows the travel time from a location x to another location y. There are total 10788x10788 combinations in the travel time matrix.

roadway speeds per hour for each day in the week. For each OSM ID we took the speed at the 5th percentile and updated the edge weights accordingly. On average, the congested travel time matrix had speeds that were 30% slower than the free-flow speed.

We evaluated DRLSA, MA-RTV, greedy-budget and MC-VRP (budget) on the 15 day evaluation set using the congested travel time network and compared the resulting service rates to those generated without congestion. We observe that MC-VRP (budget) incurs less of a decrease in service rate compared to DRLSA, MA-RTV and greedy-budget across all sizes of vehicle fleets as shown in figure 3.8. As expected, DRLSA had the greatest decreases in service rate due to its reliance on a value function that was trained offline using the free-flow travel time matrix. While hybrid approaches such as DRLSA have access to the real-time travel conditions, it is difficult to re-train offline components, resulting in a higher degradation in performance. Both MA-RTV and greedy-budget outperform DRLSA when evaluated for robustness. This result is expected since MA-RTV and greedy-budget are both online approaches, it is able to adapt to the updated matrix. We also observe that MC-VRP (budget) performs better MA-RTV and greedy-budget, showing that our non-myopic online solution improved upon the myopic online approaches as well.

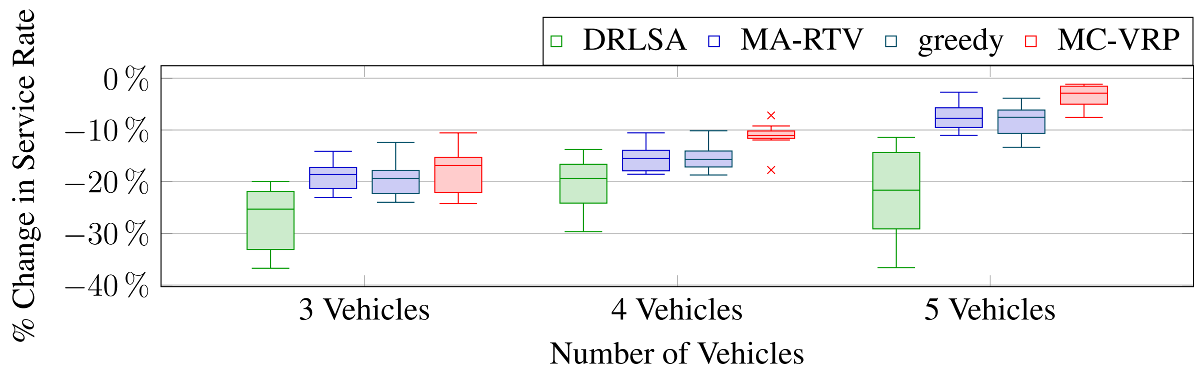


Figure 3.8: Robustness to roadway congestion: Percent change in service rate for DRLSA, MA-RTV, greedy-budget and MC-VRP (budget) on test set using congested travel time matrix compared to free-flow speed matrix. MC-VRP (budget) had less of a decrease in service rate for 3, 4 and 5 vehicle fleets compared to the baselines.

3.6 Related Work

Vehicle Routing Problems (VRPs) can be broadly classified as either static or dynamic [67]. In static VRPs, all inputs are received before optimizing routes, whereas in dynamic VRPs inputs are updated concurrently with the determination of the route. The focus of this paper is on dynamic VRPs (DVRP), which can be either dynamic-deterministic, in which no stochastic information about future inputs is known, or dynamic-stochastic, in which some probabilistic information is known about the inputs that dynamically evolve [68]. These stochastic inputs can include models over travel times, demands, and customer information [69]. DVRPs can be solved to maximize myopic rewards [13] or a non-myopic utility function [15, 16]. Exact methods for solving DVRPs seek to find an optimal solution, but are often constrained to small problem instances due to computational complexity, e.g., such approaches include column-generation [70] and the set-partitioning method [71]. Metaheuristic approaches have also been applied to DVRPs, including particle swarm optimization [72], genetic algorithms [73], and tabu search [74, 75]. Decision theoretic approaches have also been applied to DVRPs. DVRP is a sequential decision-making problem, and can be modeled as a Markov Decision Process (MDP). One approach is to use a conventional MDP structure, where actions consist of determining the next customer to serve at each decision epoch. Another approach, proposed by Ulmer et al. [50], is based on using a route-based MDP where the state-action space includes not only assigning an incoming request to a vehicle, but also optimizing the vehicles' routes.

There are three ways to solve the DVRP MDPs: offline, online, and hybrid solution methods. Offline methods pre-compute a policy that is queried while executing a plan [16, 52, 48]. Offline policies can be slow to learn, but can make decisions very quickly at execution time, and are therefore useful when there are strict time constraints on decision-making. Online solution methods perform computations during plan execution. These are generally sampling approaches, and only focus on the states of the system relevant to the current decision being made. Online methods include rollout algorithms [76, 77] and multiple scenario approach (MSA) [46]. Online approaches have typically been applied to problem settings without strict time constraints on decision-making (the time it takes to compute a decision), and in dynamic environments where policies generated offline can become stale. Hybrid solution methods attempt to combine offline and online approaches to leverage the strengths of both. For example, Ulmer et al. [60] propose an approach that embeds a value function learned using ADP into an online rollout algorithm. Joe and Lau [15] propose a similar approach, which combines Deep Reinforcement Learning to approximate a value function offline with an online simulated annealing approach.

3.7 Conclusion

We design a non-myopic online approach for DVRP for paratransit services that is robust to environmental changes by construction and scalable to real-world applications. To tackle the intractable action space, we leverage the structure of the problem to design heuristics that can sample promising actions for evaluation through MCTS. Our experimental results demonstrate superior performance and increases robustness against state-of-the-art baselines.

There are many interesting ways in which this work can be extended. One way would be to extend this approach to work in microtransit or other settings. The primary limitation is scalability. In this work we leveraged the structure of the paratransit setting to adapt online search to work in this setting. For other demand-responsive settings we could look to apply online planning in the same way - by adapting the feasible action generation process to the domain in a way that is scalable and robust. Alternatively, we could look adapt our framework to operate in decentralized or hierarchical settings in a way that would improve scalability compared to the centralized decision-making engine presented in this work.

Chapter 4

Energy Prediction Using Multi-Task and Inductive Transfer Learning

4.1 Overview

As we have discussed, real-time decision-making for optimizing transportation systems requires reasoning over complex state-action spaces covering large geographic regions. In Chapter 3, we showed how clever heuristics can be used to prune the search space for adaptive, non-myopic search algorithms for MoD. Our approach was data-driven in that it utilized historical request patterns to build a *generative* model to evaluate current actions in the context of expected future demand. Similarly, *predictive* models have great potential to improve offline and online optimization procedures by learning efficient representations from vast volumes of data where traditional analytical methods fall short. These models utilize supervised learning to learn abstract representations of the model environment and are often referred to in the AI literature as discriminative or descriptive models. There are primarily two use cases for predictive models in the context of real-time transportation applications. First, predictive models can learn efficient representations of state in a way that can be used directly in the optimization algorithms to model future scenarios. For example, an online DVRP algorithm might require estimates of future traffic conditions or travel times which can be learned through supervised learning. Second, these methods can be used to directly learn policy or utility functions through offline simulation that can be queried efficiently at run-time. The learned functions can evaluate current actions in the context of the context of expected future reward, and therefore are non-myopic. These methods collectively fall under the field of reinforcement learning (RL).

In this chapter we address key challenges in utilizing predictive models for offline and online optimization. The key challenge we aim to address is how to overcome issues of data sparsity and quality. We focus on the problem of designing predictive models for energy and emissions in the context of fixed-line public transit. Currently, public transit agencies are focused on making their fixed-line bus systems more energy efficient by introducing electric (EV) and hybrid (HV) vehicles to their fleets. However, because of the high upfront cost of these vehicles, most agencies are tasked with managing a mixed-fleet of internal combustion vehicles (ICEVs), EVs, and HVs. In managing mixed-fleets, agencies require accurate predictions of energy use for optimizing the assignment of vehicles to transit routes, scheduling charging, and ensuring that emission standards are met. The current state-of-the-art is to develop separate neural network models to predict energy consumption for each vehicle class. Although different vehicle classes' energy consumption depends on a varied set of covariates, we hypothesize that there are broader generalizable patterns that govern energy consumption and emissions. In this paper, we seek to extract these patterns to aid learning to address two problems faced by transit agencies. First, in the case of a transit agency which operates many ICEVs, HVs, and EVs, we use multi-task learning (MTL) to improve accuracy of forecasting energy consumption. Second, in the case where there is a significant variation in vehicles in each category, we use inductive transfer learning (ITL) to improve predictive accuracy for vehicle class models with insufficient data. As this work is to be deployed by our partner agency, we also provide an online pipeline for joining the various sensor streams for fixed-line transit energy prediction. We find that our approach outperforms vehicle-specific baselines in both the MTL and ITL settings.

The work comprising this chapter has been published in the Proceedings of the 2021 Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML) [44]:

- Michael Wilbur, Ayan Mukhopadhyay, Sayyed Vazirizade, Philip Pugliese, Aron Laszka, and Abhishek Dubey. Energy and emission prediction for mixed-vehicle transit fleets using multi-task and inductive transfer learning. In *Joint European*

Conference on Machine Learning and Knowledge Discovery in Databases, 2021

4.2 Introduction

Context: Public transit agencies are focused on finding ways to make their fixed-line bus systems more energy efficient by introducing electric vehicles (EVs) and hybrid vehicles (HVs), which have reduced impacts on the environment in comparison to traditional vehicles with internal combustion engines (ICEVs). However, EVs and HVs are expensive and in practice, transit agencies have to manage a mixed-vehicle fleet, requiring complex scheduling and assignment procedures to maximize the overall energy efficiency [78, 79, 11] while satisfying the expectations of transit demand. This is turn requires the ability to estimate energy and emissions of vehicles on assigned routes and trips. Energy prediction models can be categorized based on their modeling scale. Microscopic models aim to estimate vehicle energy consumption at a high frequency [80]; however, this comes at the cost of reduced accuracy. For most system level optimization, macroscopic models that aim to predict energy consumption at an aggregated spatial or temporal span are sufficient [7, 80].

State of the Art: There has been significant research on macroscopic models for EVs over recent years. For example, De Cauwer et al. used a cascade of ANN and linear regression models for energy consumption prediction for EVs using vehicle speed, voltage, current, SoC, road network characteristics, altitude, and weather [81]. Their model, however, did not use traffic data and the approach had a mean absolute error (MAE) of 12-14% of average trip consumption. Vepsäläinen et al. used a linear model using temperature, driver behavior, and roadway characteristics and found that EV energy consumption was 15% lower on suburban routes compared to city routes. A recent study by Pamula et al. used a DNN with stacked autoencoders and an multi-layer perceptron to predict energy consumption between stops. Their model used travel time, elevation change, and modeled weather as categorical variables [82]. However, most of the prior work relied on learning separate models for each vehicle class [7, 82, 83].

Challenges: There are several unresolved challenges for public transit operations teams. First, modern public bus fleets include not only a mix of vehicle classes (ICEV, HV, and EV), but also different vehicle models within each class. For example, out partner agency, Chattanooga Area Regional Transportation Authority (CARTA), manages a total of six ICEV models, two HV models, and two EV models. Training separate models for each type of vehicle ignores generalizable information that is not explicitly modeled in the feature space. For example, Ayman et al. modeled EVs and ICEVs without sharing model parameters between classes [7]. Second, the number of vehicles in each class varies greatly, which leads to an uneven distribution of data available for training the energy or emission prediction models. Third, and similar to the second problem in principle, when a new vehicle class is added to an existing fleet, the agency must deploy *some* vehicles, obtain data, and then learn a new predictive model from scratch.

Our Contributions: We address these challenges as multi-task learning (MTL) and inductive transfer learning (ITL) problems. Although different vehicle classes' energy consumption depends on a varied set of covariates through different non-linear functions, we hypothesize that there are broader generalizable patterns that govern the consumption of energy and vehicle emission. That is, if an agency has access to *many* vehicles, and consequently data, from each vehicle class (ICEVs, HVs, and EVs), we formulate emission (and energy) forecasting as an MTL problem. We show that this approach improves the predictive accuracy for all vehicle classes compared to a baseline where separate networks are trained to predict emissions (and energy) for each class. In a situation with imbalanced data or when an agency introduces a new model or class, we show that it is possible to learn a model for classes with sufficient data, and *transfer* the learned abstraction to improve the predictive accuracy for the class with insufficient data. The benefit of ITL is the ability to deploy the model earlier than the time required to collect enough samples to train a separate model for the new class. Finally, we highlight that real-world transit problems require collecting, cleaning, and joining data from various sources, formats, and precision. We provide a general online pipeline for joining the various sensor streams (vehicle telemetry and trajectory data with external data sources such as weather, traffic, and road infrastructure) for training and maintaining the fixed-line transit energy prediction models. We evaluate our MTL and ITL models using real-world data from our partner agency's mixed-fleet of EVs, HVs, and ICEVs. We show that in both the MTL and ITL settings, our approach outperforms state-of-the-art methods. The greatest improvements over baselines were in the ITL setting when the target vehicle class suffers from a lack of data. However, we also find that in some cases ITL does not work well, such as when transferring learned abstractions from EV to ICEV.

4.3 Model

4.3.1 Predicting Energy Consumed and Emissions

Transit agencies are concerned with reducing a) costs by limiting energy used, and b) the impact of their vehicles on the environment by reducing emissions. For ICEVs and HVs, energy expended by a vehicle is a function of the fuel consumed, measured in liters. On the other hand, the energy expended by an EV is a function of the dissipated charge of its battery, which is the change in its state-of-charge (SOC). This presents a problem since transit agencies primarily use prediction models to optimize the assignment of vehicles to trips. As a consequence, they require a common metric to compare across vehicle classes in their mixed-fleet for both energy consumed and emissions. For energy, we use kWh. For ICEVs and HVs, we convert liters of diesel fuel consumed to kWh using a conversion rate of 10.639 kWh/liter [84]. For EVs, we multiply the change in SOC and the capacity of the battery. We measure emissions as kg of CO_2. For ICEVs and HVs, fuel consumed in liters can be converted to emissions (kg CO_2) at a rate of 2.689 kg/liter [85]. For EVs, dissipation in charge can be converted to emissions (kg CO_2) at a rate of 0.707 kg/kWh [85]. As shown

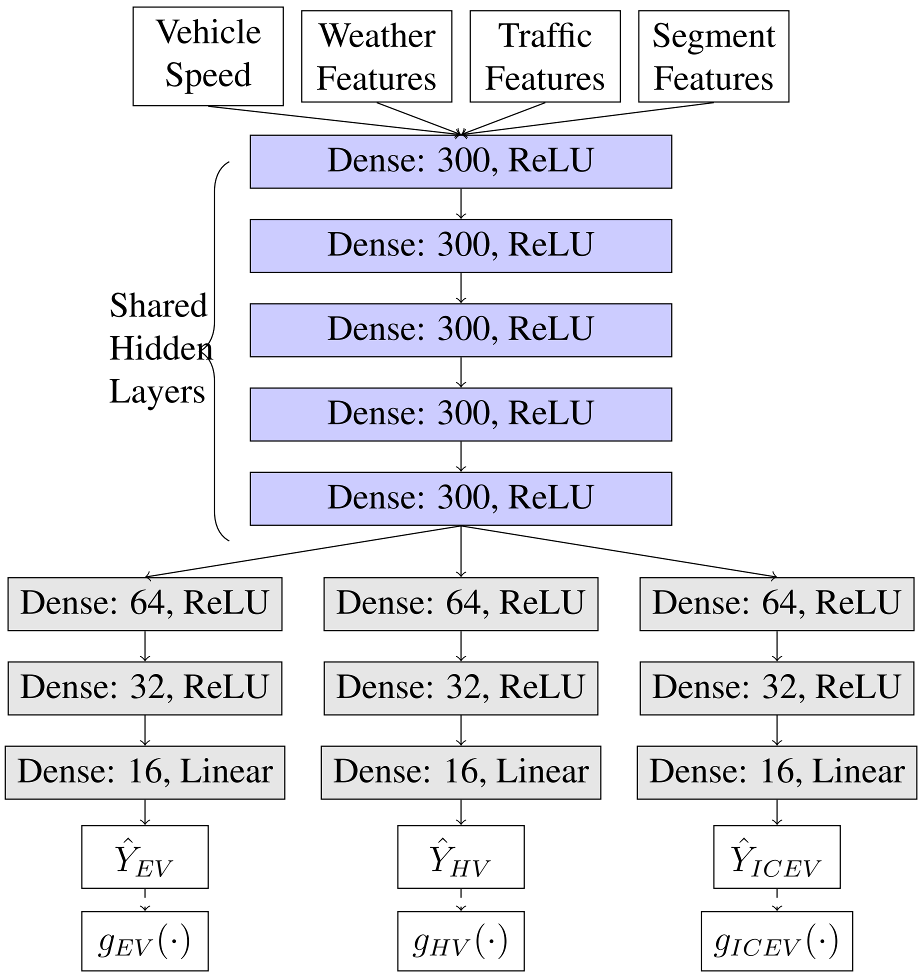


Figure 4.1: MTL Model: DNN with hard parameter sharing for predicting emissions (kg CO_2) of EVs ($\hat{Y}_{EV}$), HVs ($\hat{Y}_{HV}$) and ICEVs ($\hat{Y}_{ICEV}$). Energy consumed (kWh) is a linear function $g_i(\cdot)$ for vehicle class i, separate of the neural network per the conversion discussed in Section 4.3.1.

in Figure 4.1 and Figure 4.2, the function $g_i(\hat{Y}_i)$ represents the linear conversion between the predicted target (emission) and energy consumed for an arbitrary vehicle class denoted by the index i.

4.3.2 Preliminaries and Model Formulation

Our goal is to learn energy consumption and emissions in a mixed fleet of vehicles conditional on a set of relevant determinants (Figure 4.1 and Figure 4.2). We refer to learning prediction models as *tasks*, consistent with the terminology in the area of trans-

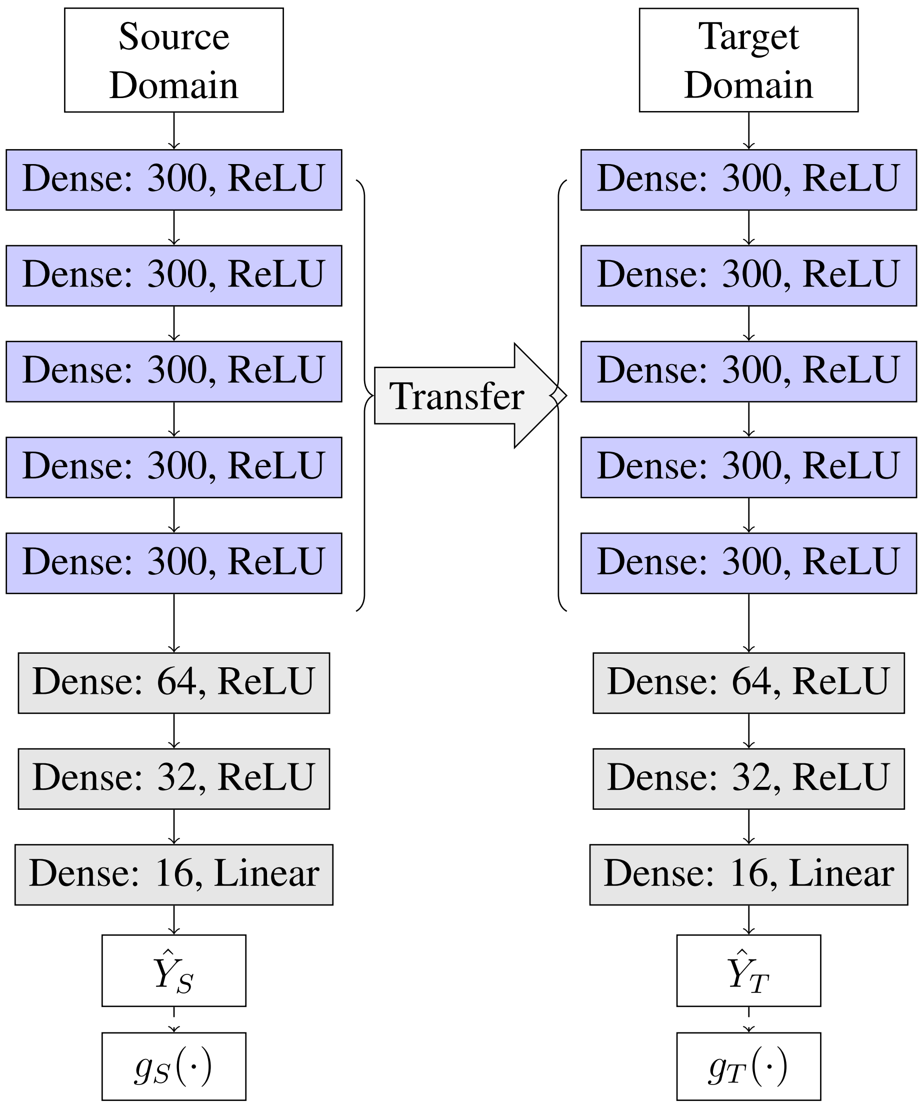


Figure 4.2: ITL Model: shared-hidden layer parameters are frozen and transfered to the target model.

fer learning [86]. We introduce the formalism for our problem next. We define a *domain* $\mathcal{D}$ as the combination of a feature space $\mathcal{X}$ and a probability distribution $P(X)$, where $X = \{x_1, x_2, \dots\} \in \mathcal{X}$. For example, $\mathcal{X}$ can include features like vehicle speed and weather. Given a specific domain, a *task* is then defined as $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$ where $\mathcal{Y}$ is the space of output labels, and f is a predictive function over $y \in \mathcal{Y}$ conditional on x. Probabilistically, f denotes the probability of a realization of y given x ($P(y \mid x)$). For example, Y can denote the energy consumed by a vehicle, and subsequently, the function f can be used to denote a distribution on the energy consumed conditional on the determinants. Typically, f is unknown; instead, we assume access to observations (data) in the form of input-output pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$. We deal with a scenario with multiple tasks (and associated domains). Specifically, there are three vehicle classes, and therefore three domains $\mathcal{D}_{EV}, \mathcal{D}_{HV}, \mathcal{D}_{ICEV}$ representing the domains of EVs, HVs and ICEVs, respectively. Similarly, we have three output label spaces $\mathcal{Y}_{EV}$, $\mathcal{Y}_{HV}$, and $\mathcal{Y}_{ICEV}$, and three predictive functions f_{EV}, f_{HV}, and f_{ICEV}, which need to be learned.

The functions f_{EV}, f_{HV}, and f_{ICEV} are parameterized by a set of parameters θ, that we seek to learn by minimizing a predefined loss function given the observed data. The input features for each of the vehicle class domains are derived from the characteristics of the road segments, weather, traffic features, and vehicle dynamics. Therefore, we can state that the feature spaces are equivalent, $X_{EV} = X_{HV} = X_{ICEV}$. Additionally, the marginal probability distributions over the features are independent of vehicle class and therefore, the marginal probability distributions over the features are equivalent, $P(X_{EV}) = P(X_{HV}) = P(X_{ICEV})$. Finally, given that the feature spaces and marginal probability distributions are the same for all vehicle classes, we have that $\mathcal{D}_{EV} = \mathcal{D}_{HV} = \mathcal{D}_{ICEV}$.

As the energy consumed for ICEV and HV vehicles are measured in fuel (liters) consumed, the space of output labels $\mathcal{Y}$ is the set of positive real numbers $\mathbb{R}_+$. On the other hand, EV vehicles have regenerative braking, therefore the energy consumed can take negative values and the task space for EV vehicles is $\mathbb{R}$. Additionally, since the performance of the three vehicle classes varies greatly, we consider that the predictive functions for each vehicle class are different; as the conditional probability distributions are not equal and $P(Y_{EV}|X_{EV}) \neq P(Y_{HV}|X_{HV}) \neq P(Y_{ICEV}|X_{ICEV})$. Finally, we can generalize the problem to n classes of transit vehicles in the fleets (e.g. the classes can be categorized based on the model and year as well); such a generalization will focus on learning the tasks $\{\mathcal{T}_1 \neq \mathcal{T}_2, \cdots, \neq \mathcal{T}_n\} \in \mathcal{T}$, given the domains $\{\mathcal{D}_1 = \mathcal{D}_2, \cdots, = \mathcal{D}_n\} \in \mathcal{D}$.

Table 4.1: Data description; data collected from Jan 1 2020 to July 1 2020.

Data Source	**Description**	**Features**	**Frequency**	**Scope**
ViriCiti - ICEVs	vehicle telemetry	fuel level, GPS	1 Hz	3 vehicles
ViriCiti - HVs	vehicle telemetry	fuel level, GPS	1 Hz	4 vehicles
ViriCiti - EVs	vehicle telemetry	current, voltage, GPS	1 Hz	3 vehicles
Clever Devices	automated vehicle location	trip ID, vehicle ID	0.1 Hz	all vehicles
HERE	traffic (per TMC)	jam factor, current speed, free flow speed	0.0166 Hz	major roads, highways
DarkSky	weather	visibility, wind speed, precipitation intensity, humidity, wind gust, temperature	0.0033 Hz	whole city
Static GTFS	transit schedule	routes, trip IDs, stop sequences, stop locations (latitude, longitude), schedule trip times, trip shape (GeoJSON)	static	whole city
GIC - Elevation	LiDAR elevation	location, elevation (meters)	static	whole city
Trip Segments	multiple sources	segment length, time to travel, average speed, roadway type	static	whole city

4.4 Approach

We now discuss our approach to learning the energy prediction functions (f_{EV}, f_{HV}, and f_{ICEV}). In order to perform data-driven learning, we first need to accumulate data from various sources. In real-world problems pertaining to public transportation, creating a data pipeline is often an arduous task due to the variety of data sources, formats, recording precision, and data collection frequency. As a result, we begin by discussing the data sources (Table 4.1) and the data pipeline. We gather data from 3 ICEVs, 4 HVs, and 3 EVs from our partner agency for a period of six months from January 1, 2020 to July 1, 2020. Each vehicle has a telematics kit produced by ViriCiti LLC [37], that provides speed and GPS positioning at a minimum of 1Hz resolution. In addition, for ICEVs and HVs the sensors provide fuel consumed (liters) while for EVs we collect current and voltage levels which are used to calculate energy consumed as well as emissions. Each vehicle is equipped with a kit from Clever Devices [36]. The Clever Devices feed provides a unique vehicle ID corresponding to the vehicle ID in the ViriCiti feed, as well as the unique trip ID which maps to scheduled trips in the static General Transit Feed Specification (GTFS) [27].

We collect weather data from multiple weather stations within the transit region at 5-minute intervals using the DarkSky API [31], including temperature, humidity, wind speed, and precipitation. Traffic data was collected at 1-minute intervals using the HERE API [38], which provides speed recordings for segments of major roads. The traffic data is reported per *TMC* (Traffic Message Channel), which is a custom geographical mapping unit. We perform map matching similar to prior work [7] to obtain traffic data for each road segment of interest to us. Road network map data was collected from OpenStreetMaps [24]. Lastly, we collect static GIS elevation data from the state Geographic Information Council which provides high-resolution digital elevation models (DEMs) derived from LiDAR elevation imaging, with a vertical accuracy of approximately 10cm.

Fixed line transit vehicles travel at pre-determined times (trips) covering a sequence of stops along a route. The latitude and longitude of each stop and the geographical shape of the path (the route segment) that the vehicles travel by visiting each stop is specified using the static GTFS schedule published by CARTA. Using this information, it is straightforward to divide the path taken by a bus during a given trip into a sequence of segments $\langle SEG \rangle$, where each segment is marked by a start stop and an end stop. As the specific characteristics of segments are important, a unique segment is created for every spatial path that exists between a pair of stops. Note that effectively, each SEG_i is described using a discrete sequence of points (latitude and longitude), close enough to draw the shape of the road on the map. We use these segments as the fundamental spatial unit for which we predict emissions (or energy). This has two advantages: first, the generation of route segments for prediction can be derived directly from a transit agency's schedule, rather than relying on external infrastructure data such as OSM [7] or time intervals [83], and second, segments can be shared between trips thereby providing additional data for learning.

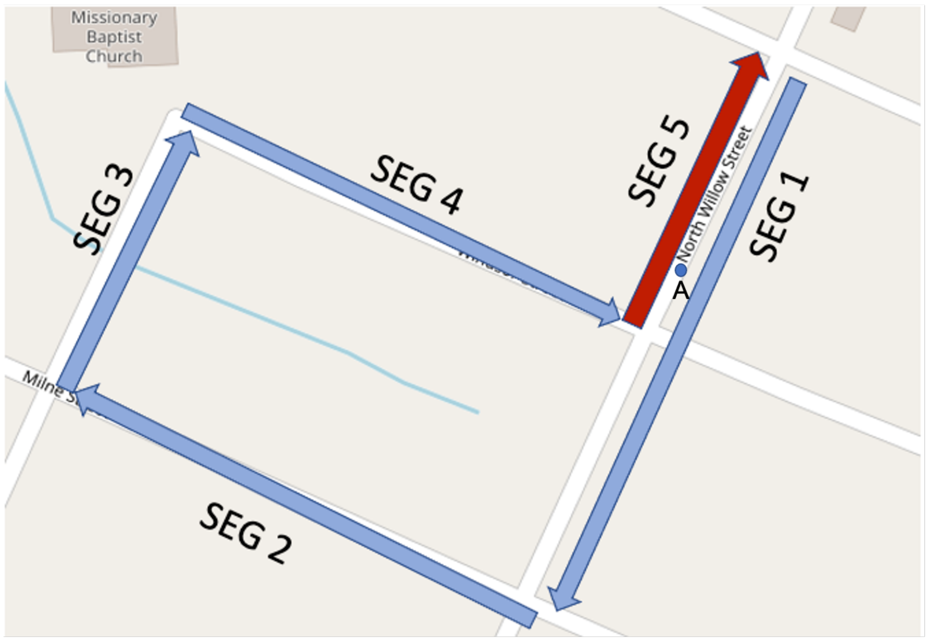


Figure 4.3: Overlapping segments. Segments 1 and 5 traverse the same section in opposite directions.

4.4.1 Mapping Vehicle Trajectories to Route Segments

To generate the joined data samples, we first map the vehicle trajectories to segments. By joining the ViriCiti and Clever Device feeds, we determine a set of GPS points that a vehicle traverses. We refer to this ordered sequence of points as a trajectory T consisting of spatial points $\{l_1, l_2, \ldots\}$. Consider that the trajectory T serves the trip R. The goal of the mapping process is to label each location $l_i \in T$ to a corresponding segment $SEG_j \in R$, thereby representing the specific segment that each vehicle traverses at a specific point in time.

In principle, it is possible to perform an exhaustive search on the segments to identify the one that matches (or is the closest to) each point in a trajectory. However, such an approach does not work in practice with real-word trajectory feeds due to two reasons. First, routes often traverse segments between spatial points close to each other during trips. For example, consider the overlapping segments in Figure 4.3 in which the vehicle passes through SEG_1 relatively early in the trip and through SEG_5 later. Due to noise in the

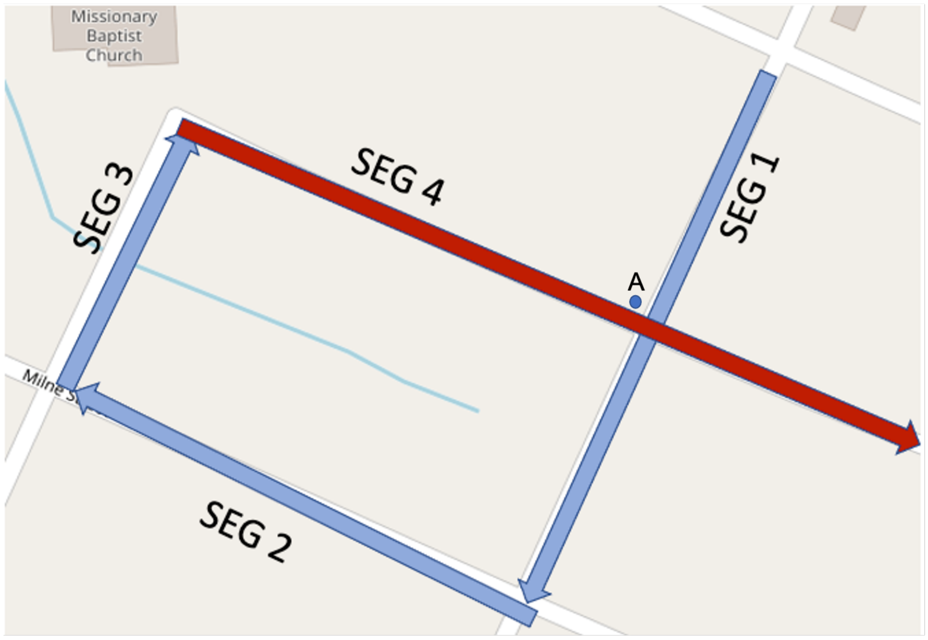


Figure 4.4: Overlapping segments. Segments 1 and 5 traverse the same section in opposite directions. Intersecting segments. Vehicle locations near the intersection of segments 1 and 4 can lead to incorrect mapping. Stops not shown.

measurement, a point early in the trip can erroneously get mapped to SEG_5, resulting in incorrect representation of the features that are induced by the segment. Similar problems arrise when segments cross each other as shown in Figure 4.4. Our exploratory analysis on the data obtained from our partner agency showed several examples of such incorrect mappings. Second, the mapping of trajectory data to segments is computationally challenging for transit agencies. As an example, consider our partner agency CARTA, which operates a total of 60 vehicles. The number for bigger cities is larger in orders of magnitude; for example, the New York Metropolitan Transit Authority (NY-MTA) operates more than 5000 buses [87]. Considering location data collected at the frequency of 1 Hz for 3 years, the matching must be done for over 3.5×10^9 spatial locations, each of which could potentially be mapped to one out of hundreds of segments (for a larger city like New York, the number of matches is 3×10^{12}).

To alleviate these concerns, we propose an algorithm for mapping vehicle trajectories to route segments (Algorithm 3). The algorithm takes the trajectory T of the vehicle traversing

Algorithm 3 Mapping Trajectories to Route Segments

```
1: **Input:**
2: $R \leftarrow$ sequence of segments $\{\textit{SEG}_0, \ldots, \textit{SEG}_N\}$ for each trip
3: $T \leftarrow$ set of vehicle GPS locations $l$ along the trip
4: $W \leftarrow$ number of segments to lookahead
5: $B \leftarrow$ max distance between segment and vehicle GPS
6: **Output:**
7: $TrajSegMap \rightarrow$ list of segments for each *SEG* in $R$
8: **Initialization:**
9: $c \leftarrow 1$, index of current segment
10: $TrajSegMap \leftarrow []$
11: **for** $i \in \{1, \ldots, |T|\}$ **do**
12:     $SegWindowDist \leftarrow []$
13:     **for** $j \in \{c, \ldots, c + W\}$ **do**
14:         test
15:         **if** $j \leq |R|$ **then**
16:             $SegWindowDist.push(dist(\textit{SEG}_j, l_i))$
17:         **else**
18:             $TrajSegMap[i] \leftarrow None$
19:         **end if**
20:     **end for**
21: **end for**
22: **Return:** $TrajSegMap$
```

the sequence of segments $\langle SEG \rangle$ of trip R. During matching, we maintain a lookahead window, denoted by W, that represents the number of segments to consider for the match. For example, if a location $l_i \in T$ is already matched to segment $\textit{SEG}_c$ in a route, then for matching the next location $l_{i+1} \in T$, we consider the set $\{\textit{SEG}_c, \ldots, \textit{SEG}_{c+W}\}$. By maintaining a short lookahead, we alleviate duplicate matches from segments further away in the route. Also, a shorter lookahead provides computational efficiency as opposed to an exhaustive search. We maintain a tolerance distance B for matching where a segment is matched to a location from a trajectory only if the distance between them is less than or equal to B. The function $dist(\textit{SEG}_j, l_i)$ is used to calculate the minimum distance between segment $\textit{SEG}_j$ and GPS point l_i.

4.4.2 Generating Samples

To generate the joined data samples, we split each of the trajectories T based on the locations mapped to trip segments. We create one data sample per continuous travel on a trip segment, providing average speed and the total fuel/energy consumption and emission on that segment. For ICEVs and HVs, the fuel consumed is provided in liters. While EVs provide state-of-charge (SOC) readings, the precision is too low to use for representing energy consumed. Therefore, we estimate the amount of energy from the battery current A and voltage V. The energy used between consecutive data points is given by $E_i = A_i \cdot V_i \cdot (TS_i - TS_{i-1})$, where E_i, A_i, and V_i are the consumed energy (Joule), current (Ampere), voltage (Volt) at time step i, respectively, and TS_i is the timestamp (in seconds) at time step i. To get the energy on a segment, the energy consumed between each sample is accumulated for all locations of the vehicle mapped to that segment.

Weather features for each sample are taken from the weather reading closest to the time at which the vehicle starts traversing a segment. For traffic features, we take the average jam factor (JF) and speed ratio (SR) of all TMCs mapped to the segment traversed by the vehicle when the vehicle enters a segment. Speed ratio is defined as the traffic speed divided by the free flow speed.

4.4.3 Learning

Recall that our goal is to address two specific problems. The first scenario is where a transit agency has access to *many* vehicles, and consequently data, from each vehicle class. In this case, our goal is to improve the predictive accuracy of f for all tasks. One method of addressing this problem is to learn a predictive model f over each vehicle class. However, we hypothesize that there are generalizable patterns between vehicle classes that can be leveraged to aid learning. Consequently, we formulate a MTL model as shown in Figure 4.1. We use hard parameter sharing to learn a common representation of the input

Table 4.2: Data processing summary.

Class	Model Year	Vehicles	Raw Samples	Distance Filtering	Final Samples
ICEV	2014	3	139,652	127,212	114,348
HV	2014	4	235,671	223,913	201,491
EV	2018	3	48,969	47,804	43,022

Table 4.3: Pearson's correlation coefficient of input features with emissions.

Class	Length Segment	Time to Travel	ΔEle vation	max ΔE levation	Speed Ratio	Visibi lity	Wind Speed	Precip itation	Hum idity	Wind Gust	Jam Factor	Tempe rature	Avg Speed
EV	0.860	0.752	0.523	0.222	0.038	0.008	-0.002	-0.003	-0.009	-0.012	-0.015	-0.037	-0.093
HV	0.916	0.838	0.505	0.135	0.038	0.006	0.004	-0.008	-0.008	-0.002	-0.026	0.013	-0.134
ICEV	0.886	0.865	0.539	0.103	0.028	0.004	0.011	-0.005	0.001	≈ 0	-0.016	-0.005	-0.262

features which enables us to extract generalizable patterns across the tasks. Additionally, each task (vehicle class) has a vehicle-specific set of hidden layers which outputs the predicted energy consumed/emissions for EVs ($\hat{Y}_{EV}$), HVs ($\hat{Y}_{HV}$), and ICEVs ($\hat{Y}_{ICEV}$) along route segments. At each training iteration, a batch of samples from EVs, HVs, and ICEVs is fed through the network and mean-squared error (MSE) loss is calculated between the predicted target and true target for each vehicle class. The gradient of the loss is then propagated back through the network.

The second problem we seek to address is where an agency has significant variation in the number of vehicles from each class. In such a case, while a common model can be learned using the MTL framework, the tasks with a significantly larger number of samples are likely to *dominate* learning. Also, learning a model solely for the task with few samples can result in overfitting. In this case, we seek to learn f for classes with sufficient data (source model) first, and *transfer* the learned abstraction to improve the predictive accuracy for the class with insufficient data (target model). Our ITL framework is shown in Figure 4.2. When training the target model, the transferred layers are frozen and only the vehicle-specific layers are updated during training.

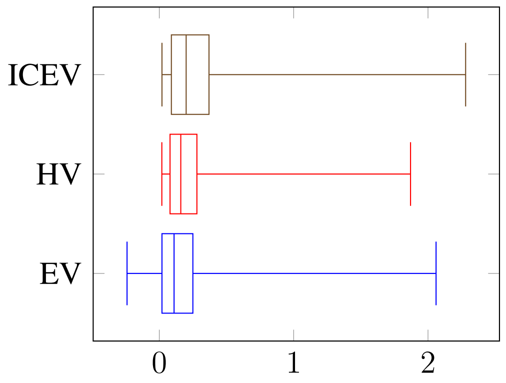


Figure 4.5: Distribution of emissions (kg CO_2) per trip segment for each vehicle class.

4.5 Experiments and Results

Vehicle telemetry, weather, and traffic data is collected for a six-month period between January 1, 2020 and July 1, 2020 for 10 vehicles as shown in Table 4.2. We include two post-processing steps in generating the final datasets for each respective vehicle class. First, we remove partial trajectories by eliminating samples where the total distance traveled was less than 50% of the segment length and greater than 150% of the segment length. Second, to address outliers and potential errors in the mapping process, we remove samples with the target value (energy/emission) in the bottom 2% and top 2% quantiles. The final data size is shown in last column of Table 4.2.

The distributions of emissions (kg CO_2) and energy (kWh) consumption are shown in Figure 4.5 and Figure 4.6. As energy consumption for ICEVs and HVs is derived from liters of diesel fuel consumed, emissions must be greater than 0 kg CO_2 for these vehicle classes. The EVs in the fleet have regenerative braking, which allows for energy consumed, and thus emission, to be negative. We predict energy/emissions per route segment. The distributions over energy and emission for each of the vehicle classes has a long right tail and the average varies between vehicle classes. Therefore, the *task* of energy/emission prediction is also different, by virtue of having a different distribution over the space of output labels $\mathcal{Y}$.

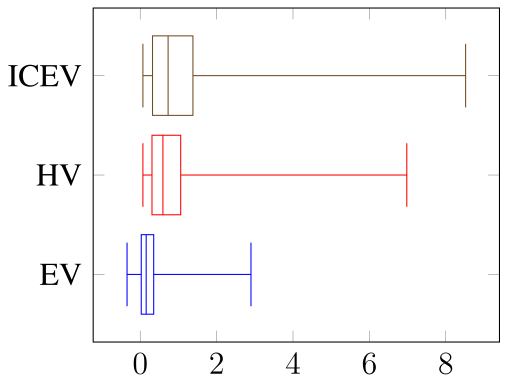


Figure 4.6: Distribution of energy (kWh) consumption per trip segment for each vehicle class.

The Pearson correlation coefficient between input features and emissions is provided in Table 4.3. Distance traveled and time to traverse the segment have a strong positive correlation with emissions. Δ Elevation, which is the change in elevation from the start to the end of the segment, also has a strong correlation with emissions for all vehicle types. Max Δ elevation, which is defined as the difference between the maximum and minimum elevation along the segment, has a relatively weaker correlation. Additionally, the average vehicle speed has a stronger negative correlation of -0.262 with emissions for ICEVs than with HVs (-0.134) and EVs (-0.093).

4.5.1 Hyperparameter Tuning and Baseline Models

We randomly select 43,022 samples from each vehicle class. For each vehicle class, we use 80% of the samples for training and 20% for testing. Of the training samples, 10% are withheld from training and used as a validation set to identify the best set of hyperparameters for the subsequent analyses. We perform the hyperparameter search using the model derived from the MTL formulation.

We tested shared hidden layer widths of $\{200, 300, 400\}$ and shared hidden layer depths of $\{3, 4, 5\}$. We use 3 vehicle-specific layers and tested the configurations of $\{128, 64, 32\}$ and $\{64, 32, 16\}$. Mean-squared error (MSE) is used for the loss function and the networks are optimized using the Adam algorithm [88]. We test learning rates of $\{0.01, 0.005, 0.001, 0.0005, 0.0001\}$ and batch sizes of $\{64, 128, 256, 512\}$. The best performing configuration is shown in Figure 4.1, which consists of 5 shared hidden layers of 300 fully connected neurons with ReLU activation functions [89], and 3 vehicle-specific hidden layers of 64, 32, and 16 hidden neurons respectively. For the output layer we test using ReLU as well as linear activation functions for ICEVs and HVs and linear activation function for EVs, however we find that using a linear activation function as the output layer for all 3 vehicle classes provides the best performance. An early stopping strategy was performed, where we stopped training if MSE on the validation set did not improve for 10 epochs. The best performing learning rate was 0.0005 and the best batch size was 256.

In the baseline model no layers are shared between vehicle-classes resulting in a separate neural network for each vehicle class. The same grid search from the proposed models was used to find the hyperparameters of the baseline models. In all experiments, we use Kaiming initialization [90] to initialize the weights of the networks.

4.5.2 Multi-task Model Evaluation

First, we investigate the performance of the MTL model compared to vehicle-specific baseline models. To evaluate the robustness of the models, we train 10 MTL models (30 vehicle-specific models, 10 for each vehicle class) and present the average MSE and MAE in Figure 4.7 and Figure 4.8. Models are trained for up to 150 epochs. We find that for all vehicle classes, the MTL model outperforms the vehicle-specific baseline models. The mean percent improvement in MSE is 8.6%, 17.0%, and 7.0% for ICEVs, HVs, and EVs, respectively. The mean percent improvement in MAE is 6.4%, 9.0% and 4.0% for ICEVs, HVs, and EVs respectively.

Even with improved accuracy, it is important to investigate the bias and the variance of the proposed approaches. Therefore, we repeat the entire evaluation using 30 datasets creating through bootsrapping [91] from the original data. At each iteration, we sample

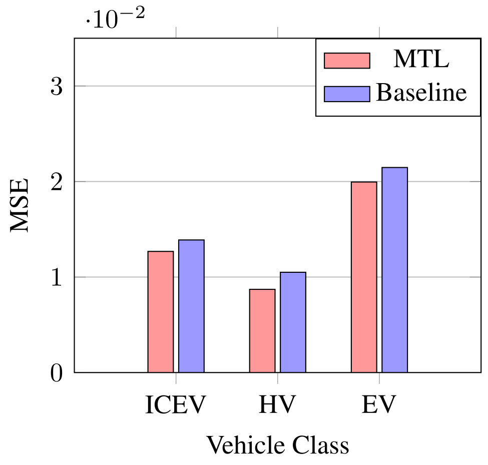


Figure 4.7: MSE of MTL model compared to vehicle-specific neural network models (baseline) on testing set. Prediction target: emissions (kg CO_2).

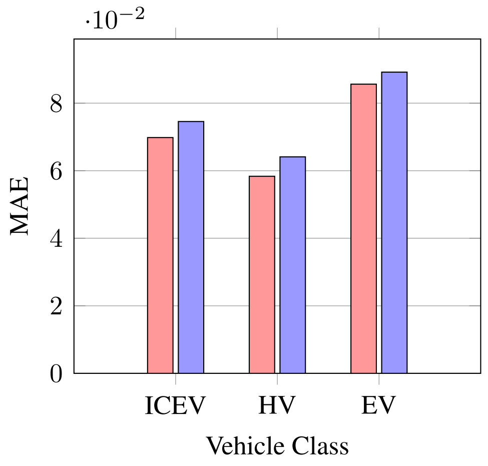


Figure 4.8: MAE of MTL model compared to vehicle-specific neural network models (baseline) on testing set. Prediction target: emissions (kg CO_2).

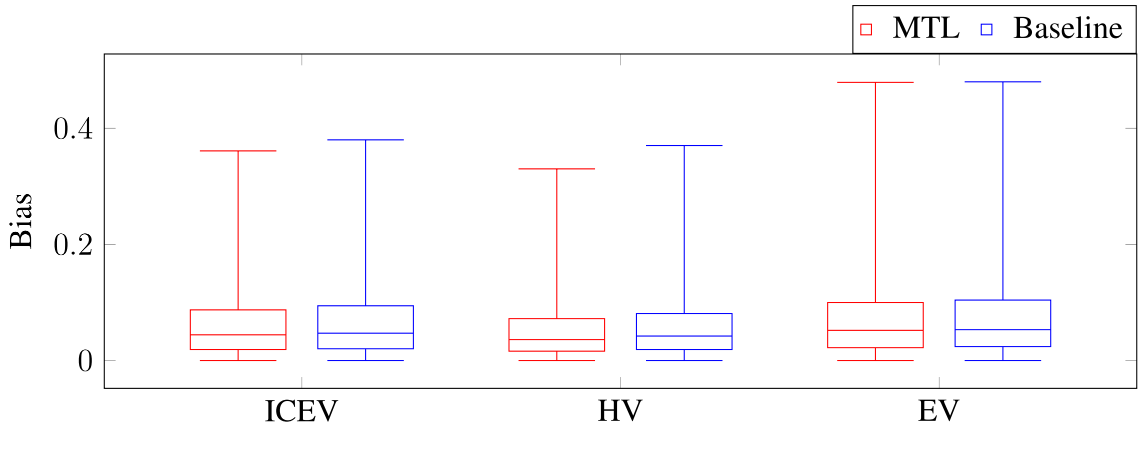


Figure 4.9: Distribution of MTL and baseline model bias per sample for each vehicle class from bootstrap evaluation, 30 bootstrap iterations. Prediction target: emissions.

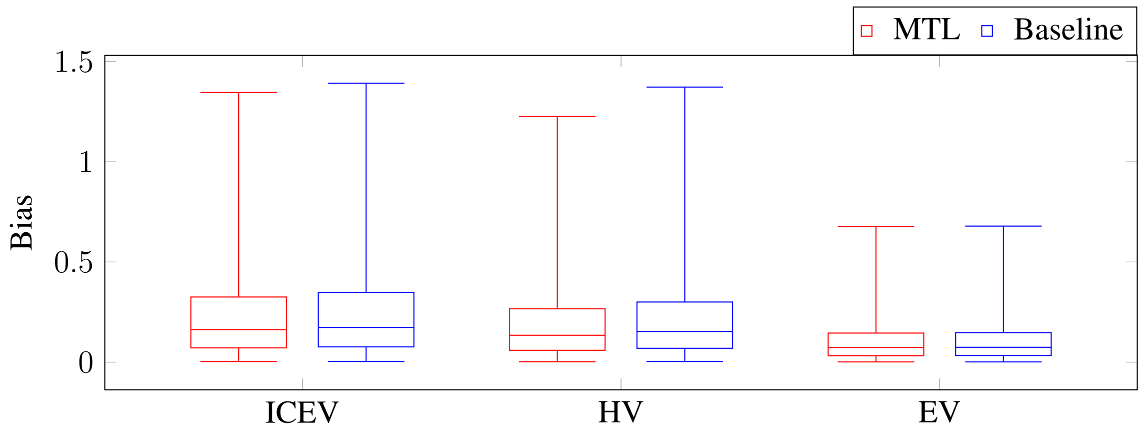


Figure 4.10: Distribution of MTL and baseline model bias per sample for each vehicle class from bootstrap evaluation, 30 bootstrap iterations. Prediction target: energy.

a training set, with replacement, from the ICEV, HV, and EV datasets. The samples not selected for each training set are used as the testing set for that iteration. For each iteration, we train a single MTL model and vehicle-specific baseline models on the training set and evaluate on the testing set. The distribution of empirical bias per sample for the MTL and baseline models is presented in Figure 4.9 and Figure 4.10. We observe that the MTL model results in a lower bias for each vehicle class compared to the baseline models. The MTL model also results in lower median variance per sample.

4.5.3 Inductive Transfer Learning Evaluation

Next, we evaluate the performance of the ITL model formulated in Figure 4.2. To train the ITL models, we use data from all of the three vehicle classes, each of which contains 43,022 samples, as outlined in Section 4.5.1. For each pair of source and target task, we first train the source model, freeze the shared hidden layers, and transfer to the target model. Then, we optimize the target model's vehicle-specific layers. For each model, the available sample size to train the target model is varied from 2%, 5%, 10%, and 15% of the total number of available samples to investigate the influence of sample size in training of the target models. This is consistent with what transit agencies might face in practice; as a new

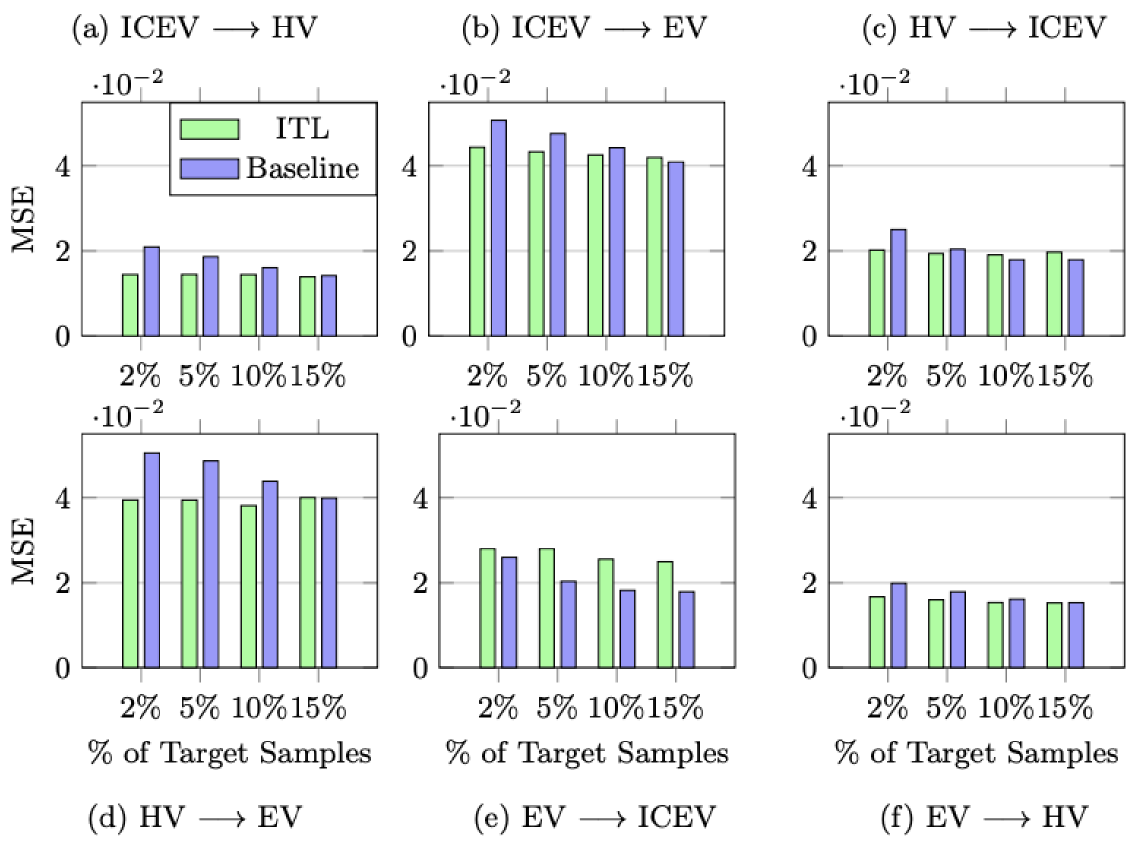


Figure 4.11: ITL models compared to corresponding baselines. ITL model is trained on full dataset in the source vehicle class and is evaluated on the target vehicle class (source ⟶ target). Average MSE compared to fraction of data samples used for training in the target vehicle class. Prediction target: emissions (kg CO_2).

vehicle is introduced, agencies gradually collect more data from it. We test our approach for all pairs of vehicle classes.

To compare the performance of the models, we train baseline models that only use the training data from the target domain. For example, while evaluating inductive transfer from EV to ICEV with 2% of the target data available, the baseline model is trained exclusively on the same amount data from ICEV class. In order to consider the randomness in training process, when evaluating the target and baseline models, we trained each model 10 times on 10 random samples from the target domain's dataset and 10 different initial values for the parameters using Kaiming initialization [90].

We provide the results of the proposed ITL approach in Figure 4.11. We observe that in

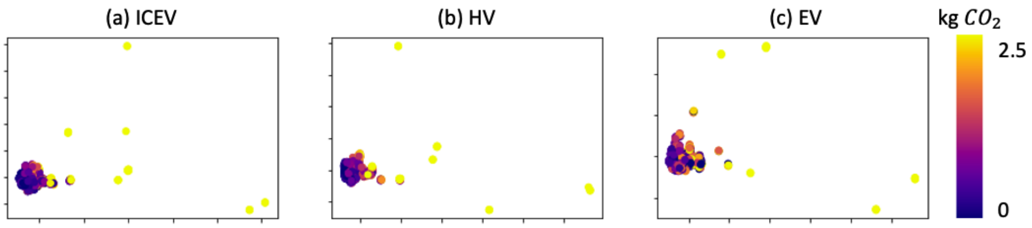


Figure 4.12: t-SNE on raw input features for each data sample from the source domain. t-SNE parameters: number of components=2, perplexity=10, initialization=PCA, number of samples=860 (2% of dataset)

general, the proposed approach results in improved forecasting accuracy across the tested scenarios (except when EV is used as source and ICEV is used as target). We also observe that as the amount of data from the target domain increases, both the ITL and the baseline method show improved forecasting accuracy; however, the baseline methods shows relatively higher improvement, to the extent of outperforming the ITL framework in some cases (15% data from target domain in Figure 4.11 b, c, e and f).

Additionally, we seek to understand the role of the shared-hidden layers in our proposed approach. Conceptually, the role of such layers in the target model is to extract generalizable patterns across the spectrum of tasks to aid learning in the target task. We use t-distributed stochastic neighbor embeddings (t-SNE) [92] to visualize the separation of multi-dimensional information in a two-dimensional space. In Figure 4.12, we show t-SNE on the raw input features of the three vehicle classes color coded by emissions (kg CO_2). All three plots are very similar, thereby corroborating our assumption that the input features are similar across the tasks ($\mathcal{D}_{EV} = \mathcal{D}_{HV} = \mathcal{D}_{ICEV}$). We separately apply t-SNE on the output of the shared-hidden layers across all pairs of source and target tasks and show the results in Figure 4.13. We observe that the ICEV source model and HV source model (plots (a) to (f) of Figure 4.13) effectively discriminate the samples with high emissions and low emissions (increasing the distance between light points and dark points). On the other hand, EV source model (plots (g) to (i) in Figure 4.13) shows poor discrimination, reflecting the negative transfer.

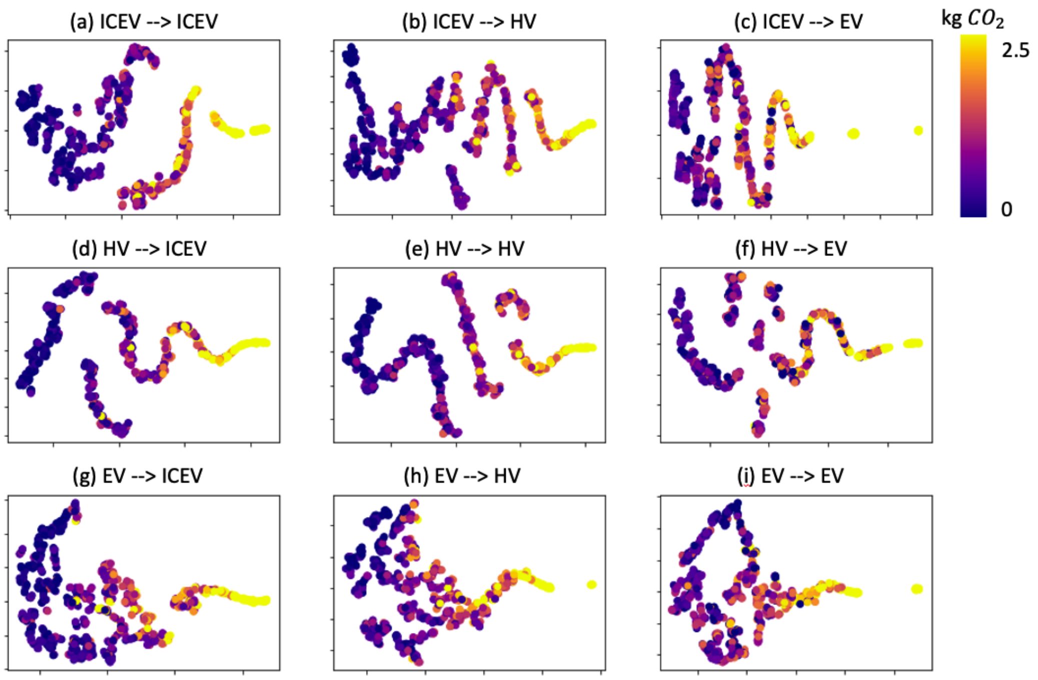


Figure 4.13: t-SNE on the output of shared-hidden layers for each data sample from the target domain. t-SNE parameters same as Figure 4.12.

4.5.4 Discussion

We now present the key takeaways from the experiments. First, we observe that in general, both the MTL and the ITL framework outperform the baseline methods, thereby resulting in improved emission (and consequently energy) predictions for transit agencies that operate mixed-fleet vehicles. Second, we observe that the MTL, ITL, and baseline models are less accurate in predicting EV emissions compared to HV and ICEV, most likely due to the complexity of the energy cycle in EV engines. Third, a key finding for practitioners is that the greatest improvements over baselines are seen when the target vehicle class suffers from lack of data. However, it is important to switch to standard models once sufficient data is collected for the class. The point at which such a switch should be made depends on the specific task and data at hand. In our work with CARTA, we implement a periodic check to facilitate such a switch. Fourth, we find that when the goal is to predict the emissions for ICEV class using a source model trained based on EV class dataset (this situation rarely arises in practice due to precedence of the ICEV class), ITL models underperformed baseline models, irrespective of the size of training data from the target domain. This indicates negative transfer between the EV domain and the ICEV domain.

Lastly, while this work is a general approach that can be used by cities to improve their energy prediction models there are a couple limitations agencies should be aware of. First, our models were trained on data from Chattanooga, TN, which is a mountainous city in the southern United States with a warm climate and limited snowfall or freezing temperatures. Therefore any direct transfer of our pre-trained models to other cities should take into account potential biases in these determinants. Second, like most macroscopic energy prediction models we do not take into account the impact of delays at stops or the number of passengers on the vehicles. We intend on incorporating these parameters into future work.

Code, data, and supplementary results of this study are available at https://github.com/smarttransit-ai/ECML-energy-prediction-public

4.6 Conclusion

By framing emission (and energy) forecasting as an MTL problem, we showed that an agency with access to *many* vehicles can improve the predictive accuracy for EVs, HVs, and ICEVs over current state-of-the-art, vehicle-specific models. We also showed that in a situation with imbalanced data the predictive accuracy of classes with insufficient data can be improved by transferring a learned abstraction from vehicle classes with sufficient data through ITL. Lastly, we provided a general online pipeline for joining the various sensor streams for emission and energy prediction of mixed-vehicle transit fleets.

Chapter 5

Time-dependent Decentralized Routing Using Federated Learning

5.1 Overview

In Chapters 3 and 4 we addressed challenges in the design of real-time MoD systems from the perspective of algorithm design (sequential decision-making) and AI respectively. In the next two chapters we shift our focus to deployment methods and architectures for MoD. In particular, we focus on the potential, and address challenges related to, deploying MoD applications on decentralized edge-computing architectures.

Current research largely assumes that algorithms and model training are performed in a centralized system such as the cloud. In centralized settings, the edge devices (vehicle telemetry kits, mobile devices ect.) stream various sensor data to a centralized cloud where algorithms evaluate the state of the system and return actions. An example of this is routing applications such as Google Maps which ingest large-scale data in real-time to update congestion models while providing congestion-aware routes for users. Route planning algorithms have progressed in line with the cloud environments in which they run. Current state of the art solutions assume a shared memory model, hence deployment is limited to multiprocessing environments in data centers. By centralizing these services, latency has become the limiting parameter in the technologies of the future, such as autonomous cars. Additionally, these services require access to outside networks, raising availability concerns in disaster scenarios. Therefore, this work provides a decentralized route planning approach for private fog networks. We leverage recent advances in federated learning to collaboratively learn shared prediction models online and investigate our approach with a simulated case study from a mid-size U.S. city.

The work comprising this chapter has been published in the Proceedings of the 2020

IEEE 23nd International Symposium on Real-Time Distributed Computing (ISORC) [19]:

- Michael Wilbur[1], Chinmaya Samal[1], Jose Paolo Talusan, Keiichi Yasumoto, and Abhishek Dubey. "Time-dependent decentralized routing using federated learning", in *2020 IEEE 23nd International Symposium on Real-Time Distributed Computing (ISORC). IEEE, 2020*

5.2 Introduction

Cities are evolving at a rapid pace. Over half the world's population currently lives in urban areas [93]. Along with a growing population, new ch allenges are emerging as an increase in housing density, population, and traffic strain city services. To meet these demands, both cities and private companies have turned to data-intensive applications to maximize efficiency of existing resources.

Cloud environments provide near real-time scalability of processing and storage resources, making cloud deployment the standard model for data-intensive applications. One such case is route planning services, which process millions of queries a day to guide vehicles from point A to point B. These services take into account the global transportation infrastructure and current traffic conditions to return the shortest route to the end user. In this context, routing is done on a time-dependent graph where the shortest path depends on the departure time and the current location of the end user.

Current approaches consist of deploying classic shortest path algorithms on either a single server or using a parallel approach in which a full road network is partitioned into multiple processes. The parallel approach assumes a shared memory model in which there is constant time communication between processes. Hence, its deployment is limited to multiprocessing environment such as in a data center or the cloud. By assuming constant time communication between processes, adapting these models to fog or edge deployments is intractable.

[1]These authors have contributed equally

There are numerous disadvantages in deploying route planning services to the cloud. First, the end user must send each request to a distance data center through a WAN network, inducing significant latency. This is satisfactory for current routing applications in which it is assumed that latency can be sustained for each request. However, latency demands of future technologies such as autonomous vehicles are already showing the limitations of centralized cloud-base route planning models. Additionally, cities typically use third party services [94] for dispatching emergency services. By relying on centralized services in remote data centers, cities risk service availability issues during disaster scenarios, precisely when such services are of most importance.

One solution to these issues is to move routing services to the network edge with a fog computing model. In this case, routing services are moved to road side units (RSUs), which are low powered raspberry pi like devices [95] scattered throughout a city. Each RSU provides a limited amount of processing and storage. By linking these devices together in a private city owned sub-network, a reliable network can be created for smart city route planning services that can remain in operation without connection to outside networks. Additionally, moving processing to the edge allows end users to connect to nearby RSUs and potentially reduce latency.

The primary concern in moving route services to a fog computing model is that memory is not shared between processes and communication between fog nodes is a primary source of latency to be accounted for. Therefore, we aim to provide a decentralized route planning approach that accounts for communication latency and is well-suited for deployment in private fog networks.

While the use of prediction models helps to limit communication at inference time, training of the models can induce significant bandwidth requirements as data is transferred to a centralized cloud for training. Therefore we use federated learning to collaboratively learn shared prediction models online for the route planning problem. In our approach, all training occurs at the RSUs and only the model weights are shared between processes, therefore reducing communication overhead during training. We use a case study from a mid-size U.S. city to demonstrate this work. The specific contributions of this paper are as follows:

1. We provide a decentralized route planning approach for time-dependent transportation networks. Our approach handles all routing at the network edge within a private RSU fog network and thus can remain active in scenarios where the centralized cloud is unavailable.

2. Prediction models are used to condense the search space and limit communication between fog nodes, and thus minimize response time.

3. We apply federated learning [96] to collaboratively learn shared prediction models online for the route planning problem. All training occurs on the RSUs and by sharing model weights our system avoids costly transfer of raw data.

4. We apply our approach using a simulated case study from a mid-size U.S. city and compare it to current state of the art methods.

We find that our approach reduces latency and memory requirements compared to current state of the art parallel route planning approaches. The outline of this work is as follows: Fundamental notation, related work and limitations of existing approaches is provided in Section 5.3. Section 5.4 covers the system model and deployment while 5.6 outlines our decentralized route planning approach. Lastly a simulated case study is provided in Section 5.7.

5.3 Related Work

In this section we provide necessary background on modeling transportation networks in the context of this work. Table 5.1 summarizes the symbols we used throughout this paper.

5.3.1 Transportation Networks as a Graph

Transportation networks are naturally modelled as time-dependent graphs [97]. Let $G_\tau = (V_\tau, E_\tau)$ be the time-dependent, directed graph, where V_τ is the set of vertices, $E_\tau \subseteq V_\tau \times V_\tau$ the set of edges of a road network at time interval τ.

Since the graph G_τ is time-dependent, the travel time on an edge $e \in E_\tau$ varies with time. All edges in a transportation network are weighted by a periodic time-dependent travel time function $T(e, \tau) : \Pi \rightarrow \mathbb{N}_0$ where Π depicts a set of time points or time-period (seconds, minutes or hours of a day).

The function $T(e, \tau)$ is impacted by the routes taken by all the vehicles in the road network or graph G_τ and often can be modeled as a latency function [98]. It can be learned from historical states of the network $\{G_0, G_1, ..., G_{\tau-1}\}$ and the current state of the network G_τ, to get expected travel times for time intervals $\{\tau + 1, \tau + 2, \cdots, \tau + f\}$, where f is the number of time intervals in the future. To differentiate this from the actual travel time function, we denote the learned travel time function as $\hat{T}(e, \tau)$.

By modeling the transportation network as a graph, routing becomes a problem of finding the shortest path between two nodes. In time-dependent graphs, the shortest path depends on the departure time at the source node. Hence the route query is defined by a tuple (s, d, τ_s), where $s \in V_{\tau_s}$ is the source, $d \in V_{\tau_d}$ is the destination, τ_s is the departure time from s and τ_d is the arrival time at destination d. This might result in shortest paths of different length for different departure times or even a completely different route. In contrast to the time-independent graph, here the directed route R for the query (s, d, τ_s) involves finding a sequence of edges along the time $[(e_1, \tau_1), (e_2, \tau_2), \cdots, (e_n, \tau_n)]$ from source s at time τ_s to reach destination d at time τ_d. The cost of the route $len(R)$ is defined as the sum of the time-dependent weights from the function $T(e, \tau)$ for each edge in route, such that $len(R) = \sum_{(e,\tau)\in R} T(e, \tau)$.

Table 5.1: List of symbols

Symbol	Description
$\mathbb{R}$	Real Numbers
$\mathbb{N}_0$	Natural numbers
G	Static graph, $G = (V, E)$
V	Set of network vertices
E	Set of network edges
τ_i	Actual time interval i of a day
$\hat{\tau}_i$	Estimated time interval i of a day
Π	Set of time points or time-period (seconds, minutes or hours of a day)
RSU_i	Road Side Unit i
R	Directed path from source vertex $s \in V$ to destination vertex $d \in V$, at time interval from source τ_s
R_d^s	Partial route from source vertex $s \in V$ to destination vertex $d \in V$, at time interval from source τ_s
$len(R)$	Travel time of the route R
G_τ	State of time-dependent graph (V, E) at time τ
V_τ	Set of vertices at at time interval τ
E_τ	Set of edges at at time interval τ
T	Travel time function
$\hat{T}$	Travel time predictor
$\hat{E}$	Equivalent Grid Routing predictor
SP	Shortest path algorithm that uses Travel time predictor $\hat{T}$ to find route with minimum travel time
g_i	Grid i
G^i	Subgraph whose each vertices and edges maps to grid i
$\hat{t}_v^u$	Estimated travel time from vertex $u \in V$ to vertex $v \in V$ using Travel time predictor $\hat{T}$

5.3.2 Current State of the Art Routing

Current state of the art route planning is typically deployed in centralized cloud-based systems [99, 100, 101]. In this architecture, data is stored in distributed databases while a travel time model represents the current state of the transportation network. Vehicles query a central router for routes.

We draw on two bodies of work in routing algorithms that heavily influence the central architecture for route planning. The first is single server routing where it is assumed that the routing network resides entirely in a single physical node or server which handles all routing queries. The second is parallel routing where the network is partitioned between multiple nodes and routing queries are processed concurrently using multiple processes.

The single server approach relies on work from Dijkstra [102] and Bellman and Ford [103, 104], who proposed some of the first algorithms to solve the routing problem in a single server. Many advanced route planning algorithms that exist today are variants of these works. While these algorithms compute optimal shortest paths, they are too slow to process real-world data sets such as those derived from large-scale road networks. To address this issue, there are many techniques aimed at speeding up these algorithms. Such techniques often are based on clever heuristics that accelerate the basic shortest paths algorithms by reducing their search space. Bi-directional search [105, 101] not only computes the shortest path from the source s to the target t, but simultaneously computes the shortest path from t to s on the backward graph. Guided search approaches such as A* [106] use heuristics to guide the search and limit the search space. Goldberg et al. proposed the ALT approach in which they enhance A* by introducing landmarks to compute feasible potential functions using the triangle inequality [101, 107]. In other work, contraction techniques are used to speed-up the shortest path computation. This includes highway hierarchies [108, 99] which exploits the hierarchical in road networks, while contraction hierarchies [100] contract the graph in a pre-processing stage.

To parallelize the routing problem, a full road network is typically partitioned into multiple processes and the edge expansion proceeds similarly to Dijkstra. In parallel versions of Dijkstra, the priority queue is based on a shared memory model where it is assumed that communication between processors is constant [109, 110]. The parallel priority queue supports simultaneous insertion and deletion of an arbitrary sequence of elements ordered according to key, in addition to find-minimum and single element delete operations [109, 111]. Techniques to parallelize advanced routing algorithms such as contraction hierarchies are only limited to the pre-processing step where the contraction of nodes can be done in parallel [112].

5.3.3 Limitations of Centralized Route Planning

Cloud based route planning models assume near unlimited processing and network availability. This makes current approaches poorly suited for deploying route planning services on private fog networks where resources are constrained and access to outside cloud resources can be intermittent, particularly in disaster scenarios.

Some of the prior work discussed earlier, for parallel route planning [109, 110, 111], assume that the graph network has static weights, which doesn't hold in real transportation network where traffic congestion changes with time. In a time-dependent network, edge expansion depends on arrival/departure time at each edge, hence processing is sequential. There are some approaches [112] which model the time-dependent nature of the network but the parallelization is only limited to the pre-processing phase and not during real-time query.

All of these approaches use a parallel shared memory model where an assumption is made that the shared memory allows a constant time direct communication between each pair of processors. This holds in a multiprocessing system and possibly in a data center, but this assumption is not realistic in a decentralized setting where communication between processes can add significant latency. While previous work of ours has focused on data integrity in distributed RSU networks [18], limited work has been done on routing in such networks.

5.4 System Model

In this section we outline the system architecture and data collection for decentralized route planning.

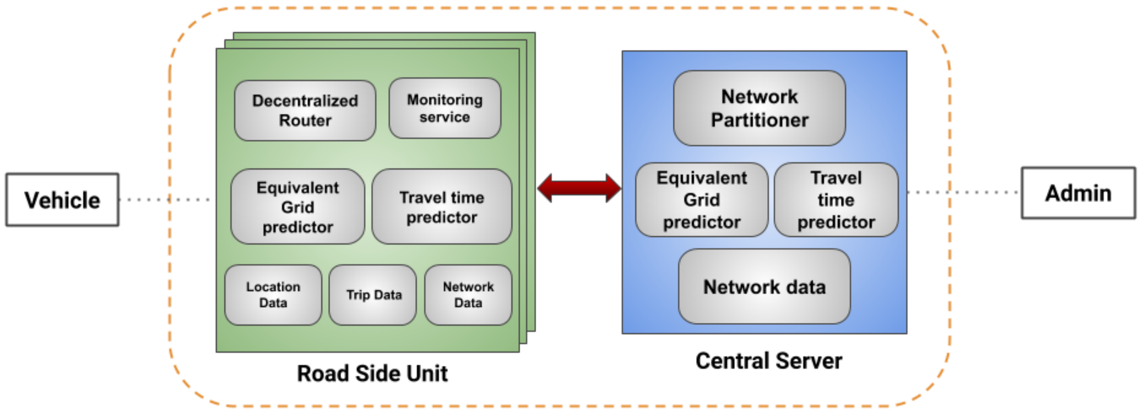


Figure 5.1: Decentralized architecture for route planning

5.4.1 Architecture

Figure 5.1 shows our decentralized architecture for route planning. The fundamental components are outlined as follows:

1. *Road Side Unit (RSU)*: low-powered compute nodes [95] located near roads and highways throughout the transportation network. These nodes are assumed to have computational resources similar to Raspberry Pis. Linked together, the RSUs form a private fog network.

2. *Central Server*: The central server is assumed to be a cluster of compute and storage nodes with horizontal on demand scaling. The primary role of the central server is as the central administrator for the RSU network meta-data and resources. It is assumed that access to the central server is intermittent and can fail in disaster scenarios.

3. *Vehicle*: Vehicles are assumed to be GPS equipped and network enabled. They provide two functions. First, they periodically send location data and travel speed to the RSU network. Second, vehicles can query the network for routes from their current location to a destination.

4. *Admin*: Maintains the global transportation network graph $G = (V, E)$ and helps divide the network into sub-graphs as outlined in Section 5.5.

5.4.2 Data collection

Our system consists of the following types of data:

1. *Location data*: This data is periodically collected from multiple vehicles and are stored as a set $\{(e_1, \tau_1, t_1), \cdots, (e_n, \tau_n, t_n)\}$, where e_i is the edge traversed, τ_i is the time interval of the day.

2. *Trip data*: This data consists of a set of route plans $\{R_1, \cdots, R_n\}$ where each route is a sequence of edges $R_i = [(e_1, \tau_1), (e_2, \tau_2), \cdots, (e_p, \tau_p)]$ from the location of the vehicle to its destination.

3. *Network data*: Each RSU maintains a list of subgraphs of the global routing graph $G = (V, E)$. Location data and trip data from vehicles travelling on this subgraph is maintained at that RSU.

5.5 System Deployment

In this section we outline the procedure for deploying the system using Algorithm 4 and Algorithm 5.

5.5.1 Central Server Setup

First, Admin sends the full graph $G = (V, E)$ and a geohash precision $prec$ to the Network Partitioner in Central Server. The geohash precision value determines the resolution or area of the desired grids. For this we use geohash encoding [113] to encode the geographical coordinates.

Algorithm 4 Partition Network

```
1: **Data:** $G = (V, E)$, $g = \{g_1, g_2, \cdots, g_n\}$, $prec$ =geohash precision value
2: **for all** $v \in V$ **do**
3:     $v.grid = gh.encode(v, prec)$; geohash encode
4: **end for**
5: **for all** $e_i \in E$ **do**
6:     $e_{u,v}$ = edge between vertices u and v;
7:     **if** $u.grid \neq v.grid$ **then**
8:         $w = intersection(e_{u,v}, u.grid, v.grid)$
9:         $w.grid = (u.grid, v.grid)$
10:        $G.add(w, e_{u,w}, e_{u,v})$
11:        $G.remove(e_{u,v})$
12:    **end if**
13: **end for**
```

The Network Partitioner receives the graph $G = (V, E)$ and $prec$ from the Admin and partitions the network into grids $\{g_1, g_2, \cdots, g_k\}$ using Algorithm 4. The algorithm proceeds by first annotating each vertex v with the grid they belong to using the geohash encoding function $gh.encode(v, prec)$, where $prec$ represents the precision of the encoding. Then for each edge $e_{u,v}$ it checks if both the vertices u and v are in the same grid. If they are in different grids a boundary vertex w is found at the intersection between the two grids through which the edge passes using the function $intersection(e_{u,v}, u.grid, v.grid)$. Vertex w splits edge $e_{u,v}$ into edges $e_{u,w}$ and $e_{w,v}$ and the graph G is updated accordingly.

5.5.2 Deployment to RSUs

First, each RSU requests the subgraphs for a set of grids $\{g_1, g_2, \cdots, g_k\}$ from the Central Server. The Central Server receives the set of grids $\{g_1, g_2, \cdots, g_k\}$ and maps the network for each grid using Algorithm 5 and ultimately returns the set of subgraphs $\{G^1, G^2, \cdots, G^k\}$ to the RSU.

Algorithm 5 Grid Network Mapping

```
1: Data: A graph $G = (V, E)$, $g = \{g_1, g_2, \cdots, g_k\}$
2: Result: $\{G^1, G^2, \cdots, G^k\}$
3: Initialize GridNetworkList
4: for all $g_i \in g$ do
5:     Initialize graph $G^i$
6:     for all $v \in V$ do
7:         if $v.grid == g_i$ then
8:             $G^i.add(v)$
9:         end if
10:    end for
11:    for all $e \in E$ do
12:        if $e.grid == g_i$ then
13:            $G^i.add(e)$
14:        end if
15:    end for
16:    GridNetworkList.append($G^i$)
17: end for
18: Return: GridNetworkList
```

5.6 Decentralized Routing

Our system needs to provide a shortest route from an origin location to destination. Any RSU can receive a route query between two points, which may be within one of the RSU's subgraphs or require communication with neighboring RSUs.

In this section we outline prediction models for estimating both travel times throughout the network as well as predicting which next RSU should be contacted to find the current shortest path for that query. We then provide the full decentralized routing algorithm.

5.6.1 Training With Federated Learning

Model training occurs on the RSUs. We achieve this by leveraging recent advances in federated learning [114] which enable the RSUs to collaboratively learn a shared prediction model without transferring raw data from the devices to the centralized cloud.

The goal of federated learning is to learn a model with parameters embodied in a real matrix $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$, from data stored across the RSUs. Here $\mathbf{W}$ is a 2D matrix representing the parameters of each layer in a fully-connected feed-forward network [115]. d_1 and d_2 represents the output and input dimensions respectively. The tasks proceed in rounds and each round alternates between local and global model updates:

1. *Distribute global model*: Admin randomly initializes the weights $\mathbf{W_0}$ of the prediction model and stores it in Central Server. In round $t \geq 0$, the Central Server distributes the current model $\mathbf{W}_t$ to a subset S_t of n_t RSUs.

2. *Local update*: Each RSU then independently updates the model based its local data. Let the updated local models be $\mathbf{W_t^1}, \mathbf{W_t^2}, \cdots, \mathbf{W_t^{n_t}}$, so the update of each RSU i can be written as $\mathbf{H_t^i} := \mathbf{W_t^i} - \mathbf{W_t}$, for $i \in S_t$. For this update we use stochastic gradient descent (SGD) [116]. Each RSU then sends the update back to the Central Server.

3. *Global update*: aggregation of local updates.

$$\mathbf{H_t} := \frac{1}{n_t} \sum_{i \in S_t} \mathbf{H_t^i}, \mathbf{W_{t+1}} := \mathbf{W_t} + \eta_t \mathbf{H_t}.$$

   Here η_t is learning rate. For simplicity we can choose $\eta_t = 1$.

5.6.2 Prediction Models

Our decentralized routing approach requires two prediction models described as follows.

5.6.2.1 Travel Time Predictor

A travel time predictor model estimates travel time on an edge e_i in time interval τ_i. Let edge e_i define a directed edge from v_i to v_{i+1}, then the travel time function is defined as $T(v_i, v_{i+1}, \tau_i)$ where τ_i refers to the departure time at vertex v_i.

Table 5.2: Input features for Travel Time and Equivalent Grid predictors.

Feature	**Dim**	**Description**
From location	42	Geohash encoded binary of start loc
To location	42	Geohash encoded binary indication of To coordinate of an edge
Week of year	52	One-hot encoded binary indication of Week of year used to sample travel time data
Day of week	7	One-hot encoded binary indication of Day of week used to sample travel time data
Hour of day	24	One-hot encoded binary indication of Hour of day used to sample travel time data
Minutes	60	One-hot encoded binary of Minutes of hour used to sample travel time data

Table 5.3: Target features for Travel Time and Equivalent Grid predictors.

Feature	**Dim**	**Description**
Travel time	1	One-hot encoded binary indication of the true travel time data collected from HERE API
Next Grid	28	Geohash encoded binary indication of a Grid

For training, we build an input feature set described in Table 5.2. The resulting feature space has 228 dimensions. We used one-hot encoding to map the travel time τ_i to one-hot encoded binary for Week of Year, Day of Week, Hour of Day and Minutes of Hour. The output of this model is travel time as shown in Table 5.3 which is a scalar value representing the travel time on an edge.

Geohash encoding is used to map v_i and v_{i+1} to the From location and To location respectively. An important aspect of the feature set is the geohash resolution. We used a resolution of 9.5m which matches the width of most major road segments in the United States. We found that reducing resolution to greater than 9.5m caused some vertices to belong to the same geohash while increasing resolution added complexity to the model and did not noticeably improve model performance. A resolution of 9.5m was represented by 42 bits.

5.6.2.2 Equivalent Grid Routing Predictor

The search procedure can extend to multiple RSUs. As more RSU are included in the search, it can incur huge delays from communication costs. Hence, the goal is to minimize the number of message exchanges required during the search. Therefore the equivalent grid routing model predicts the best neighboring grid through which the shortest path likely resides for a particular route. By iteratively finding the next grid the optimal route will pass through, the search space is reduced and communication is optimized. Hence we learn an Equivalent Grid Routing Predictor $\hat{E}$ such that $\hat{E}(s, d, \tau_s)$ gives the next best possible grid to travel to destination d and τ_s is the departure time from s. For this model, we use the same feature set as shown in Table 5.2. The output of this model is the next grid as described in Table 5.3 which represents the next grid in the route.

5.6.3 Decentralized Route Planning Algorithm

The goal of the decentralized route planning is to distribute the query among different RSUs. One of the problems we discussed earlier is that state of the art solutions for parallelizing the query fails in a time-dependent network. We mitigate this problem by using Travel Time Predictor $\hat{T}$. To minimize communication between RSUs during the search we use Equivalent Grid Routing Predictor $\hat{E}$.

Algorithm 6 handles decentralized routing queries from vehicles as well as from other RSUs as the query is propagated forward through the network. Algorithm 7 iteratively builds the route as the results propagates back to the RSU from which the query started.

We first list some utility functions that are used in the algorithm and then discuss the algorithm in detail. The utility functions are:

1. $gh.encode(v, prec)$: Uses geohash encoding to find the grid to which the vertex belongs to with geohash precision $prec$.

2. $GetRSU(g_i)$: Finds the RSU mapping for any grid i.

Algorithm 6 Handle Query

```
1: Data: A graph $G = (V, E), s \in V, d \in V$, $\tau_s$ = departure time from vertex $s$, $RSU_o$ = RSU from where the query origins, $id$: Unique identifier to identify this query.
2: $save(id, (s, d, \tau_s))$
3: $g_s = gh.encode(s)$
4: $g_d = gh.encode(d)$
5: if $g_s \neq g_d$ then
6:     $g_{\text{next}}$ = $\hat{E}(s, d, \tau_s)$
7:     $RSU_{\text{next}}$ = GetRSU$(g_{\text{next}})$
8:     $\{v_1, v_2, \cdots, v_b\}$ = $g_s.intersect(g_d)$
9:     for all $v \in \{v_1, v_2, \cdots, v_b\}$ do
10:        $\hat{t}^s_v$ = $\hat{T}(s, v, \tau_s)$
11:        $msg($"query"$, RSU_{\text{next}}, \{id, v, d, \tau_s + \hat{t}^s_v, RSU_o\})$
12:    end for
13:    for all $v \in \{v_1, v_2, \cdots, v_b\}$ do
14:        $R_p$ = $SP(G, s, v, \tau_s)$
15:        $msg($"partial path"$, RSU_o, \{id, s, v, \tau_s, R^s_v\})$
16:    end for
17: else
18:    $R_p$ = $SP(G, s, d, \tau_s)$
19:    $msg($"partial path"$, RSU_o, \{id, s, d, \tau_s, R^s_d\})$
20: end if
```

3. $GetRoute(G, s, d, \tau_s)$: This function uses network G, Dijkstra [102] and Travel time Predictor $\hat{T}$ to find the route from source s at departure time τ_s to destination d with minimum travel time.

4. $msg(type, RSU_i, val)$: An async call for communicating the type of message ($type$) and actual message (val) to a RSU i. There are two types of messages:

   (a) *query*: Upon receiving this message, Algorithm 6 is executed. The message, represented by (val) is passed as an argument to this function.

   (b) *partial path*: Upon receiving this message, Algorithm 7 is executed, with the message represented by (val) as input. If the final route plan is returned as a response, it is communicated to the client which made the routing request.

Algorithm 7 Handle Partial Path

```
1: Data: A graph $G = (V, E), u \in V, v \in V$, $\tau_u$ = departure time from vertex $u$, $R$ =
   Route from $u$ to $v$, starting at $\tau_u$, $id$: Unique identifier to identify this query.
2: Return: Final Route plan $(id, R_{final})$ or NULL
3: $(s, d, \tau_s)$ = $get(id)$
4: if $v == d$ then
5:     $R_{final}$ = $GetRoute(G_{id}, s, d, \tau_s)$
6:     Return: $(id, R_{final})$
7: else
8:     $G_{id}$ = GetGraph(id)
9:     if $G_{id} ==$ NULL then
10:        Initialize Graph $G_{id}$
11:    end if
12:    for all $(e_i, \tau_i) \in R$ do
13:        $G_{id}.add(e_i)$
14:        $G_{id}[e_i] = (\tau_i, \tau_{i+1} - \tau_i)$
15:    end for
16:    SaveGraph(id, $G_{id}$)
17:    Return: NULL
18: end if
```

5.6.4 Decentralized Route Planning Example

To demonstrate execution, Figure 5.2 shows an example where a network is partitioned into 4 RSUs. Figure 5.3 shows a sequence diagram for this example.

In this example, RSU_1 receives a route query (id, s, d, τ_s) from a client and calls Algorithm 6 to find the route. Since the source s and destination d do not belong to the same grid, Equivalent Grid Predictor is called to find the next best possible grid to reach destination d. Then the algorithm finds the nodes at the intersection of the current grid and the next best possible grid. After getting all the boundary vertices, for each boundary vertex, Travel Time Predictor estimates the time it will take to reach that vertex. An asynchronous message $(\text{“}query\text{”}, id, v_{12}, d, \hat{\tau}_{v_{12}}, RSU_1)$ is sent to RSU_2.

After sending the message, RSU_1 proceeds to find the actual route from s to boundary vertex v_{12}. After getting the route, denoted by R^s_{12}, a message $(\text{“}partialpath\text{”}, id, s, v_{12}, \tau_s, R^s_{12})$ is prepared and sent to the requesting RSU informing the starting RSU that the operation

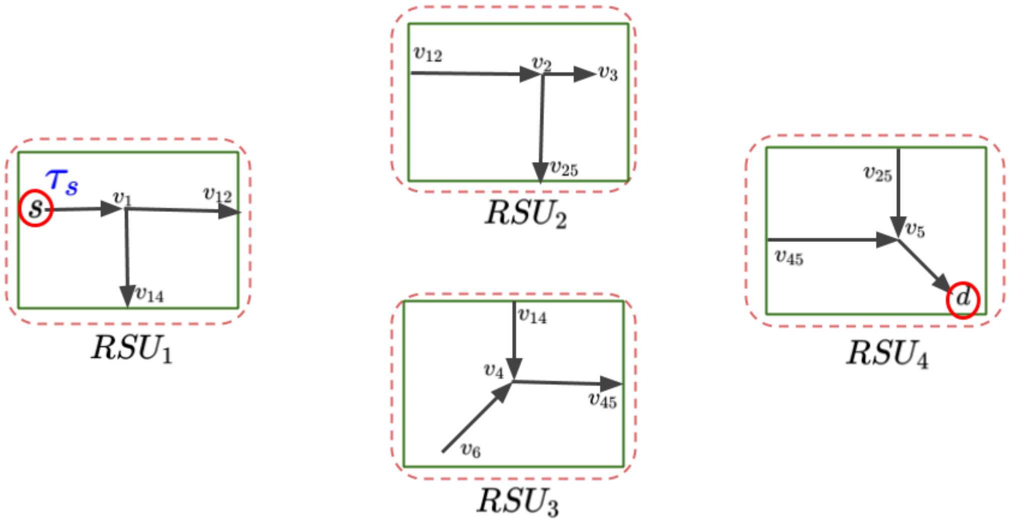


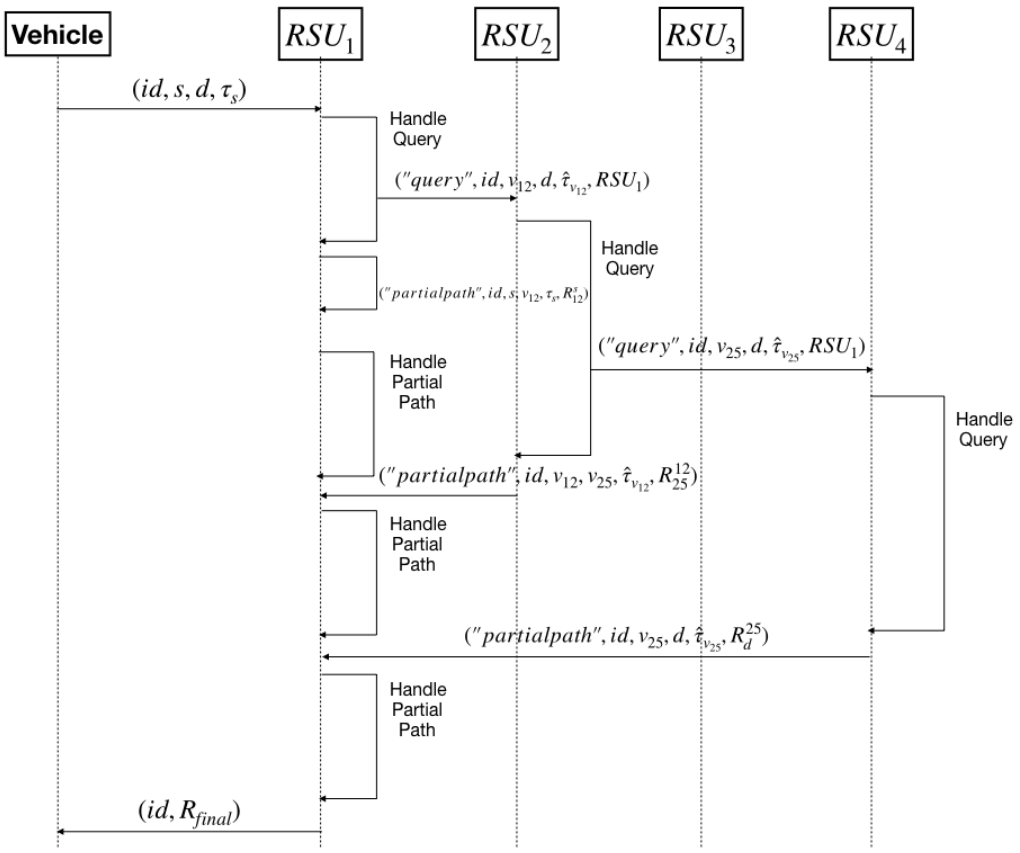
Figure 5.2: Decentralized Route Planning example

resulted in a partial path, i.e. the destination is not located in that RSU.

This message is meant to be sent to the RSU to which the client sent the request. Since it's RSU_1, which is itself, a function call is made to handle this message where the function arguments are the same as the message. This function implements Algorithm 7 which handles the partial routes or paths. We call it partial route because this is still not the final route that needs to be given to the client.

The goal of Algorithm 7 is to create a new graph with the id of the request or if it already exists, add all partial routes to the graph and finally, do a simple shortest path routing on it.

At this step, RSU_1 waits for partial routes from the other RSUs for this request, identified by its id, and executes Algorithm 7 if it receives a message with the partial route in it. This process continues until RSU_1 gets a partial route which has the destination in it. When a partial route contains the destination vertex Algorithm 7 executes $GetRoute(G_{id}, s, d, \tau_s)$ to obtain the final route.



Vehicle
RSU_1
RSU_2
RSU_3
RSU_4
(id, s, d, τ_s)
Handle Query
$("query", id, v_{12}, d, \hat{\tau}_{v_{12}}, RSU_1)$
Handle Query
$("partialpath", id, s, v_{12}, \tau_s, R^s_{12})$
$("query", id, v_{25}, d, \hat{\tau}_{v_{25}}, RSU_1)$
Handle Partial Path
Handle Query
$("partialpath", id, v_{12}, v_{25}, \hat{\tau}_{v_{12}}, R^{12}_{25})$
Handle Partial Path
$("partialpath", id, v_{25}, d, \hat{\tau}_{v_{25}}, R^{25}_{d})$
Handle Partial Path
(id, R_{final})

Figure 5.3: Sequence Diagram of Decentralized Route Planning example

5.6.5 Decentralized Route Planning Properties

As discussed in Section 5.3, A^* is a classic algorithm for informed search, which relies on a heuristic function to guide the search procedure. In this context, our approach for route planning is an informed search procedure where Equivalent Grid Routing Predictor acts as a heuristic function.

It is well established that if a graph G is finite and edge weights are non-negative, then A^* is guaranteed to terminate and is complete, i.e. it will always find a route from source to destination if one exists. Our approach is similar to A^*, but it cannot give guarantees on its termination and hence may not be complete. This is due to our use of learned models in the Equivalent Grid Routing Predictor. Therefore as with all machine learning models, the accuracy of our approach is tied to the accuracy of the learned models.

5.7 Experiments and Results

In this section, we evaluate our decentralized architecture for route planning with a case study from a mid-size U.S. city.

5.7.1 Experiment Setup

1. *RSUs*: Cluster of 5 RSUs simulated by Docker [117] containers. RSUs are static as their location does not change with time.

2. *Global Network Graph*: We use OpenStreetMap to generate the underlying routing graph $G = (V, E)$. For the region in this study there are a total of $233, 123$ nodes and $474, 213$ edges.

3. *Graph Partitioning*: We used a geohash precision value of 28 bits, which results in a grid area of $1.44km^2$. A total of 1034 grids are generated as a result of the partition. These grids are assigned to the 5 RSUs as shown in Figure 5.4.

4. *Location data*: To simulate vehicle locations in the region, we use historical traffic data collected at an interval of 1 minute from the HERE API [38] for the region. Traffic data from January 1 to January 31, 2018 was used for training and data from Feb 1 to Feb 7, 2018 was used for testing.

5. *Trip data*: To simulate routing queries from vehicles, we synthetically generate 1,000,000 source and destination pairs which are chosen randomly within the region. For the source-destination pairs, departure times were chosen uniformly from 9am-5pm.

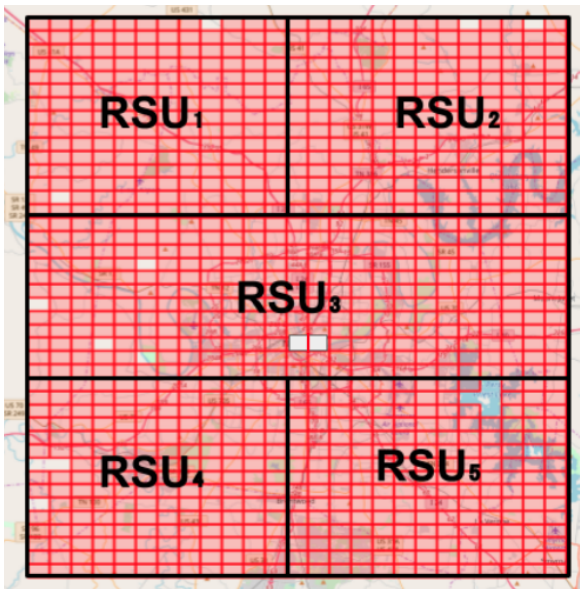


Figure 5.4: Partition of city into grids of area $1.44km^2$ and placement of grids in RSUs. Grids are represented by the 1034 square grids while the RSU regions are represented by bolded black lines.

5.7.2 Evaluation of Prediction Models

5.7.2.1 Travel Time Predictor - Training Evaluation

For the travel time predictor we used a deep feed-forward neural network (DNN) [118] for regression to estimate travel time of an edge. We used SGD [116] as the optimizer and a hidden layer configuration of $[200, 190, 170, 150, 100, 50, 20, 10]$. Early stopping criteria was implemented to avoid over-fitting. Fig. 5.5 shows the change in validation Mean Absolute Error (MAE) during the training. We found that the Federated learning model took longer to train than the Central learning model.

Table 5.4 evaluates the resource consumption of the model during training for Federated learning and Central learning. As the Central learning model was trained on a single large server, we divided the resource consumption of the Central model by the number of fog nodes for a direct comparison with the Federated training model. Results show that Federated learning used less CPU per node as well as 3.4 - 3.7 times less memory per node

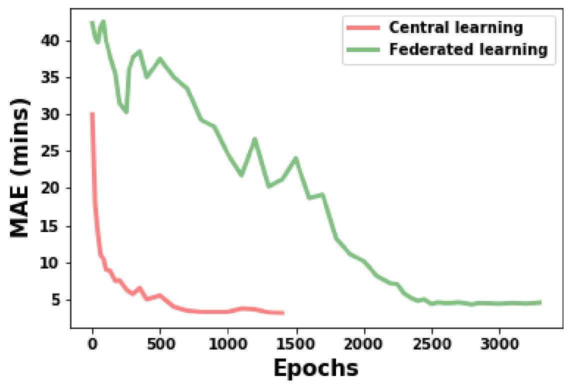


Figure 5.5: MAE vs Epoch curve during training of Travel time predictor

Table 5.4: Resource consumption for Travel time predictor.

	CPU (%/RSU)	**Memory (MB/RSU)**	**# Messages**
Central Learning	78% (median) 97% (max)	191 (median) 307 (max)	N/A
Federated Learning	67% (median) 84% (max)	51 (median) 88 (max)	6255

than Central learning. Lastly, federated learning sent 6255 messages while training.

5.7.2.2 Equivalent Grid Routing Predictor - Training Evaluation

We use a deep feed-forward neural network (DNN) [118] for a binary classification that gives the next best possible grid for a given pair of source, destination along with the time interval. For binary classification, we used a sigmoid function [119] for the output layer and an Adam optimizer was used as the optimizer for the model. The configuration for hidden layers was $[250, 200, 170, 100, 50, 20, 10]$. Fig. 5.6 shows the loss vs epoch curve during the training phase for this predictor. Federated learning took more time to train than Central learning. The loss for a model trained with Central learning was 0.32 which is less than the model trained from Federated learning which was 0.37.

Table 5.5 evaluates the resource consumption of the Federated learning and Central learning models during training. We find that Federated learning used less CPU than Central learning per node on average. Additionally, Federated learning used 3.3 - 3.6 times

Table 5.5: Resource consumption for Equivalent Grid Routing predictor.

	CPU (%/RSU)	Memory (MB/RSU)	# Messages
Central Learning	81% (median) 93% (max)	217 (median) 336 (max)	N/A
Federated Learning	74% (median) 97% (max)	64 (median) 91 (max)	9543

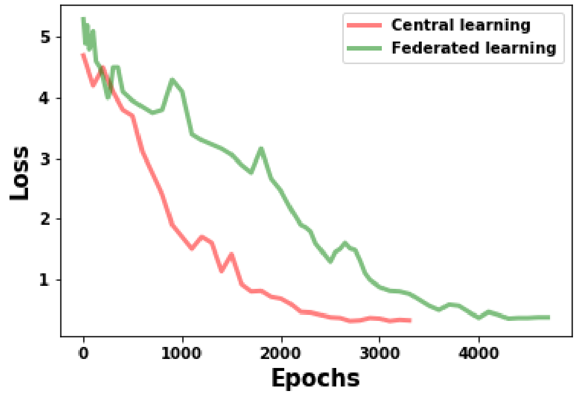


Figure 5.6: Loss vs Epoch curve during training of Equivalent Grid Routing predictor

less memory per node than Central learning. Federated learning sent 9543 messages while training.

5.7.3 Evaluation of Decentralized Route Planner

During testing we monitored CPU and memory consumption per node compared to current state of the art solutions. The resource requirements of each method is presented in Table 5.6. To evaluate performance of our approach we measured the query response time per request as well as the accuracy of the returned routes compared to the optimal route.

5.7.3.1 Resource Consumption

We find that our approach uses more CPU than Single server Dijkstra or Parallel Dijkstra because in our approach shortest paths are calculated between boundary nodes in parallel when the request is received. In terms of memory, we find that our approach uses less memory per node than Single server Dijkstra and slightly more memory than Parallel

Table 5.6: Evaluation of routing algorithms.

Algorithm	**CPU per RSU (% used)**	**Memory per RSU (MB)**	**Query time per trip request (s)**
Single server Dijkstra	23% (median) 31% (max)	5.78	0.97 (median)
Parallel Dijkstra	27% (median) 36% (max)	0.76 (median) 1.14 (max)	9.2 (median) 19.13 (max)
Contraction Hierarchies	18% (median) 23% (max)	13.36	0.016 (median)
Parallel Contraction Hierarchies	13% (median) 21% (max)	3.31 (median) 5.79 (max)	5.78 (median) 10.21 (max)
Our approach	52% (median) 67% (max)	0.94 (median) 1.31 (max)	2.43 (median) 5.81 (max)

Dijkstra. We find that Contraction Hierarchies and Parallel Contraction Hierarchies use the most memory due to caching shortcut edges.

5.7.3.2 Query Response Time

Single server Dijkstra and Contraction Hierarchies result in the lowest query response times as expected since this simulation was done on one machine. It is expected that in production cloud environments the latency between vehicles and the cloud would factor into this result. Parallel algorithms such as parallel Dijkstra and parallel Contraction hierarchies have higher query times than our approach since their search proceeds sequentially.

5.7.3.3 Accuracy

We found that our approach returned no route for 0.8% of the queries and a sub-optimal route (i.e. longer than the shortest route) for 7.6% of the queries. Therefore we found that 91.6% of routes from our model were the shortest route. As our approach is reliant on trained models, it is expected that our model improves as more data is available for training.

5.8 Conclusion and Future work

In this work we provided a decentralized route planning approach and deployment model for fog networks. Our approach uses prediction models to limit communication between fog nodes and thus improve latency and memory demands over current parallel approaches to route planning. The core of our architecture relies on data-driven models that estimates travel times and guides the search procedure during query time.

Additionally, to limit communication during training we used recent advances in federated learning to train the models. Through this approach all training occurs on the RSUs and only model weights are shared between nodes. Therefore costly transfer of raw traffic data is avoided, reducing bandwidth stress during training. This work was evaluated through a simulation using real traffic data for a mid-sized U.S. city.

Potential extensions of this work include investigating ways to improve travel time and grid prediction models to mitigate the impact of errors on user trips, as well as expanding the architecture to handle node failures. Additionally, our approach can be extended to allow multiple modes of transportation.

Chapter 6

A Decentralized Approach for Real Time Anomaly Detection in Transportation Networks

6.1 Overview

While decentralized deployment models have great potential for real-time transportation applications, a key challenge is ensuring the integrity of the sensor readings on which these algorithms rely. A particular concern is orchestrated data-integrity attacks in which an adversary attempts to compromise a subset of sensors with the goal of maximizing the effect of the attack on the global transportation system. Transportation systems are particularly vulnerable to data-integrity attacks since well orchestrated in a specific area can have cascading effects throughout the transportation network. Decentralized deployments can improve reliability by removing the single point of failure associated with centralized models. However, decentralized deployments also increase the risk by potentially exposing computation methods and functionality on numerous nodes throughout the edge network.

In Chapter 5 we introduced the concept of a roadside-unit (RSU) network and proposed a decentralized routing algorithm within this framework. In this Chapter we propose a multi-tiered anomaly detection framework which utilizes spare processing capabilities of the distributed RSU network in combination with the cloud for fast, real-time detection. Additionally, we focus on implementation of our framework in smart-city transportation systems by providing a constrained clustering algorithm for RSU placement throughout the network. Extensive experimental validation using traffic data from Nashville, TN demonstrates that the proposed methods significantly reduce computation requirements while maintaining similar performance to current state of the art anomaly detection methods.

The work comprising this chapter has been published in the 2019 Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP) [18]:

- Michael Wilbur, Abhishek Dubey, Bruno Leão, and Shameek Bhattacharjee. A decentralized approach for real time anomaly detection in transportation networks. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 274–282. IEEE, 2019

6.2 Introduction

Emerging trends and challenges: Internet of Things (IoT), edge/fog computing, and the cloud are fueling rapid development in smart connected cities. Given the increasing rate of urbanization, the advancement of these technologies is a critical component of mitigating demand on already constrained transportation resources. Recent research on smart transportation systems has focused on optimal route planning for congestion reduction, which has shown huge potential impact on maximizing existing transportation resources [120]. The costs of optimizing route planning are relatively low compared to large scale infrastructure upgrades, making this an attractive option for city planners and transportation experts.

Approaches to optimal route planning are typically data-driven [121], [122], [123]. The scale and real-time nature of these systems require shared computing architectures to handle the high velocity and volume of data originating from small sensors placed throughout the network. One solution is edge/fog computing. In this case, services are moved to road-side units (RSUs), which are low-powered edge devices [95] situated between the sensor level and the cloud. Each RSU hosts various computation services for a collection of sensors, and communicates with the cloud. Implementing a network of RSUs moves computation to the edge of the network, creating a decentralized data processing system.

Data-driven approaches are susceptible to data integrity attacks. The dynamic nature of real-time routing systems means that the effects of such an attack have immediate impact and substantial cascading consequences [124]. Additionally, the distributed and shared nature of the underlying architecture provides multiple points of entry, making data integrity attacks even more likely. Given the potential human and economic impacts of such an attack, the trustworthiness of data in smart transportation networks is of critical importance. While there is substantial research regarding anomaly detection in transportation networks [125], these approaches are often computationally costly and do not adapt well to the real-time nature of distributed smart transportation data networks. Despite the critical importance of data integrity in such systems, research in this area remains underdeveloped.

Current state of the art statistical detection methods typically rely on measures of central tendency such as median and mean or their variants. While this approach works for *deductive* attacks and *additive* attacks, in which sensor readings are decreased or increased respectively, it fails for *camouflage* attacks in which sensor readings are increased at some sensors and decreased at other sensors. Camouflage attacks are of particular importance when working with data-integrity attacks in transportation networks, as such an attack would aim to divert traffic and resources to specific regions or roads and thus, maximize the effects of an attack.

Therefore we aim to improve data integrity in decentralized smart transportation systems by proposing a novel real-time anomaly detection algorithm for deductive and camouflage data integrity attacks. Our approach maintains similar accuracy to traditional methods, while addressing two critical components of scaling anomaly detection to decentralized systems. First, it reduces the computational costs associated with computationally expensive traditional anomaly detection by avoiding continuous computation on all sensors in real time. This is accomplished by continuously monitoring anomalies at the RSU level using a statistical means approach for aggregate anomaly detection and reserving the more computationally costly sensor level detection for cases in which anomalies are found at the RSU level. Second, our approach is designed so that the anomaly detection process itself is distributed, mirroring the natural architecture of modern decentralized smart networks and allowing seamless integration with such systems.

We also provide a constrained hierarchical clustering algorithm for RSU placement in an existing transportation system fitted with traffic sensors. As shown later, this approach improves zone level detection while also maximizing spare processing capacity at the RSU level.

Contributions: This paper presents a decentralized anomaly detection approach and architecture for distributed smart transportation systems. Our focus is on *orchestrated data-integrity attacks*, in which an organized attacker falsifies data in a systematic attack process. This paper's main contributions are as follows:

- Anomaly detection is framed as a decentralized computational system allowing for real-time processing and scalability.
- An algorithm is provided for RSU placement which maximizes processing capacity of the RSU network and optimizes central tendency anomaly detection methods.
- We present a novel real time anomaly detection algorithm which reduces the computational costs associated with traditional anomaly detection methods while maintaining similar accuracy.

Outline: We start by defining the problem and model assumptions in Section 6.3. Related work is covered in 6.4. The System Model is covered in Section 6.5, while the Sensing Architecture is covered in Section 6.6. RSU placement is outlined in Section 6.7 and the anomaly detection framework is proposed in Section 6.8. Finally, simulations and results are provided in Section 6.9.

6.3 Problem Statement

Our primary concern is orchestrated data integrity attacks in smart, decentralized transportation systems.

6.3.1 Problem Overview

The goals of our system are the following:

- *Real-time* identification of orchestrated data-integrity attacks.
- *Decentralized implementation.* The system should integrate easily with modern smart-city infrastructure and be optimized for common hardware limitations in such systems.
- *Deductive and Camouflage Attacks* - extend traditional statistical means anomaly detection to camouflage attacks in which mean and median are unchanged.
- Reduce computation requirements compared to traditional anomaly detection methods.

6.3.2 Assumptions

To achieve the above goals, we make the following assumptions.

1) Sensor Model: We assume that the city has sensors capable of transmitting traffic speed data wirelessly to an RSU. In our investigation, speed data is collected by the sensor and sent to its associated RSU.

2) Road-side Units: RSUs are low-powered fog nodes [95] placed throughout the transportation network which are capable of collecting and transmitting data from a collection of sensors to a centrally located cloud-based routing system.

3) Centralized Cloud: A centralized cloud network is available to provide additional processing capabilities for sensor level anomaly detection.

4) Attack Model: The attacker is capable of compromising a subset of sensors or RSUs by manipulating their outputs. These attacks occur at the sensor level. As the focus of this paper is orchestrated data-integrity attacks, sensor or RSU faults from physical failures is outside the scope of this paper.

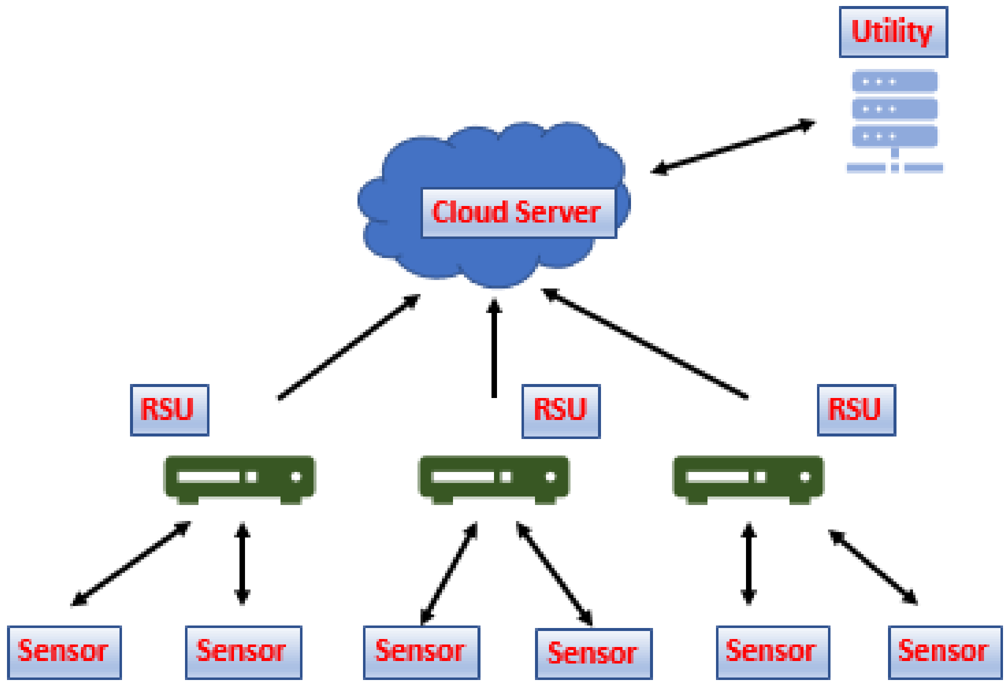


Figure 6.1: System Architecture

6.3.3 Our Approach

The architecture for our system is detailed in Figure 6.1, and consists of three fundamental components: the Sensor level, RSU level and Cloud. Our anomaly detection framework thus consists of two components, zone level detection and sensor level detection. Zone level detection is run at the RSU level, while the more computationally expensive sensor level detection runs at the cloud. Framing detection in this way maximizes existing hardware resources while reducing computation requirements compared to traditional detection approaches.

A major focus of this paper is on the integration and implementation of the anomaly detection framework in decentralized smart transportation networks. As RSUs are fog nodes, a fundamental question is how to deploy these devices throughout the network. We identify three critical considerations to answering this question. First, RSUs should be located as close to possible to the sensors streaming to it in order to minimize network latency.

Second, as RSUs are low-powered devices, the maximum number of sensors mapped to a single RSU is to be constrained. Lastly, we look to group sensors together as to maximize the efficiency of our anomaly detection approach.

6.4 Related Work

Smart city research has advanced rapidly in recent years. A large focus of this research has focused on implementation of sensor systems for transportation, communication and infrastructure monitoring [126], [127], [128], [129], [130]. In general, anomaly detection is focused on finding deviations in single (point) or sequence (collective) values from normal expected behavior. Traditional anomaly detection is based on classification, statistical, state based, clustering or information theory [125]. Classification methods are usually based on Support Vector Machines (SVM), Bayesian Models, Gaussian Processes or Neural Networks [131]. These methods require large scale, detailed and accurate models of system behavior. Additionally, supervised classification models require careful consideration regarding user data privacy. This is of particular concern when dealing with transportation systems and the specific movement of users over time. State based methods use Kalman Filtering [132] to estimate normal behavior. These methods require making realistic assumptions on data distributions, a challenging task. Additionally hardware considerations must be accounted for [133].

Our primary concerns regarding the anomaly detection problem are accuracy, computational requirements and easy distribution over a decentralized network. For this reason, our zone level detection uses a statistical approach. Related statistical approaches include auto-regressive, exponential or cumulative weighted moving averages (ARMA, EWMA, CWMA) and Cumulative Sum Control Chart (CUSUM) of data as metrics under normal operating behavior. These approaches are light weight, and do not necessarily require anomalous data. Our work presents a hybrid approach which uses a statistical mean ratio that has proven effective in detecting data-integrity attacks in power grid networks [134] and Gaussian Processes for sensor level detection [135].

Hierarchical anomaly detection has shown to be useful in monitoring large scale distributed web architectures [136]. The advantage of hierarchical anomaly detection is that the detection computation can be balanced between low-powered edge devices and central computation clusters. One approach is to keep a central model of expected data behavior to compare with current data [137]. In this case, when anomalous patterns are found in the system the second, more computationally expensive, procedure of identifying anomalous nodes within the subsystem is performed [138], [139].

RSU placement has been studied in relation to maximizing connectivity for smart cities using intersection-priority [140], minimizing event reporting times along highways [141] and maximizing information flow in urban areas [142]. Approaching RSU placement through the context of anomaly detection efficiency is a new topic.

6.5 System Model

6.5.1 Data Overview

To simulate the framework provided in Figure 6.1, historical data is collected from the HERE API [38] for use as real-time sensor data for Nashville, TN. Two months of data was extracted from February 12, 2018 to April 12, 2018 for use as historical training and reference data. Additionally, two weeks of data from April 16, 2018 to April 27, 2018 was extracted for testing and simulation. Only weekdays (Monday-Friday) are considered.

The HERE data is composed of time stamped speed recordings, identified by its Traffic Message Channel identification (TMC ID), [143]. Each TMC represents a segment of road in which the speed was recorded. In our framework each TMC ID acts as a sensor which provides speeds for optimal routing. There are 9,979 TMCs, and therefore sensors, in our data set.

6.5.2 Data Integrity Attack Overview

Traditional anomaly detection in transportation systems focus on detecting faulty sensors [131] [135], whether from hardware failure or software issues in the collection of data. In this model, anomaly detection is run in the cloud for each sensor in isolation. Data-integrity attacks on the other hand are orchestrated from a collection of sensors simultaneously to maximize the effect of the attack on the global transportation system.

The shared nature of computing resources in smart connected cities provide multiple entry points for attackers, making attacks likely events. Additionally, the dynamic real-time nature of such systems means that well designed attacks will have substantial cascading effects throughout the system. In this sense, focused localized attacks on a collection of sensors will propagate throughout the network quickly.

While traditional anomaly detection operate at the sensor level, the identification of orchestrated attacks requires aggregate detection across groups of sensors. In this context, organized data integrity attacks spanning multiple sensors within a selected region can have cascading effects throughout the transportation system.

Our focus is primarily on two types of data integrity attacks. In the first type of attack a selected percentage of sensors have their speed values reduced and is referred to as a deductive attack. These attacks aim to diverge traffic away from attacked sensors by convincing the routing system that certain roads have more congestion than in reality.

The second type of attack is camouflage attacks, in which an organized attacker balances additive and deductive attacks to evade detection and exert certain behaviors on the system. Camouflage attacks are of particular concern in transportation routing systems as an attacker can deviate network behavior at a fine granular level to maximize impact of the attack. One scenario would be an attack aimed at gathering vehicles along a specific road segment or crowding drivers in a highly dense area. Identifying attacks of this nature is of critical importance to first responders and the defense industry, yet as this approach would leave mean and median unchanged, camouflage attacks evade traditional central tendency approaches.

6.5.3 Simulated Deductive and Camouflage Attacks

To simulate deductive attacks and camouflage attacks, we use the historical standard deviation of a sensor's speed, represented by σ_s, as a basis for altering speed value d at attacked sensor s. Therefore d_s^a represents the speed value at sensor s when attacked while d_s is the actual speed recorded at sensor s when not attacked. The severity of the attack is governed by δ. Equation 6.1 represents the process for altering speeds from a deductive attack at a single sensor while Equation 6.2 represents the process for altering speeds from an additive attack at a single sensor.

$$d_s^a = d_s - \delta * \sigma_s \tag{6.1}$$

$$d_s^a = d_s + \delta * \sigma_s \tag{6.2}$$

Each RSU r is responsible for a subset of sensors $S^r \subset S$ where S^r is the subset of sensors at RSU r and S represents all the sensors in the network. Therefore, if we look to simulate a deductive attack at RSU r during time window k affecting p percentage of sensors, then p percentage of sensors are randomly selected for the attack.

Conversely, to simulate a camouflage attack during time window k, then p percentage of sensors at that RSU are selected for attack and each of the attacked sensors is randomly assigned to have its speed readings altered by a deductive attack from Equation 6.1 or an additive attack from Equation 6.2.

6.6 Sensing Architecture

In this section, we present a decentralized system architecture for efficient sensing over a large city in real time. The system is comprised of three central components as shown in Figure 6.1.

6.6.1 Road Sensor System - Sensor Level

Traffic information is maintained by sensors distributed throughout the network edges. The sensor units are responsible for capturing current speed values at each road. Together, the sensor network provides real-time monitoring of the transportation network. In the context of our data, each TMC ID [143] represents a sensor streaming real-time vehicle speed information.

6.6.2 Roadside Unit System - RSU Level

Roadside Units (RSUs) are small, low powered devices with wireless capabilities [95]. RSUs have two main responsibilities. First, the RSU level is responsible for communicating data from the sensors to the central cloud. Second, spare processing capacity is used for zone level anomaly detection described in Section 6.8.1. A depiction of the interaction between the sensor level and RSU level is shown in Figure 6.2.

6.6.3 Utility System and Cloud Service

The cloud service is a broad term incorporating the utility system, routing services and long term data storage. For this work we are primarily concerned with the utility system, which is a collection of high powered computation nodes residing in the cloud. The role of the utility system is providing processing for sensor level detection.

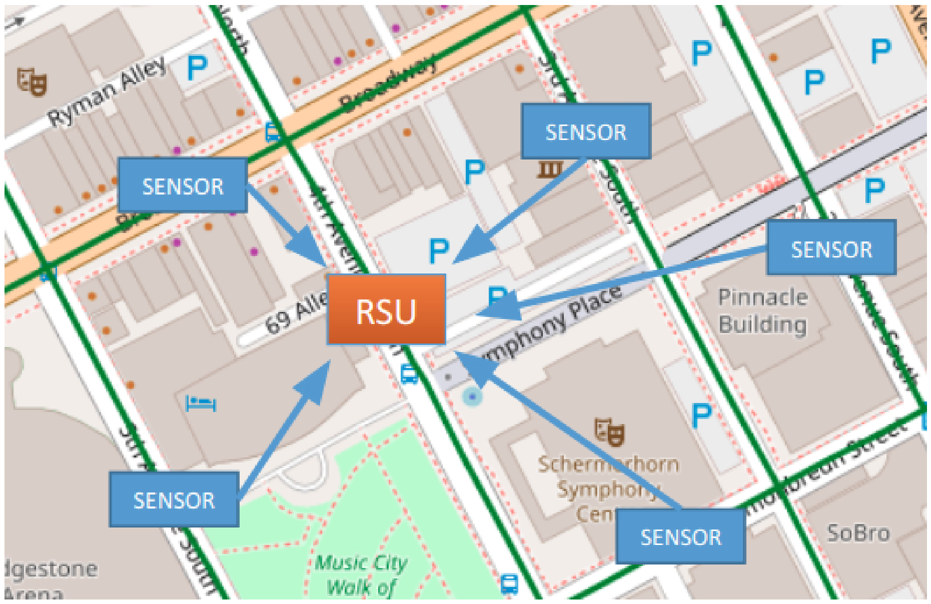


Figure 6.2: Data Collection Framework - RSU-Sensor Interaction

6.7 RSU Deployment - Clustering Procedure

The way in which RSU devices are deployed affects resource utilization and network efficiency. Therefore in this section we provide a constrained hierarchical clustering algorithm for RSU deployment.

As each RSU is responsible for a subset of sensors, ultimately the goal of the algorithm is to match each sensor s_i with an RSU. Through a mapping process, each RSU r will be responsible for the data collected from a subset of sensors $S^r \subset S$ where S^r is a collection of sensors mapped to RSU r.

Since zone level detection outlined in Section 6.8.1 is optimized when sensors with similar traffic patterns are grouped together (see Section 6.9.3), feature sets are generated for each cluster using training speed data from the HERE API. For cluster c consisting of sensors S^c, the speed data from these sensors is broken into 30 minute time windows from 7:00AM to 9:00PM, resulting in 28 features total. By taking the mean speed at each time window k, the feature set for cluster c is represented by $F^c = \{f^c(k_1), ..., f^c(k_{28})\}$. Clusters are grouped together by similarity. We therefore use euclidean distance to measure

Algorithm 8 RSU Clustering

```
1: **Input:** $m, \eta$
2: **Initialize:** $C \leftarrow S$
3: **while** $len(C) > m$ **do**
4:     $l_{min} = \infty$
5:     **for** $i \in \{0, \ldots, len(C)\}$ **do**
6:         $c_j \leftarrow nearest(c_i, C)$
7:         **if** (len($S^{c_i}$)+len($S^{c_j}$)) $\leq \eta$ **then**
8:             $l_{(i,j)} \leftarrow euclideanDist(F^{c_i}, F^{c_j})$
9:             **if** $l_{(i,j)} < l_{min}$ **then**
10:                $l_{min} \leftarrow l_{(i,j)}$, $c_v \leftarrow c_i$, $c_w \leftarrow c_j$
11:            **end if**
12:        **end if**
13:    **end for**
14:    $c_{new} \leftarrow merge(c_v, c_w)$
15:    add $c_{new}$ to $C$
16:    remove $c_v$ and $c_w$ from $C$
17: **end while**
```

the similarity between two clusters.

The clustering procedure is detailed in Algorithm 8. Algorithm 8 relies on three helper functions:

- $nearest(c_i, C)$: returns the cluster whose centroid is geo-spatially closest to the centroid of cluster c_i, according to haversine distance.

- $euclideanDist(F^{c_i}, F^{c_j})$: returns the euclidean distance between the feature sets of clusters c_i and c_j.

- $merge(c_v, c_w)$: returns a new cluster. The feature set of the new cluster is recalculated using the combined set of sensors in the new cluster.

Line one specifies the input parameters where m is the target number of clusters and η is the maximum number of sensors in a cluster. In the initialization step, C represents the set of all clusters. C is initially set such that each cluster consists of a single sensor.

The clustering procedure starts at line (3) and continues until the number of clusters equals m. As we loop through each cluster c_i, the geographically nearest cluster c_j is

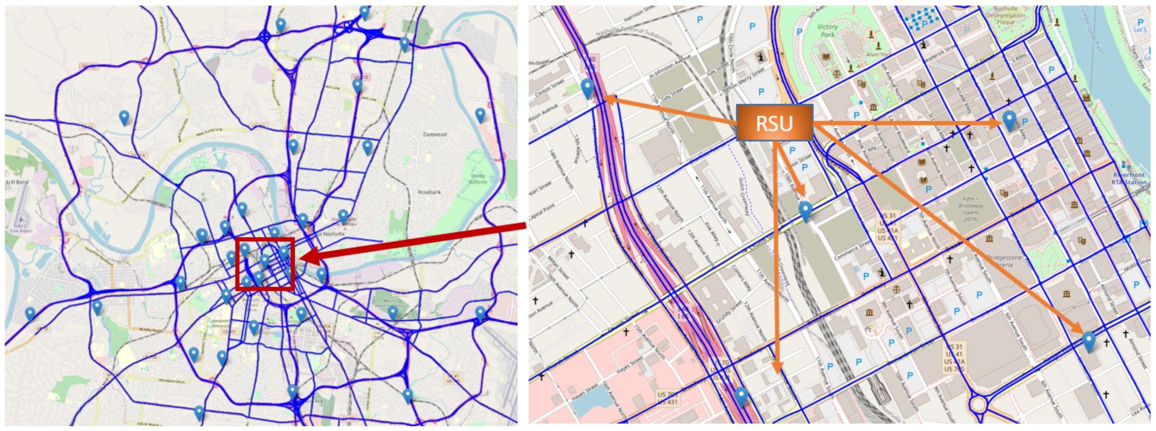


Figure 6.3: Cluster RSU - full layout and downtown Nashville. The clustering approach results in multiple RSUs in the highly travelled downtown area, allowing for resources to be deployed according to demands of the sensor network.

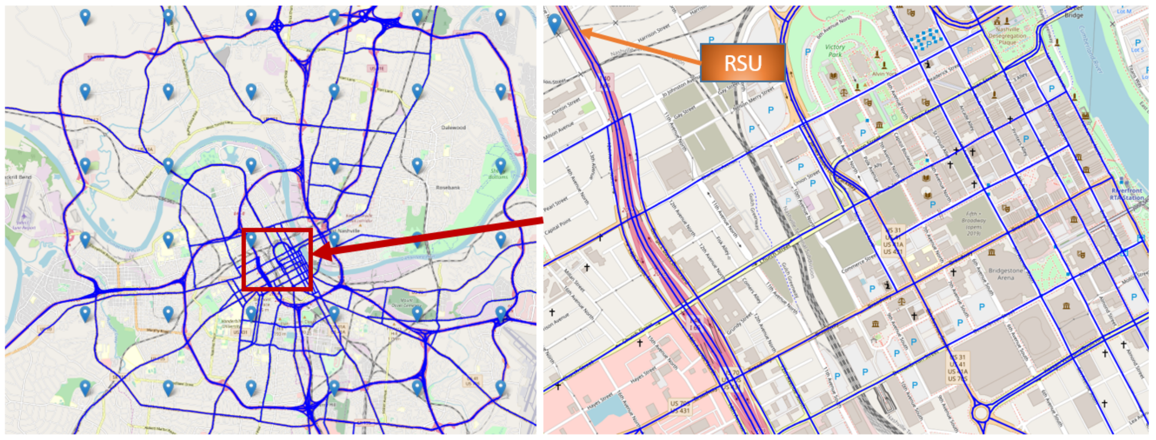


Figure 6.4: Grid RSU - full layout and downtown Nashville. The grid layout results in only one RSU in the highly travelled downtown area.

identified. If the η constraint is satisfied and the euclidean distance between F_i and F_j is less than l_{min} then we reassign l_{min} to $l_{(i,j)}$ and update c_v and c_w accordingly. After each cluster is iterated through, (c_v, c_w) are merged into a single cluster c_{new} which is added to C and c_v, c_w are subsequently removed.

The visual representation of the cluster RSU network is provided in Figure 6.3. For comparison, a grid RSU layout where the geo-spatial boundary of the sensor network is divided into a square grid with an RSU located at the center of each grid was generated as shown in Figure 6.4.

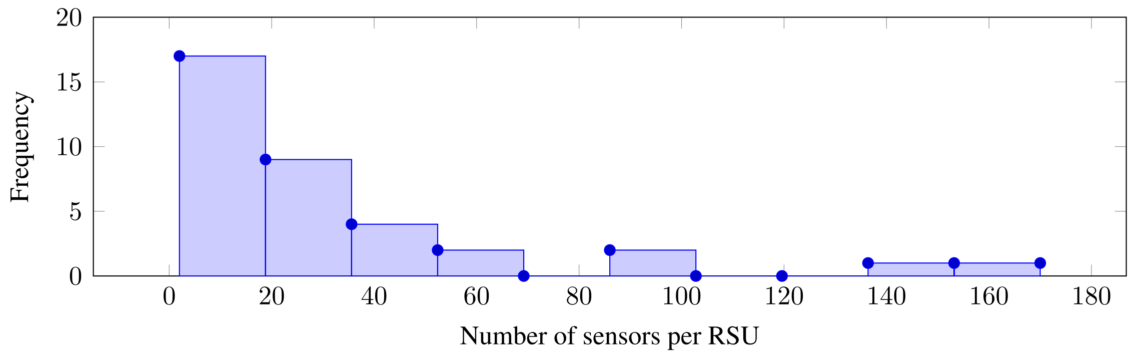


Figure 6.5: Grid RSU layout histogram - sensors per RSU distribution. This layout places high stress on a small number of RSUs while under-utilizing the full processing capabilities of the network.

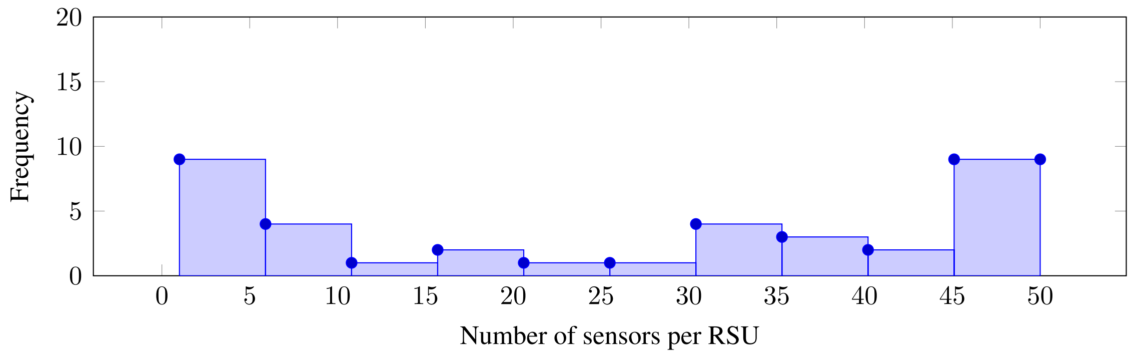


Figure 6.6: Cluster RSU layout histogram - sensors per RSU distribution. Number of sensors in an RSU does not exceed $\eta = 50$. Constraining the number of sensors at an RSU places an upper bound on processing demand and ensures processing requirements do not exceed the capacity of RSU hardware.

Comparing the two layouts, the cluster RSU layout does a better job concentrating RSUs in areas where there are a high number of sensors. Additionally, by only merging spatially adjacent clusters, the subset of sensors at each RSU maintains a connected subgraph of road edges.

The effect of η is illustrated by the sensor distributions in Figures 6.5 and 6.6. By limiting the maximum number of sensors in each RSU, the processing and networking demands placed on each RSU can be controlled. Conversely, the grid layout includes two RSUs that taken together, are responsible for approximately 30% of all the sensors in the network. This imbalance in sensor distribution creates high stress on a few RSUs while under-utilizing the resources at the remaining RSUs.

6.8 Anomaly Detection

This section describes our novel two-tiered anomaly detection approach in which zone level detection is continuously run at the RSU network and sensor level detection is used to identify sensors compromised by data integrity attacks. Sensor level detection is only performed on a set of sensors when an attack is first identified at the zone level.

6.8.1 Zone Level Detection

The zone level detection provides a mechanism for identifying data integrity attacks at the RSU level. Zone level detection is processed at the RSUs.

Each sensor continuously transmits time-stamped speed data to its RSU. Since each RSU r is responsible for a subset of sensors, the RSU collects the data from its set of sensors in the last time window k. At each time window k, the harmonic mean $HM^r(k)$ and arithmetic mean $AM^r(k)$ are calculated per Equations 6.3 and 6.4 respectively. The statistical metric used for anomaly detection is the ratio of $HM^r(k)$ to $AM^r(k)$, as shown in Equation 6.5.

$$HM^r(k) = \frac{S}{\sum_{s=1}^{S} \frac{1}{d_s}} \tag{6.3}$$

$$AM^r(k) = \frac{\sum_{s=1}^{S} d_s}{S} \tag{6.4}$$

$$Q^r(k) = \frac{HM^r(k)}{AM^r(k)} \tag{6.5}$$

While traditional central tendency detection methods based on arithmetic mean or median are effective in detecting additive or deductive attacks, camouflage attacks go unde-

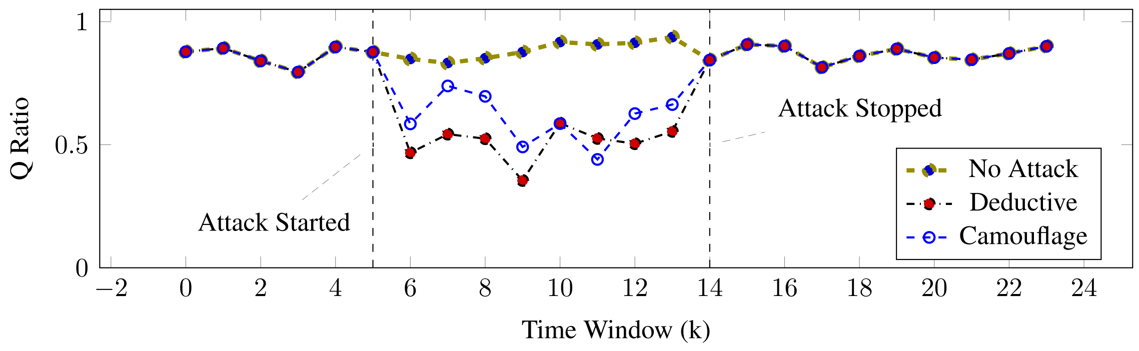


Figure 6.7: $Q^r(k)$ under deductive and camouflage attacks at a single RSU. Time Window (k) set to 30 minute intervals, δ=2.5 and p=35%.

tected since arithmetic mean and median remain the same. As shown in Figure 6.7, where speed readings for 35% of sensors at a selected RSU were subjected to a δ attack of 2.5, Q^r responds to camouflage attacks as well as deductive attacks.

For detection, $Q^r(k)$ is compared to the historical average and standard deviation of $Q^r(k)$ at time window k as shown in Equations 6.6 and 6.7. ϵ^r is a threshold that is unique to each RSU. An investigation for determining ϵ^r is provided in Section 6.9.1

$$Q^r(k) < Q^r_{ave}(k) - \epsilon^r * Q^r_{std}(k) \tag{6.6}$$

$$Q^r(k) > Q^r_{ave}(k) + \epsilon^r * Q^r_{std}(k) \tag{6.7}$$

6.8.2 Sensor Level Anomaly Detection

For many smart transportation applications, such as optimal routing systems, we must identify which sensors are attacked to mitigate the effects of data integrity attacks in real time. Therefore sensor level detection is required.

For sensor level detection we use Gaussian Processes to get the expected speed and standard deviation at a given sensor using the 15 sensors closest to that sensor. This approach assumes a high correlation between speed readings at nearby sensors [135]. We use

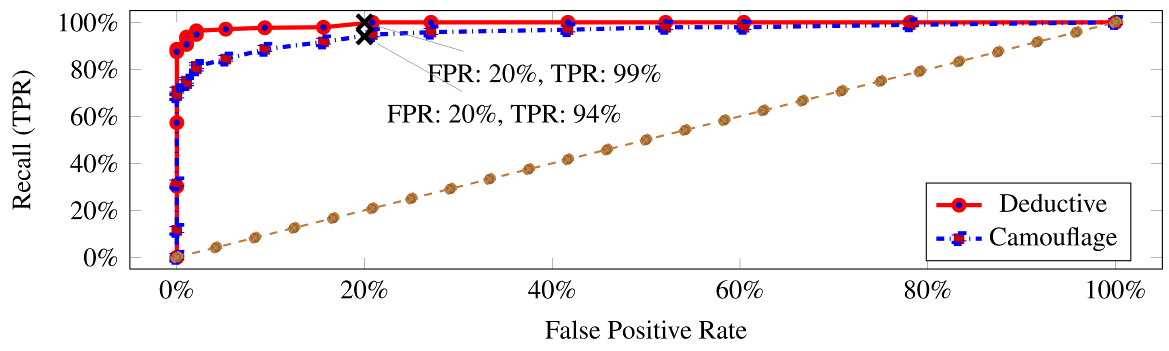


Figure 6.8: Epsilon true positive rate (TPR) vs false positive rate (FPR) for a single RSU, deductive and camouflage attacks. These curves were generated for each RSU and the value of ϵ^r was selected such that FPR was 20%. Attack parameters: $\delta = 2.5, p = 35\%$

CUSUM for detection, however as sensor level detection is not continuous in our two-tiered anomaly detection approach the process is restricted to two windows.

As a kernel function, the commonly used RBF (squared exponential) kernel is used. A study of detection accuracy and computation time between continuous sensor level detection compared to two-tiered anomaly detection is provided in Section 6.9.

6.9 Simulations and Results

6.9.1 Parameter Tuning For Zone Level Detection

The effectiveness of zone level detection is highly dependent on ϵ. For each RSU we simulated 100 deductive and 100 camouflage attacks with δ held constant at 2.5 in which 35% of the sensors at an RSU are attacked. For each attack, a random time window in the testing set between 7:00AM and 9:00PM was attacked and zone detection was performed. To obtain false positive and true negative results, zone detection was also run at the same time window without the presence of a data integrity attack. The process was repeated for ϵ^r values ranging from 0 to 10 and recall (TPR) and false positive rate (FPR) were recorded at each simulation step.

The cost of false positives at the zone level in two-tiered anomaly detection is only in terms of the increased computation time required to run sensor level detection. Since the cost of false positives is low, the value of ϵ^r is set such that the number of false positives is approximately 20% at each RSU. Therefore while the exact value of ϵ^r is unique at each RSU, we can expect the resulting false positive rate to be roughly 20%.

Figure 6.8 provides a graphical representation of this process for an example RSU, where recall was 99% and 94% for deductive and camouflage attacks respectively when FPR was 20%. As discussed in the following section, recall at the zone level remains relatively consistent across the RSU network following this procedure.

6.9.2 Zone Level Detection - Investigation of Attack Methods

Here we investigate the bounds for which our zone level detection is viable. There are two primary considerations in quantifying the severity of a data integrity attack. First, the severity of the attack on each affected sensor is represented by δ (see Equations 6.1 and 6.2). Second, the percentage of sensors affected by the attack (p) represents the breadth of an attack at each RSU.

Two simulations are configured. First, p was held constant at 35% and δ was varied from 0 to 3.5. For each δ value, 100 deductive attacks and 100 camouflage attacks were again simulated at each RSU in the network. However for this simulation, true and false positives and negatives at each RSU were aggregated together at each value of δ, resulting in a single recall value for the entire network at every δ. The results of this simulation are provided in Figure 6.9, and show that for both deductive and camouflage simulations the recall is greater than 90% when δ is greater than 2.25.

For the second simulation the same procedure was followed except this time δ was held constant at 2.5, while p was varied from 0% to 60%. As shown in Figure 6.10, zone level detection retains 90% accuracy for attacks affecting as low as 25% of the sensors at an RSU.

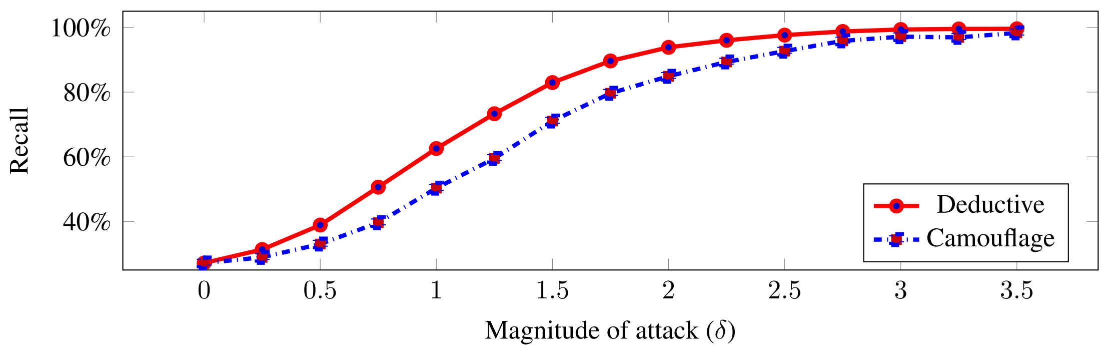


Figure 6.9: Recall (TPR) aggregated across all RSUs with five or more sensors vs magnitude of attack (δ). ϵ^r used for detection and the percentage of sensors attacked p is held constant at 35%. Recall for the network is greater than 90% when δ is greater than 2.25 for both deductive and camouflage attacks.

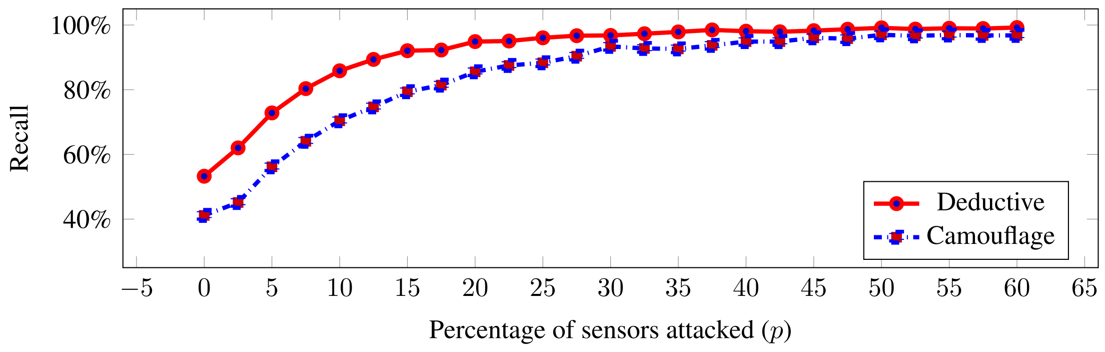


Figure 6.10: Recall (TPR) aggregated across all RSUs with five or more sensors vs percentage of sensors attacked at each RSU (p). Full network simulation - each unique ϵ^r used for detection and δ is held constant at 2. Recall for the network is greater than 90% for attacks affecting 25% or more sensors for both deductive and camouflage attacks.

6.9.3 Zone Level Detection Comparison - Grid vs Cluster RSU Deployment

In Section 6.7 we discussed the advantages of constrained hierarchical clustering for RSU placement in terms of maximizing hardware resources. Here we investigate the benefits of this approach in terms of anomaly detection.

The same zone level attack simulation as detailed in Section 6.9.1 was applied to the cluster RSU and grid RSU networks respectively, with $\delta = 2.5$ and $p = 35\%$. For detection, ϵ^r, generated from Section 6.9.1, is used. To find ϵ^r for each RSU in the grid network, the process in Section 6.9.1 was repeated for the grid network. Recall statistics for each RSU

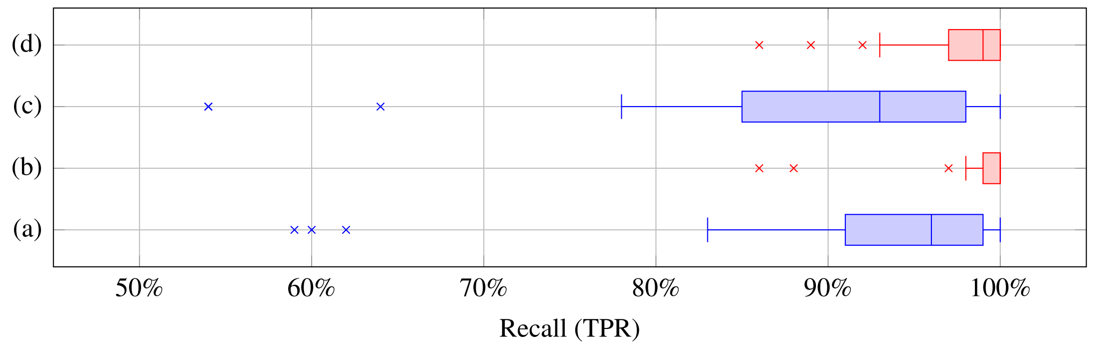


Figure 6.11: Zone level recall of (a) grid RSU layout - deductive attack simulation, (b) Cluster RSU layout - deductive attack simulation, (c) grid RSU layout - camouflage attack simulation, (d) cluster RSU layout - camouflage attack simulation. Each data point represents recall at a single RSU in the network. Only RSUs with more than 5 sensors considered. Attack parameters: $\delta = 2.5, p = 35\%$

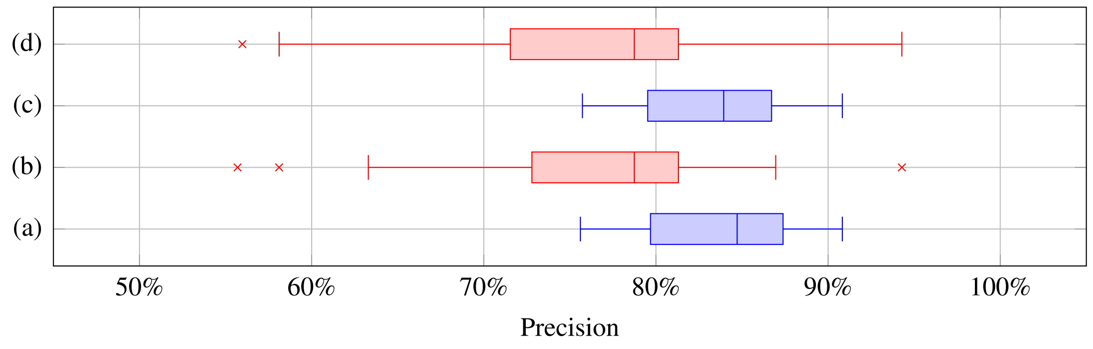


Figure 6.12: Zone level precision of (a) grid RSU layout - deductive attack simulation, (b) Cluster RSU layout - deductive attack simulation, (c) grid RSU layout - camouflage attack simulation, (d) cluster RSU layout - camouflage attack simulation. Each data point represents precision at a single RSU in the network. Only RSUs with more than 5 sensors considered. Attack parameters: $\delta = 2.5, p = 35\%$

is provided in Figure 6.11 while precision is shown in Figure 6.12.

Both networks are capable of running zone level detection, as average recall was over 90% for both RSU configurations. This implies that our zone level detection algorithm is an adequate solution regardless of RSU layout. However, recall is higher for the cluster RSU configuration showing that clustering groups of sensors by traffic pattern similarity has a positive effect on zone level anomaly detection.

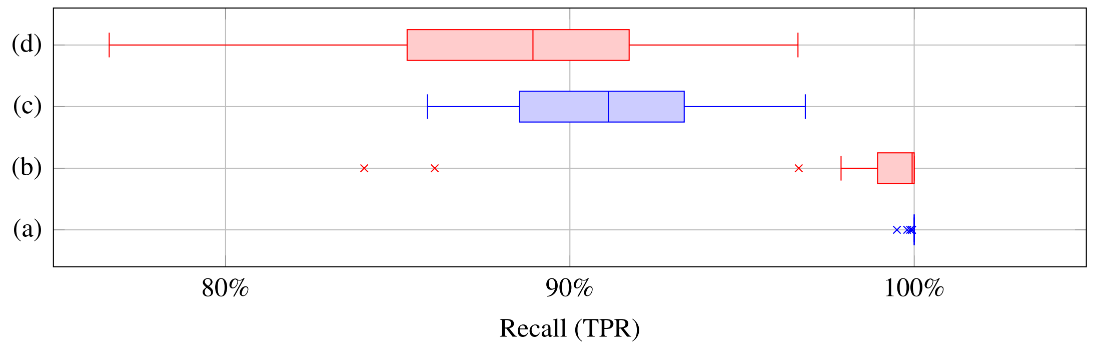


Figure 6.13: Sensor level recall of (a) sensor only (GP) - deductive attack simulation, (b) two-tiered detection - deductive attack simulation, (c) sensor only (GP) - camouflage attack simulation, (d) two-tiered detection - camouflage attack simulation. Each data point represents aggregate recall of sensor level detection at a single RSU in the network. Only RSUs with more than 5 sensors considered. Attack parameters: $\delta = 2.5, p = 35\%$

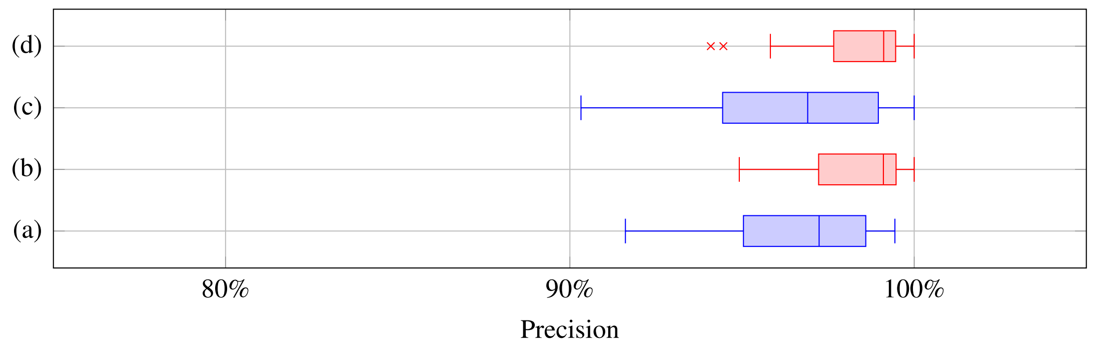


Figure 6.14: Sensor level precision of (a) sensor only (GP) - deductive attack simulation, (b) two-tiered detection - deductive attack simulation, (c) sensor only (GP) - camouflage attack simulation, (d) two-tiered detection - camouflage attack simulation. Each data point represents aggregate precision of sensor level detection at a single RSU in the network. Only RSUs with more than 5 sensors considered. Attack parameters: $\delta = 2.5, p = 35\%$

6.9.4 Two-Tiered Anomaly Detection vs Sensor Only Detection

In Section 6.8 two-tiered anomaly detection was outlined. We now move on from zone level detection and investigate two-tiered anomaly detection compared to continuous sensor only detection with Gaussian Processes. The simulation procedure remains the same as outlined in Section 6.9.3, however now we find true positives, true negatives, false positives and false negatives at the *sensor level*.

Recall and precision are provided in Figures 6.13 and 6.14 respectively. Note that

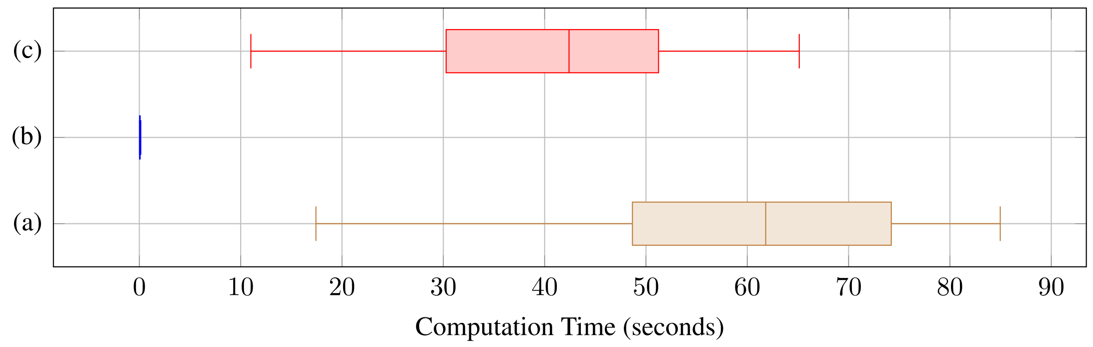


Figure 6.15: **Computation time** (seconds) of (a) sensor level detection (GPs), (b) zone level detection, (c) two-tiered detection

while true and false positives and negatives were calculated at the sensor level, recall and precision as shown in Figures 6.13 and 6.14 are aggregated at the zone level. This procedure allows us to visualize the interaction between zone level detection in the previous section and two-tiered detection provided here, as well as directly compare computation time per RSU as shown in Figure 3.5.

We find that precision and recall are similar between two-tiered and sensor only detection. The major difference between these approaches is in computation time, where two-tiered detection is 35% less than sensor only detection and the computation time of zone level detection was negligible in relation to the other two approaches.

An important observation is that our simulation procedure effectively attacked 50% of the time, a much higher percentage than can be expected in an actual deployment scenario. As two-tiered detection only requires sensor level detection when an attack is detected at the zone level and the computation time from zone level detection is negligible, it can be assumed that a 35% reduction in computation time between two-tiered and sensor only detection is a conservative estimate.

6.10 Conclusion and Future Work

In this paper we presented a novel two-tiered anomaly detection framework that maintains similar accuracy to current state of the art systems with a significant reduction in processing requirements. Additionally we covered the integration of our anomaly detection framework in decentralized smart transportation systems and provided a constrained hierarchical clustering algorithm for RSU deployment.

Our current work focuses on deductive and camouflage attacks. We would like to extend this to a variety of potential attacks. Therefore, future work will include extending this work to additive attacks as well as strategic attacker events in which the attacker has a comprehensive understanding of transportation system behavior. Additive attacks have worked in other aggregate anomaly detection cases [134]. These attacks cause an increase in the Q value, which is capped at one. By taking the inverse of our metric in relation to a jam factor, which is also available as one of the data streams from the HERE API, we can extend this work to such attacks.

Additionally, we would like to investigate the cascading effects of data-integrity attacks on routing systems. We also plan to use additional information, such as weather or planned events, to predict anomalies ahead of time.

Chapter 7

SmartTransit-AI: A Software Framework for Mobility-on-Demand Services

7.1 Overview

The focus on this work has been on how to design real-time, adaptive transit solutions with a focus on MoD and fixed-line operations. In the preceding chapters we discussed the importance, and our contributions, related to data management, integrity, AI-driven modelling and algorithms in this context. We showed how these contributions can improve and extend existing transit by providing more demand-responsive and efficient transportation services. To evaluate our contributions, we emphasized the potential of these applications by testing them compared to state-of-the-art approaches on real-world datasets from our partner CARTA (Chattanooga). However, to truly impact operations transit agencies need ways to train their own models and access these algorithms through software. As it currently stands, the operations software used by transit agencies is provided by various private companies. These offerings are closed in that the agencies themselves have limited access to underlying algorithms and don't have an easy way to apply agency-specific constraints. These one-size-fit-all solutions severely limit the ability of transit agencies to customize their services to their unique populations as well as implement state-of-the-art algorithms from the academic and open-source communities.

Therefore, in this chapter we present the SmartTransit-AI application framework for real-time MoD operations management. SmartTransit-AI fits within the broader category of Mobility-as-a-Service (MaaS) software - however with two important distinctions. First, like existing MaaS, SmartTransit-AI provides a software ecosystem where various private and public mobility options can be managed under one system. It allows agencies to plug in private providers as well as manage their own fleets and schedules in one interface.

However, SmartTransit-AI is more than a wrapper joining existing mobility options. It also allows direct access to modular optimization components including offline VRP and real-time DVRP algorithms. These modular components provide standardized input and output interfaces such that new algorithms can be incorporated into the framework provided they implement the corresponding offline VRP or online DVRP interfaces. Second, we make limited assumptions regarding operational constraints. We provide an easy to use configuration framework that gives transit agencies full control over system constraints and objectives. This includes constraints related to vehicle configurations and capacities, pickup and dropoff time windows, waiting time and detour time. Additionally, agencies can specify custom objective functions related to common metrics such as vehicle miles travelled (VMT), vehicle detour miles (VDM) or service rate.

First, we will discuss the SmartTransit-AI software ecosystem which was published in the Proceedings of the 14th IEEE International Conference on Cyber-Physical Systems (ICCPS) [144]. This work was also awarded top honors in the demonstration track for SmartComp 2023 [145].

- Michael Wilbur, Maxime Coursey, Pravesh Koirala, Zakariyya Al-Quran, Philip Pugliese, and Abhishek Dubey. Mobility-on-demand transportation: A system for microtransit and paratransit operations. In *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)*, ICCPS ’23, pages 260–261, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700361. doi: 10.1145/3576841.3589625. URL https://doi.org/10.1145/3576841.3589625

- Michael Wilbur, Sophie Pavia, Maxime Coursey, Pravesh Koirala, Zakariyya Al-Quran, Philip Pugliese, and Abhishek Dubey. Microtransit optimizer for mobility-on-demand. In *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2023

Second, we will discuss our implementation and deployment for CARTA which was used to run a pilot for their paratransit services in August 2023. Currently, we are working with CARTA to adapt SmartTransit-AI for all of their paratransit services going forward as well as to pilot microtransit operations in early 2024. Personally Identifiable Information (PII) has been removed where noted. Names in this section were generated with the ”names” python package [146].

7.2 Introduction

Large-scale adoption of smart phones and sensing technologies has given rise to new user-centric transportation modes including rideshare and shared-mobility services. We collectively refer to these new transportation modes as mobility-on-demand (MoD). Led by companies such as Uber, Lyft and Via, MoD provides users with reliable point-to-point travel options through smart phone applications. As MoD continues to transform urban mobility, cities are looking for ways to utilize new technologies to improve existing public services and adapt to this new environment.

One domain in which public transit agencies are looking to better utilize is the case of microtransit and paratransit services. These services fit under the umbrella of *ridepooling*, which refers to MoD that utilizes high-capacity vehicles to service trip requests. From the city's perspective, microtransit services are available to all residents and can be thought of as a low-cost extension of their public transit system. They can be used for direct point-to-point travel as well as in hybrid transit systems where the vehicles shuttle passengers to and from fixed-line transit [147]. Similarly, paratransit is a ridepooling service run by a transit agency that provides curb-to-curb service for passengers that are unable to use fixed-route transit (e.g. passengers with disabilities).

Many transit agencies operate both microtransit and paratransit services. However, the objectives and constraints for implementing these services varies greatly between agencies. This means that most agencies have to either design their own software or manually augment their workflows to adapt existing off-the-shelf software. This ad-hoc process makes it hard for researchers to implement new ridepooling algorithms and approaches in real-world settings.

This paper describes our MoD transportation software for microtransit and paratransit operations. The target audience is transit agencies managing these services and was initially designed for an upcoming pilot to test our ridepooling algorithms with our partner agency the Chattanooga Area Regional Transportation Authority (CARTA) in Chattanooga, TN. An overview of the software design and components is provided in Figure 7.1. The software includes three interfaces - an operations manager web application for dispatchers, a vehicle operator (or driver) mobile application and the user mobile application for residents to book requests. Additionally, it includes two modular optimization components - 1) an offline VRP solver for ahead-of-time scheduling and 2) an online DVRP solver for same-day trip requests. The offline VRP solver is used to batch all requests ahead of time and assign vehicles to service the routes all at once. The online DVRP solver is used for same-day operations where a trip request must be assigned to a route as it arrives. The optimization components can be replaced with new solvers over time or augmented to handle various constraints specific to each transit agency by implementing programming interfaces we have designed for the offline and online tasks.

7.3 Software Overview

As shown in Figure 7.2, we provide an operations manager which is a web application that allows the transit agency to manage clients, take bookings, update schedules and monitor real-time operations. Through the operations manager the agency is provided with what they need to handle both microtransit and paratransit services and is general enough to be used for any centrally managed ridepooling service. The operations manager consists of four sub-modules. There is a bookings module that allows transit agencies to take bookings over the phone or manage existing reservations booked through the user mobile application.

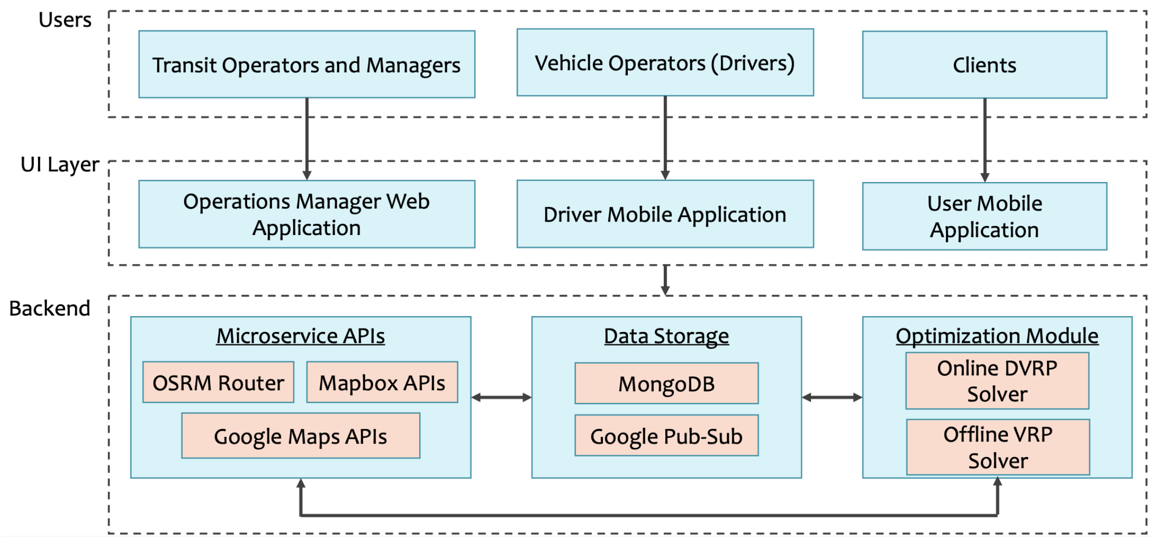
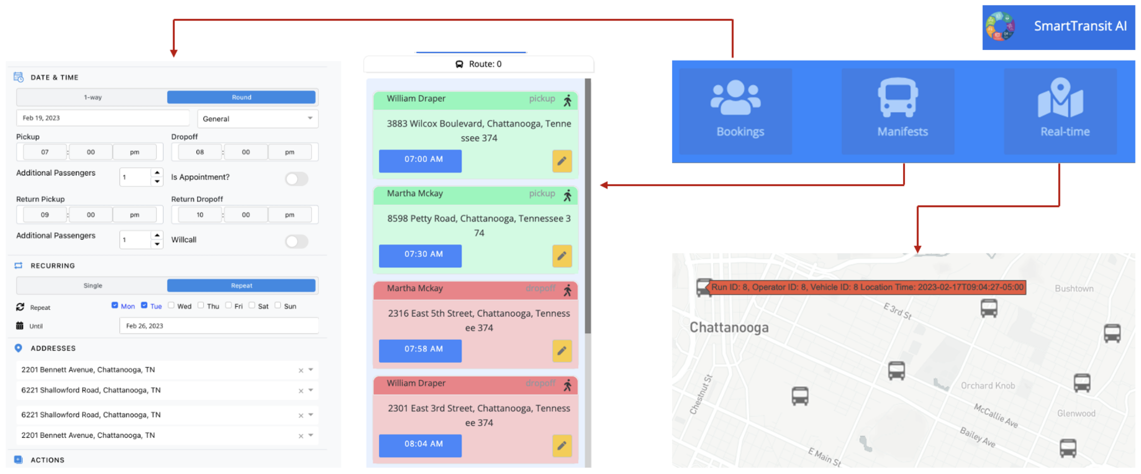


Figure 7.1: SmartTransit-AI software design. It includes three frontend interfaces connected to a common backend.

The manifests module is for managing vehicle schedules and trip-to-vehicle assignments. Through the real-time component dispatchers can monitor and track active vehicles for more informed decision making. The real-time component is connected to the backend in a way that shows operators the real-time location of their fleet as well as the upcoming locations for each vehicle in the fleet. Lastly, the reporting module allows operators to export various components of their operations to Excel for offline or manual changes. They also have access to optimization components that can automate or recommend trip-to-vehicle assignment as discussed in Section 7.4.

We also provide two mobile applications that can run on tablet or phone. As shown in Figure 7.3, the driver application allows vehicle operators (or drivers) to manage their routes for the day. It allows a driver to login and get their route for the day which is a schedule of users to pickup and dropoff. The driver application interacts directly with our backend to get up-to-date routes and communicate with the dispatchers as drivers service their schedules. GPS locations are published every second to our backend so the operations managers, dispatchers and real-time algorithms have access to vehicle locations and status in real-time. Lastly, we also provide a mobile application for users to schedule trips through



DATE & TIME

1-way | [✓ Round]

Feb 19, 2023 | General

Pickup 07 : 00 pm | Dropoff 08 : 00 pm

Additional Passengers 1 | Is Appointment?

Return Pickup 09 : 00 pm | Return Dropoff 10 : 00 pm

Additional Passengers 1 | Willcall

RECURRING

Single | [✓ Repeat]

Repeat [✓ Mon] [✓ Tue] Wed Thu Fri Sat Sun

Until Feb 26, 2023

ADDRESSES

2201 Bennett Avenue, Chattanooga, TN

6221 Shallowford Road, Chattanooga, TN

6221 Shallowford Road, Chattanooga, TN

2201 Bennett Avenue, Chattanooga, TN

ACTIONS

Route: 0

William Draper pickup
3883 Wilcox Boulevard, Chattanooga, Tennessee 374
07:00 AM

Martha Mckay pickup
8598 Petty Road, Chattanooga, Tennessee 374
07:30 AM

Martha Mckay dropoff
2316 East 5th Street, Chattanooga, Tennessee 374
07:58 AM

William Draper dropoff
2301 East 3rd Street, Chattanooga, Tennessee 374
08:04 AM

Run ID: 8, Operator ID: 8, Vehicle ID: 8 Location Time: 2023-02-17T09:04:27-05:00

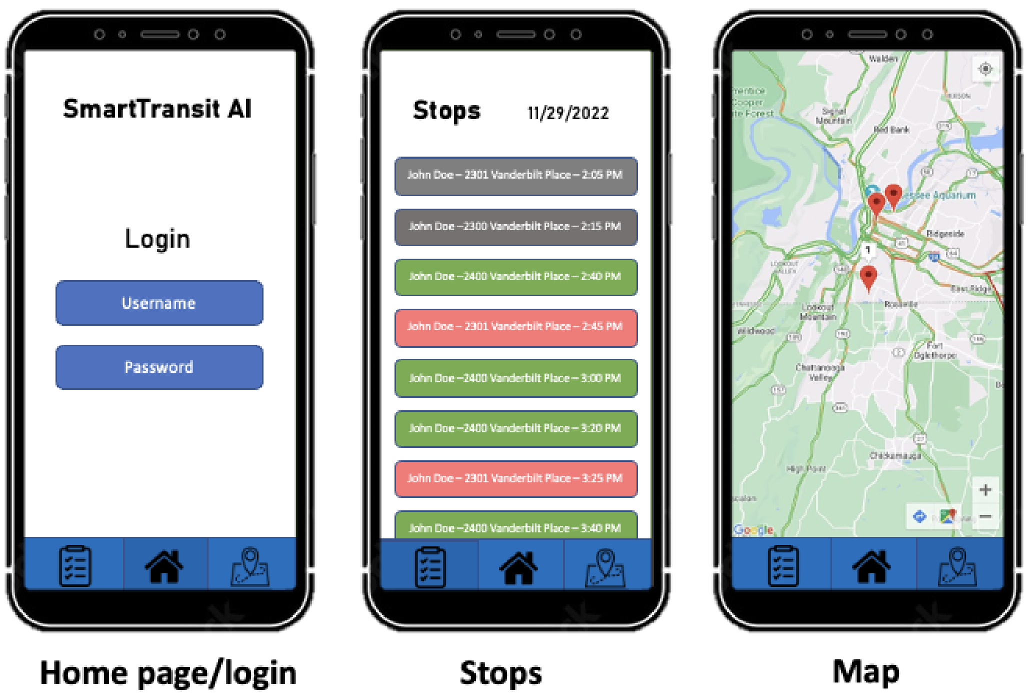
Figure 7.2: Operations manager web application: allows the transit agency to manage clients, take bookings, update route manifests and monitor real-time operations and dispatching.

their smart phone. Users can also call to request trips over the phone which is then booked through the operations manager interface.

The interfaces rely on a set of APIs to manage the various automated processes and to help inform decision-making. We are running a customized Open Source Routing Machine (OSRM) deployment that is augmented with historical traffic conditions. For same-day operations we rely on Mapbox for routing with real-time traffic conditions. For managing bookings we integrated Google Maps Places Autocomplete in the text inputs related to addresses and we use a combination of Mapbox and Google Maps APIs for geo-encoding. The primary data store is MongoDB and we utilize Google Pub-Sub for pushing updates to drivers and users as well as processing real-time vehicle locations. The software is deployed on Google Cloud Platform (GCP).

7.4 Ridepooling Algorithms, Optimization and Interfaces

A key problem for transit agencies that manage ridepooling services is designing algorithms to assign requests to vehicles. In microtransit and paratransit, requests can be for some day in the future, which we refer to as ahead-of-time requests, or can be for the



Stops 11/29/2022

John Doe – 2301 Vanderbilt Place – 2:05 PM
John Doe –2300 Vanderbilt Place – 2:15 PM
John Doe –2400 Vanderbilt Place – 2:40 PM
John Doe – 2301 Vanderbilt Place – 2:45 PM
John Doe –2400 Vanderbilt Place – 3:00 PM
John Doe –2400 Vanderbilt Place – 3:20 PM
John Doe – 2301 Vanderbilt Place – 3:25 PM
John Doe –2400 Vanderbilt Place – 3:40 PM

Figure 7.3: Vehicle operator mobile application: allows driver to manage their route for the day.

same day. Therefore, we need two optimization components. An offline VRP solver is run ahead of time and bulk assigns trip requests to vehicle routes for the upcoming day. In this way, the offline VRP solver generates the initial vehicle schedules for the start of a day. Then when a new same-day request arrives, an online DVRP solver assigns the request to a vehicle that can accommodate the trip without violating the constraints set by the transit agency.

Trip-to-vehicle assignment is fundamentally a vehicle routing problem (VRP). Most research focuses on a set of common VRP formulations that can be classified by the types of constraints applied to the system including vehicle capacities, pickups and dropoffs, and time-related restrictions (time windows) [148]. However, in real-world scenarios the constraints and objectives vary between setting (microtransit vs paratransit) as well as between agencies. This makes it hard to use off-the-shelf algorithms from the research community. Therefore, our offline and online solvers are modular. We defined structured interfaces

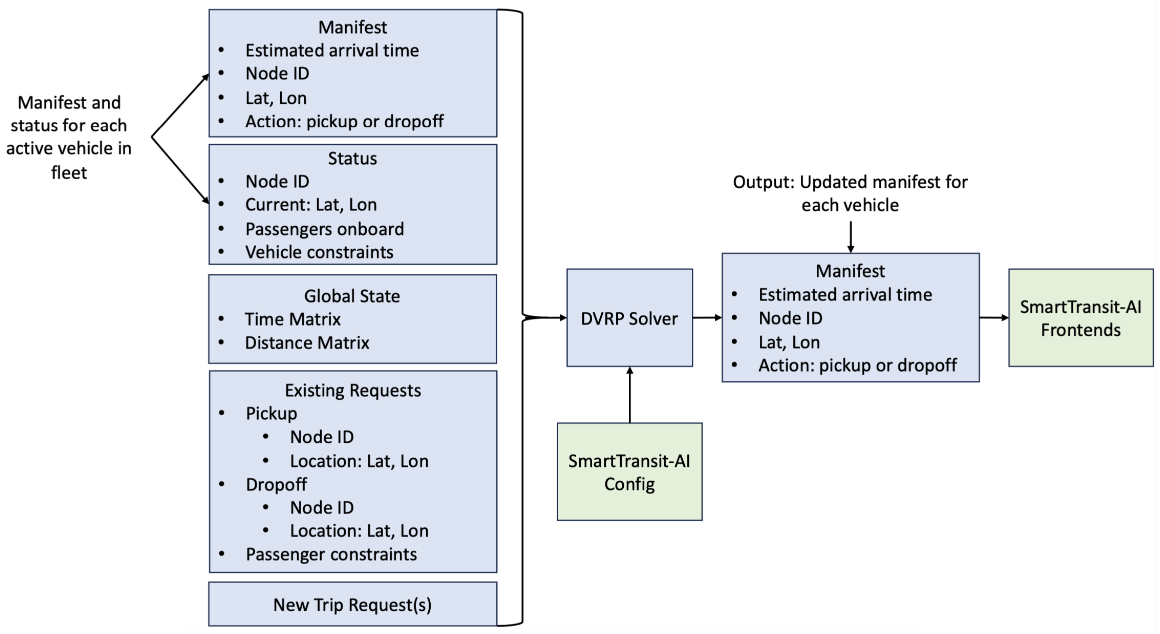


Figure 7.4: DVRP Interface: we defined a common interface for the input and output for incorporating real-time ride-pooling algorithms within SmartTransit-AI so that new algorithms can be quickly adapted and included within the software framework.

for both solvers so that the implementation details of the solver are decoupled from the software itself. In this way, new solvers can be added or constraints can be adapted to fit different transit agency requirements. The solvers are made available by a REST API which means for a new solver to be incorporated the solver endpoint needs to be changed in a configuration file.

A visual representation of the real-time DVRP interface is shown in Figure 7.4. The input consists of a manifest and the status for each vehicle in the fleet that is currently active. The manifest is an ordered list of locations the vehicle will visit. Each location is either a pickup or dropoff for a passenger as well as the estimated arrival time at that location. The status of the vehicle includes the current location of the vehicle, the passengers currently onboard as well as any constraints on the vehicle. Example vehicle-level constraints include the time in which the vehicle can leave the depot or must return to the depot as well as capacities for different types of passengers. The locations in the manifest correspond with existing trip requests that are currently, or scheduled to be serviced. Each request has a pickup and dropoff location as well as any constraints applied to that request. Common constraints may be time windows for which the pickup or dropoff must be serviced, the number of passengers on this trip as well as types of passengers (wheelchair passengers, ambulatory passengers). The new trip request or set of new trip requests includes the same information as the existing requests but these requests are not yet assigned to any vehicle yet. The goal of the DVRP solver is to assign these new trip requests to vehicles in a way that does not violate any constraints and optimizes an objective function. Both the constraints and objective function are set by the transit agency through a SmartTransit-AI configuration file. For routing, we generate a travel time and distance matrix indexed by Node ID. A Node ID is the set of all pickup and dropoff locations for both existing and new requests as well as the depot. In this way the DVRP solver is provided with travel times and distances to be used when optimizing the manifests without having to rely on an external shortest path module. Finally, the DVRP solver must return the updated set of manifests which is processed by the SmartTransit-AI backend and pushed to the various SmartTransit-AI frontend (driver applications so that drivers have the updated routes as well as the operations manager web UI). The offline VRP interface follows a similar structure except all requests are considered new (unassigned) requests and the vehicle manifests are initially empty.

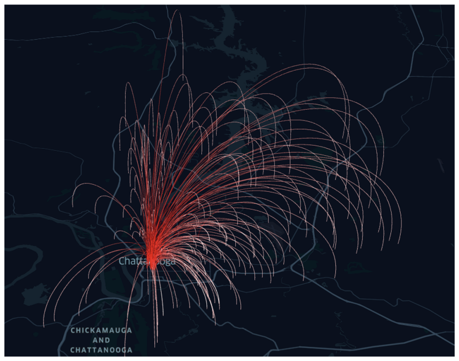
We provide two offline solver implementations included with the software. First is a heuristic solver implemented with Google OR-Tools that was customized to CARTA's paratransit requirements. The second solver is based on our recent work using temporal decomposition which we had to adapt for the real-world constraints of our partner agencies [2]. Additionally, we provide a greedy online solver as well as a state-of-the-art non-myopic online solver for the paratransit setting based on recent work of ours [43]. For the microtransit setting we are currently working on implementing a highly scalable batch solver based on shareability graphs [13].

7.5 Evaluation of Offline VRP Solvers

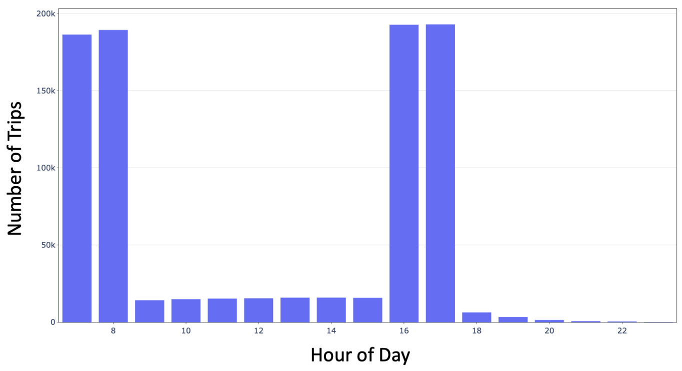
As discussed in the previous section, we implemented two offline VRP solvers within SmartTransit-AI. The first is a heuristic solver implemented with Google OR-Tools, which we refer to as the OR-Tools heuristic solver. The second is a temporal decomposition method based on our work in this space which we refer to as Rolling Horizon [2]. Before running the pilot, which we discuss in Section 7.6, we evaluated the solvers through a set of experiments outlined here. First, in Section 7.5.2 we evaluate the offline VRP solvers in a paratransit setting using synthetic data generated from mobility data in Chattanooga, Tennessee. Second, in Section 7.5.3 we evaluate the offline VRP solvers using real, historical data from microtransit services in Chattanooga (CARTA Go).

7.5.1 Experiment Setup

For both settings, we applied constraints based on CARTA's paratransit setting. This included the following time-window constraints: 1) if the request was appointment-based, the passenger must be dropped off by appointment and picked up no earlier than 60 minutes before the appointment, and 2) if the request was not an appointment-based request the passenger must be picked up within 15 minutes before or after the requested pickup time and then dropped off no later than 60 minutes after the requested pickup time. We also applied to vehicle capacity constraints, in line with CARTA's paratransit vehicles which had a maximum capacity of eight ambulatory passengers and two wheelchair passengers. If any of these constraints could not be met, a request could be dropped - therefore we provide a service rate metric which is the percentage of trips that could be serviced compared to all trip requests for that day.



(a)



(b)

Figure 7.5: Spatial and temporal distribution of synthetic OD datasets from our generative demand model in Chattanooga, TN. (a) Spatial distribution of trip requests for one day in the synthetic paratransit dataset. (b) Temporal distribution trip requests over 24 hours.

7.5.2 Evaluation with Synthetic Origin-Destination Data

To evaluate our approaches, we utilized a generative demand model that generates synthetic trip requests based on movement data. Each trip is represented as an origin-destination (OD) pair with a start and end location and the requested time of day. The generative demand model generates an OD dataset for a day and the number of trips in the dataset can be scaled up or down based on the use-case. The model can scale over 80,000 requests per day, capturing a significant percentage of trips in the region. To evaluate our offline optimization algorithms, we generated a week's worth of OD datasets each with 200 trip requests which represents CARTA's typical weekday paratransit operations. The spatial and temporal breakdown of trip requests in the dataset is shown in Figure 7.5.

We evaluated three offline VRP solvers on the synthetic paratransit dataset. The first solver was a heuristic solver implemented with Google OR-Tools as discussed in Section 7.4. The objective of the Google OR-Tools heuristic solver was to minimize vehicle miles travelled (VMT) with an additional large penalty term for dropping a trip request. We also evaluated our Rolling Horizon Solver [2] with a penalty of 1000 and 2000. The larger penalty represented a more significant impact on dropping a trip request.

We assumed a vehicle capacity of 8 and applied a tight time window constraint in which

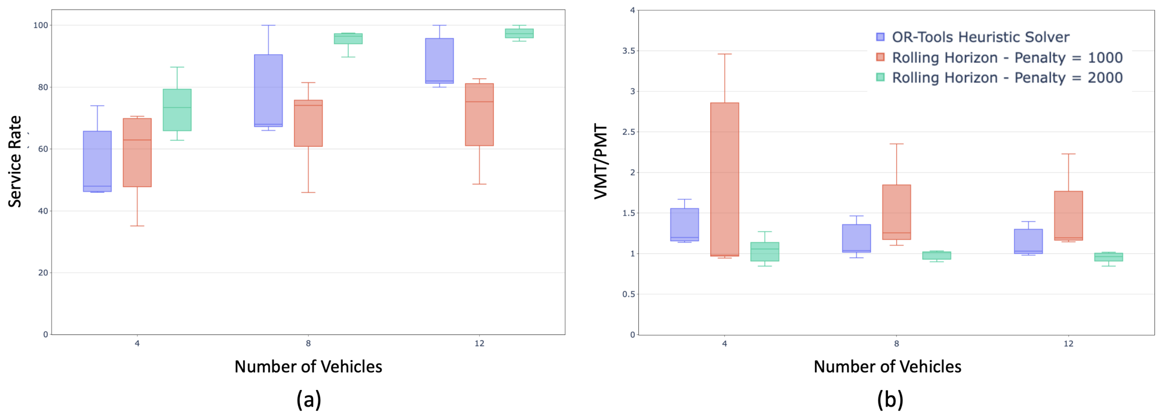


Figure 7.6: (a) Service Rate and (b) VMT/PMT ratio for our two offline VRP solvers - Google OR-Tools and Rolling Horizon [2] evaluated on a synthetic dataset based on mobility patterns in Chattanooga, TN. Rolling Horizon was evaluated with a penalty for dropping a booking of 1000 and 2000.

requests had to be picked up within 15 minutes before or 15 minutes after the requested pickup time and the maximum detour time was 15 minutes. Any trip request that could not be serviced within these time windows was dropped. Therefore, the first metric to consider was service rate, defined as the number of trip requests serviced compared to the total number of trip requests on that day provided in Figure 7.6a.

Figure 7.6b shows the Vehicle Miles Travelled/Passenger Miles Travelled (VMT/PMT ratio). PMT (Passenger Miles Travelled) is the summation of the shortest path between origin and destination for each OD pair in the dataset for a specific day. PMT therefore represents the vehicle miles that would be required if each passenger drove directly between their corresponding origin and destination. VMT represents the total vehicle miles travelled by the vehicles over a day – including the miles a passenger or passengers were on board and when a vehicle was travelling between locations without anyone on–board. Modelling VMT/PMT in this way provides a measure of efficiency of the vehicle route plans by comparing actual miles driven compared to the miles associated with the direct shortest path of each passenger that was served. Therefore, a lower VMT/PMT ratio is more efficient and a VMT/PMT close to 1 is highly efficient and must be evaluated in the context of service rate since VMT/PMT only considers passengers served. As shown in

Figure 3 the best performing model was the Rolling Horizon with a penalty of 2000 both in the context of service rate as well as VMT/PMT - which was approximately one for 4 and 8 vehicles and less than one for 12 vehicles.

Current CARTA operations average a VMT/PMT ratio between 1.7 and 2, therefore Rolling Horizon with a penalty of 2000 has the potential to reduce VMT/PMT by 40%-50% while servicing the same number of requests with the same vehicle configurations. In this way, our methods can service the same number of requests with fewer miles travelled and thus reduces the emissions from the fleet. Alternatively, the improvement in VMT/PMT ratio indicates that we can service more requests with the same number of vehicles – assuming the new trip requests are sampled from the same distribution.

7.5.3 Evaluation with CARTA Go Data (Microtransit)

We also evaluated the SmartTransit-AI Google OR-Tools solver in the microtransit setting using three weekdays of real data in July 2023 from CARTA's microtransit operations (CARTA Go). The results are presented in Table 7.1. The average number of Microtransit requests was 170 per day and there were four vehicles operating in the morning and four vehicles operating in the afternoon/evening. One important note is that we applied the constraints derived from the paratransit setting outlined in Section 7.5.1. However, CARTA Go did not require hard time window constraints in this respect, and therefore the column Time Constraint Violations refers to trips that would have violated these constraints. Conversely, this is reflected when the Smarttransit-AI routes did not service 100% of the requests since some requests could not be serviced without violating the QoS constraints. Note that all requests in the microtransit setting are non-appointment-based requests.

7.6 Chattanooga Implementation and Pilot

To test and evaluate the software in a real-world setting we ran a pilot in August 2023 with CARTA's paratransit team - CARTA Care-a-Van (CAV) in Chattanooga, TN. The goal

Table 7.1: Microtransit results – evaluation of scheduled routes for CARTA's current microtransit operations (CARTA Go) compared to routes generated by Smarttransit-AI. Results were evaluated with real microtransit trip data over three week days in July 2023.

Solver	**VMT/PMT**	**Shared Rate**	**Service Rate**	**Time Constraint Violations**
CARTA Go	1.51	60%	100%	31
SmartTransit-AI (OR-Tools)	1.28	94%	91%	0

of the pilot was to test and evaluate the software framework in a real-world paratransit setting. We chose paratransit over microtransit due to the complex system requirements with paratransit services that are often overlooked when designing general-purpose VRP solvers for MoD. This includes tight time-windows associated with ADA requirements in paratransit and limited vehicles.

We identified three key tasks to evaluate during the pilot. First, we aimed to evaluate the integration of our offline VRP optimizer with the CARTA CAV service. Second, we wanted to evaluate the software during real-time operations. This involved equipping drivers with the driver application on a tablet mounted in the vehicle and monitoring operations through the real-time web interface with members of the CARTA CAV operations team. Third, we wanted to gain feedback from schedulers and drivers on system usability and identify possible improvements going forward.

7.6.1 Pilot Design and Setup

We selected two days for the pilot - August 3, 2023 and August 10, 2023. First, for both days we exported the trip requests, driver schedules, vehicles and scheduled manifests from CARTA's existing system. The data was imported into SmartTransit-AI. For the test dates we then generated new schedules with the Google OR-Tools offline VRP solver. On both days there were 15 vehicles available with schedules staggered between morning and afternoon shifts according to CARTA's driver and vehicle availability. There were 159

Table 7.2: Metrics recorded for system testing on August 3, 2023 and August 10, 2023. VMT: Vehicle Miles Travelled, VDM: Vehicle Detour Miles, VMT/PMT: Vehicle Miles Travelled to Passenger Miles Travelled, Shared Rate: percentage of trip requests that shared their trip with another passenger.

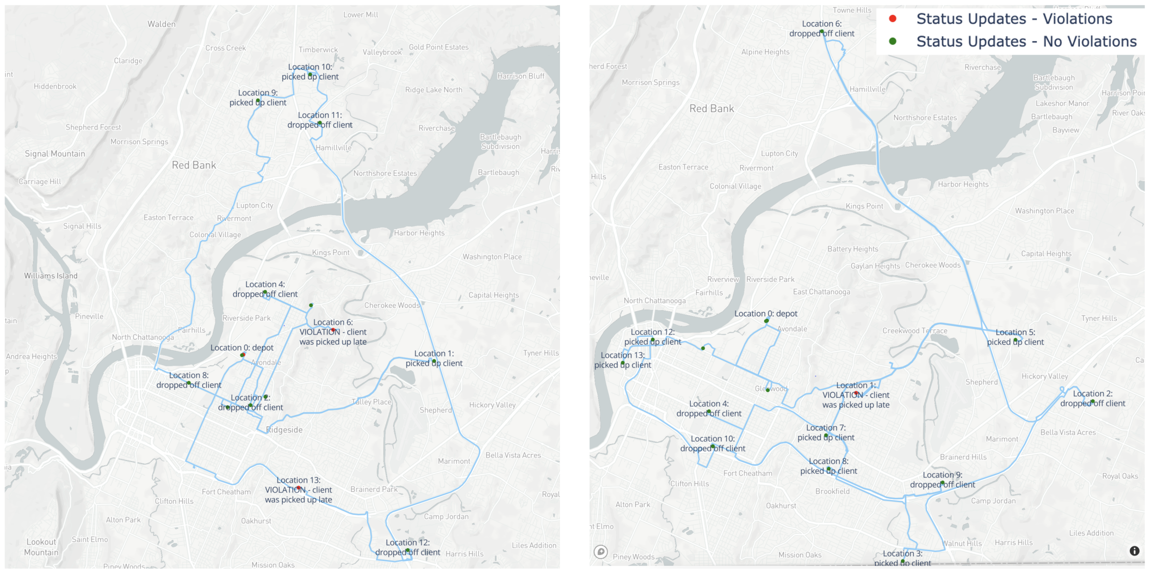
Date	**Solver**	**VMT**	**VDM**	**VMT/PMT**	**Shared Rate**	**Passengers Served**
8-3-2023	CARTA CAV	1531	601	1.41	61%	159
	SmartTransit-AI	1175	300	1.07	86%	159
8-10-2023	CARTA CAV	1269	517	1.27	68%	129
	SmartTransit-AI	1061	281	1.06	84%	129

passengers total on August 3, 2023 and 129 passengers total on August 10, 2023.

CARTA CAV has strict time window constraints for two types of passenger requests. Pickup-constrained requests must be picked up within a 15 minute window before or after the requested pickup time and the passenger must be dropped off within an hour of the requested pickup time at their destination. Dropoff-constrained requests represent appointments where a passenger must be dropped off before their appointment and must be picked up no earlier than one hour before the appointment. Additionally, each vehicle had two capacity constraints - no more than 8 ambulatory passengers and 2 wheelchair passengers could be on a vehicle at any given time. The problem is fundamentally a resource-constrained VRP where all of the passengers known ahead-of-time must be serviced. In this way, CARTA CAV was a useful setting to test our software due to the complex set of constraints as mentioned. All of the constraints were defined in the SmartTransit-AI config as shown in Figure 7.4.

7.6.2 Evaluation of Offline VRP Solver

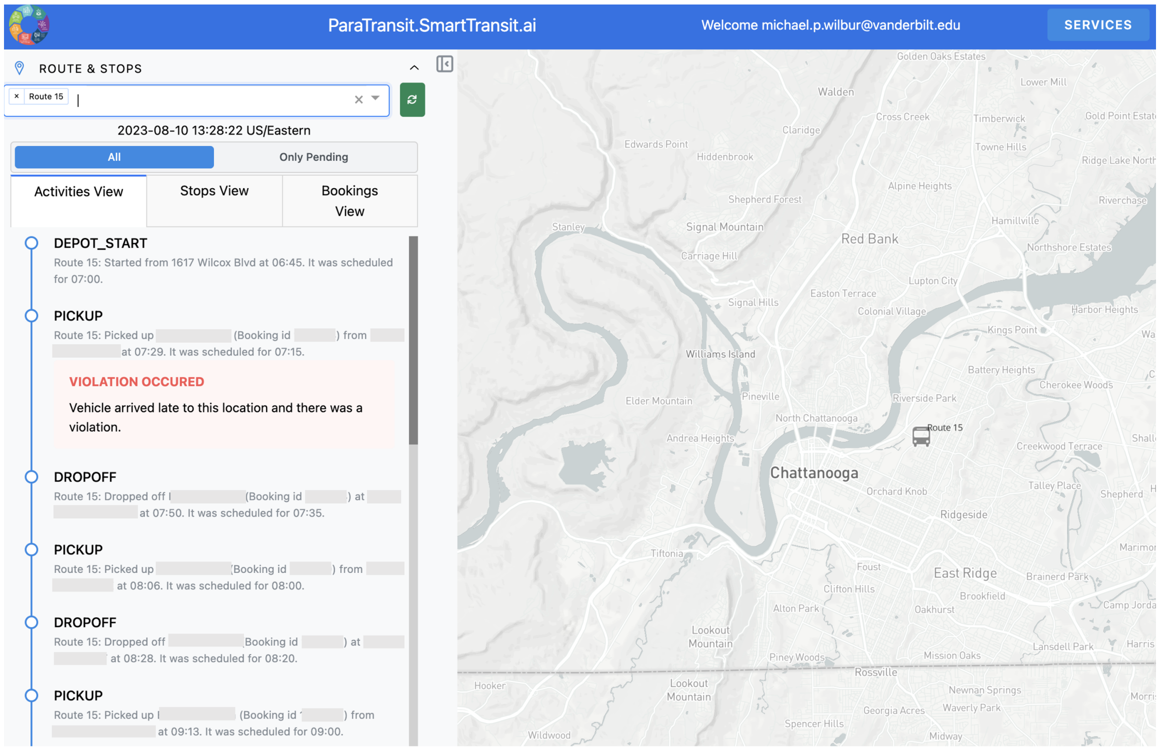
The objective was to minimize Vehicle Miles Travelled (VMT) while servicing all trip requests. Key metrics related to performance of CARTA's original schedule compared to the schedule generated by the SmartTransit-AI offline VRP solver for both days is provided in Table 7.2. As shown, the SmartTransit-AI VRP solver reduced VMT by 356 miles on


Status Updates - Violations
Status Updates - No Violations
Location 0: depot
Location 1: picked up client
Location 2: dropped off client
Location 4: dropped off client
Location 6: VIOLATION - client was picked up late
Location 8: dropped off client
Location 9: picked up client
Location 10: picked up client
Location 11: dropped off client
Location 12: dropped off client
Location 13: VIOLATION - client was picked up late
Location 0: depot
Location 1: VIOLATION - client was picked up late
Location 2: dropped off client
Location 3: picked up client
Location 4: dropped off client
Location 5: picked up client
Location 6: dropped off client
Location 7: picked up client
Location 8: picked up client
Location 9: dropped off client
Location 10: dropped off client
Location 12: picked up client
Location 13: picked up client
Red Bank

(a) (b)

Figure 7.7: GPS trace and status updates during the pilot with CARTA CAV (paratransit). A driver serviced a route the morning of (a) August 3rd, 2023 and (b) August 10, 2023. Violations in which a location was visited outside the time window constraints set by CARTA CAV are shown in red and GPS trace is shown in blue. Names and times were removed due to PII.

August 3, 2023 and by 236 on August 10, 2023. We use Vehicle Miles Travelled to Passenger Miles Travelled VMT/PMT as the metric to represent normalized efficiency where PMT was the total shortest path distance between origin and destination for all trip requests. There was a 24% and 17% improvement in VMT/PMT over CARTA CAV's initial schedule for August 3, 2023 and August 10, 2023 respectively. The efficiency gain correlates with the finding that our implementation had a much higher Shared Rate, which is the percentage of passengers who shared their trip with at least one other passenger compared to CARTA CAV's schedule (86% compared to 61% for August 3, 84% compared to 68% for August 10).



2023-08-10 13:28:22 US/Eastern

DEPOT_START
Route 15: Started from 1617 Wilcox Blvd at 06:45. It was scheduled for 07:00.

PICKUP
Route 15: Picked up [redacted] (Booking id [redacted]) from [redacted] at 07:29. It was scheduled for 07:15.

VIOLATION OCCURED
Vehicle arrived late to this location and there was a violation.

DROPOFF
Route 15: Dropped off [redacted] (Booking id [redacted]) at [redacted] at 07:50. It was scheduled for 07:35.

PICKUP
Route 15: Picked up [redacted] (Booking id [redacted]) from [redacted] at 08:06. It was scheduled for 08:00.

DROPOFF
Route 15: Dropped off [redacted] Booking id [redacted]) at [redacted] at 08:28. It was scheduled for 08:20.

PICKUP
Route 15: Picked up [redacted] (Booking id [redacted]) from [redacted] at 09:13. It was scheduled for 09:00.

Figure 7.8: A screenshot of the real-time operations view in the operations management web application while a driver served route 15 on August 10, 2023. Names and locations of stops hidden to remove PII.

7.6.3 Real-Time Operations

For the pilot we deployed a version of the driver application to Apple TestFlight and installed the application on a set of three iPad's for the CAV team. We used two iPads for testing with a series of test drives with drivers, operators and dispatchers on August 1st. After the tests we selected a route and driver for Thursday August 3rd to run with the driver application and repeated the process for August 10th. The iPad was mounted in one of the standard paratransit vehicles and the driver used the application to follow his route and all updates were watched by the operations team through the web application.

A screenshot of the route and updates from the test is provided in Figure 7.7 for both days. The figures were generated after the runs ended through our reporting module in the operations management web application. This report function joined the GPS traces of the vehicles for each route serviced with the status updates input from the driver when a location was visited to generate an HTML file for download. Violations where the passenger was serviced outside of the time window constraints set by CARTA CAV are shown in red while stops visited on-time are shown in green. The GPS trace is shown in blue.

Figure 7.8 shows a screenshot of the real-time view in the web operations application while a driver serviced route 15 on August 10, 2023. As shown, the real-time view shows the current location of the vehicle servicing the route as well as the status of all locations in the route manifest. We also provide real-time tags to alert the operations team when a 1) violation occurs, 2) there was a no-show because the rider did not board at a pickup location and 3) warnings related to future locations where the vehicle is anticipated to arrive late and may be a potential violation. As shown in Figure 7.8, the driver was late to the first pickup location but quickly was able to make up time and remain on schedule for the subsequent locations. This functionality allows the operations team to both know what violations have occurred in real-time as well as anticipate future delays. For this we use real-time congestion-aware routing from Mapbox for accurate forecasting.

7.7 Conclusion and Outlook

In this chapter we presented SmartTransit-AI, a software framework for real-time MoD operations management. SmartTransit-AI extends current MaaS to allow for customizable constraints, objectives and algorithms so that transit agencies can better manage their operations while incorporating state-of-the-art algorithms and approaches from academia and the open-source community. We also provided an overview of a pilot run with our partner agency CARTA in Chattanooga, TN. We show that the algorithms implemented can result in more efficient services and the software ecosystem has the ability to improve ease-of-use for transit agencies.

Going forward, we would like to test the software in new settings such as microtransit as well as hybrid systems that utilize demand-responsive transit with fixed-line. Given the customizable interfaces, there are minimal changes required to service these settings.

Therefore, we look forward to partnering with our existing partners for further evaluation.

Chapter 8

Conclusion

This dissertation presented a computational framework for demand-responsive urban mobility. We focused on data-driven methods which aim to utilize vast sums of data generated from the now ubiquitous presence of network-connected sensing technologies in urban areas. We studied the role of data and demand-responsive transportation in the context of two high-impact modalities for transit agencies - MoD and fixed-line transit. Our computational framework modelled demand-responsive transportation as a resource allocation problem comprising of four concrete subproblems - 1) Planning, 2) Prediction, 3) Deployment and 4) Software Design. We made contributions to three key challenges. First, we proposed a non-myopic, online algorithm for DVRPs that leveraged the structure of the paratransit problem to combine domain-specific heuristics and MCTS for adaptive planning in this setting. Second, we showed how multi-task and transfer learning can be used to overcome issues of data quality for energy and emissions prediction in the context of fixed-line services. Third, we resolved challenges related to anomaly detection and application design for edge-network deployments in the transportation domain.

Lastly, results of this project were codified into a cloud application for managing and deploying demand-responsive transportation operations. The application framework targets transit agencies and allows them to manage their fleets, schedules and bookings through a collection of web and mobile interfaces. The software exposes the state-of-the-art optimization algorithms and tools proposed in this work with the goal of easing technology transfer between research and practice. We presented the results of a pilot with CARTA's paratransit operation using our software framework.

There are many potential ways in which to extend the work presented in this dissertation. One important extension is in the domain of multi-modal transportation where MoD vehicles provide service to and from fixed-line transit. The goal of these services is to increase utilization on fixed-line transit while extending coverage of public transit to areas of a city with limited access to existing transit service [10]. We are currently looking at developing algorithms and middle-ware that allows transit agencies to service trip requests with multiple public and private operators that provide service in different regions of the city. Additionally, multi-modal services can then be incorporated within our software framework to allow transit agencies to expand their demand-responsive offerings.

Bibliography

[1] Michael Wilbur, Philip Pugliese, Aron Laszka, and Abhishek Dubey. Efficient data management for intelligent urban mobility systems. In *Proceedings of the Workshop on AI for Urban Mobility at the 35th AAAI Conference on Artificial Intelligence (AAAI-21)*, 2021.

[2] Youngseo Kim, Danushka Edirimanna, Michael Wilbur, Philip Pugliese, Aron Laszka, Abhishek Dubey, and Samitha Samaranayake. Rolling horizon based temporal decomposition for the offline pickup and delivery problem with time windows. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI-23)*, 2023.

[3] Eda Beyazit. Evaluating social justice in transport: lessons to be learned from the capability approach. *Transport Reviews*, 31(1):117–134, 2011.

[4] David Harvey. *Social justice and the city*, volume 1. University of Georgia press, 2010.

[5] Federal Highway Administration. Status of the nation's highways, bridges, and transit: 2002 conditions and performance report, 2003.

[6] Thibault Séjourné, Samitha Samaranayake, and Siddhartha Banerjee. The price of fragmentation in mobility-on-demand services. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):1–26, 2018.

[7] Afiya Ayman, Amutheezan Sivagnanam, Michael Wilbur, Philip Pugliese, Abhishek Dubey, and Aron Laszka. Data-driven prediction and optimization of energy use for transit fleets of electric and ICE vehicles. *ACM Transations of Internet Technology*, 2020.

[8] Tai-Yu Ma, Saeid Rasulkhani, Joseph YJ Chow, and Sylvain Klein. A dynamic

ridesharing dispatch and idle vehicle repositioning strategy with integrated transit transfers. *Transportation Research Part E: Logistics and Transportation Review*, 128:417–442, 2019.

[9] Yang Liu, Prateek Bansal, Ricardo Daziano, and Samitha Samaranayake. A framework to integrate mode choice in the design of mobility-on-demand systems. *Transportation Research Part C: Emerging Technologies*, 105:648–665, 2019.

[10] J Carlos Martínez Mori, M Grazia Speranza, and Samitha Samaranayake. On the value of dynamism in transit networks. *Transportation Science*, 2023.

[11] Amutheezan Sivagnanam, Afiya Ayman, Michael Wilbur, Philip Pugliese, Abhishek Dubey, and Aron Laszka. Minimizing energy use of mixed-fleet public transit for fixed-route service. In *35th AAAI Conference on Artificial Intelligence (AAAI)*, pages 14930–14938, February 2021.

[12] Amutheezan Sivagnanam, Salah Uddin Kadir, Ayan Mukhopadhyay, Philip Pugliese, Abhishek Dubey, Samitha Samaranayake, and Aron Laszka. Offline vehicle routing problem with online bookings: A novel problem formulation with applications to paratransit. In *31st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3933–3939, July 2022.

[13] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.

[14] Andrea Simonetto, Julien Monteil, and Claudio Gambella. Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C: Emerging Technologies*, 101:208–232, 2019.

[15] Waldy Joe and Hoong Chuin Lau. Deep reinforcement learning approach to solve

dynamic vehicle routing problem with stochastic customers. In *Conference on Automated Planning and Scheduling (ICAPS)*, volume 30, pages 394–402, 2020.

[16] Sanket Shah, Meghna Lowalekar, and Pradeep Varakantham. Neural approximate dynamic programming for on-demand ride-pooling. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 507–515, 2020.

[17] Afiya Ayman, Michael Wilbur, Amutheezan Sivagnanam, Philip Pugliese, Abhishek Dubey, and Aron Laszka. Data-driven prediction of route-level energy use for mixed-vehicle transit fleets. In *6th IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 41–48, September 2020.

[18] Michael Wilbur, Abhishek Dubey, Bruno Leão, and Shameek Bhattacharjee. A decentralized approach for real time anomaly detection in transportation networks. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 274–282. IEEE, 2019.

[19] Michael Wilbur, Chinmaya Samal, Jose Paolo Talusan, Keiichi Yasumoto, and Abhishek Dubey. Time-dependent decentralized routing using federated learning. In *2020 IEEE 23nd International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2020.

[20] Scott Eisele, Michael Wilbur, Taha Eghtesad, Kevin Silvergold, Fred Eisele, Ayan Mukhopadhyay, Aron Laszka, and Abhishek Dubey. Decentralized computation market for stream processing applications. In *2022 IEEE International Conference on Cloud Engineering (IC2E)*, Pacific Grove, CA, USA, October 2022. IEEE Computer Society.

[21] Michael Wilbur, Amutheezan Sivagnanam, Afiya Ayman, Samitha Samaranayeke, Abhishek Dubey, and Aron Laszka. Artificial intelligence for smart transportation. *arXiv preprint arXiv:2308.07457*, 2023.

[22] Berkay Aydin, Vijay Akkineni, and Rafal A Angryk. Modeling and indexing spatiotemporal trajectory data in non-relational databases. In *Managing Big Data in Cloud Computing Environments*, pages 133–162. IGI Global, 2016.

[23] Shaohua Wang, Yang Zhong, and Erqi Wang. An integrated gis platform architecture for spatiotemporal big data. *Future Generation Computer Systems*, 94:160–172, 2019.

[24] Mordechai Haklay and Patrick Weber. OpenStreetMap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

[25] Jochen Topf and Frederik Ramm. Geofabrik, 2020. URL http://www.geofabrik.de/.

[26] Project-OSRM. Open source routing machine, 2022. URL https://project-osrm.org/.

[27] Static GTFS Reference. Static GTFS reference. https://developers.google.com/transit/gtfs/reference, 2021.

[28] Chattanooga Area Regional Transportation Agency (CARTA). Carta developer center gtfs, 2022. URL https://www.gocarta.org/about/developers-center/.

[29] Nashville MTA (WeGo). Wego gtfs, 2022. URL https://data.nashville.gov/Transportation/WeGo-Transit-and-Middle-TN-RTA-Stops-and-Routes-GT/2246-gtr4.

[30] Tennessee Department of Finance and Administration. Elevation data, 2019. URL https://www.tn.gov/finance/sts-gis/gis/data.html.

[31] Dark Sky. Dark Sky API documentation. https://darksky.net/dev/docs, 2019.

[32] Meteostat. Historical weather and climate data. https://meteostat.net/en Raw data: NOAA, Deutscher Wetterdienst, 2020.

[33] Samriddhi Singla, Ayan Mukhopadhyay, Michael Wilbur, Tina Diao, Vinayak Gajjewar, Ahmed Eldawy, Mykel Kochenderfer, Ross Shachter, and Abhishek Dubey. Wildfiredb: An open-source dataset connecting wildfire spread with relevant determinants. In *35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks*, 2021.

[34] National Oceanic and Atmospheric Administration. Noaa weather data portal. https://www.noaa.gov/, 2022.

[35] Google. General transit feed specification (GTFS) real-time overview, 2022. URL https://developers.google.com/transit/gtfs-realtime/,.

[36] Clever Devices. Clever Devices API documentation. https://www.cleverdevices.com/, 2020.

[37] Viriciti. ViriCiti SDK documentation. https://sdk.viriciti.com/docs, 2020.

[38] HERE Technologies. Here traffic, 2022. URL https://www.here.com/.

[39] INRIX. Inrix traffic, 2022. URL https://inrix.com/.

[40] Xiaoyu Wang, Xiaofang Zhou, and Sanglu Lu. Spatiotemporal data modelling and management: a survey. In *Proceedings 36th International Conference on Technology of Object-Oriented Languages and Systems. TOOLS-Asia 2000*, pages 202–211. IEEE, 2000.

[41] Alice Thudt, Dominikus Baur, and Sheelagh Carpendale. Visits: A spatiotemporal visualization of location histories. In *EuroVis (Short Papers)*, 2013.

[42] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[43] Michael Wilbur, Salah Kadir, Youngseo Kim, Geoffrey Pettet, Ayan Mukhopadhyay, Philip Pugliese, Samitha Samaranayake, Aron Laszka, and Abhishek Dubey. An

online approach to solve the dynamic vehicle routing problem with stochastic trip requests for paratransit services. In *ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, April 2022.

[44] Michael Wilbur, Ayan Mukhopadhyay, Sayyed Vazirizade, Philip Pugliese, Aron Laszka, and Abhishek Dubey. Energy and emission prediction for mixed-vehicle transit fleets using multi-task and inductive transfer learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2021.

[45] George B Dantzig and John H Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[46] Russell W Bent and Pascal Van Hentenryck. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(6): 977–987, 2004.

[47] Lars M Hvattum et al. Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. *Transportation Science*, 40(4):421–438, 2006.

[48] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V Snyder, and Martin Takáč. Reinforcement learning for solving the vehicle routing problem. *Neural Information Processing Systems*, 2018.

[49] Roy Lave and Rosemary Mathias. State of the art of paratransit. *Transportation in the New Millennium*, 478:1–7, 2000.

[50] Marlin W Ulmer et al. Route-based markov decision processes for dynamic vehicle routing problems. Technical report, 2017.

[51] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

[52] Marlin W Ulmer, Barrett W Thomas, and Dirk C Mattfeld. Preemptive depot returns for dynamic same-day delivery. *EURO Journal on Transportation and Logistics*, 8 (4):327–361, 2019.

[53] Yvan Dumas, Jacques Desrosiers, and Francois Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1): 7–22, 1991.

[54] Mykel J Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. MIT press, 2015.

[55] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[56] Ayan Mukhopadhyay, Geoffrey Pettet, Chinmaya Samal, Abhishek Dubey, and Yevgeniy Vorobeychik. An online decision-theoretic pipeline for responder dispatch. In *International Conference on Cyber-Physical Systems*, pages 185–196, 2019.

[57] Maria Gabriela S Furtado, Pedro Munari, and Reinaldo Morabito. Pickup and delivery problem with time windows: a new compact two-index formulation. *Operations Research Letters*, 45(4):334–341, 2017.

[58] Marlin W Ulmer, Dirk C Mattfeld, and Felix Köster. Budgeting time for dynamic vehicle routing with stochastic customer requests. *Transportation Science*, 52(1): 20–37, 2018.

[59] Matthew Zalesak and Samitha Samaranayake. Assignment algorithms for on-demand high-capacity ridepooling. *Technical Report*, 2021.

[60] Marlin W Ulmer, Justin C Goodson, Dirk C Mattfeld, and Marco Hennig. Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests. *Transportation Science*, 53(1):185–202, 2019.

[61] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel J Kochenderfer, and Abhishek Dubey. Hierarchical planning for resource allocation in emergency response systems. In *International Conference on Cyber-Physical Systems*, pages 155–166, 2021.

[62] OpenStreetMap contributors. https://planet.osm.org , 2017.

[63] Geoff Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[64] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. 1987.

[65] Maxim Egorov, Zachary N Sunberg, et al. Pomdps. jl: A framework for sequential decision making under uncertainty. *The Journal of Machine Learning Research*, 18 (1):831–835, 2017.

[66] Kate Keahey, Jason Anderson, et al. Lessons learned from the chameleon testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. July 2020.

[67] Harilaos N Psaraftis, Min Wen, and Christos A Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

[68] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[69] Ulrike Ritzinger, Jakob Puchinger, and Richard F Hartl. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231, 2016.

[70] Huey-Kuo Chen, Che-Fu Hsueh, and Mei-Shiang Chang. The real-time time-dependent vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review*, 42(5):383–408, 2006.

[71] Christian H Christiansen and Jens Lysgaard. A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research Letters*, 35(6):773–781, 2007.

[72] Mostepha R Khouadjia, Briseida Sarasola, Enrique Alba, Laetitia Jourdan, and El-Ghazali Talbi. A comparative study between dynamic adapted pso and vns for the vehicle routing problem with dynamic requests. *Applied Soft Computing*, 12(4): 1426–1439, 2012.

[73] Eiichi Taniguchi and Hiroshi Shimamoto. Intelligent transportation system based dynamic vehicle routing and scheduling with variable travel times. *Transportation Research Part C: Emerging Technologies*, 12(3-4):235–250, 2004.

[74] Michel Gendreau et al. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, 33(4):381–390, 1999.

[75] Andrea Attanasio, Jean-François Cordeau, Gianpaolo Ghiani, and Gilbert Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, 2004.

[76] Nicola Secomandi. A rollout policy for the vehicle routing problem with stochastic demands. *Operations Research*, 49(5):796–802, 2001.

[77] Justin C Goodson, Barrett W Thomas, and Jeffrey W Ohlmann. A rollout algorithm framework for heuristic solutions to finite-horizon stochastic dynamic programs. *European Journal of Operational Research*, 258(1):216–229, 2017.

[78] Antti Lajunen. Energy consumption and cost-benefit analysis of hybrid and electric city buses. *Transportation Research Part C*, 38:1–15, 2014.

[79] Xinkai Wu, David Freese, Alfredo Cabrera, and William A Kitch. Electric vehicles' energy consumption measurement and estimation. *Transportation Research Part D: Transport and Environment*, 34:52–67, 2015.

[80] Yuche Chen, Guoyuan Wu, Ruixiao Sun, Abhishek Dubey, Aron Laszka, and Philip Pugliese. A review and outlook of energy consumption estimation models for electric vehicles. *International Journal of Sustainable Transportation, Energy, Environment, & Policy*, 2021.

[81] Cedric De Cauwer, Wouter Verbeke, Thierry Coosemans, Saphir Faid, and Joeri Van Mierlo. A data-driven method for energy consumption prediction and energy-efficient routing of electric vehicles in real-world conditions. *Energies*, 10(5):608, 2017.

[82] Teresa Pamuła and Wiesław Pamuła. Estimation of the energy consumption of battery electric buses for public transport networks using real-world data and deep learning. *Energies*, 13(9):2340, 2020.

[83] Yuche Chen, Lei Zhu, Jeffrey Gonder, Stanley Young, and Kevin Walkowicz. Data-driven fuel consumption estimation: A multivariate adaptive regression spline approach. *Transportation Research Part C: Emerging Technologies*, 83:134–145, 2017.

[84] eia. EIA energy conversion calculator. https://www.eia.gov/energyexplained/units-and-calculators/energy-conversion-calculators.php, 2021.

[85] epa. EPA greenhouse gases calculator. https://www.epa.gov/energy/greenhouse-gases-equivalencies-calculator-calculations-and-references, 2021.

[86] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2009.

[87] NY MTA. NY MTA subway and bus facts 2019. https://new.mta.info/agency/new-york-city-transit/subway-bus-facts-2019, 2019.

[88] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[89] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

[90] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[91] William C Parr. A note on the jackknife, the bootstrap and the delta method estimators of bias and variance. *Biometrika*, 70(3):719–722, 1983.

[92] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[93] United Nations, Department Economic, Social Affairs, and Population Division. *World Urbanization Prospects: The 2018 Revision (ST/ESA/SER.A/420*. United Nations, New York.

[94] Omnigo. 911 dispatch software, 2019. URL https://www.omnigo.com/solutions/computer-aided-dispatch-software.

[95] Frank Perry, Kelli Raboy, Ed Leslie, Zhitong Huang, Drew Van Duren, et al. Dedicated short-range communications roadside unit specifications. Technical report, United States. Dept. of Transportation, 2017.

[96] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[97] Thomas Pajor. Multi-modal route planning. *Universität Karlsruhe*, 2009.

[98] Traffic Assignment Manual. Bureau of public roads. *US Department of Commerce*, 1964.

[99] Peter Sanders and Dominik Schultes. Engineering highway hierarchies. In *ESA*, volume 6, pages 804–816. Springer, 2006.

[100] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. *Experimental Algorithms*, pages 319–333, 2008.

[101] Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

[102] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[103] Lester R Ford Jr. Network flow theory. Technical report, RAND CORP SANTA MONICA CA, 1956.

[104] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1): 87–90, 1958.

[105] George Dantzig. *Linear programming and extensions*. Princeton university press, 2016.

[106] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[107] Andrew V Goldberg and Renato Fonseca F Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, pages 26–40, 2005.

[108] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In *European Symposium on Algorithms*, pages 568–579. Springer, 2005.

[109] Gabriele Di Stefano, Alberto Petricola, and Christos Zaroliagis. On the implementation of parallel shortest path algorithms on a supercomputer. In *International Symposium on Parallel and Distributed Processing and Applications*, pages 406–417. Springer, 2006.

[110] Yuxin Tang, Yunquan Zhang, and Hu Chen. A parallel shortest path algorithm based on graph-partitioning and iterative correcting. In *2008 10th IEEE International Conference on High Performance Computing and Communications*, pages 155–161. IEEE, 2008.

[111] Andreas Crauser, Kurt Mehlhorn, Ulrich Meyer, and Peter Sanders. A parallelization of dijkstra's shortest path algorithm. In *International Symposium on Mathematical Foundations of Computer Science*, pages 722–731. Springer, 1998.

[112] Christian Vetter. Parallel time-dependent contraction hierarchies. *Student Research Project*, page 134, 2009.

[113] Tibor Vukovic. Hilbert-geohash-hashing geographical point data using the hilbert space-filling curve. Master's thesis, NTNU, 2016.

[114] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[115] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[116] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[117] docker. docker, 2019. URL https://www.docker.com/.

[118] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

[119] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[120] Chinmaya Samal, Liyuan Zheng, Fangzhou Sun, Lillian J Ratliff, and Abhishek Dubey. Towards a socially optimal multi-modal routing platform. *arXiv preprint arXiv:1802.10140*, 2018.

[121] Fangzhou Sun, Chinmaya Samal, Jules White, and Abhishek Dubey. Unsupervised mechanisms for optimizing on-time performance of fixed schedule transit vehicles. In *Smart Computing (SMARTCOMP), 2017 IEEE International Conference on*, pages 1–8. IEEE, 2017.

[122] Fangzhou Sun, Yao Pan, Jules White, and Abhishek Dubey. Real-time and predictive analytics for smart public transportation decision support system. In *2016 IEEE*

International Conference on Smart Computing (SMARTCOMP), pages 1–8. IEEE, 2016.

[123] Vijay K Shah, Shameek Bhattacharjee, Simone Silvestri, and Sajal K Das. Designing sustainable smart connected communities using dynamic spectrum access via band selection. In *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments*, page 12. ACM, 2017.

[124] Meital Ben Sinai, Nimrod Partush, Shir Yadid, and Eran Yahav. Exploiting social navigation. *arXiv preprint arXiv:1410.0151*, 2014.

[125] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[126] Gerhard P Hancke, Gerhard P Hancke Jr, et al. The role of advanced sensing in smart cities. *Sensors*, 13(1):393–425, 2012.

[127] Michael Batty, Kay W Axhausen, Fosca Giannotti, Alexei Pozdnoukhov, Armando Bazzani, Monica Wachowicz, Georgios Ouzounis, and Yuval Portugali. Smart cities of the future. *The European Physical Journal Special Topics*, 214(1):481–518, 2012.

[128] Hafedh Chourabi, Taewoo Nam, Shawn Walker, J Ramon Gil-Garcia, Sehl Mellouli, Karine Nahon, Theresa A Pardo, and Hans Jochen Scholl. Understanding smart cities: An integrative framework. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 2289–2297. IEEE, 2012.

[129] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1): 22–32, 2014.

[130] Scott Eisele, Istvan Mardari, Abhishek Dubey, and Gabor Karsai. Riaps: resilient information architecture platform for decentralized smart systems. In *2017 IEEE*

20th International Symposium on Real-Time Distributed Computing (ISORC), pages 125–132. IEEE, 2017.

[131] Fangzhou Sun, Abhishek Dubey, and Jules White. Dxnat—deep neural networks for explaining non-recurring traffic congestion. In *Big Data (Big Data), 2017 IEEE International Conference on*, pages 2141–2150. IEEE, 2017.

[132] Simon J Julier and Jeffrey K Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[133] Rong Jiang, Rongxing Lu, Ye Wang, Jun Luo, Changxiang Shen, and Xuemin Sherman Shen. Energy-theft detection issues for advanced metering infrastructure in smart grid. *Tsinghua Science and Technology*, 19(2):105–120, 2014.

[134] Shameek Bhattacharjee, Aditya Thakur, and Sajal K Das. Towards fast and semi-supervised identification of smart meters launching data falsification attacks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 173–185. ACM, 2018.

[135] Amin Ghafouri, Aron Laszka, Abhishek Dubey, and Xenofon Koutsoukos. Optimal detection of faulty traffic sensors used in route planning. In *Proceedings of the 2nd International Workshop on Science of Smart City Operations and Platforms Engineering*, pages 1–6. ACM, 2017.

[136] Constantine Manikopoulos and Symeon Papavassiliou. Network intrusion and fault detection: a statistical anomaly approach. *IEEE Communications Magazine*, 40(10): 76–82, 2002.

[137] Vasilis Chatzigiannakis, Symeon Papavassiliou, Mary Grammatikou, and B Maglaris. Hierarchical anomaly detection in distributed large-scale sensor networks. In *Computers and Communications, 2006. ISCC'06. Proceedings. 11th IEEE Symposium on*, pages 761–767. IEEE, 2006.

[138] Alberto Gonzalez Prieto, Daniel Gillblad, Rebecca Steinert, and Avi Miron. Toward decentralized probabilistic management. *IEEE Communications Magazine*, 49(7), 2011.

[139] Gautam Biswas, Hamed Khorasgani, Gerald Stanje, Abhishek Dubey, Somnath Deb, and Sudipto Ghoshal. An application of data driven anomaly identification to spacecraft telemetry data. In *Prognostics and Health Management Conference*, 2016.

[140] Jeonghee Chi, Yeongwon Jo, Hyunsun Park, and Soyoung Park. Intersection-priority based optimal rsu allocation for vanet. In *Ubiquitous and Future Networks (ICUFN), 2013 Fifth International Conference on*, pages 350–355. IEEE, 2013.

[141] Baber Aslam and Cliff C Zou. Optimal roadside units placement along highways. In *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*, pages 814–815. IEEE, 2011.

[142] Baber Aslam, Faisal Amjad, and Cliff C Zou. Optimal roadside units placement in urban areas for vehicular networks. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000423–000429. IEEE, 2012.

[143] Traffic Message Channel. Traffic message channel documentation. https://wiki.openstreetmap.org/wiki/TMC, 2018.

[144] Michael Wilbur, Maxime Coursey, Pravesh Koirala, Zakariyya Al-Quran, Philip Pugliese, and Abhishek Dubey. Mobility-on-demand transportation: A system for microtransit and paratransit operations. In *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)*, ICCPS '23, pages 260–261, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700361. doi: 10.1145/3576841.3589625. URL https://doi.org/10.1145/3576841.3589625.

[145] Michael Wilbur, Sophie Pavia, Maxime Coursey, Pravesh Koirala, Zakariyya Al-Quran, Philip Pugliese, and Abhishek Dubey. Microtransit optimizer for mobility-on-demand. In *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2023.

[146] Python package - names. URL https://pypi.org/project/names/.

[147] Mauro Salazar, Federico Rossi, Maximilian Schiffer, Christopher H Onder, and Marco Pavone. On the interaction between autonomous mobility-on-demand and public transportation systems. In *2018 21st international conference on intelligent transportation systems (ITSC)*, pages 2262–2269. IEEE, 2018.

[148] Grigorios D Konstantakopoulos, Sotiris P Gayialis, and Evripidis P Kechagias. Vehicle routing problem and related algorithms for logistics distribution: a literature review and classification. *Operational research*, pages 1–30, 2020.

LIST OF ABBREVIATIONS

CARTA	Chattanooga Area Regional Transportation Authority
CBSE	Component-based Software Engineering
CPS	cyber physical systems
DDoS	distributed denial-of-service
DSL	Domain Specific Language
EV	Electric Vehicle
FMLM	First-Mile-Last-Mile
GTFS	General Transit Feed Specification
HV	Hybrid Vehicle
ICEV	Internal Combustion Engine Vehicle
IID	independent and identically distributed
ILP	Integer Linear Program
ITL	Inductive Transfer Learning
ITS	Intelligent Transportation Systems
LP	Linear Program
MCTS	Monte-Carlo tree search
MDP	Markov decision process
MIC	Model Integrated Computing
ML	Machine Learning

MoD	Mobility-on-Demand
MTL	Multi-task Learning
NLP	natural language processing
OD	Origin-Destination
OSM	Open Street Maps
OSRM	Open Source Routing Machine
PII	Personally Identifiable Information
QoS	Quality of Service
RL	Reinforcement Learning
RSU	Road-side Unit
RSU	Roadside Unit
RTV	Request-Trip-Vehicle Graph
RV	Request-Vehicle Graph
SMDP	semi-Markov decision process
VDM	Vehicle Detour Miles
VMT	Vehicle Miles Travelled
VMT/PMT	Vehicle Miles Travelled to Passenger Miles Travelled