Design of a synthetic data generation and simulation framework for mobility on demand applications

Mohammadmahdi Zaeimi

A Thesis

in

The Department

of

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Quality Systems Engineering) at

Concordia University

Montréal, Québec, Canada

November 2022

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mohammadmahdi Zaeimi**

Entitled: **Design of a synthetic data generation and simulation framework for mobility on demand applications**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Quality Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Dr. Anjali Awasthi

_____ Examiner
Dr. Dr. Andrea Schiffauerova

_____ Supervisor
Dr. Chun Wang

Approved by _____
Dr. Zachary Patterson, Graduate Program Director

March 6, 2023

_____
Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Design of a synthetic data generation and simulation framework for mobility on demand applications

Mohammadmahdi Zaeimi

Urbanization increases issues such as traffic congestion, lack of parking spots, and underutilized vehicles. In recent years, mobility-on-demand (MOD) concept has been proposed to effectively mitigate these issues. However, a common issue with MOD research is the lack of precise traffic data for conducting transportation-related studies and improving the proficiency of MOD systems. This is mainly because of data privacy concerns, GPS device limitations or errors, and expensive infrastructures for collecting real-time traffic data. Given the constraints, traffic simulations could be a reasonable solution for simulating the dynamic MOD activities such as distributing vehicles in the cities of interest and mimicking their movement behaviours. Despite the features that existing traffic simulators provide, they are not designed to support MOD use cases explicitly. For instance, background traffic generated by these simulators mostly follows random algorithms and the traffic flow is not based on real traffic patterns of the region. Another issue could be the lack of integration APIs to accept user inputs while the simulation is running to adapt the behaviour of the simulation. In this thesis, a synthetic MOD data generation framework is proposed. This framework takes a map region, real traffic data, and service vehicles trip plan as input. Using the ARIMA machine learning algorithm, we could predict demand and generate background traffic, followed by simulating the service vehicles in the region. The proposed framework generates synthetic traffic based on real traffic patterns and then simulates the service vehicles' movements on the map. While the simulation is running, the framework monitors the vehicles and collects real-time trajectory data. This framework leverages the features of SUMO as a microscopic simulation engine. In addition, established HTTP APIs enable third-party integration and allow users to control vehicles and trips

on the map before and during the simulation execution. The offered simulation features include and are not limited to, the importation of a trip plan for numerous vehicles and the update of vehicle destinations. In addition to integration APIs, the proposed framework provides a graphical user interface to facilitate simulation setup and execution. The provided user interface enables users to explore a map, specify a region on the map, and then choose it as a simulation boundary. Throughout the simulation, the software core captures and stores real-time data on vehicle movement in a database that might be utilized for mobility-on-demand research. This simulation framework returns comprehensive service vehicle trajectories, departure time, destination time, travel duration, route length, and service vehicle status. The proposed software is open-source and publicly available, and its capabilities could be improved for future study.

Acknowledgments

I want to sincerely thank my sweet wife Simindokht Samavat for her unwavering support during all of the challenging and frustrating circumstances we encountered while working toward our objectives. Her kind assistance has had a significant impact on this effort.

Also, I appreciate the effort and attention that my supervisor, Dr. Chun Wang of the Concordia Institute for Information Systems Engineering, has put into my study. He supported me at every level of this study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

1.1 The proliferation of mobility on demand applications

Global urbanization is expanding dramatically. According to a recent United Nations research, 68 percent of the world's population would dwell in urban areas by 2050, up from 55 percent currently (*UN population projection*, 2018). Urbanization increases urban travel, which exacerbates its negative impacts (Zardini, Lanzetti, Pavone, & Frazzoli, 2022). Because of their predominant use in urban areas, private automobiles are often fairly technologically advanced and capable of travelling large distances; but, because these automobiles spend the vast majority of their time parked, they are considered to be quite underused (Mitchell, Borroni-Bird, & Burns, 2010). The mobility-on-demand (MoD) concept has a high potential to decrease such issues. An MoD system distributes vehicles around urban regions, and upon receipt of a request, the vehicle is allocated to the requester. The mobility-on-demand systems reduce problems such as pollution, congestion, and the lack of parking spots while satisfying individual transportation requirements (Mitchell et al., 2010; Pavone, 2015).

We can study mobility-on-demand on a supply-and-demand framework. The United States Department of Transportation presented a concept of the MOD illustrating how multi-modal transportation operations management may interact with the supply and demand sides of the MOD. (Shaheen et al., 2017).

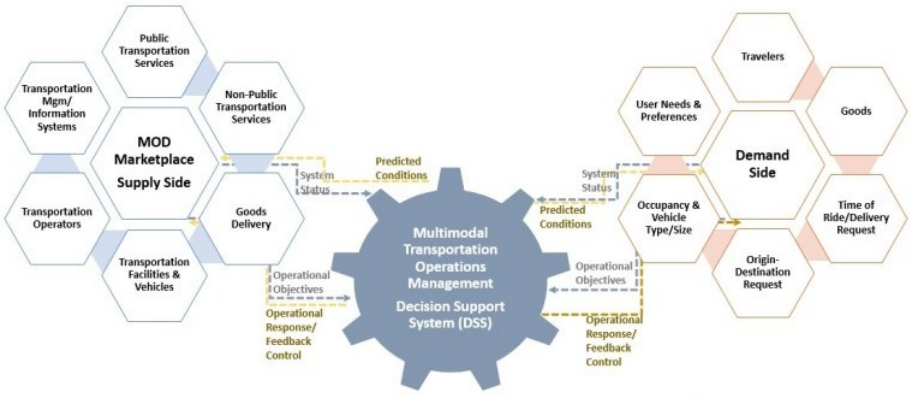

Figure 1.1: US Department of Transportation MOD Vision (Shaheen et al., 2017)

1.1.1 Supply Side

All the participants, operators, and equipment involved in providing transportation services for the delivery of people or products make up the ecosystem's supply side. Public transportation facilities, private transportation services such as Uber, roads, and any type of vehicle that provides the service are the supply side of the MOD system.

1.1.2 Demand Side

All system users (travellers and couriers), along with their preferences and decisions, make up the ecosystem's demand side. All travellers, including, pedestrians, riders, drivers, origin-destination requests, and traveller preferences, are all categorized on the demand side of the MOD systems.

Despite its benefits, MOD systems have limitations, such as a tendency to become unbalanced due to unequally distributed transportation demand. To address these problems, ride-hailing businesses began offering a variant of MOD services in which central management allocates passengers to an on-demand driver, who subsequently transports them to their final destination (Berger, Chen, & Frey, 2018). Although central management can help preserve the system's balance, it cannot mandate the allocation of vehicles to passengers because drivers make the final decision and seek to maximize their revenue (Zardini et al., 2022). Understanding and modelling such systems is quite challenging and necessitates a vast array of skills. While modelling the relationships between

the many parties involved in two-sided platforms has proven to be difficult, their manifestation in the setting of dense and congested urban transportation networks adds another layer of complexity (Kucharski & Cats, 2020b). Researchers frequently investigate the following topics: supply-demand interactions (Nourinejad & Ramezani, 2020; Xu, Yin, & Ye, 2020), optimal matching of drivers to requests (Cui, Makhija, Chen, He, & Khani, 2021; Zuniga-Garcia, Tec, Scott, Ruiz-Juri, & Machemehl, 2020), driver repositioning tactics (Afeche, Liu, & Maglaras, 2018; Holler et al., 2019), pooled rides (Bischoff, Maciejewski, & Nagel, 2017; Kucharski & Cats, 2020a), driver engagement, working shifts, and platform selections (Ashkrof, de Almeida Correia, Cats, & van Arem, 2020; Bokányi & Hannák, 2020). Each of these study fields is comprised of a number of critical and difficult research issues. The greatest difficulty is in portraying the entire system with its internal dependencies, as well as its non-determinism and adaptive development. In the lack of a comprehensive simulation platform, the majority of research has focused on a particular component.

1.2 The need for an explicit mobility-on-demand simulation platform

Mobility trajectory datasets are crucial for mobility-on-demand experimental studies. However, limitations such as privacy considerations have restricted the accessibility of real trajectory data sets. This has led to the development of traffic simulators, that mimic moving entities on the maps to generate datasets of pseudo-realistic trajectories. To execute a traffic simulation regardless of the simulation tools, having a realistic traffic scenario is significant. Although it is possible to generate random traffic on the map, the distribution of vehicles does not follow real-world patterns. As a result, simulations based on random traffic cannot be used as an appropriate platform for mobility-on-demand studies. On the other hand, various GPS sensors and mobile devices have gathered massive amounts of location data from vehicles and individuals, drawing a lot of interest in spatial-temporal data like trajectory. However, because these trajectories are collected from human and vehicle mobility, it is highly sensitive data and privacy concerns restrict access to these data. Another approach to accessing traffic data could be traffic counting devices installed on the roads. This approach is not flexible and is highly cost and time-oriented and needs infrastructure. The aforementioned problems have led to the development of a synthetic MOD data generation and

simulation framework. This framework can help us to generate more realistic trajectories based on available historical data. We require a vehicle traffic simulator and an acceptable scenario to assess new concepts in a realistic setting in order to analyze mobility patterns, traffic congestion, or innovative communication protocols. There are several traffic simulators available, and they vary in the kind of simulations they can do. Macroscopic traffic simulators concentrate on the traffic flows but ignore the impact of a single car. Microscopic simulators are typically favoured in the vehicular networking sector because the behaviour of a single vehicle is frequently of interest and requires meticulous modelling. Although microscopic simulators could be a proper solution for studying MOD applications, generating scenarios, testing, and viewing the results need very deep knowledge of simulation. The microscopic simulators need high levels of expertise for configuring the system and it could be dramatically sophisticated to execute a simulation (Codeca, Frank, & Engel, 2015). Urban traffic simulators such as SUMO provide many tools for conducting the simulations, however, we need simulation and visualization tool which is easy to configure and provides the required features of MOD services (Lopez et al., 2018).

1.2.1 Dispatching and matching in dynamic environments

Many problems still exist in the present MOD systems and need to be thoroughly investigated. Addressing the imbalance between demand (i.e., the trip requests) and supply (i.e., the vehicles), which is brought on by the asymmetric demand distribution over time and geography, is one of the key issues for a large-scale MOD system with a lot of trip demands from a metropolitan region (X. M. Chen, Zheng, Ke, & Yang, 2020). To address the issue, two categories of solutions are mostly being researched right now. The first approach uses price strategies to affect the demand side (X. Wang, Liu, Yang, Wang, & Ye, 2019). Surge pricing, however, is seen to have a considerable negative impact on trip demand but a weak positive influence on vehicle supply, suggesting that it may not greatly increase platform profit (L. Chen, Mislove, & Wilson, 2015). In order to rebalance cars, the second type of approach concentrates on the supply side and involves vehicle relocation. To more effectively meet demand, the MOD platform moves cars to areas where it anticipates future demand to be higher (Liu, Xie, & Chen, 2021). As the MOD systems grow, the matching and dispatching strategies need improvements to support the growing demand and limited supply. To

overcome these challenges, not only innovative solutions and algorithms are needed, stakeholders need to have access to sufficient tools for testing and studying their ideas on a simulated system.

1.2.2 Communicating with stakeholders and general public

The solutions should benefit the MOD stakeholders in order to provide novel enhancements to current MOD systems. Since there are so many different stakeholders in such dispersed systems, communication tactics are crucial when introducing new concepts. Stakeholders must comprehend the system, its problems, and potential remedies. Having said that, it appears to be crucial to have the right tools for interacting with stakeholders. Ideas would be presented more effectively if they were visualized and replicated for the audience. Additionally, because not all stakeholders have the same degree of understanding, the visualization tools aid in making the ideas appear more credible to other stakeholders, including the general public.

1.2.3 Common requirements for simulation and visualization

The MOD applications usually perform the matching of supply and demand. As a result, these applications provide matching results that include sets of origin and destination pairs. In order to be able to test these origin-destination pairs, an ideal simulation and visualization platform must be capable of providing the following features to be beneficial for the MOD applications:

- Map-based MOD region determination flexibility
- Accepting inputs of origin-destination pairs
- Background traffic based on real traffic patterns of the region
- Supporting on-demand changes to the simulation
- Allow integration with MOD applications through APIs

As MOD applications are not limited to specific areas or neighbourhoods, the simulation and visualization platform should be flexible enough to execute scenarios in any user-defined region. Also, it is expected that the simulation platform takes simple inputs and performs any required

conversions internally. This includes converting addresses from text to GPS coordinates which are referred to as geocoding. Another expectation would be the flexibility of traffic configuration. This feature allows users, to define the density of the traffic to observe the results in different traffic scenarios. Also, it is highly probable that an MOD application needs to change a trip plan while it is in progress. The change might be changing the destination or ignoring a specific destination. API integration is another important feature. The MOD applications could leverage their algorithm results if they can connect to the simulation platform through APIs, perform simulations, and optimize their algorithms based on the results of the simulation. The simulation results consist of the travel time of the specific vehicles between user-defined departure and destination.

1.2.4 Lacking of tools

Although the existing simulation platforms have a variety of tools to design and execute traffic simulation scenarios, there is no clear and easy-to-use tool to directly support MOD application requirements. The focus of this thesis is to design and implemented a simulation platform to enable its users to use an interface to easily execute a simulation. These features include defining the simulation region on the map, defining traffic density while it follows an approximate real-world traffic pattern, accepting a set of origin and destination for vehicles, accepting on-demand changes while the simulation is running and collecting real-time trajectories of specific vehicles on the simulation network. Furthermore, the existing simulation engines cannot integrate with other MOD applications because they usually do not provide APIs for integration purposes. Since these features are not provided by existing simulation platforms, it could be beneficial to have a simulation platform that supports such features.

1.3 Outline of thesis

The thesis is organized into 7 chapters with an overall structure presented in Figure 1.2. Following Chapter 1 of introduction and motivation, Chapter 2 will first provide the literature review of the traffic simulation approaches. A few simulation tools are also discussed. Furthermore, synthetic traffic data generation approaches are reviewed in this chapter. Chapter 3 discusses the primary

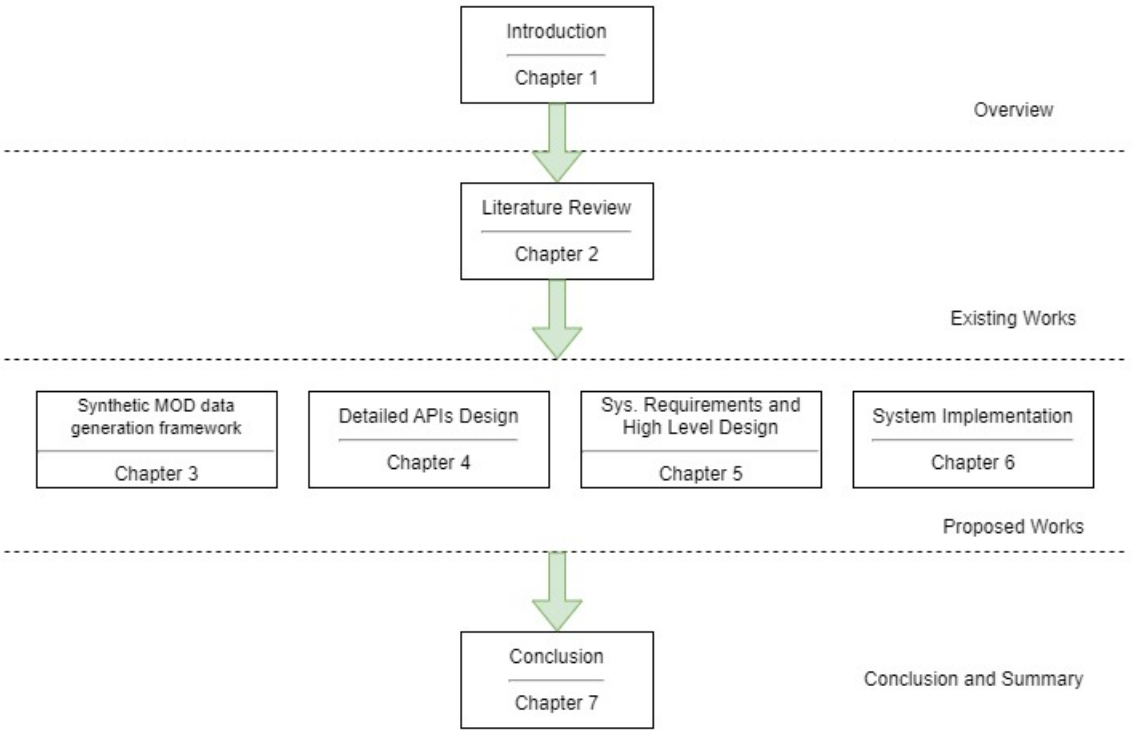

Figure 1.2: Overall structure of the thesis

requirements of the VSP system and provides use cases of these requirements. In chapter 4, the APIs that are developed in the VSP are discussed. Chapter 5, describes how the designed system produces synthetic traffic data and the tools that are used. In this chapter, the role of demographic data is discussed as well. In chapter 6, we provide details of implementing the VSP, including the programming languages, tools, and libraries. Also, the modules of the application and their relationship are presented in this chapter. Chapter 6 will finish by reviewing a test case scenario and describes how we can follow the steps for a complete simulation execution. Chapter 7 concludes our work and presents future works.

Chapter 2

Background and Literature Review

This chapter briefly introduces discrete event simulation, then presents a few existing simulation and visualization tools in the area of urban mobility. Following this traffic, simulation approaches are reviewed followed by a literature review on traffic synthetic data generation.

2.1 Discrete Event Simulation

In a discrete event system, state changes (also known as events) take place instantly and at distinct points in time. Between two successive events, it is assumed that nothing (i.e. nothing interesting) occurs, meaning that the system's state does not change (in contrast to continuous systems where state changes are continuous) (Varga, 2001). The systems that could be viewed as discrete event systems, could be simulated using discrete event simulation (DES). The time when events occur is often called *event timestamp*.

When analytical calculations fail or a simple model structure is needed to communicate findings to stakeholders through animation, simulation, especially DES, is an effective way to address the unresolved problems (Kogler, Rauch, et al., 2018). Urban traffic could be considered as a discrete event model and it could be simulated using discrete event simulation (Abohashima, Gheith, & Eltawil, 2019). For example, in the urban mobility context, some of the events for a vehicle could be as follows:

- Vehicle starts moving from its departure.

- Vehicle reaches its destination.
- Vehicle changes its route.

2.1.1 Simulation Time

In contrast to real-time or CPU time, which refers to how long the simulation software has been operating and how much CPU time it has been used, the time within the model is frequently referred to as simulation time, model time, or virtual time (Varga, 2001). Discrete event simulation can execute the simulation with configurable speed, but the simulation time should represent the real-world time duration that takes to complete an event or simulation.

2.1.2 The Event Loop

Discrete event simulation maintains a set of events that will occur in future time steps. This event data structure is often called FES (Future Event Set) or FEL (Future Event List). With this set of events, the following pseudo-code would be used by simulation (Varga, 2001):

```
initialize -- this includes building the model and
inserting initial events to FES
while (FES not empty and simulation is not yet complete)
{
retrieve the first event from FES
t:= timestamp of this event
process event
(processing may insert new events in FES or delete existing ones)
}
finish simulation (write statistical results, etc.)
```

Firstly, the initialization step builds models and data structures for processing the simulation and inserts the initial events into the FES to make sure that the simulation can start. In this step, the simulation reads inputs such as road network, and the input demand if exists. Then, events from the FES are consumed according to their timestamp. The simulation terminates when there are no more events or when it is no longer required to continue the simulation because model time or CPU time has reached a predetermined limit. Also, the termination might occur when the statistics have

reached the specified level of accuracy. In this step, the simulation stores the statistics and logs regarding the simulation results into a user-defined or default output.

2.2 Traffic simulation approaches in the literature

One of the primary focuses of traffic simulation is to analyze traffic behaviour by providing a road network, vehicle initial state, and behaviour model of traffic. Modelling and simulation of traffic flows could be classified as microscopic (Shen & Jin, 2012), macroscopic (Sewall, Wilkie, Merrell, & Lin, 2010), and mesoscopic (Sewall, Wilkie, & Lin, 2011).

Microscopic approaches simulate the dynamics of each vehicle under the impact of its surroundings, in contrast to macroscopic methods, which consider the collection of cars as a continuous flow. Mesoscopic models, on the other hand, mimic traffic at various degrees of complexity by combining the best aspects of microscopic and macroscopic models. A basic issue in traffic simulation is the development and representation of road networks (Chao et al., 2020). The availability of empirical traffic flow data sets in the form of video, LiDAR, and GPS sensors is growing. Such advancement leads to more data-driven traffic flow reconstruction. Reconstructing traffic flows from Spatio-temporal data from sensors (Li, Wolinski, & Lin, 2017), generating synthetic data from limited trajectory samples (Chao, Deng, Ren, Ye, & Jin, 2017), and using learning methods to learn behavioural patterns in traffic monitoring data sets (Bi, Mao, Wang, & Deng, 2019).

2.2.1 Macroscopic methods

Macroscopic methods, also known as continuum methods, do not consider details of every individual vehicle, instead describe vehicles' interaction and their behaviours in an overall view. The purpose of this model is to study traffic in large-scale road networks. The primary results of these models are aggregated data such as flow density. Macroscopic models are not efficient in simulating street-level traffic because they are limited to highway networks and cannot model lane-changing behaviours of a vehicle which mostly happens on streets (Chao et al., 2020). Lighthill, Whitham (Lighthill & Whitham, 1955), and Richard (Richards, 1956) developed one of the earliest macroscopic models which are called the LWR model. According to this model, the traffic flow rate solely

depends on traffic density, which represents the relation between flow and density.

2.2.2 Microscopic methods

Microscopic traffic models may mimic traffic in both continuous lanes and junctions by simulating specific vehicle behaviours. The cost of computing is typically the bottleneck, particularly when a large-scale simulation is required (Chao et al., 2020). By simulating a subject vehicle's reactions to its front vehicle, several modifications and expansions of the car-following model have been created throughout the years. Optimal Velocity Model (OVM) (Bando, Hasebe, Nakayama, Shibata, & Sugiyama, 1995) and intelligent driving model (IDM) (Treiber & Helbing, 2001). The subject vehicle is supposed to keep moving at its ideal speed under the OVM concept. The difference between its velocity and the front vehicle's ideal velocity determines how quickly it accelerates. According to the vehicle's present speed, relative speed, and position to the vehicle in front of it, the IDM model calculates the acceleration or deceleration of the vehicle. The IDM model can replicate a variety of vehicle kinds and driving styles thanks to vehicle-specific characteristics.

2.2.3 Mesoscopic methods

Between macroscopic and microscopic techniques, mesoscopic models represent a middle ground. The main goal of mesoscopic models is to aggregately characterize traffic flow dynamics and depict individual driver behaviours using probability distribution functions. Mesoscopic models have three categories: cluster models, headway distribution models and gas-kinetic models. The most well-known models among mesoscopic techniques are gas-kinetic models, which compare traffic dynamics to gas dynamics (Hoogendoorn & Bovy, 2001).

2.3 Traffic Simulation and Visualization Tools

2.3.1 SUMO

In 2001, the German Aerospace Center (DLR) began work on the open-source traffic simulation program SUMO. Since then, SUMO has grown into a full-featured suite of traffic modelling utilities that includes a road network that can read a variety of source formats, demand generation and

routing utilities from a variety of input sources (origin-destination matrices, traffic counts, etc.), and a high-performance simulation that can be used for single junctions or entire cities, as well as a ”remote control” interface (TraCI) that allows the simulation to be adjusted while running (Behrisch, Bieker, Erdmann, & Krajzewicz, 2011). SUMO is a microscopic traffic simulation so that each vehicle is specified clearly, including at a minimum an identification (name), a departure time, and the vehicle's network path. For large-scale traffic studies, it is possible to provide the traffic flow definitions using origin/destination matrices (O/D matrices). These matrices define the number of vehicles during the defined time for a specific region on the map. Maps could be imported to SUMO from a variety of sources such as:

(1) Open Street Map (OSM)

(2) Manually Created Maps using Netedit tool

(3) VISUM

(4) Vissim

(5) OpenDRIVE

SUMO supports geo-coordinates and it is possible to convert road networks to their equivalent GPS points. This feature could be very useful in simulating real-world scenarios. Besides the network flexibility of SUMO, researchers can generate traffic flows in several methods. A few traffic generation methods are as follows:

(1) OD-matrices

(2) Turn count and probabilities

(3) Activity based on demographic data

(4) Random traffic

SUMO is capable of executing the simulation scenario either with or without visualization. For visualization, it offers a graphical user interface (GUI) that can illustrate the simulation steps and events on the simulation network. The SUMO GUI is not a web-based application and only runs on

the local computer operating system. So, users can only access the GUI and the simulation features by using the machine on which the simulation is running. This could be one of the disadvantages of SUMO because it limits flexibility and performance.

2.3.2 CityFlow

H. Zhang et. al proposed and implemented CityFlow which is a microscopic traffic simulator. it can simulate the behaviour of each vehicle at each time step (Zhang et al., 2019). CityFlow uses multi-threading, a new data structure, and a new simulation algorithm to optimize and speed up the engine. CityFlow is primarily designed to support multi-agent reinforcement learning. It has a python interface to interact with the simulator and get or put data into the simulation. Although CityFlow is way faster than SUMO in large-scale scenarios, its feature set is not as complete as SUMO. Maps cannot be imported from sources such as OSM, traffic is not definable by direct OD matrices, and many more limitations in comparison to SUMO. Regardless of its limited feature set, it provides a web-based Graphical User Interface which makes it more convenient to view large-scale networks. The authors stated that the current CityFlow could be used for traffic signal control problems.

2.4 Synthetic traffic data generation approaches in the literature

With the presence of smart devices that are equipped with global-positioning (GPS) functionality and internet connectivity, there are huge collections of mobility data, including individuals and vehicles. This data could be used to train the prediction models for city planning and management. However, due to privacy implications, it is not possible to share such datasets with third parties. Furthermore, because of issues such as data breaching and security vulnerabilities, these datasets could not be used for development and research purposes (Kulkarni & Garbinato, 2017). As a result, one of the primary obstacles to Intelligent Transportation Systems (ITS) is the availability of reliable data for training the models. By using larger and more accurate data, the accuracy of forecasting models could be improved. Due to the lack of data, generating synthetic data could be a point of interest for many researchers.

Hermoupolis is a data generator proposed by Pelekis et. al. This data generator is able to produce synthetic datasets along with their spatiotemporal features. It generates annotated trajectories of moving objects that follow predefined mobility profiles, as well as network-constrained simulated GPS-like recordings. Hermoupolis input consists of a road network, a set of points on the network, and a set of mobility profiles (Pelekis, Sideridis, Tampakis, & Theodoridis, 2015). The proposed method cannot generate synthetic trips and needs user-defined data to generate their synthetic trajectories.

To generate synthetic traffic data, Sapre et. al used a minimal dataset of GPS traces and a map of the city from OpenStreetMap (OSM). They manually extracted trajectories from OSM and calculated the percentage of trips between every two districts to the total number of trips and used the results to scale the traffic demand by randomly generating the origin-destination pairs. The output of this work is a synthetic dataset of randomly generated origin and destination pairs. In this work, the authors validated their model by combining image processing based on MATLAB libraries and Google Map Traffic Layer data. For exporting the GPS traces they used the JOSM tool to export specific area data. Because the GPS traces on OSM is not well structured they had to clean data. The trace dump is a list that is made up of individual travels. The arrangement of the trace points is kept the same during each journey. The list, on the other hand, does not distinguish between different excursions. The authors mention that they had a lack of high-quality trace data. (Sapre, Kalambur, Sitaram, & Bastian, 2018). The proposed model solely relies on the random trip generation and cannot estimate the pattern of traffic.

Xie et al. introduce SMARTS as a traffic simulation software that generates a dataset of three different data types, namely, trajectories, travel times, and route plans. The input of SMARTS is the user-defined trip plans. For the background traffic demand, SMARTS is not using any base realistic dataset as input, therefore its methodology is based on random origin-destination generation (Xie et al., 2019). In this work, authors rely on user-defined trips and cannot generate realistic traffic on the map.

Simulation tools such as SUMO support several types of input to generate traffic demand. Among these tools available in the SUMO simulation package, DFROUTER is designed to generate traffic based on vehicle counter sensors. J. Zambrano et. al used DFROUTER and *Induction Loop*

Sensors data available in Spain, Valencia (Zambrano, Calafate, Soler, Cano, & Manzoni, 2016). J. Zambrano et al. used the traffic sensors datasets of Valencia, Spain as a real data source of traffic. They used SUMO traffic simulation available tools in combination with proposed heuristics to generate origin-destination matrices from the input sensors data. The authors validated the result with other cities to determine the distribution level of vehicles in cities. The proposed methodology in this work is highly dependent on the traffic sensors data (Zambrano et al., 2016). These data are not publicly available and are not scalable.

H. Song et al. proposed STDG as a method for estimating and generating traffic based on machine-learning L2 Logistic Regression models. The input of this study is a realistic trajectory dataset of Seol, Korea. The authors use logistic regression and generate a synthetic dataset of origin-destination pairs as the output of the study. Also, in the next step, SUMO is used to route the vehicles (Song & Min, 2018). The proposed model, cannot generate synthetic trajectories and only has estimated trips for the purpose of simulation.

L. Codeca et al. proposed a complete traffic scenario for Luxembourg city based on realistic demographic data. They analyzed the input demographic data, extracted map data from OpenStreetMap then used SUMO traffic simulation software to generate origin-destination pairs (trips) based on population. After generating trips, SUMO helps to route the vehicles toward the destination and calculate the routing algorithms. The output of this study is a complete traffic demand scenario for 24 hours exclusively for the city of Luxembourg (Codecá, Erdmann, Cahill, & Harri, 2020). The authors in this study did not generate any synthetic trajectory and only build the trips for 24 hours for Luxembourg. Synthetic trajectories are missing from this work.

Another approach for generating synthetic traffic data is conducted by V. Kulkarni and B. Gabrinato (Kulkarni & Garbinato, 2017). In this study, the authors apply machine learning to learn the patterns of traffic from a real dataset and use these patterns to generate a new and larger dataset. Their proposed model is based on RNN (Recurrent Neural Networks) for extracting the behavioural patterns of users. By designing the network model to learn complex sequences and extending it to make predictions in the spatiotemporal domain, the system architecture is based on Long Short-term Memory (LSTM) recurrent neural networks (RNN).

Deep neural networks have a lot of potentials when it comes to traffic prediction. Traffic data

Table 2.1: Summary of existing works on urban traffic generation and simulation

Input	Output	Methodology	Reference
user-defined points	synthetic trajectories	simulating object movement	(Pelekis et al., 2015)
Real Trajectories	Scaled Synthetic Trajectories	Random OD based on flow percentage	(Sapre et al., 2018)
user-defined trips	trajectories and travel times	distributed simulation and object tracking	(Xie et al., 2019)
Traffic Sensors Data	OD Matrix	SUMO DFRouter	(Zambrano et al., 2016)
Population Statistics	OD pairs for 24 hours	SUMO	(Codeca et al., 2015)
Real trajectories	Synthetic OD pairs	ML Logistic Regression	(Song & Min, 2018)
Real Trajectories	Synthetic Trajectories	RNN – LSTM	(Kulkarni & Garbinato, 2017)
Parallel Real and Synthetic Data	Large Synthetic Trajectories	GANs	(Y. Chen et al., 2018)

must be enormous and diversified to construct high-accuracy traffic prediction models because collecting large and precise data is exceedingly expensive, if not impossible (Y. Chen, Lv, Wang, & Wang, 2018). Therefore, Chen et. al chose to train models using simulated data in parallel with real data. They used GANs (Generative Adversarial Networks) to generate traffic data. Chen et. al proposed improved GANs to generate traffic data. They stated that the traffic flow series is a time-serial dependency, so they cannot directly apply GANs to these series. In this proposed model they fed real data into the generator and synthetic data. However, no simulation of traffic is implemented in this study and trajectories are only based on GAN models.

2.5 Summary

According to the literature and the best of our knowledge for traffic analysis, available data is very limited and accessing accurate traffic data would violate privacy. To forecast traffic or demand, models need to be trained with a large amount of traffic data to have better accuracy. In our proposed synthetic data generator, we use a real trajectories dataset as the input and train a machine learning model to predict the demand for a specific time period. The machine learning predictions help us

to generate corresponding demand using SUMO. The proposed model not only generates the synthetic trajectories but also can provide travel times of user-defined vehicles with the synthetic traffic as background. It can provide a platform for mobility-on-demand studies. Also, our simulation platform provides APIs for MOD applications to import trip plans, apply on-demand changes to the simulation, and export the trajectories and logs after a full round of simulation.

Chapter 3

Synthetic MOD data generation framework

Our proposed simulation platform uses real demand data published by city authorities, trains a machine learning model for learning traffic patterns and predicts vehicle demand. Based on the trained model trips on the selected region are generated. Trips generated based on realistic data are considered background traffic of the region. The second section of the proposed framework is the configuration of user-defined service trips and the simulation of both background traffic and service vehicles. Our proposed framework monitors MOD service trips and collects real-time trajectories as synthetic MOD data. The collected synthetic data are stored in a database which makes it accessible for further MOD analysis purposes.

3.1 Background data generation

3.1.1 Data preprocessing

In order to train machine learning models in our proposed framework, we utilized the Montreal urban trip data of 2017 published by Montreal authorities (*Montreal Trajectories*, 2018). The reason for using this data is that every trip that is collected in this dataset, includes date and time, departure, and destination locations. Also, because the locations that are provided in this data set, are based on GPS points, it enables us to apply data processing methods based on GPS locations. For instance,

we can filter data based on a specific region on the map by defining a polygon. The trips dataset consists of 185285 trips during a one-month period from 00:00 on September 18th, 2017 until 23:59 on October 18th, 2017. We focused on the region of Downtown Montreal, and by defining a polygon on this region, could filter the trips data set. We filtered trips that either start or end within the Downtown Montreal region. After filtering trips, three types of output are demonstrated. Table 3.1 shows the number of observed trips in three categories. The first category includes trips that their start and end locations within the Downtown region. The second category has trips that start from downtown Montreal, but their destination is not within this region. The last category only includes trips that depart outside of downtown Montreal, but their destination is located within the region of downtown Montreal.

Table 3.1: Trips within Downtown Montreal

Type	**Count**
Departure and destination within the region	17868
Only departure within the region	14353
Only destination is within the region	14436
Total Trips	**46657**

For simulation purposes, we need to have a pair of departure and destination to be able to simulate the vehicle movement using SUMO software. In this study, a pair of departure and destination is referred to *Trip*. To solve the issue of the trips that their departure or destination is not within the selected region, we determined the primary roads on the boundaries of Downtown Montreal and randomly set the missing departure or destination to these specific points on the boundaries. As a result, our filtered data consists of trips that are within Downtown Montreal and all trips include departure and destination GPS locations. The procedure is described in Algorithm 1.

After running the Algorithm 1 on our data set, the total number of trips within the region is 46657. All of the trips in this data set have departures and destinations within the study region P. Table 3.2 demonstrates 5 records of data set.

In the next stage of preprocessing data, we created the charts of trip distribution in terms of distribution per day of the week and distribution per time of the day. For the time of the day, we divided the day into 24 frames of a 1-hour period and calculated the trip distribution. Figure 3.1 is

Algorithm 1 FIXING TRIPS WITH DEPARTURE OR DESTINATION OUTSIDE REGION

Require: $\mathcal{M}$, $\mathcal{L}$, $\mathcal{P}$

for each $l_j \in \mathcal{M}$ **do**
  $o_j \leftarrow$ departure point from l_j
  $w_j \leftarrow$ destination point from l_j
  if $o_j \notin \mathcal{P}$ **then**
    $o_j \leftarrow$ random point from $\mathcal{L}$
  end if
  if $w_j \notin \mathcal{P}$ **then**
    $w_j \leftarrow$ random point from $\mathcal{L}$
  end if
end for

Ensure: All departures and destinations are within the region

Table 3.2: Sample trips within the region

StartTime	Departure	Destination
2017-09-18 00:16:58	POINT(-73.651166 45.544501)	POINT(-73.654753 45.545521)
2017-09-18 02:17:46	POINT(-73.650987 45.544496)	POINT(-73.654753 45.545521)
2017-09-18 05:30:24	POINT(-73.832869 45.636068)	POINT(-73.624171 45.530941)
2017-09-18 06:02:50	POINT(-73.646385 45.54629)	POINT(-73.545318 45.550661)
2017-09-18 06:18:40	POINT(-73.745495 45.55839)	POINT(-73.720594 45.559501)

a demonstration of processed real data for Downtown Montreal.

3.1.2 Demand prediction

In DDSTG, the travel demand in each region within a time window is predicted using a machine learning algorithm based on the travel demand time-series data. At a time point t_n, DDSTG needs to predict the number of trips within the region. In order to accomplish this, time series forecasting models may be utilized to estimate future demand based on previously observed demand. For predicting a single demand value in the cases of time-varying data, the Poisson model (Y. Wang et al., 2019) and the auto-regressive model (Moreira-Matias, Gama, Ferreira, Mendes-Moreira, & Damas, 2013) could be utilized. In spite of the fact that several forecasting models may be put into our architecture, we will not focus on the model selection because it is data-dependent. One might pick a forecasting model depending on the specific attributes of their demand data. Readers can refer to (Zha, Yin, & Du, 2017) for a review of model selection in the context of urban mobility. In

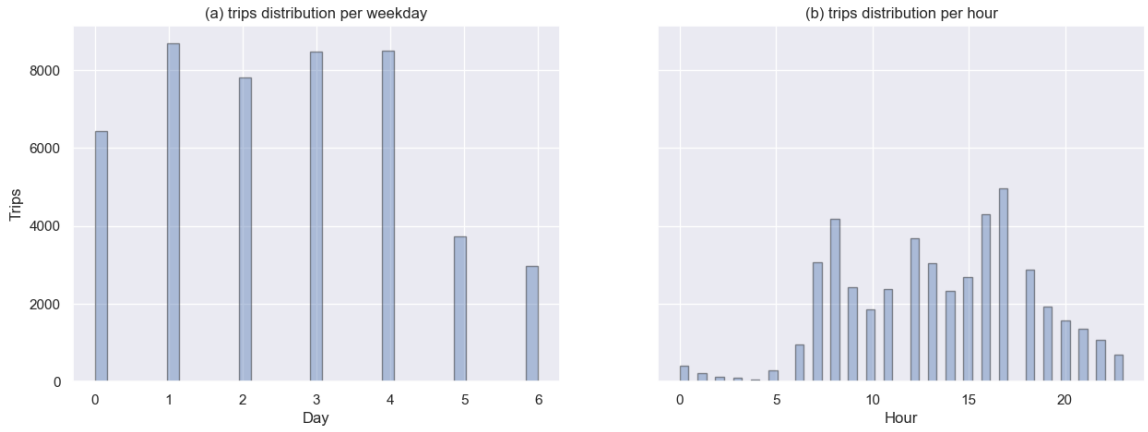

Figure 3.1: Trips distribution in Downtown Montreal. (a) Distribution of trips on each day of the week, starting from Monday = 0. (b) Distribution of trips in each hour of the day for 24 hours.

DDSTG, We use the ARIMA model which is widely used in transportation demand prediction.

We considered that a day is divided into a set of time windows with an equal and fixed size of ΔT. Each time window is denoted as $\{T_1, T_2, ..., T_n, ...\}$, where $T_n = [t_n, t_n + \Delta T]$ refers to the nth time window. Let $\mathcal{J}$ denote the set of trips in a time window T_n. The trip $j \in \mathcal{J}$ is a four-tuple (ID_j, o_j, w_j, t_j), where ID_j is the trip identification number, o_j and w_j are the departure and destination locations. t_j is the time when the vehicle starts moving toward its destination, $t_n \leq t_j \leq t_n + \Delta T$. Let T_n be a time window in a day, $\hat{r}^{T_n}$ is the predicted number of vehicles within the time window T_n. Table 3.3 shows the list of notations used in this study. We chose a set of primary locations on the boundaries of the region, denoted as $\mathcal{M}$. Each point $m \in \mathcal{M}$ is located on the boundaries of the study region. Also, the traffic region in this study is denoted as $\mathcal{P}$.

After preparing the data set of trips, which is described in 3.1.1, the ARIMA model is trained to learn the data patterns from the real data set. At a given time t_n, DDSTG predicts the number of vehicles for $T_n = t_n + \Delta T$ using the ARIMA machine learning model. The predicted demand is denoted as $\hat{r}^{T_n}$. For training the ARIMA model, we divided our data set to train and test data sets. The ARIMA model is trained based on the trip data from September 17 until October 10. The rest of the data set is dedicated to testing purposes. Figure 3.2 illustrates the division of data into train and test data sets.

Table 3.3: Description of notations

Notation	Description
T_n	A time window
t_n	The start time of time window
ΔT	The size of time window
$\mathcal{M}$	A set of points on the boundaries of the region, indexed by m
$\mathcal{J}$	A set of trips within the region $\mathcal{P}$, indexed by j
$\lvert\mathcal{J}\rvert$	Total number of trips in $\mathcal{J}$
$\mathcal{L}$	A set of trips with either departure or destination outside of region $\mathcal{P}$
$\lvert\mathcal{L}\rvert$	Total number of trips in $\mathcal{L}$
$\mathcal{P}$	A polygon defining the study region
o_j, w_j	Departure and destination location of trip
$\hat{r}^{T_n}$	Predicted number of trips within the time window

By using the ARIMA diagnostics tools we derived the training details plots. The plots demonstrate how the model is trained against the input data set. The plots are as follows:

- **Standardized residual plot:** This plot is used to check for the presence of any patterns or outliers in the residuals, which could indicate a lack of fit or a violation of the assumptions of the model.
- **Histogram plus estimated density:** This plot is used to check if the residuals are approximately normally distributed, which is an assumption of many time series models.
- **Normal Q-Q:** This plot is a Q-Q plot (Quantile-Quantile plot) of the residuals against a normal distribution. This plot is used to check if the residuals are approximately normally distributed, which is an assumption of many time series models.
- **Correlogram:** This plot is also known as the autocorrelation plot, it shows the correlation between the residuals and lags of the residuals. This plot is used to check for autocorrelation in the residuals, which could indicate that the model is not able to capture all the dependencies in the data.

Figure 3.3 illustrates the diagnostics plots of our trained ARIMA model.

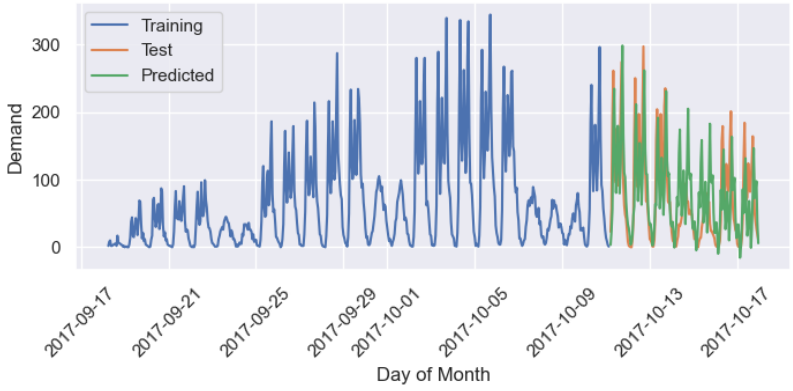

Figure 3.2: Train, test, and prediction demand data

Fig. 3.4 shows the comparison results in terms of the average value of demand over real Downtown Montreal data and predictions based on historical demand information. A 24-hour day is divided into 24 consecutive time frames. The demand value for each time window is averaged over 28 weekdays of one month.

Since the predicted demand follows the real traffic pattern based on our real trips data set, we can use the values of predicted demands, to generate reasonable trips as background traffic for our framework. In the next step, we describe how background trips are generated based on predicted demands.

3.1.3 Generate trips based on predicted demand

Every trip consists of a departure and a destination. In our proposed framework, we use the predicted demand value, and the real data set of trips, to create a collection of trips for simulation purposes. Therefore, we avoid randomized departures and destinations and we follow real data patterns. For the simulation of traffic, we utilize the trips data set $\mathcal{J}^{T_n}$ and the predicted demand $\hat{r}^{T_n}$. Given a set of trips $\mathcal{J}^{T_n}$, predicted demand value $\hat{r}^{T_n}$, and time window T_n, DDSTG picks trips from $\mathcal{J}^{T_n}$ so that the number of chosen trips is equal to $\hat{r}^{T_n}$. If $\hat{r}^{T_n}$ is less than or equal to the number of total available trips in $\mathcal{J}^{T_n}$, DDSTG can select the trips and utilize them for simulation. However, if predicted demand $\hat{r}^{T_n}$ is greater than $\mathcal{J}^{T_n}$, then the algorithm picks all available trips in $\mathcal{J}^{T_n}$, and selects the rest of trips from $\mathcal{L}$. As stated in Table 3.3, $\mathcal{L}$ is a set of trips that are

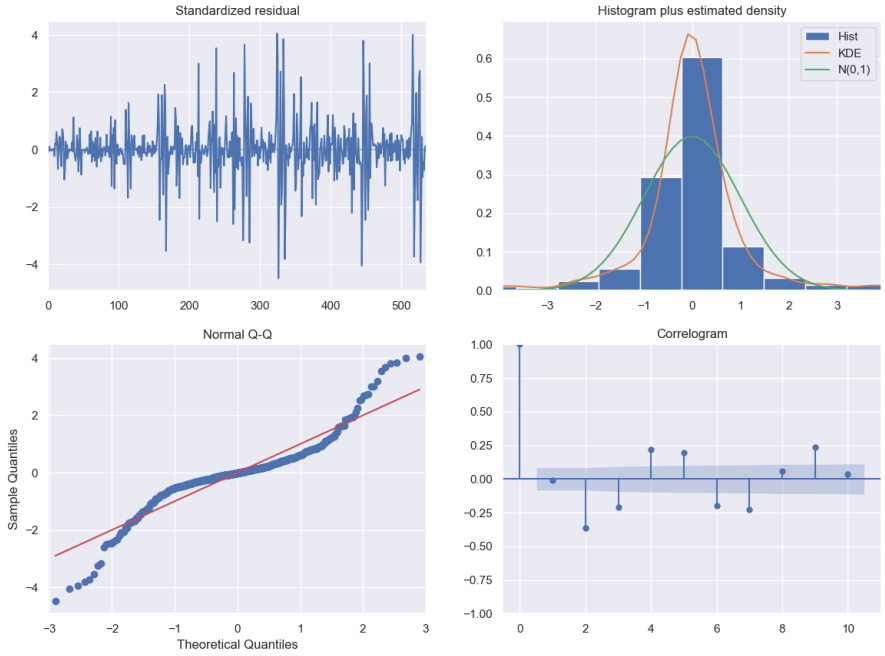

Figure 3.3: ARIMA diagnostic plots

not completed within the region. However, it is important to consider that, these trips have either departure or destination on the boundaries of the study region. This is because of preventing to use of trips that their departure or destination is not within the region. Algorithm 2 shows how we picked trips to generate the background traffic for the framework.

Algorithm 2 SELECTING TRIPS FROM AVAILABLE TRIPS DATA SET

Require: $T_n = [t_n, t_n + \Delta T]; \forall j \in \mathcal{J}^{T_n}; \forall l \in \mathcal{L}^{T_n};$

1: **while** $t = t_n$ **do**
2: Predict the number of trips $\hat{r}^{T_n}$ in the time window T_n
3: Based on the predicted demand, select trips from $\mathcal{J}^{T_n}$
4: **if** $|\mathcal{J}^{T_n}| \leq \hat{r}^{T_n}$ **then**
5: Select the rest of trips from $\mathcal{L}^{T_n}$
6: **end if**
7: **end while**
8: Send selected trips to SUMO for simulation

Ensure: Selected trips for simulation follow the real data distribution pattern

By running the Algorithm 2 on preprocessed trips data set, and using the predicted demand $\hat{r}^{T_n}$, we can have a data set of trips that could be used as the background traffic for the simulation.

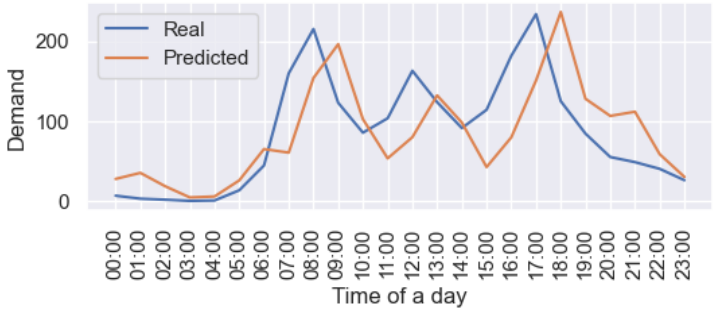

Figure 3.4: Comparison of real demand and predicted demand based on historical trips data of Downtown Montreal

3.2 MOD trip data generation

3.2.1 Service trips configuration

Besides background traffic that is generated based on realistic data and by using machine learning techniques, service trips are user-defined tuples that describe trips from a departure to a destination, starting at a specific time. The service trip tuple is defined as (o_j,w_j,t_j). Let o_j be trip departure and w_j be trip destination, the vehicle j begins moving from departure address toward the destination at time t_j. A list of service trips that is defined by user is considered as one of the inputs of the framework. The proposed framework, can read the input service trips and configure the simulation by following the Algorithm 3.

Algorithm 3 CONFIGURING MOD SERVICE TRIPS

Require: $T_n = [t_n, t_n + \Delta T]; \forall j \in \mathcal{J}^{T_n}$;

while $t = t_n$ **do**
    Validate $o_j \in \mathcal{P}$
    Validate $w_j \in \mathcal{P}$
    Convert o_j , w_j addresses to (x_j, y_j) points on the SUMO network
    if $t = t_j$ **then**
        add a vehicle to simulation map, departing from o_j toward w_j
    end if
    Collect all inserted vehicles GPS locations at each simulation step
end while

Ensure: All service trips movement attributes are stored in database

By loading the service trips side-by-side with background traffic, the service vehicles that are

considered as MOD vehicles, need to adapt their movement behaviors based on the traffic on the road. For instance, in a congested road, service vehicle cannot move with its maximum speed limit and at various circumstances needs to adapt its speed. SUMO controls the speed, routing, and movement behaviors of all vehicles on the roads of the simulation network. Because the proposed framework, utilizes realistic traffic data to generate background traffic, and also takes service trips as input from user, the synthetic MOD data that is driven from the framework has been affected by realistic factors of traffic. In the following sections, simulation is described in more details.

3.2.2 Running SUMO

For simulation purposes, we use SUMO software. SUMO is able to mimic the movement of individual vehicles on the roads of a user-defined map. For a successful traffic simulation, SUMO requires a map and trip data. Then SUMO routes the vehicle from departure to its destination on the map. For a realistic traffic scenario, we need to generate trips that are based on realistic traffic data. Also, the number of vehicles should follow a pattern that demonstrates the real scenario in the simulation region. In section **??**, we explained the details of the approach of extracting and choosing trips from real data.

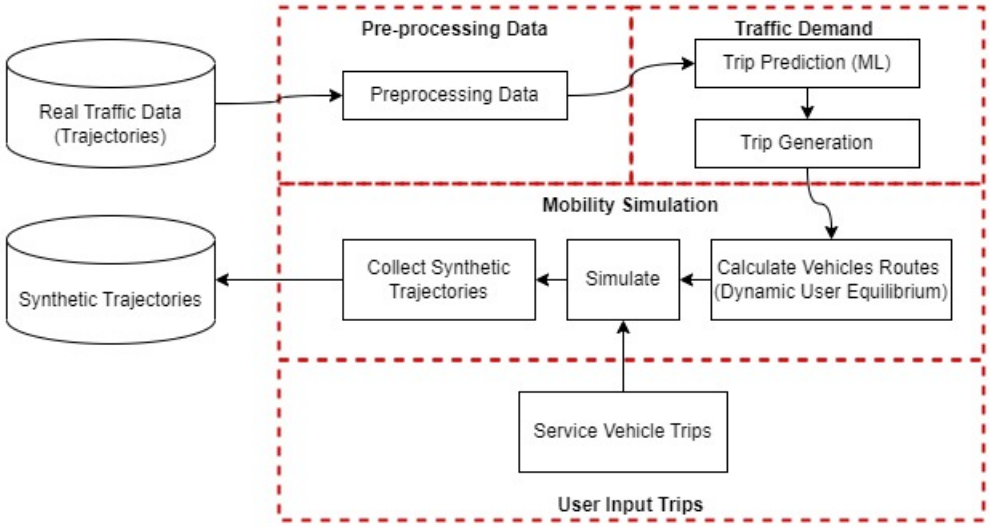

Figure 3.5: Synthetic Data Generating Approach

For a given set of vehicles with origin-destination relationships (trips), the simulation must

calculate the network routes (list of edges) utilized to reach the destination from the origin edge. Using a routing algorithm such as Dijkstra or A* to compute the shortest or quickest paths through the network is the easiest way to locate these routes. These techniques need assumptions about the travel time for each network edge, which is typically unknown before conducting the simulation because travel durations are dependent on the number of cars in the network. User assignment is the problem of selecting suitable routes that account for travel times in a traffic-loaded network (Lopez et al., 2018). SUMO package has various traffic demand generation methods. In this study, we used the Iterative Assignment (Dynamic User Equilibrium) method to generate the routes for the origin and destination pairs. This method tries to calculate a user equilibrium such that the vehicle cannot reduce its travel time by using a different route. This is an iterative process. SUMO chooses routes based on Gawron and Logit algorithms. The Gawron method computes the probability of each driver's selection from a list of possible routes. The Logit mechanism calculates the new probability for each path using a preset formula. It disregards previous costs and probabilities and calculates the route cost as the total of the edge costs from the most recent simulation (Lopez et al., 2018).

3.2.3 Mechanism to collect real-time service vehicle data

By using the generated trips and routes in the previous steps, we can configure the SUMO to execute the simulation with the generated vehicles. While the simulation is executing the scenario, at every step of the simulation SUMO generates information such as vehicle position on the map, collisions, delays, etc. The proposed platform is connected to the simulation engine and collects these data on each simulation step and then sends these data to the data layer. The data layer of the framework stores vehicle GPS location in a database. After storing the data in the database we can have access to vehicles' geo-coordinates at each timestamp. This collection could be referred to as synthetic trajectories. Table 3.4 illustrates a sample of this data. Figure 3.6 shows the overview of the mechanism for collecting real-time data from the simulation.

Based on the synthetic demand on the map and the imported service vehicle plans, we can produce other valuable data such as travel time for each vehicle. This information can help MOD applications simulate the vehicle travel plan based on the time of the day and see the travel time for

Table 3.4: Collected Data Sample

Veh. Id	timestamp	Longitude	Latitude
sv_18#1	41198	-73.56128192818187	45.49122707846640
sv_18#1	41197	-73.56121477272544	45.49117204969210
sv_18#1	41196	-73.56114474108236	45.49111466394739
sv_18#1	41195	-73.56107819541623	45.49106013455619
sv_18#1	41194	-73.56100783730017	45.49100248098101

such a plan.

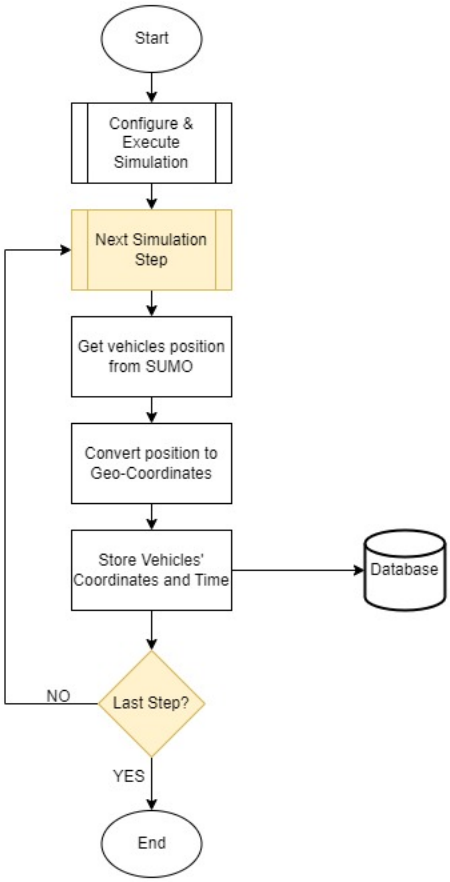

Figure 3.6: Real-time data collection algorithm flowchart

Chapter 4

System Requirements and High-level Design

This chapter describes the requirements of the system as well as a high-level design of the system features. After collecting the system requirements through surveys and meetings with the MOD researchers in the team, I found out that the system requires 8 primary use cases and APIs. Each use case is described using the standard use case description method. Then, in the design section, I explained the overview of the system design, including the primary components, followed by the APIs access requirements.

4.0.1 Requirements Collection and Methods

The formulation of system requirements may be considered to be the phase of systems design that is most important. For the reasons listed below, it is crucial. First and foremost, it is crucial for both the technical designers and the users of the new system to be clear on what they want the system to achieve. Only when the system requirements are made explicit The technical experts can match current systems and software to a set of desired outputs. Similar to this, users won't have a system in place to enable them precisely meet their expectations unless system needs are well specified (Mumford, 1985). In order to identify and collect the most significant requirements, we need to choose a proper method for requirements collection. Barata et. al studied the available

methods of requirements collection in the software industry and they concluded that interviews with stakeholders are the most efficient method among the studied group (Barata, Lisboa, Bastos, & Neto, 2022). In order to collect the requirements of the system we provided a list of questions and team members answered these questions during the meetings:

(1) What is your MOD application? Ride-sharing, ride-hailing, crowd-shipping, etc.

(2) What kind of data do you need for your research?

(3) If you could have a simulation and visualization platform for your research, what features would you like to have?

4.1 System Requirements

After collecting the responses and analyzing the discussions in the meetings, it seems that traffic data is not easily accessible to the teams. On the other hand, the available data might not meet the requirements of particular teams and it would limit the experiments and results of studies. One of the important issues that are noticeable is that researchers mostly use the traffic data of other countries and cities. Such data might not reflect the driving behaviours of other cities, which could affect the research results. We can conclude that we need to resolve the issue of lacking data. Furthermore, if we can provide a platform besides the flexible data, which allows researchers to change the traffic attributes and data using a visualization solution, it could be more beneficial for future studies. Therefore, in this thesis, the proposed system provides 9 primary functions. These functions could be mapped to the system requirements and they can assist the researchers to facilitate their studies. The functions are categorized as simulation and visualization. Figure 4.1 illustrates a conceptual model of primary requirements.

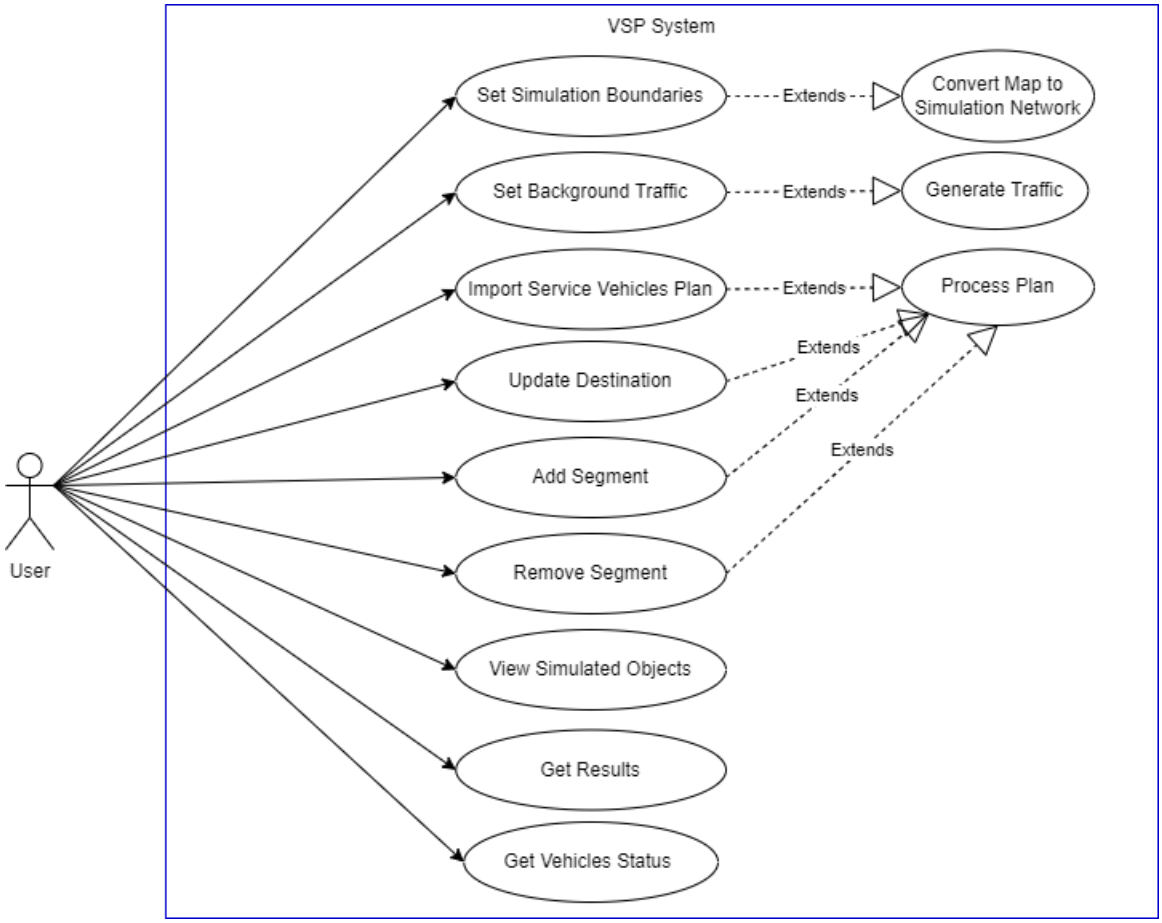

Figure 4.1: System Requirements Overview - Use Case Diagram

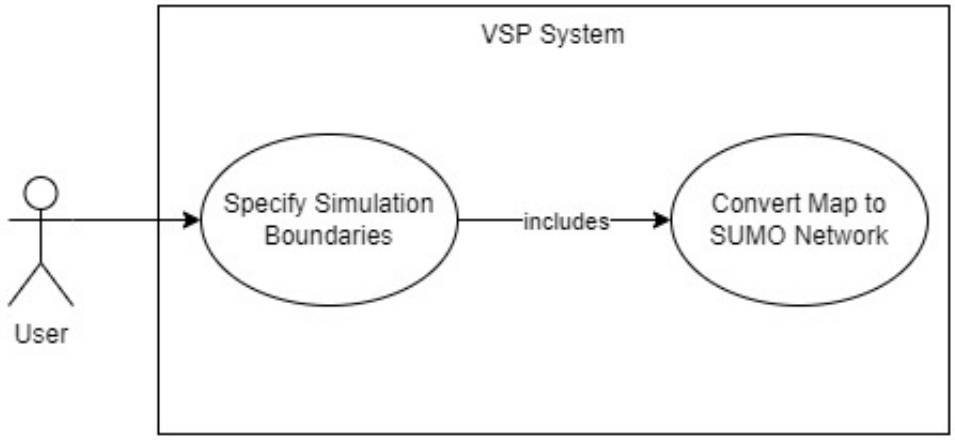

Figure 4.2: Simulation Boundaries Selection Use Case Diagram

4.2 Simulation Use Cases

4.2.1 Simulation Boundaries Selection

The simulation system should be able to execute the simulation on real-world maps in order to have more realistic results. The simulation should execute the scenarios on predefined areas of the map because the boundaries cannot be unlimited, due to processor limitations. To address such requirements, the system should be flexible enough to get the user-defined boundaries and set up the simulation area. The proposed system takes the boundaries as an array of GPS points, then uses this array to set up the simulation boundaries on the real-world maps. Figure 4.2 illustrates this use case followed by the use case description in Table 4.1.

4.2.2 Background Traffic Setup

Once the simulation starts, it can show the moving vehicles on the map. However, to achieve more realistic results from the simulation, we would like to have realistic traffic on the simulation network. In this case, while the simulation is running, based on the time of the day in the simulation, the traffic density will not be constant. Also, it is important to consider that the traffic density in a city varies in different neighbourhoods. In this use case, the actor can provide basic demographic

Table 4.1: Use Case Description for Simulation Boundaries Selection

Use-Case ID	1	
Use-Case Name	simulation-boundaries-selection	
Actors	Any MOD planning application can call this API.	
Description	API takes an array of GPS points tuples [(lat1,lon1),(lat2,lon2),. . .]. Then, the system exports selected boundaries and generates the simulation network by converting the map to the simulation network.	
Preconditions	Selected boundaries should meet system limitations. It should cover a valid urban area. System should not accept boundaries on the sea or poles.	
Postconditions	The selected boundary should be set as the simulation network, and the SUMO network file should exist in system paths to be used by SUMO.	
	Actor Action	System Response
Normal Flow	1.0. the user provides an array of tuples. Each tuple is a GPS point (lat,lon).	• Exports a map from a map source such as OpenStreetMap within the selected boundaries • Converts the map to simulation network (.net.xml) • Saves the network file in the simulation system path to be used by SUMO
	Actor Action	System Response
Alternative Flow	1.1. The user provides a region name such as city name	• Exports a map from a map source such as OpenStreetMap within the selected region • Converts the map to simulation network (.net.xml) • Saves the network file in the simulation system path to be used by SUMO
	Actor Action	System Response
Exceptions Flow	1.0.E.1 User inputs are not GPS point tuples or are less than 4 points	• Prompts user that inputs are invalid • Logs error
	1.0.E.2 Conversion Failed	Prompts the user for failure reason and logs errors
Special Requirements	Should be already connected to OpenStreetMap.	

information about the selected boundaries and the platform generates the background traffic. Table 4.2 demonstrates the use case description.

4.2.3 Import Service Vehicles Plan

The focus of the proposed simulation and visualization platform is to provide software for researchers to be able to test and monitor their traffic decisions in a virtual traffic environment. To meet this requirement, the system should be able to get a plan of vehicles including their departure, destination, and probable stop points, and simulate their movements on the map. These vehicles are referred to as **Service Vehicles**. Because these vehicles are moving through a network that already contains realistic traffic, their travel time and travel distance will be variable based on the routing decisions that happen during the simulation. Table 4.3 shows the use case description and the expected flows.

4.2.4 Update Destination

In real-world scenarios, it is possible that a vehicle changes its destination while has departed. Our proposed platform considers this scenario and it is possible to apply changes to vehicle plans while the simulation is running. So, there is no need to stop the simulation and import a new plan to see the results. Table 4.4 shows the use case description of this capability of the system.

4.2.5 Add Segments to Plan

While the simulation is running and the plan has been imported into the system, the simulation keeps reading and executing the plan. At some point in time, the planner might need to add a new stop point for the vehicle. In this case, this use case is designed to accept the input from the actor and applied the changes to the running simulation. Table 4.5 illustrates this use case and its descriptions.

4.2.6 Remove Segment

In our proposed platform being flexible to changes is one of the objectives. Segments are the stop points between the departure and destination of the vehicle. The actor can add and remove these

Table 4.2: Use Case Description for Background Traffic Setup

Use-Case ID	2	
Use-Case Name	setup-traffic	
Actors	Any MOD planning application can call this API.	
Description	API takes input data such as demographic information and generates background traffic on the map.	
Preconditions	Selected boundaries should meet system limitations. It should cover a valid urban area. System should not accept boundaries on the sea or poles.	
Postconditions	The selected boundary should be set as the simulation network, and the SUMO network file should exist in system paths to be used by SUMO.	
	Actor Action	System Response
Normal Flow	1.0. the user provides an array of tuples. Each tuple is a GPS point (lat,lon).	• Exports a map from a map source such as OpenStreetMap within the selected boundaries • Converts the map to simulation network (.net.xml) • Saves the network file in the simulation system path to be used by SUMO
	Actor Action	System Response
Alternative Flow	1.1. User provides a region name such as city name	• Exports a map from a map source such as OpenStreetMap within the selected region • Converts the map to simulation network (.net.xml) • Saves the network file in the simulation system path to be used by SUMO
	Actor Action	System Response
Exceptions Flow	1.0.E.1 User inputs are not GPS point tuples or are less than 4 points	• Prompts user that inputs are invalid • Logs error
	1.0.E.2 Conversion Failed	Prompts the user for failure reason and logs errors
Special Requirements	Should be already connected to OpenStreetMap.	

stop points and the system should adapt accordingly, meaning that if the vehicle has not reached the stop point, it should change its next destination to the next stop point in its list. Table 4.6 describes this use case in more details.

4.2.7 Get Simulation Results

The simulation platform should provide a result of the vehicles and the GPS points that they passed from. The objective is to provide a database of GPS tracking points, including the stop points, total travel time, reroutes, delays, and many more properties of each vehicle on the map. This data could be one of the most useful data for researchers of Mobility on Demand subjects. Because they can test their algorithms and scenarios on the proposed platform and see more realistic results based on the realistic traffic on the map. Table 4.7 is the use case description of this feature.

4.2.8 Get Service Vehicles

The proposed platform is considered to be able to connect to other third-party applications developed by the researchers. Using its available APIs, other applications can call and retrieve the vehicles' data on the simulation map. So, if they need to apply any changes to a specific service vehicle, they can access the vehicles' identifiers and their last status on the map. Table 4.8 demonstrates this use case description.

4.3 Visualization Use Cases

4.3.1 View Simulation

One of the important features of the simulation engines is to be capable of presenting the moving objects on the map. Our proposed simulation and visualization platform is based on the SUMO engine and uses its engine to show the moving objects on the map. Table 4.9 describes more details about this use case.

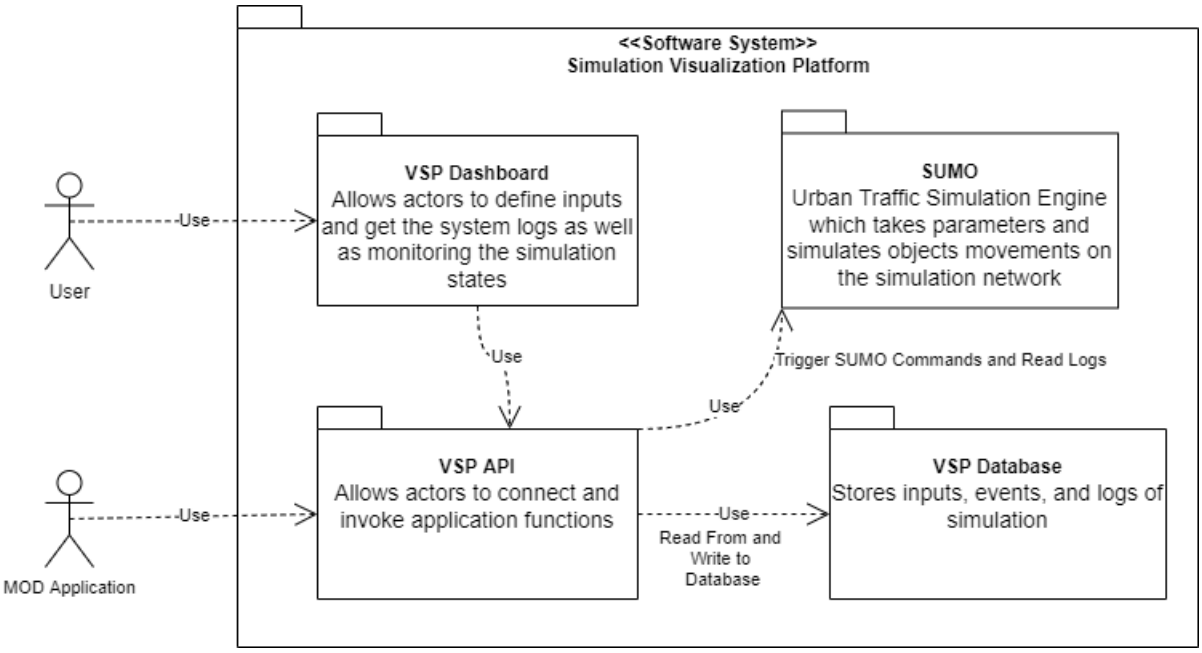

Figure 4.3: Overall VSP Structure Design

4.4 Overall Structure Design

The VSP system gets inputs from the users using a dashboard. The dashboard passes the user inputs to the system using the available APIs, then after processing the inputs and generating the required configuration files for the SUMO, the VSP API starts the SUMO simulation core to execute the simulation scenario. While the simulation is running, the VSP gets outputs from the SUMO and logs to the database. As a result, to achieve these tasks, the VSP system needs to have at least 4 primary components, including the VSP dashboard, VSP API, VSP Database, and SUMO. Figure 4.3 illustrates the design structure and their communications.

4.4.1 VSP Dashboard

The VSP involves a user interface that enables users to define the simulation inputs such as boundaries, traffic information, and service vehicles plan. The VSP dashboard connects to the application modules and transfers data between the internal layers and the user.

4.4.2 VSP API

This component is a middle layer between the actors and SUMO. In this component, the application's inputs will be validated and processed. Furthermore, this component does the conversions and processing of data to make it readable by the SUMO or make the SUMO results readable by the actor.

4.4.3 VSP Database

The proposed VSP application stores user inputs, events, logs, and the results of the simulation in a relational database. So, even after the simulation is terminated, the results and logs are available for the actors. Furthermore, the GPS coordinates, vehicles' status, and travel details of vehicles are stored in the database and can be retrieved by the system.

4.4.4 SUMO

The VSP application is based on the SUMO simulation engine. The SUMO component takes the inputs and executes the simulation. While the simulation is running, the engine logs events and passes the logs to the VSP API component. Also, the SUMO component takes the commands from the VSP APIs and adapts the simulation accordingly.

4.5 APIs Access Summary

APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols. API stands for Application Programming Interface and it allows applications to communicate regardless of their programming language and technology. There are four different ways that APIs can work:

- **SOAP APIs:** This API transfers messages in XML format and is not popular anymore.
- **RPC APIs:** Remote Procedure Calls that the client completes a procedure and sends the request to the server, then the server responds with the requested data.

- **REST APIs:** These are widely used and adaptable APIs available right now online. Requests are sent to the server as data by the client. The server launches internal processes using this client input and sends the results back to the client.

- **Websocket APIs:** This kind of API supports two-way communication. The server and client transmit data in JSON format on a live connection.

In our proposed VSP application, the APIs are implemented based on REST API and are accessible without any authentication or authorization. The benefits of providing the VSP services using REST APIs are as follows:

- **Integration:** New apps are integrated with current software systems through APIs. Because each functionality doesn't need to be created from start, development time is sped up. APIs can be used to benefit from preexisting code.

- **Easy Maintenance:** A gateway between two systems is created via the API. Each system is required to implement internal adjustments to ensure that the API is not harmed. In this manner, any upcoming code modifications by one party won't affect the other side.

Table 4.3: Use Case Description for Importing Foreground Traffic (Plan)

Use-Case ID	3	
Use-Case Name	import-initial-plan	
Actors	Any MOD planning application can call this API.	
Description	This API takes a JSON formatted input as schedule plan. The JSON input is an array of trips. Also, each trip has an array of segments. The System creates corresponding vehicles with defined information and simulates their moving on the map. These vehicles are called Service Vehicles and have an identifiable colour. At the end of the simulation, the system provides the logs of these vehicles.	
Preconditions	Simulation boundaries are set.	
Postconditions	Service vehicles with identifiable colours moving on the map.	
Normal Flow	**Actor Action**	**System Response**
	3.0. The user provides a JSON formatted input containing trips and segments and posts to the API.	• Validates and processes input • Converts text-address to geo-coordinates • Adds new vehicles to the simulation
Exception Flow	**Actor Action**	**System Response**
	3.0.E.1 Input is invalid	• Validates and processes input • Converts text-address to geo-coordinates • Adds new vehicles to the simulation
	3.0.E.2 Address Conversion Failed	• Prompts user the reason for failure • Logs error
Special Requirements	• Simulation is running • precise and powerful geocoder should exist to convert the address to GPS points	

Table 4.4: Use Case Description for Updating Existing Vehicles Destination

Use-Case ID	4	
Use-Case Name	update-destination	
Actors	Any MOD planning application can call this API.	
Description	This API applies dynamic changes to each segment while the simulation is running. The actor should provide a JSON formatted input to apply the updates. Each segment is defined by a unique id. So, it is required for the actor to provide the segment id in the input. Also, a new destination is required in this API. If the segment id and new destination are valid values, the system applies the change to the vehicle and for better demonstration, the colour or shape of the vehicle will change as well.	
Preconditions	Simulation boundaries are set and the vehicle exists on the network.	
Postconditions	Updated vehicle should be marked by a new colour or shape.	
Normal Flow	**Actor Action**	**System Response**
	4.0. The user provides values such as vehicle id, segment id, new destination, and depart time in JSON format and posts the input to the API.	• System reads and validates inputs • Identifies vehicle on the network • Converts address to GPS point • Finds the closest edge to the new destination • Updates destination and changes colour
Exception Flow	**Actor Action**	**System Response**
	4.0.E.1 Invalid Inputs	• Prompts the user • Shows a sample of valid values if possible
	4.0.E.2 Vehicle not found	• Prompts user that vehicle does not exist on the network • Logs error
Special Requirements	• Simulation is running • Vehicle Id should be valid and available • Destination should be valid and within the boundaries of simulation	

Table 4.5: Use Case Description for Adding Segments to Trips

Use-Case ID	5	
Use-Case Name	add-segment	
Actors	Any MOD planning application can call this API.	
Description	Actor can add a new segment to each vehicle that exists on the simulation and has not reached to its last destination. The system defines a new stop point for the vehicle. Inputs should be validated before adding the segment to make sure the address is not already reached and not processed by previous segments.	
Preconditions	Simulation boundaries are set and the vehicle exists on the network.	
Postconditions	The vehicle should have the segment in its segment list.	
Normal Flow	**Actor Action**	**System Response**
	5.0. The actor provides input values formatted in JSON and posts the API.	• System reads and validates inputs • Identifies vehicle on the network • Converts address to GPS point • Finds the closest edge to the new destination • Adds a new stop for the vehicle
Exception Flow	**Actor Action**	**System Response**
	5.0.E.1 Invalid Format	• Prompts the user • Shows a sample of valid values if possible
	5.0.E.2 Invalid input	• Prompts user the reason for failure • Logs error
Special Requirements	• Simulation is running • Vehicle Id should be valid • Segment should be valid and within the boundaries of simulation	

Table 4.6: Use Case Description for Removing Existing Segments

Use-Case ID	6	
Use-Case Name	remove-segment	
Actors	Any MOD planning application can call this API.	
Description	Actor can remove a non-processed segment for any vehicle that exists on the simulation and has not reached to its last destination.	
Preconditions	• Simulation boundaries are set. • Simulation is running. • Vehicles exist on the network. • Segment is not passed by the vehicle.	
Postconditions	Segment should not exist on the destinations list of the vehicle.	
Normal Flow	**Actor Action**	**System Response**
	6.0. The actor can list the segments of each vehicle using APIs. Then, provides a segment id and calls the deleted segment API.	• Removes the segment • Replaces the removed segment with the next segment
Exception Flow	**Actor Action**	**System Response**
	6.0.E.1 Segment is already processed	• Prompts actor • Logs the error
Special Requirements	Simulation is running	

Table 4.7: Use Case Description for Collecting Simulation Results

Use-Case ID	7	
Use-Case Name	collect-simulation-results	
Actors	Any MOD planning application can call this API.	
Description	The actor can get simulation results. Results consist of: • Vehicles GPS locations • Travel distance • Travel Time • etc.	
Preconditions	Simulation boundaries are set and simulation is running.	
Postconditions	Simulation logs and results should be accessible after the simulation is finished.	
Normal Flow	**Actor Action**	**System Response**
	7.0. The user calls the Get logs API to get all simulation logs. Filtering logs are not provided by the system.	• System stores all logs into the database • Database is accessible after simulation execution
Exception Flow	**Actor Action**	**System Response**
	7.0.E.1 Simulation has not started yet	• Prompts the actor that simulation has not started • Logs error
	7.0.E.2 Log not available	• Prompts user that log is not ready yet and could be retrieved in a while • Logs error
Special Requirements	Simulation should be started and at least one vehicle should reach its destination	

Table 4.8: Use Case Description for Accessing Service Vehicles Data

Use-Case ID	8	
Use-Case Name	get-service-vehicles	
Actors	Any MOD planning application can call this API.	
Description	Actor can get list of service vehicles that are on the simulation map now. This API could be useful for retrieving the information on trips and segments. This information could be useful for calling other APIs such as Update Destination, Remove Segment, etc.	
Preconditions	Simulations have already started.	
Postconditions	Actor can get a list of service vehicles including their segments.	
Normal Flow	**Actor Action**	**System Response**
	8.0. User calls the Get service vehicles API to get all vehicle information.	• System creates list of service vehicles including their data • Status, id, and other information of each vehicle and segment are included in the response.
Exception Flow	**Actor Action**	**System Response**
	8.0.E.1 Simulation has not started yet	• Prompts actor that simulation should be running • Logs the error
Special Requirements	Simulation is running	

Table 4.9: Use Case Description for Viewing Running Simulation

Use-Case ID	9	
Use-Case Name	view-simulation	
Actors	Human actor can use this use case	
Description	Actor can execute the simulation either using the GUI or No-GUI versions. If the actor uses the GUI version, the selected map, background traffic, and service vehicles can be viewed in the SUMO GUI. Also, the changes to the simulation scenario are identifiable using the GUI.	
Preconditions	SUMO should be installed on the Host machine.	
Postconditions	SUMO GUI should be executed.	
Normal Flow	**Actor Action**	**System Response**
	9.0. The actor chooses to run the simulation using the GUI.	• System configures the simulation and reads the map • System displays the user-selected boundaries and the moving vehicles in this area.
Exception Flow	**Actor Action**	**System Response**
	9.0.E.1 SUMO GUI Not Found	• Prompts actor that system cannot connect to the simulation engine • Logs the error
Special Requirements	SUMO GUI should be available on the host.	

Chapter 5

Detailed Design of Platform APIs

This section discusses the design details of different system components and explores the relations to deliver the expected API results.

The VSP system consists of five primary components: the dashboard, the Open Street Map (OSM), the VSP Core, the SUMO Engine, and the database. These components communicate with each other to provide the expected services. The relations between these components are illustrated in Figure 5.1.

As illustrated in Figure 5.1, the VSP user interacts with the system using the dashboard component. The dashboard component provides tools to select a particular area on the map as the simulation boundary. By selecting an area on the map, the coordinates are passed to the VSP Core to download the map. After downloading the map, the VSP Core converts the map into a simulation network and prepares the files for the SUMO simulation engine. Furthermore, any commands that the user sends to the system are received by the VSP Core and this component stores the command in an event queue. While the SUMO executes a simulation scenario, the VSP Core gets the simulation step data from the SUMO. At each simulation step, if any change event (User Command) is found, the VSP Core takes the command out of the queue and applies the change to the simulation engine. Figure 5.2 illustrates an overview of how the simulation starts and accepts the events. Once the simulation starts, the user can call the functions of the VSP using the dashboard or APIs. The requests will be passed to the VSP Core using the available protocols (HTTP/Socket). After receiving a request from the user, the VSP Core stores and processes the message, and returns the results

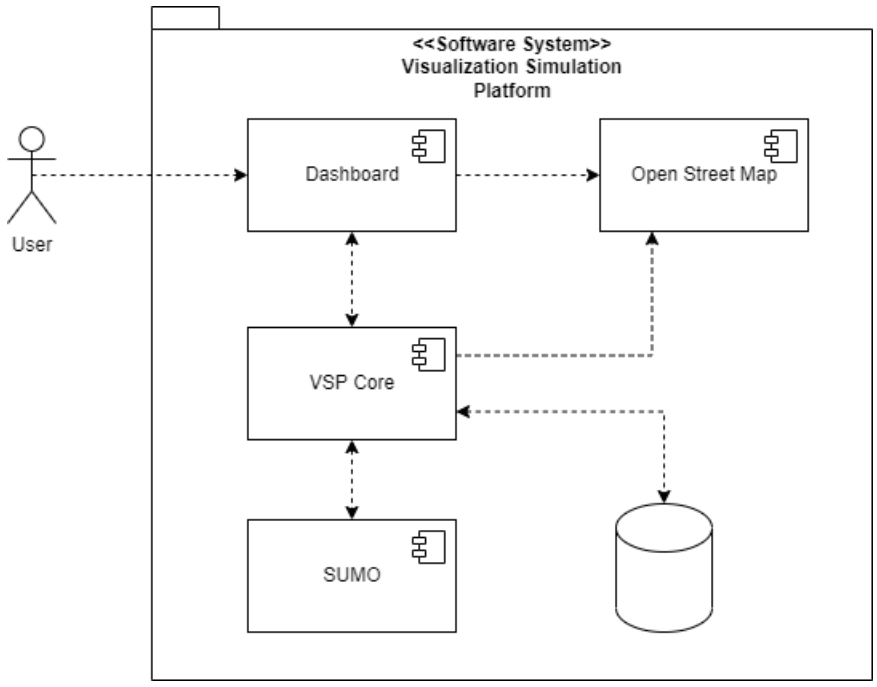

Figure 5.1: The VSP Components Overview

to the user using the same connection. Figure 5.3 demonstrates a general sequence diagram of the system.

5.1 Boundary Selection API Design

Boundary API is the first use case in the VSP and is intended to enable the user to define an area on the map and use it as the simulation boundary. The functionality of this API is illustrated in Figure 5.4. The actor uses the user interface tools to select an area on the map. This area is considered to be the boundary of the simulation. Once the user wants to start the simulation, the dashboard sends the selected coordinates to the VSP Core. The VSP Core downloads the map from the open street map (OSM) which is an open-source mapping system. The files downloaded from the OSM have a specific format and they are called .osm files. The VSP Core converts the OSM file to the simulation network and stores the required files for the simulation. Once the simulation is started, it will read the stored simulation networks on the disk to execute its simulation scenario.

The technical specifications of this API is as follows:

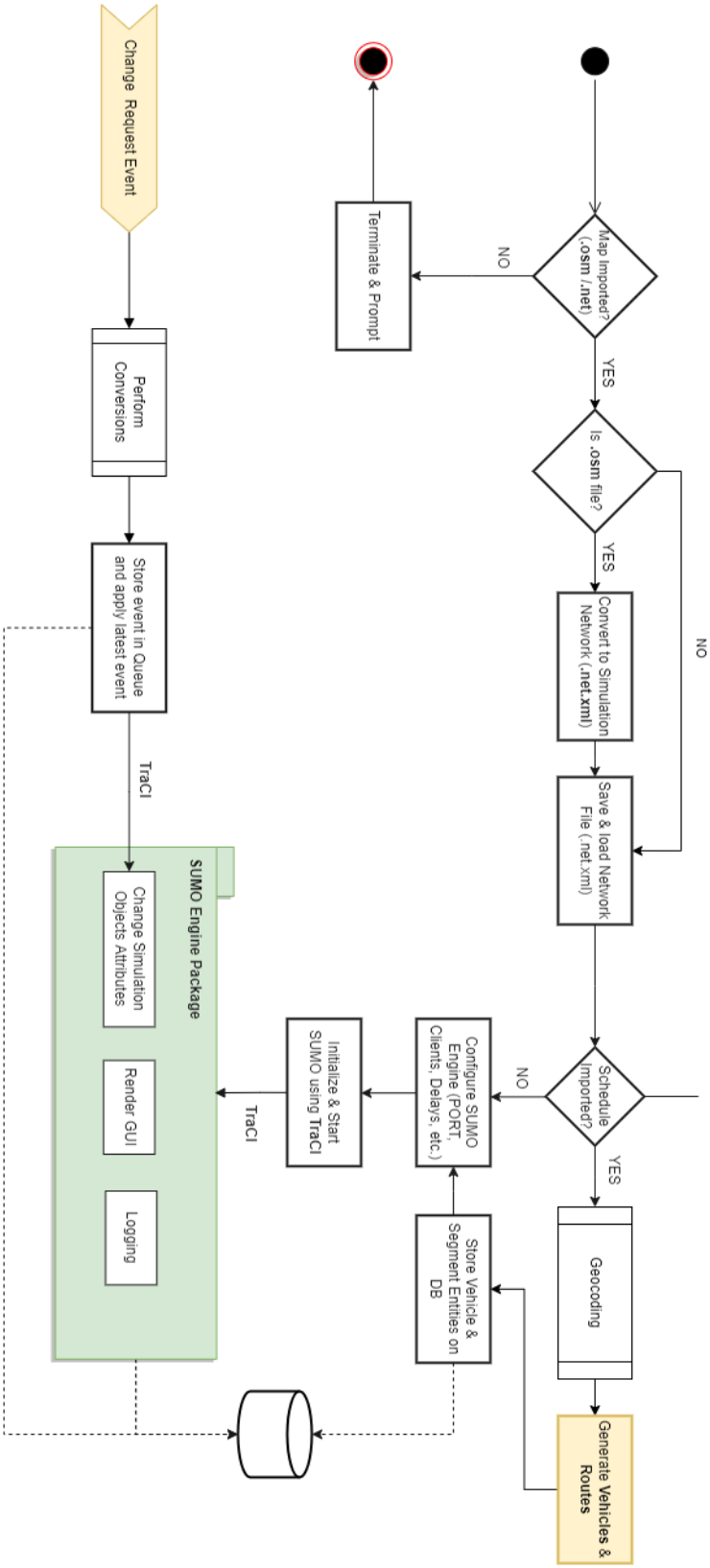

Figure 5.2: Start Simulation Activity Diagram

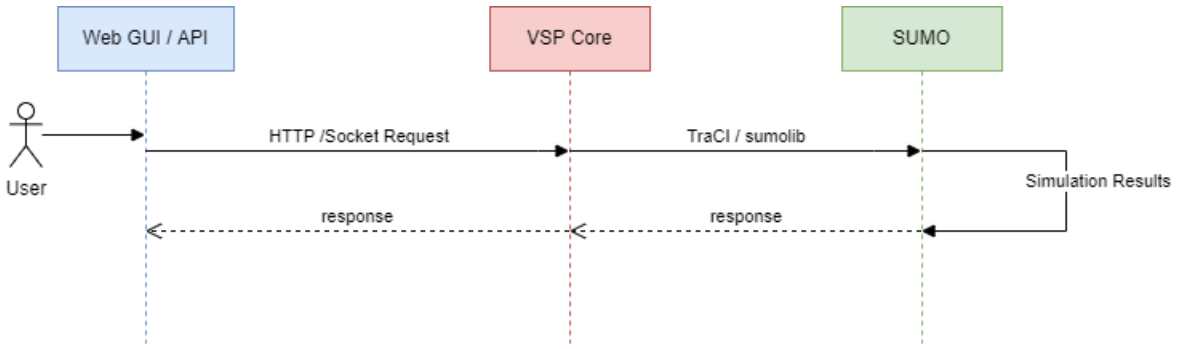

Figure 5.3: VSP Events Sequence Diagram Overview

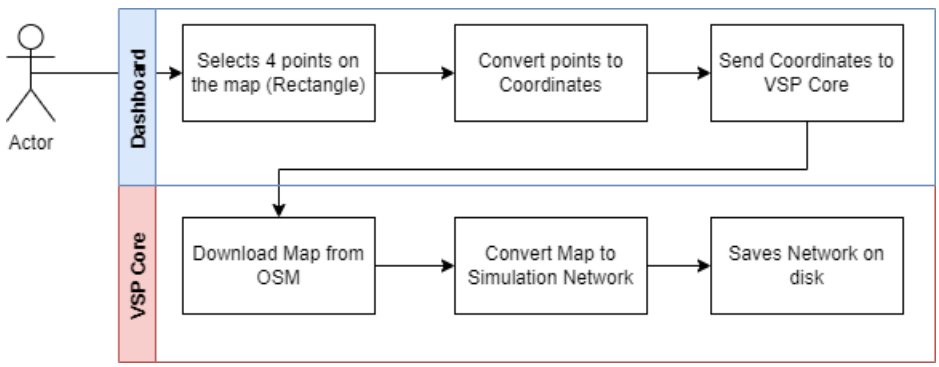

Figure 5.4: Boundary Selection Activity Diagram

Url: **[POST]** http://127.0.0.1:5000/map

Table 5.1: Boundary Selection API Request Specs

Parameter	Type	Description
[]	Array of decimal	An array of 4 coordinate points in the format of decimal numbers

Sample Request:

```
curl --location --request POST 'http://127.0.0.1:5000/map' \
--header 'Content-Type: application/json' \
--data-raw '[73.59059936523396, 45.48301230113723, -73.54940063476586,
    45.496986832059754]'
```

5.2 Traffic Setup API Design

In the previous section, the application downloaded, converted, and stored the network file on the disk. In this section, the traffic API can use the stored network file to generate the traffic according to the user inputs. In this work, I have used the available demographic data which is published on the Canada Statistics internet sites (Statistics Canada, 2022). By using such realistic data, we can have more realistic traffic patterns in the simulation. I used the ACTIVITYGEN and ScenarioFromOSM tools in the SUMO package to generate traffic. These SUMO tools use a road network model and a population description to create a traffic demand for a scenario. In order to determine everyday activities like work, school, and spare time, it employs an activity-based traffic model that depends on a multi-modal trip planner encompassing buses, vehicles, bicycles, and pedestrians. The traffic setup API gets demographic data from the user and generates the traffic accordingly. The traffic files, which are stored on the disk, contain the list of vehicles, followed by their departure time, departure, and destination locations, and probable stops. As illustrated in figure 5.5, this API is using the existing simulation network which is generated in the previous step, then stores the traffic files on the disk to be used by the simulation core.

The technical specifications of this API is as follows:

Url: **[POST]** http://127.0.0.1:5000/traffic

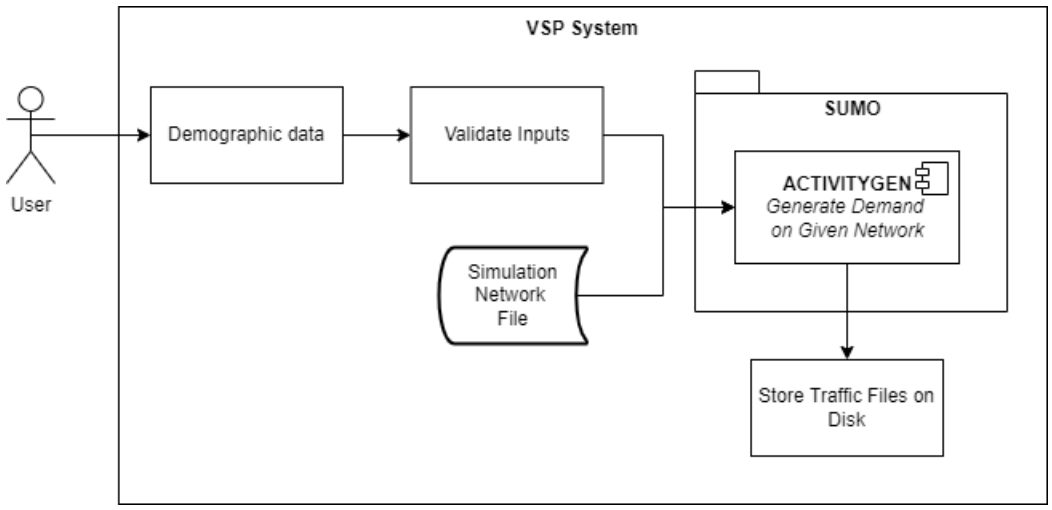

Figure 5.5: Traffic Setup Flow

Table 5.2: Traffic Setup API

Parameter	Type	Description
population	Integer	Population of the selected area
density	Integer	Number of people per square kilometer

Sample Request:

```
curl --location --request POST 'http://127.0.0.1:5000/traffic' \
--header 'Content-Type: application/json' \
--data-raw '{
    "population": 1000,
    "density": 300
}'
```

5.3 Import Service Vehicle Plan API Design

This API is designed to accept a list of vehicles and add the vehicles to the simulation map. Each vehicle in this list has a departure time, departure, and destination address. The API processes the input file and extracts the data. Then, based on the data creates and add the vehicle to the simulation map. The colour of the service vehicle which is added by this API differs from the background traffic vehicles. This makes it easier to identify the service vehicles on the map when the simulation

is running and many moving objects exist on the map. The technical specifications of this API is as follows:

Url: **[POST]** http://127.0.0.1:5000/plan

Table 5.3: Import Plan API

Parameter	Type	Description
vehicle_id	String	Unique identifier of the vehicle
sequence_number	Integer	Segment Order Index
begin_time	Integer	Time to start movement
departure	String	Departure address
destination	String	Destination address

Sample Request:

```
curl --location --request POST 'http://127.0.0.1:5000/plan' \
--header 'Content-Type: application/json' \
--data-raw '[
    {
        "vehicle_id": "1",
        "sequence_number": 1,
        "begin_time": 30100,
        "departure": "2132 tupper, montreal",
        "destination": "176 Peel, montreal"
    }
]'
```

Addresses in the plan file are in postal address format, but I need to convert these addresses to values that are recognizable by the SUMO simulation engine. In the SUMO, streets and intersections have their specific identifiers and it is not possible to find an address in the SUMO by its postal format text. So, I needed to have a conversion between text-based addresses and SUMO points. To do this, firstly, I convert the text-based address to valid GPS coordinates. This process is known as Geocoding. The Nominatim is an open-source tool for searching the addresses on the Open Street Map data. I used this tool to find the GPS coordinates equivalent to the address. Then, I converted the GPS point to the SUMO point on the simulation network. But, I noticed that the SUMO does not

allow me to put the cars on these points. Because mostly the converted points are located on sidewalks which are not available for vehicles. To solve this problem, I designed an algorithm to find the nearest available street to the converted point which is permitted for vehicles. I use this point as the result of converting the address to the points in the simulation. The conversion algorithm flowchart is illustrated in Figure 5.6. After converting the address to a point on the simulation network, a new vehicle is inserted into the network with a specified start time. Once the simulation time reaches the specified time for the vehicle, the vehicle can be seen on the map and it starts moving toward its pre-defined destination.

5.4 Update Destination API Design

In the MOD applications, it is probable that a vehicle changes its destination. In the VSP there is an API for applying changes to the destination of vehicles that exist on the simulation network. The objective is to apply these changes while the simulation is running. This feature allows the simulation user to apply on-demand changes without the need to stop or restart the simulation execution. The validations in this API, check if the input vehicle identifier is valid and if the address is within the simulation boundaries. Figure 5.7 shows the overall algorithm of updating the destination. As shown in the figure, the address is converted using the AddressConverter module which connects to the Nominatim service and the final result is a point in the simulation network where a car could be located at.

The technical specifications of this API is as follows:

Url: **[POST]** http://127.0.0.1:5000/destination

Table 5.4: Change Destination API

Parameter	Type	Description
vehicle_id	String	Unique identifier of the vehicle
destination	String	Destination address

Sample Request:

```
curl --location --request POST 'http://127.0.0.1:5000/destination' \
--header 'Content-Type: application/json' \
```

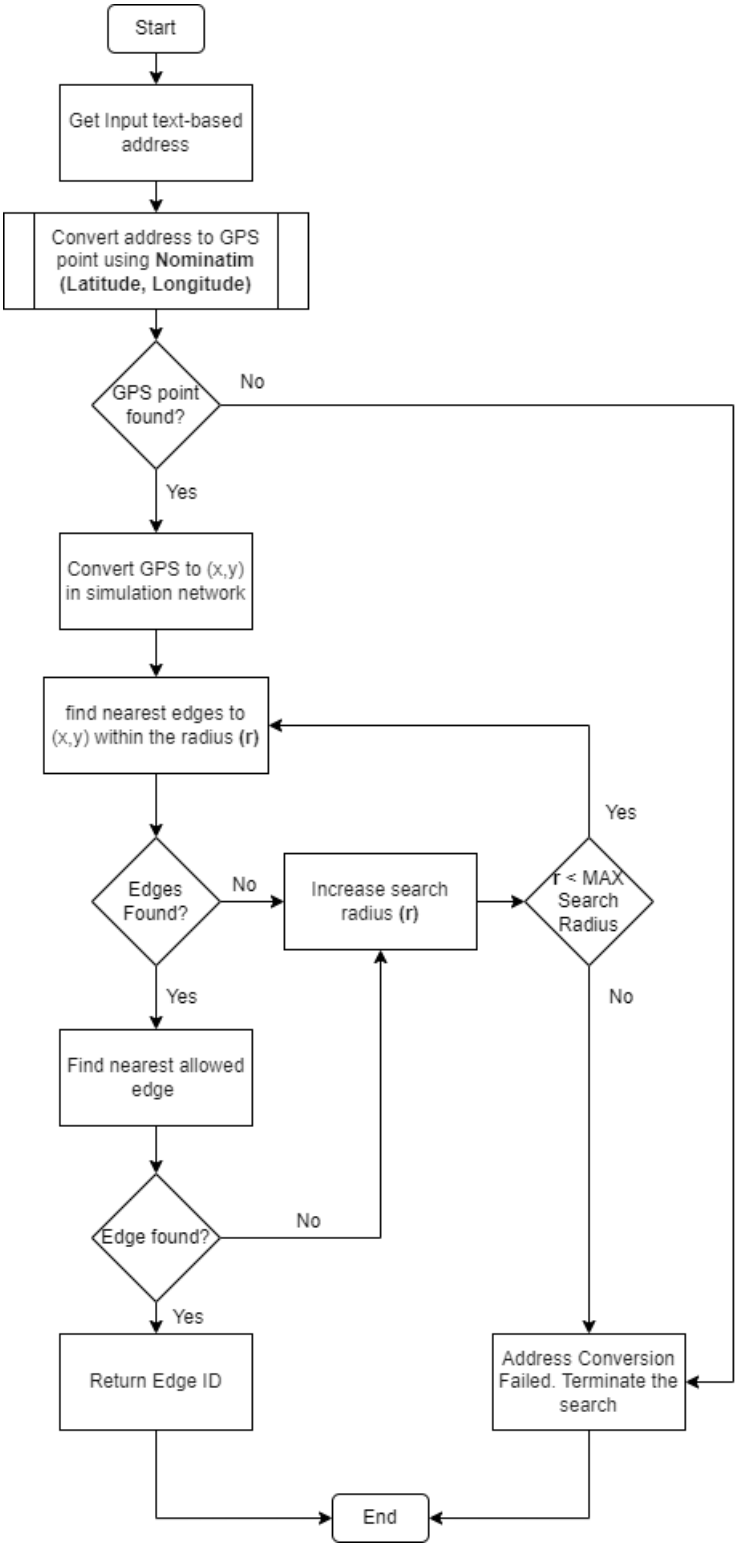

Figure 5.6: Convert text-based address to network points algorithm

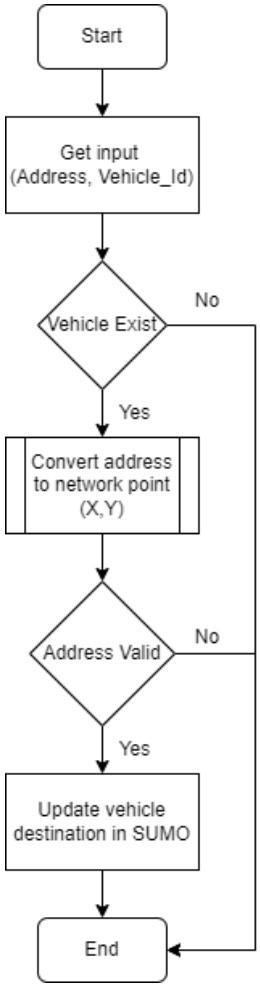

Figure 5.7: Update vehicle destination algorithm

```
--data-raw '{
    "vehicle_id": "sv_1#1",
    "destination": "2132 tupper, montreal"
}'
```

5.5 Add segment API Design

The VSP system does not put limitations on updating the vehicles' travel plans during the simulations. In the other words, while the simulation is running, the actor can add stop points for each vehicle. As the focus of this VSP system is mostly on MOD-based applications, in this study each

stop point is declared as a segment. So, a vehicle might have one or more segments in its trip to the final destination. The flowchart 5.8 shows how this API processes inputs.

The technical specifications of this API is as follows:

Url: **[POST]** http://127.0.0.1:5000/segments

Table 5.5: Add Segment API

Parameter	Type	Description
vehicle_id	String	Unique identifier of the vehicle
begin	String	start time for moving from departure
departure	String	Departure address
destination	String	Destination address

Sample Request:

```
curl --location --request POST 'http://127.0.0.1:5000/segments' \
--header 'Content-Type: application/json' \
--data-raw '{
    "vehicle_id": "1",
    "begin": "100",
    "departure": "4545 monkland",
    "destination": "4040 terrebone"
}'
```

5.6 Remove segment API design

It is probable for any MOD application that the user wants to cancel a trip or remove a stopping point from the whole trip plan. In this case, the simulation platform should be dynamic enough to be able to remove a segment while the simulation is running. So, the vehicle will not move toward the point that is removed from the original plan imported at the beginning of the simulation. While this feature could be useful for dynamic planning, the system should be smart enough to validate the request and avoid deleting the segment that is already passed. Also, it should be noted that because each vehicle could have a queue of segments, the VSP should rearrange the segments to make sure the vehicle will pass all the segments during the simulation. Figure 5.9 shows the internal steps in this API.

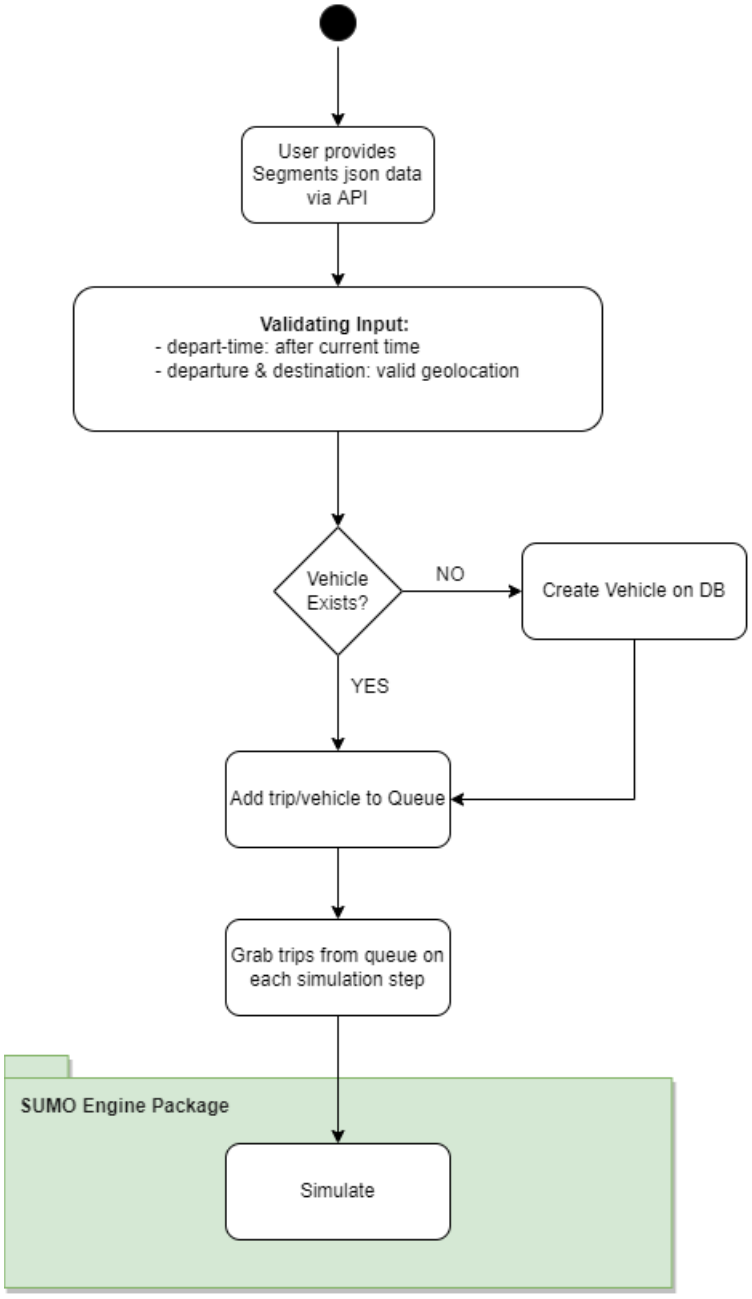

Figure 5.8: Add segment API flowchart

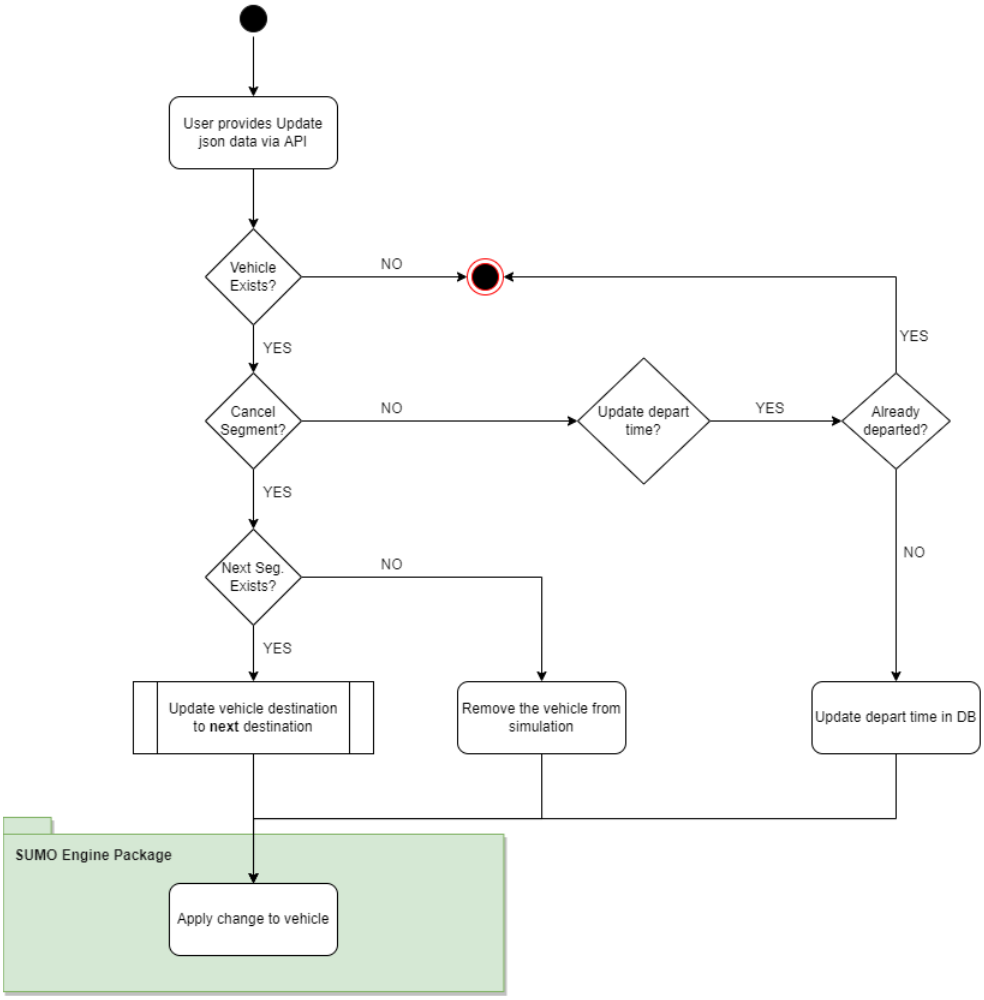

Figure 5.9: Remove segment API flowchart

The technical specifications of this API are as follows:

Url: **[DELETE]** http://127.0.0.1:5000/segments

Sample Request:

```
curl --location --request DELETE 'http://127.0.0.1:5000/segments' \
--header 'Content-Type: application/json' \
--data-raw '{
    "vehicle_id": "1",
    "destination": "4040 terrebone"
}'
```

Table 5.6: Delete Segment API

Parameter	Type	Description
vehicle_id	String	Unique identifier of the vehicle
destination	String	Destination address

5.7 Service Vehicle Status API Design

The VSP system is connected to a database and the database keeps the persistent data. Once the simulation starts and the cars appear on the simulation network, at each simulation step, the VSP system gets the vehicles' locations on the network. It converts these values to GPS coordinates. Then, inserts a new log for each vehicle on the database including the vehicle identifier, vehicle coordinates, and timestamp. As the simulation goes forward, the logs in the database increase. At the end of the simulation, the VSP database lists trajectories related to each vehicle on the simulation network. Simultaneously, the VSP monitors service vehicles on the simulation network and if a vehicle reaches its destination, the VSP changes the vehicle status in the database. The VSP provides an API for retrieving the status of the vehicle by using its vehicle identifier.

Sample Request: The following code snippet shows how to use the API.

```
curl --location --request GET 'http://127.0.0.1:5000/status?vehicle_id=1' \
--header 'Content-Type: application/json'
```

Response: The status API is designed for retrieving data from the VSP system. The response is formatted in JSON. Table 5.7 shows the output parameters of this API.

Table 5.7: Vehicle Status API Response Format

Parameter	Type	Description
vehicle_id	String	Unique identifier of the vehicle
virtual_vehicle_id	String	vehicle_id + segment_id
departure_address	String	Departure address
destination_address	String	Destination address
status	Integer	Status of vehicle. 1: Pending, 2: Processing, 3: Arrived

Sample Output: The following json string, presents a sample output of the status API. According to the design of the system, each vehicle can have many segments. Therefore, the status API

returns the status of the vehicle for all of its predefined segments.

```
[{
    "vehicle_id": "1",
    "virtual_vehicle_id": "sv_1#1",
    "status": 3,
    "departure_address": "1905 Bassins, Montreal",
    "destination_address": "1201 rue Guy, montreal "
}]
```

5.8 Summary

The VSP system has an API layer which allows the users or MOD applications to connect using the HTTP or socket protocols and utilize the internal features of the VSP. The available APIs take the inputs, validate, convert the values if needed, and pass the values to internal layers for further processing. While the simulation is running, it is possible to call the APIs and the VSP system applies the inputs to the running simulation engine. During the simulation, the VSP collects logs from the SUMO engine and after processing the logs, stores them in the database for further utilization.

Chapter 6

System Implementation and Verification

This chapter covers the technical implementation details, including the programming language, database, tools, and code structure. Furthermore, to verify the implementation a test scenario is designed to test the application functionality. Finally, a case study is presented in which the VSP could be used to perform required simulations.

6.1 Implementation Details

The VSP is an open-source application that is publicly available to use and contribute to. The source is available on GitHub which is a well-known source control system. Table 6.1 lists the code metadata of the VSP.

Table 6.1: Code metadata

Metadata	Description
Permanent link to code/repository	https://github.com/arax-zaeimi/vsp-sumo
Legal Code License	MIT License
Code Versioning System	Git
Software code language	Python, HTML, JavaScript
Compilation Requirements, dependencies	Python 3.9+, SUMO 1.14.1+, PostgreSQL 10.0+
Developer Documentation/Manual	https://github.com/arax-zaeimi/vsp-sumo/blob/main/README.md
Support email for questions	m_zaeimi at live.concordia.ca

6.1.1 Simulation Engine

The VSP is based on the SUMO application and libraries. This traffic simulation software is free and open source. Since its introduction in 2001, it has enabled the modelling of multi-mode traffic networks, which include pedestrians, public transportation, and road vehicles. Numerous auxiliary tools, such as network import, route calculations, visualization, and emission calculation, are included with SUMO and automate fundamental processes for developing, executing, and assessing traffic simulations. Custom models may be added to SUMO, and it offers a number of APIs for controlling the simulation (Lopez et al., 2018). To use, reference, and call the SUMO components it should be installed on the local computing machine. The SUMO application could be installed on Windows, Linux, and MAC operating systems. I started this research with version 1.9.1 and at the time of writing this thesis, SUMO introduced version 1.14.1. So, I upgraded the engine to the latest available version. In Figure 6.1 the installed SUMO version is presented using the command line.

```
PS C:\> sumo --version
Eclipse SUMO sumo Version 1.14.1
 Build features: Windows-10.0.17763 AMD64 MSVC 19.29.30133.0 Release SumoUtilsLibrary FMI Proj GUI SWIG GDAL FFmpeg OSG
GL2PS Eigen
 Copyright (C) 2001-2022 German Aerospace Center (DLR) and others; https://sumo.dlr.de

Eclipse SUMO sumo Version 1.14.1 is part of SUMO.
```

Figure 6.1: SUMO Installed Version

The SUMO contains several functional components that could be used in a variety of scenarios. According to the requirements of the project, the users choose the right components to use. For the implementation of the VSP we have used the following components of the SUMO:

sumo and *sumo-gui* are the primary components that take a configuration as input and start to execute the simulation based on the parameters defined in the configurations. Both components have the same functionality and output and the only difference is that the *sumo-gui* offers a graphical user interface and shows the moving objects on the simulation network. So, the user can visually observe the map, objects, and simulation behaviours while the simulation is running. Also, it is possible to change many visualization options using the interface. Figure 6.3 is an screenshot of the *sumo-gui* whilst executing a scenario.

As is shown in Figure 6.3, the graphical user interface is showing the simulation map, objects

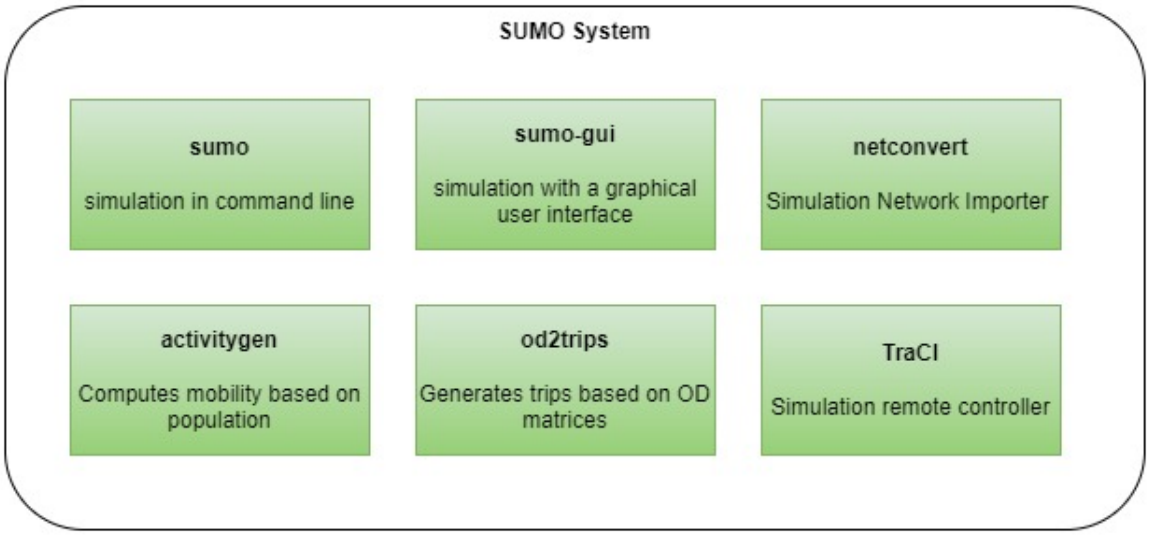

Figure 6.2: SUMO Components

that are loaded and moving on the roads, and brief statistics at the bottom of the interface. Furthermore, it is possible to change the simulation speed, scale the traffic, and view the live logs. Although the SUMO GUI has more features, they are beyond the scope of this study and we only focus on the features that are mostly used for the objectives of the thesis.

netconvert is another component of the SUMO that allows importing realistic maps to the SUMO. One of the requirements for simulation is a road map. In the SUMO we call it a simulation network. It is possible to create a personalized network from scratch, but in this study, I needed to have a realistic map in the simulation. To do so, I download a particular area map from the OpenStreetMap website, a free and open-source map system. Then, used the *netconvert* to convert the map to the simulation network. If the simulation network needs any changes, the *netedit* could be used to apply the changes to the network. *netedit* is included in the SUMO installation package.

activitygen and *od2trips* are other useful tools included in the SUMO package which allow us to generate the background traffic on the map. By using these features we can develop more realistic demand on the map because the vehicles are inserted into the map based on predefined patterns that are generated from real-world population statistics.

TraCI is the primary tool that allows the VSP application connects to the SUMO simulation

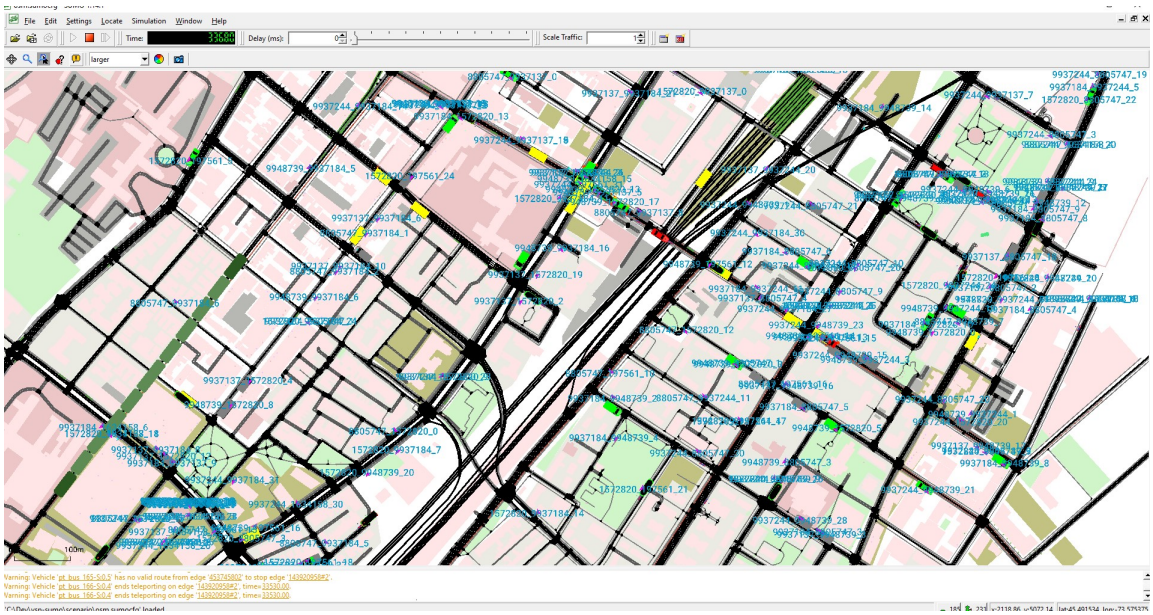

Figure 6.3: SUMO GUI while running a scenario

engine, retrieve simulation properties and status, and apply the required changes to the simulation that is already running. According to the official website of the SUMO, the TraCI is the short term for ”Traffic Control Interface”. Accessing a running road traffic simulation allows us to retrieve values of simulated objects and manipulate their behaviour ”online”.

6.1.2 Programming Language

The SUMO libraries are available in several programming languages such as Python and C#, however, the most reliable and up-to-date libraries are implemented using Python. By reliability, we mean that all available features of the SUMO have been implemented and tested in Python. Therefore, we chose Python as the primary programming language to make the integration smoother and ensure we can use the SUMO libraries recommended by its development team. We need Python for the VSP development and utilization of the SUMO application and libraries. I installed the *Python 3.9* on my local computer and upgraded to *Python 3.10.2* at the time of writing this document.

6.1.3 Database

The VSP needs access to a database to read or write data. The data consists of commands that are sent from the APIs to the VSP Core or the simulation information that is collected from the SUMO while executing a simulation scenario. I started the project with a file-based database which stores data on the file on the local machine. This database engine is called *SQLite*. However, due to the limitations of this database engine I chose to continue the project with a more robust database provider. I needed a free, open-source, and high-speed database engine which supports reading and writing data at a high speed. I chose PostgreSQL as the primary database provider. According to the *PostgreSQL* website, this is a potent open-source object-relational database system that has been actively developed for over 30 years and has a solid reputation for dependability, feature robustness, and performance. The VSP is using *PostgreSQL*, version *"PostgreSQL 10.17, compiled by Visual C++ build 1800, 64-bit"*. To store data in a database I designed and created three tables, namely, commands, trajectories, and statistics. The tables and their columns are illustrated in Figure 6.4.

As shown in Figure 6.4, the tables do not need to have a relationship to be functional.

- **Simulation:** This table keeps the simulation parameters such as begin and end times. The segment and trajectory tables have a relation with the simulation table. So that, the stored data could be connected to only one instance of simulation.

- **Command:** The user sends its change commands through API. So, the VSP stores the command on this table to queue it for processing. Once the command is processed, the VSP core marks the command as processed. This helps the VSP to keep track of changes and avoid missing the processing of input commands.

- **Trajectory:** While the simulation is running and the vehicles are moving on the map, the VSP collects vehicles' GPS coordinates at each simulation step and stores them in *Trajectory* table. Once the simulation is finished, this table contains a collection of GPS points alongside the timestamp. This collection could be called the trajectories.

- **Segment:** The vehicle plans consist of segments. The VSP stores these segments in the database for further processing. The VSP reads the segments from the database, converts

addresses to the geo-coordinates, then converts them to the equivalent points in the simulation network.

6.1.4 Code Structure

The VSP code is structured into four primary python modules. These modules and their relationship is illustrated in Figure 6.5. By separating the required methods into different modules, the development of new features, maintenance, and reading of the code is more convenient. Furthermore, it keeps the code clean and easy to understand.

- **socket_server.py :** This module starts a web server for presenting the dashboard panel. Once the server is started a socket connection between the dashboard and the web server is initiated for transmitting data back and force. Furthermore, this module can take the results from the server and push them to the user interface using the available socket connection. This allows the application to present the current status of progress to the user interface instantly.

- **api.py :** This module is the entry point for all commands that come from outside of the VSP. In this module, a web server is configured to host the APIs of the VSP. Once a request is received, the module handover the message to the internal layers of the VSP for further processing.

- **osmGet.py :** This module is designed to receive an array of GPS coordinates and download the area from the OpenStreetMap servers. The downloaded map will be stored on the disk to be accessible by internal modules of the VSP.

- **data_access.py :** The VSP needs to have access to a database to do its functions. This module handles the data requests. It could read, update, or store data on the destination database. So, the VSP internal modules do not directly connect to a database. This makes the code cleaner and easier to manage and all data requests will be processed in a single module.

- **sumo_runner.py :** This is one of the core modules of the VSP in which the connections to the SUMO simulation engine are handled. This module uses one of the SUMO libraries named TraCI to connect to the SUMO and call its internal functions. TraCI allows us to manage the

simulation while a simulation scenario is running. We can add or remove vehicles, change destinations, read the statistics of the simulation and many more features. We use TraCI to collect vehicles' GPS coordinates and their travel status.

6.1.5 Control Dashboard

A control dashboard is designed for the VSP to make it easier to configure the options. The dashboard interface allows the user to select a particular area on the map and prepare it for the simulation. Furthermore, the dashboard could set demographic data, import plans, and define the simulation time. Figure A.1 presents the dashboard graphical user interface.

The designed dashboard is connected to the server with a socket connection. The user interface uses the JavaScript language to create JSON formatted data of user requests. Then, uses the available socket connection to send data to the server. Because of the designed socket connection, the server can send back the reports of the internal processes to the user interface. This feature allows the internal processes to report their progress to the user interface without any delays. The designed dashboard allows users to design and execute a simulation scenario without the need to know the internal complexities of the SUMO. The sequence of actions is as follows:

- Set Simulation Boundaries
- Set Traffic
- Set Service Vehicles Plan
- Set Begin and End Time
- Set Simulation Speed
- Turn SUMO UI on or off

6.2 Performance Evaluation

Synthetic traffic generation solutions in the literature do not integrate SUMO simulation and ARIMA and they have different approaches. Since we do not find an existing solution with the

same approach, to show the advantages of our proposed synthetic traffic generator, we compared the generated traffic patterns with other approaches without traffic prediction. We chose two approaches as benchmarks for the evaluation of our proposed framework. The first framework is SAGA which generates a 24-hour traffic scenario starting from an OSM file (Codecá et al., 2020). As the second benchmark, we chose STDG which estimates traffic based on real traffic datasets by using machine learning L2 logistic regression models, then generates synthetic trips. In this method, authors use generated trips and SUMO to produce a complete traffic scenario (Song & Min, 2018).

The first benchmark is labelled *SAGA* in our comparisons. SAGA extracts data from OSM files downloaded from OpenStreetMap and then generates activity-based traffic scenarios. For comparing DDSTG and SAGA, we consider the traffic demand patterns that are generated by the two approaches. Because SAGA is an open-source contribution library of SUMO, we could use its library to generate a 24-hour traffic scenario for the region of Downtown Montreal. Since we could use the same region for both methods, we were able to compare the demand generated by both methods. Figure 6.7 compares traffic demand generated by SAGA and DDSTG.

Figure 6.7 also shows the real demand for the same period of time, which could be clear means of comparison between, synthetic and real traffic demand data. The day is divided into 24 1-hour time windows. In this figure, we are showing the number of vehicles for a 24-hour day, starting from 00:00 until 23:59. It can be clearly demonstrated from the figure that generated demand by the DDSTG approach is following the real demand pattern during the day. Although the DDSTG demand is not identical to real demand, generally is following the increases and decreases of demand in real traffic data. However, the demand generated by the SAGA method can only follow the demand increase at the beginning of the day and the decrease in traffic density in the evening. Since DDSTG uses real traffic data to train its machine learning model, it can predict the traffic demand of the day and generate vehicles accordingly. Therefore we can see in Figure 6.7 that for DDSTG we have two demand pick hours, which is the same as the real data pattern, but the SAGA method could only generate demand with only one rush hour.

As the second benchmark, we used R-square (R2) to compare the results of our model with the results of STDG. Statistical Traffic Demand Generation (STDG) is another approach for generating synthetic trips by using machine learning techniques. The authors in this paper have published the

performance metrics of their work. In this paper, authors generate synthetic trips by generating the start and end points separately and calculating the R2 for each set (Song & Min, 2018). For comparison purposes, we consider the average values for R2 for both origin and destination points, then compare them with the value of R2 derived from our proposed approach.

$$\text{MSLE} = \frac{1}{n}\sum_{i=1}^{n}(\log(1+y_i) - \log(1+\hat{y}_i))^2 \tag{1}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}i)^2}{\sum i = 1^n (y_i - \bar{y})^2} \tag{2}$$

Equation 1 shows the formula for calculating MSLE. In this equation, n is the number of samples, y_i is the true value of the ith sample, and $\hat{y}_i$ is the predicted value of the ith sample. Equation 2 shows the formula for R2 also known as the coefficient of determination. In this equation, n is the number of samples, y_i is the true value of the ith sample, $\hat{y}_i$ is the predicted value of the ith sample, and $\bar{y}$ is the mean of the true values of all the samples. As Table 6.2 shows, we can see improvements for values of R2 and MSLE.

Table 6.2: Comparison of performance between DDSTG and STDG

Method	**R2**	**MSLE**
STDG	0.34	0.17
DDSTG	0.46	0.14

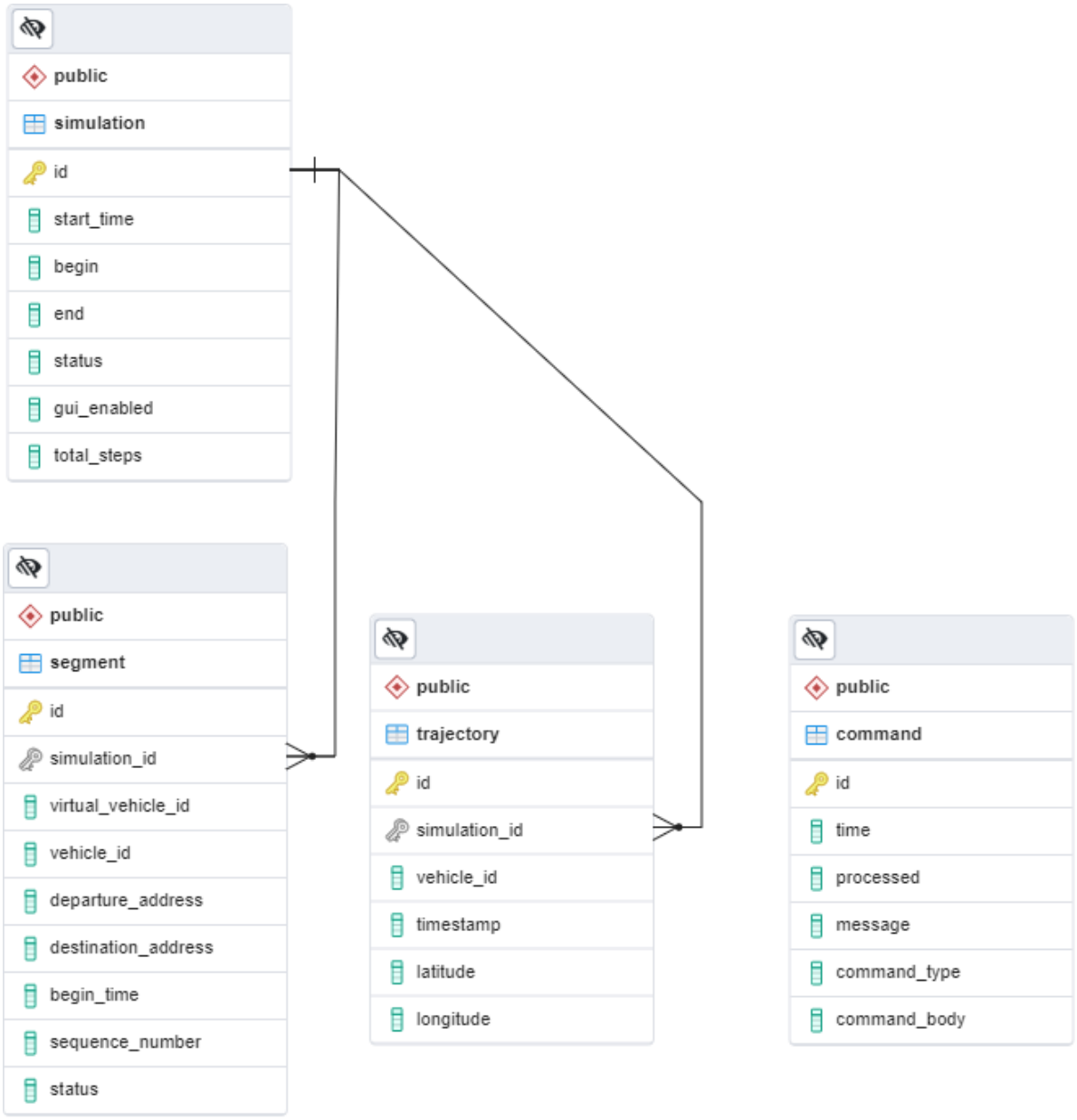

Figure 6.4: VSP Database Design

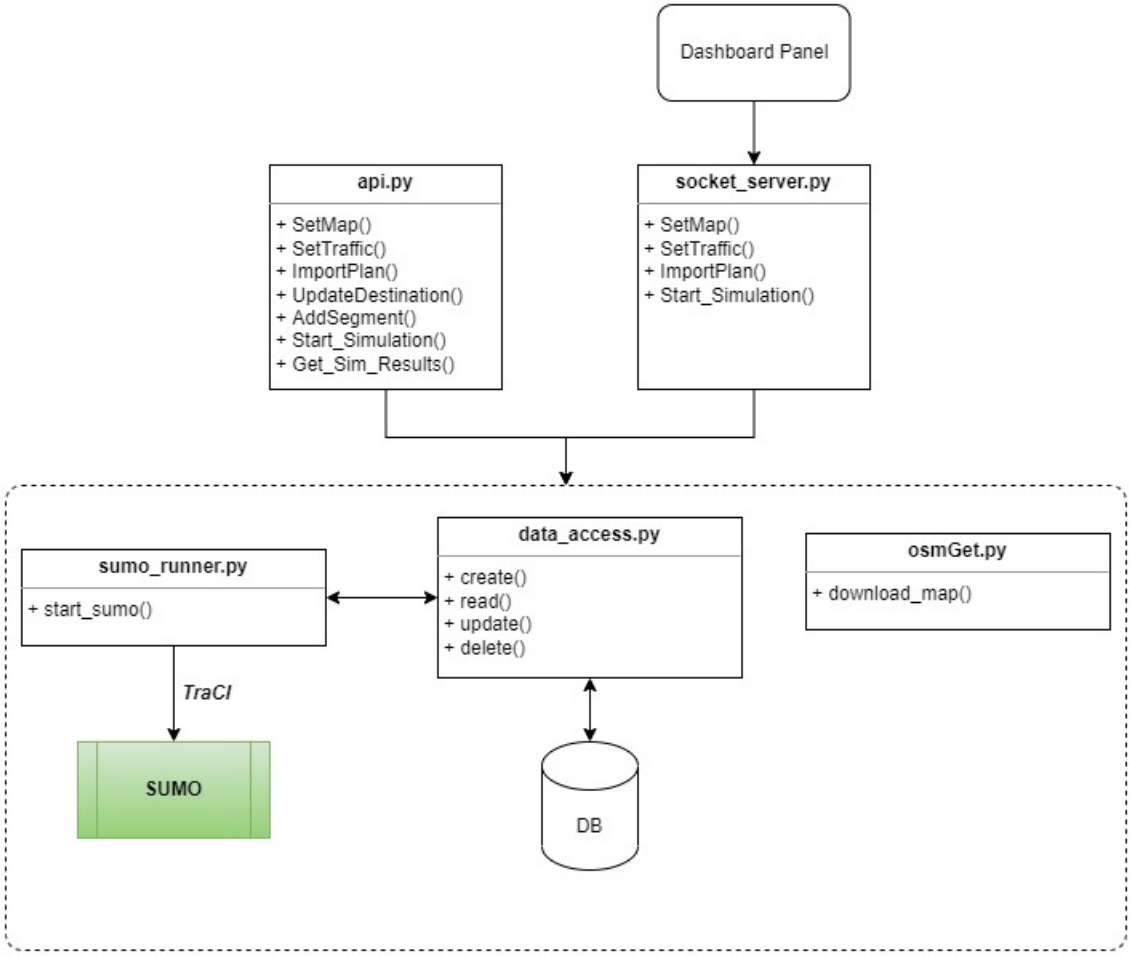

Figure 6.5: VSP Code Structure

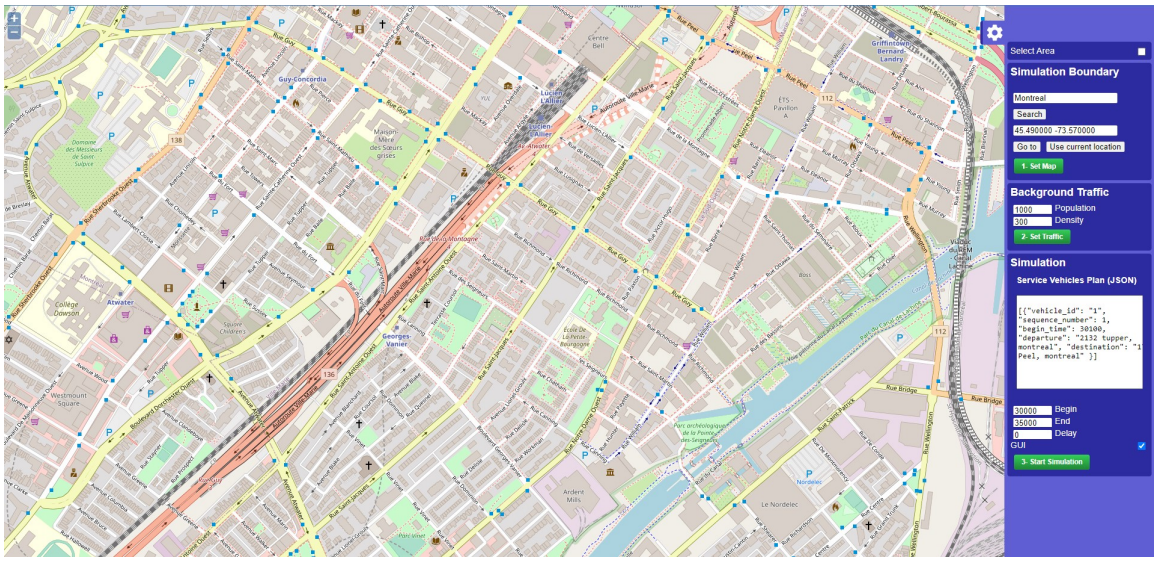

Figure 6.6: VSP Control Dashboard

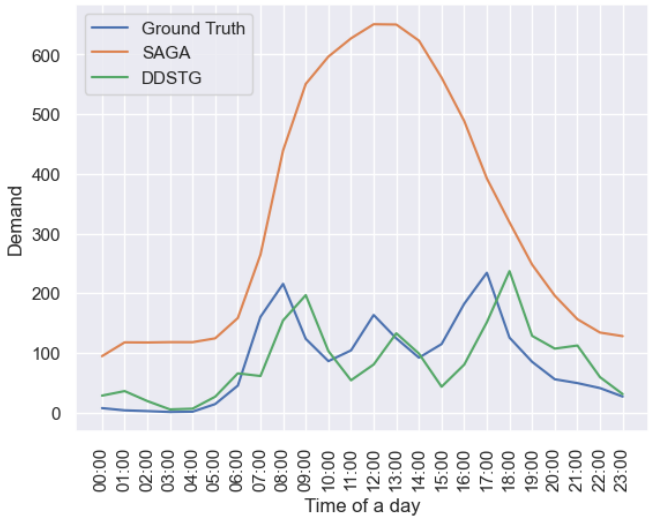

Figure 6.7: Comparison of DDSTG synthetic demand and SAGA demand

Chapter 7

Conclusion and Future Work

7.1 Conclusion

While existing urban traffic simulation platforms have a variety of features to design and execute scenarios, they are not specifically designed for MOD applications. So performing a simulation scenario in these applications needs a high level of expertise in simulation to run a simulation in a metropolitan area. It could start by downloading the maps, converting them to the simulation networks, configuring traffic with a huge amount of effort, providing meaningful inputs for the simulation engine, and even collecting the simulation results. These are a few of the many challenges that performing a simulation could have for a user. On the other hand, the existing simulation platforms cannot directly support MOD applications and do not offer any integration API. Also, the background traffic generated in the simulation applications is mainly based on random departures and destinations and times and there is no specific pattern in their generated traffic data. In this thesis, we proposed a synthetic MOD data generation framework. Synthetic MOD data includes trajectories for each MOD service vehicle moving in the simulation environment in the context of realistic background traffic. we used ARIMA to train a machine learning model for the prediction of traffic demand for 24 hours. The designed simulation framework and application are based on the SUMO engine. Besides, a user interface has been designed and implemented in this thesis to provide the framework features to the users. This resulted in a dynamic visualization simulation platform that can integrate with MOD applications through its RESTful APIs or Socket connections and provides

a user interface for collecting inputs from users, including simulation boundaries, traffic configuration parameters, and even vehicle plans. Application APIs allow users to apply on-demand changes to the simulation such as changing vehicle parameters, including their destination and adding and removing segments to/from a specific vehicle. Furthermore, this application collects GPS traces of the specified vehicles along with the time. This feature contributes to generating a synthetic tracking database for MOD vehicles.

7.2 Future Work

Although the proposed framework and application can run simulations on specific map sections, its performance may suffer when applied to broader parts of the map. On a personal computer with a Core i7 (10th Generation) CPU and 8 GB of RAM, the suggested system was created and tested. Utilizing cloud-based processors and servers to maximize the performance of the system may be a wise move. Additionally, the performance might be enhanced by spreading out the simulation processing among a number of processing devices. It could be advantageous if we can offer as many APIs as feasible for more dynamic integration. Furthermore, it could be beneficial if the application can support multiple machine learning models to be chosen by the user. Additionally, the graphical user interface (GUI) may be enhanced to accommodate all APIs. A better-designed user interface can assist the application in being more user-friendly and adhering to the finest user experience principles. The ability of the system user to select the simulation engine is another potential improvement. In order to compare the outcomes, the user may select any of the simulation engines that are accessible.

Appendix A

User Guide

This appendix is an instruction to use the VSP and its APIs. In order to run the VSP, it is required to have the required dependencies already installed on the local machine. Table A.1 lists the requirements.

Table A.1: Dependencies

#	Dependency	Description
1	Python 3.9+	The VSP and SUMO are based on Python language and libraries
2	SUMO 1.14.1+	The VSP uses SUMO as a simulation engine.
3	PostgreSQL 10.0+	The VSP stores required data and results into PostgreSQL database

The aforementioned dependencies are third-party applications and libraries that the VSP uses to perform its functions. As these applications are not provided by this thesis, please visit the official websites for detailed installation instructions.

A.1 Create Database

The VSP stores data on the PostgreSQL database. To initialize the database follow these steps:

(1) Create a database in PostgreSQL and name it *vspsumodb*.

(2) Navigate to the root directory of the project and run this script:

```
python data\_access.py
```

A.2 Start VSP Dashboard

The VSP provides a graphical user interface to collect inputs, configure and start the simulation scenarios. To start the dashboard execute the following script:

```
pyhton dashboard.py
```

Figure A.1 shows the VSP dashboard.

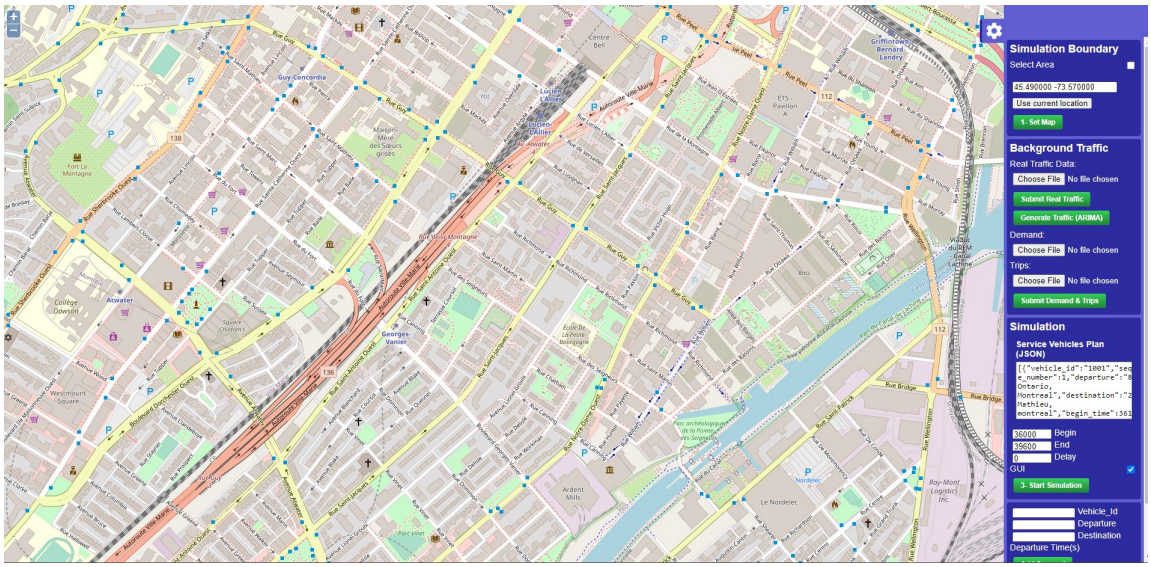

Figure A.1: VSP Dashboard

The *dashboard.py* starts a socket server and loads the user interface HTML page in an available browser. The user interface is connected to the python process using a socket connection. The socket connection sends commands to the server and receives updates.

The VSP dashboard provides 4 primary sections:

- Simulation Boundary
- Background Traffic
- Simulation

- Manage Segments

A.2.1 Simulation Boundary

This section enables users to explore the map, select an area and confirm it as the simulation region. Once the user confirms the area, the VSP downloads the region from OpenStreetMap and converts it to the simulation network.

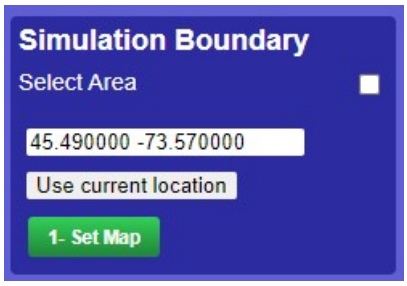

Figure A.2: VSP Dashboard - Simulation Boundary

A.2.2 Background Traffic

Background traffic is generated based on user-defined input files. There are two methods for generating traffic. The ARIMA-based model needs raw traffic files as input. Another method does not use machine learning and is based on demand and trip files. The result of both methods is the background traffic on the map. Traffic density in the simulation region is dependent on the demand values which are defined in hourly time intervals. Figure A.3 is the user interface control that enables the user to define the traffic.

A.2.3 Simulation

This section enables users to change the simulation properties, such as speed, simulation start and end times. Also, it enables users to provide a *JSON* formatted input to define the service vehicle plans. The VSP reads this plan and loads the vehicles accordingly to the simulation.

The JSON input is an **array** of segments. Each segment is formatted as follows:

Input Example:

Figure A.3: VSP Dashboard - Background Traffic

Table A.2: Service Vehicles Input Format

Parameter	Type
vehicle_id	Number
sequence_number	Number
departure	String
destination	String
begin_time	Number

```
[{
    "vehicle_id": 1,
    "sequence_number": 1,
    "departure": "1905 Bassins, Montreal",
    "destination": "1201 rue Guy, montreal ",
    "begin_time": 30100
}]
```

A.2.4 Manage Segments

This section enables users to apply changes to the service vehicles while the simulation is running. If the vehicle is not processed yet or has not reached its destination, the applied changes will take effect once the simulation processes the service vehicle on the map.

Figure A.4: VSP Dashboard - Simulation

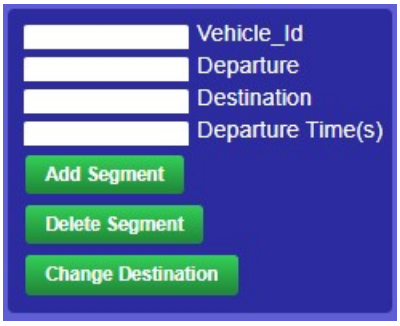

Figure A.5: VSP Dashboard - Manage Segments

A.3 Start VSP API

The VSP features are also available via REST APIs. APIs enable users to integrate the VSP with other mobility-on-demand applications. Applications can send parameters such as GPS coordinates, and trip plans to the VSP and get the simulation results without any further configurations or complexities. To start the API the following script should be executed:

```
python api.py
```

The VSP API starts the server on the local machine on port *5000*. API base address is *http://127.0.0.1:5000/*. APIs have the same functionality as the VSP dashboard, but they are capable of integration with other applications. Inputs and outputs are *JSON* formatted.

A.3.1 Simulation Boundary

This API enables users to set the simulation boundary by providing an array of four GPS coordinates. The VSP calculates the area, downloads it, and prepares the map for the simulation. The following example shows the API address, an input sample, and how to call the API.

API Address: *http://127.0.0.1:5000/map*

Method: *POST*

Sample Input:

```
[ -73.5905993, 45.48301230, -73.54940063, 45.49698683 ]
```

Example API Call:

```
curl --location --request POST 'http://127.0.0.1:5000/map' \
--header 'Content-Type: application/json' \
--data-raw '[ -73.59059936523396, 45.48301230113723, -73.54940063476586,
    45.496986832059754 ]'
```

A.3.2 Background Traffic

This API enables users to configure the background traffic. The higher the input values, the more traffic will run through the map in the simulation. The higher values need more computation performance and might impact the simulation speed.

API Address: *http://127.0.0.1:5000/traffic*

Method: *POST*

Sample Input:

```
{
    "population": 1000,
    "density": 300
}
```

Example API Call:

```
curl --location --request POST 'http://127.0.0.1:5000/traffic' \
--header 'Content-Type: application/json' \
--data-raw '{
```

```
    "population": 1000,
    "density": 300
}'
```

A.3.3 Simulation

This API enables users to configure the simulation parameters such as speed, start, and end time. It also accepts a vehicle plan as input, then processes the vehicles once their departure time is reached.

API Address: *http://127.0.0.1:5000/simulation_start*

Method: *POST*

Sample Input:

```
{
    "begin": "30000",
    "end": "35000",
    "ui_enabled": "true",
    "delay": "0",
    "segments": [
        {
            "vehicle_id": "1",
            "sequence_number": 1,
            "begin_time": 30100,
            "departure": "2132 tupper, montreal",
            "destination": "176 Peel, montreal"
        }
    ]
}
```

Example API Call:

```
curl --location --request POST 'http://127.0.0.1:5000/simulation_start' \
--header 'Content-Type: application/json' \
--data-raw '{
    "begin": "30000",
    "end": "35000",
```

```
    "ui_enabled": "true",
    "delay": "0",
    "segments": [
        {
            "vehicle_id": "1",
            "sequence_number": 1,
            "begin_time": 30100,
            "departure": "2132 tupper, montreal",
            "destination": "176 Peel, montreal"
        }
    ]
}'
```

A.3.4 Add Segment

This API enables users to add a trip segment to a vehicle that is already on the map or waiting to start its trips. The segments received from this API will be stored in the database and once the departure time is reached, the vehicle will process the segment.

API Address: *http://127.0.0.1:5000/segments*

Method: *POST*

Sample Input:

```
{
    "vehicle_id": "1",
    "begin": "100",
    "departure": "4545 monkland",
    "destination": "4040 terrebone"
}
```

Example API Call:

```
curl --location --request POST 'http://127.0.0.1:5000/segments' \
--header 'Content-Type: application/json' \
--data-raw '{
    "vehicle_id": "1",
    "begin": "100",
```

```
    "departure": "4545 monkland",
    "destination": "4040 terrebone"
}'
```

A.3.5 Delete Segment

This API enables users to remove a trip segment. The vehicle will not process this trip anymore because the user has deleted it from the list of trips for the vehicle.

API Address: *http://127.0.0.1:5000/segments*

Method: *DELETE*

Sample Input:

```
{
    "vehicle_id": "1",
    "destination": "4040 terrebone"
}
```

Example API Call:

```
curl --location --request DELETE 'http://127.0.0.1:5000/segments' \
--header 'Content-Type: application/json' \
--data-raw '{
    "vehicle_id": "1",
    "destination": "4040 terrebone"
}'
```

A.3.6 Change Destination

This API enables users to update the destination of a vehicle while it is moving toward its predefined destination. This causes the vehicle to ignore the plan and override its destination. Once the vehicle receives a change destination command, the route will be calculated again and the vehicle follows the new route toward the destination.

API Address: *http://127.0.0.1:5000/destination*

Method: *POST*

Sample Input:

```
{
    "vehicle_id": "sv_1#1",
    "destination": "2132 tupper, montreal"
}
```

Example API Call:

```
curl --location --request POST 'http://127.0.0.1:5000/destination' \
--header 'Content-Type: application/json' \
--data-raw '{
    "vehicle_id": "sv_1#1",
    "destination": "2132 tupper, montreal"
}'
```

A.3.7 Vehicle Status

This API enables users to retrieve the most recent status of the vehicle. It shows the last trip segments that the vehicle has processed and it determines whether the vehicle has reached its destination.

API Address: *http://127.0.0.1:5000/status?vehicle_id={VEHICLE_ID}*

Method: *GET*

Example API Call:

```
curl --location --request GET 'http://127.0.0.1:5000/status?vehicle_id=1' \
--header 'Content-Type: application/json'
```

Sample Output:

```
[{
    "vehicle_id": "1",
    "virtual_vehicle_id": "sv_1#1",
    "status": 3,
    "departure_address": "1905 Bassins, Montreal",
    "destination_address": "1201 rue Guy, montreal "
}]
```

A.4 Simulation Results

A.4.1 Trajectories

The VSP database contains the data collected from the simulation. The *trajectory* table holds GPS coordinates of the service vehicles alongside their timestamp. Therefore, users can investigate the location of the vehicle for a specific time of the simulation. Table A.3 represents the schema of the table which stores trajectories.

Table A.3: Trajectories

Column	Description
id	unique identifier of record
simulation_id	unique identifier of the simulation
vehicle_id	user-defined ID of vehicle
timestamp	time that record is logged
latitude	Vehicle GPS location latitude at the current timestamp
longitude	Vehicle GPS location longitude at the current timestamp

Sample Data Query:

```
SELECT * FROM public.trajectory
ORDER BY id ASC LIMIT 100
```

A.4.2 Segment Status

The VSP stores all segments in the database and keeps track of every segment while the simulation is running. Once a service vehicle reaches its destination, the VSP updates the segment status in *Segment* table. It enables users to retrieve the latest status of vehicles for each segment. Table A.4 shows the table schema for the service vehicle segments.

Sample Data Query:

```
SELECT * FROM public.segment
ORDER BY id DESC LIMIT 100
```

Table A.4: Service Vehicle Trip Segments

Column	Description
id	unique identifier of record
simulation_id	unique identifier of the simulation
virtual_vehicle_id	combination of vehicle and its segment
vehicle_id	user-defined ID of vehicle
departure_address	Text-based address of departure
destination_address	Text-based address of destination
begin_time	the vehicle starts moving at this time (in seconds)
sequence_number	defines the order of segment
status	Status of vehicle. 0: Not inserted, 1: Inserted to map, 3: Arrived
departure_edge_id	identifier of road in simulation network
destination_edge_id	identifier of road in simulation network

A.4.3 SUMO Specific Outputs

As the VSP is designed based on the SUMO simulation engine, the users can also benefit from the outputs provided by the SUMO engine. These files will be generated and stored under the following path:

```
{PROJECT_ROOT}\data\output.tripinfo.xml
```

Accessing and reading XML files cannot be integrated into integrated applications. Therefore, the VSP provides an API that reads the XML outputs and populates the results in JSON format. Because the API could be easily accessed via integrated applications, and the output is formatted in JSON, the integrated application can take the advantage of investigating simulation results with minimum effort. Table A.5 describes the output.

API Address: *http://127.0.0.1:5000/results*

Method: *GET*

Example API Call:

```
curl --location --request GET 'http://127.0.0.1:5000/results'
```

Sample Output:

```
[
    {
        "arrival": 40263.0,
```

Table A.5: Service Vehicle Trip Results Format

JSON Key	**Value Description**
id	unique identifier of vehicle_segment
depart	The real departure time (the time the vehicle was inserted into the network)
departLane	The id of the lane the vehicle started its journey
arrival	The time the vehicle reached its destination
arrivalLane	The id of the lane the vehicle was on when reaching its destination
duration	The time the vehicle needed to accomplish the route
routeLength	The length of the vehicle's route
rerouteNo	The number the vehicle has been rerouted
waitingTime	The time in which the vehicle speed was below or equal 0.1 m/s


```
        "arrivalLane": "455722372#0_1",
        "depart": 40100.0,
        "departLane": "25367003#1_1",
        "duration": 163.0,
        "id": "sv_9#1",
        "routeLength": 1192.14
    },
    {
        "arrival": 40246.0,
        "arrivalLane": "329694555#6_1",
        "depart": 40000.0,
        "departLane": "20170335#1_1",
        "duration": 246.0,
        "id": "sv_8#1",
        "routeLength": 1426.04
    },
    {
        "arrival": 30964.0,
        "arrivalLane": "-20112792#4_1",
        "depart": 30600.0,
        "departLane": "975251638_1",
        "duration": 364.0,
        "id": "sv_6#1",
```

```
        "routeLength": 3152.24
    }
]
```

Appendix B

Test Case Scenario

Based on the interviews and the meetings with the MOD research teams, we collected the requirements to design and implement a simulation platform which facilitates the design and execution of MOD simulations. The proposed application takes user inputs, generates background traffic, generates service vehicles, executes the simulation, applies on-demand changes, and produces the simulation results. To test the available features of the application, we designed a test scenario consisting of steps for a complete simulation execution. In this test we follow these steps:

(1) Start the VSP dashboard panel

(2) View, explore, or search the map

(3) Specify an area on the map for simulation

(4) Configure background traffic

(5) Define service vehicle trip segments

(6) Define begin and end time of the simulation

(7) Start simulation

(8) Access results

Inputs that are being used in the test case scenario are as follows:

Table B.1: Test case scenario inputs

Map Coordinates	Montreal Downtown selection on the map
Traffic density	Dynamic based on input demand file
Time of Day	10:00 AM - 11:00 AM

B.1 Start Application

To start the application, we provided a batch file (.bat) that runs the application servers. **app.bat** installs the application packages, executes the application, and opens a web browser showing the map explorer and available controllers. B.1 shows the user interface of the application. The user interface consists of a map explorer and a side control bar that enables users to provide the inputs and configure the simulator settings.

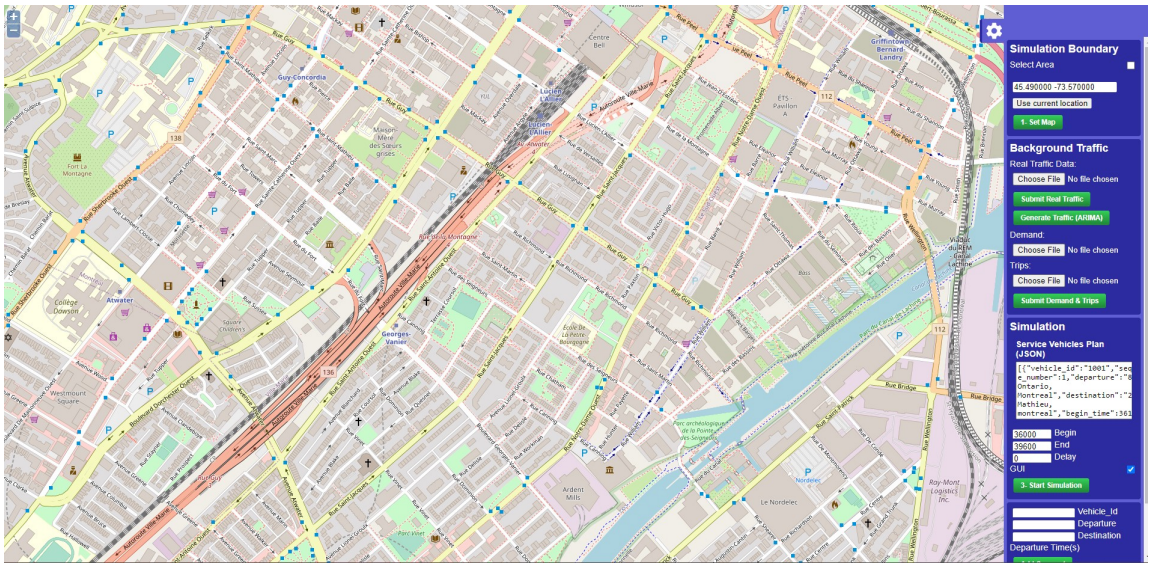

Figure B.1: Application user interface

B.2 Set simulation boundaries (Map)

The first step of the configuration is the map setup. Users can explore the map using the map explorer in the application. Then, by checking the *Select Area* check box, a polygon is drawn on the map that enables the user to select an area on the map for the simulation. By clicking on the

Set Map button, the application downloads the selected area data from OpenStreetMap and converts the map to a simulation network supported by SUMO. B.2 shows the map control. The downloaded and converted files are stored on the disk for further processing steps. The next step is configuring the background traffic.

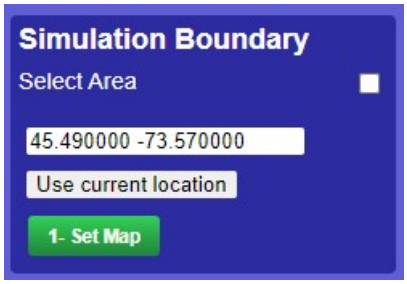

Figure B.2: Map controls on user interface

For this test case scenario, we focused on Downtown Montreal. We used the map explorer to select this area and used the provided button to set this area as simulation boundaries. Figure B.3 shows the result of the conversion of the map to the simulation network. This network is recognizable by the SUMO application and could be used for further processing.

Figure B.3: Selected Boundaries for Test Case. Downtown Montreal

B.3 Configure background traffic

In this step, the user can follow two scenarios to set up the background traffic.

(1) Raw data

(2) Processed Data

B.3.1 Raw data

In this approach, the application requires a raw trip file in .csv format. This file is a collection of trips either inside or outside of the selected region. Table B.2 shows the format of the file and required columns.

Table B.2: Raw trips data format

Column Name	**Description**
trip_id	unique identification of the trip
starttime	the start date and time of the trip
start_lon	departure location longitude
start_lat	departure location latitude
end_lon	destination location longitude
end_lat	destination location latitude

Users can upload the .csv file using the SUBMIT REAL TRAFFIC button on the user interface. Then, by clicking on the GENERATE TRAFFIC (ARIMA), the application starts processing the input file based on the ARIMA machine learning model. Two files of demand data and trip data are generated by using this approach. Then, these files will be used for generating the background traffic while the simulation is running. In this approach, it is considered that the user intends to generate demand based on the ARIMA model which is the algorithm that is used in this thesis. For our test case, we used data from a survey provided by Montreal City Authorities (*Montreal Trajectories*, 2018). This data set includes departure, destination and start time of trips within Montreal. For the purpose of our study, we preprocessed data and removed non-usable columns. The application automatically filters the trips that are within the selected region.

B.3.2 Processed Data

The proposed application is able to accept user-defined demand and trip files. In this approach, the application does not execute the ARIMA model and by using the provided inputs, generates the background traffic. To follow this approach, users need to choose trip and demand files using the file controls in the dashboard and submit the files by clicking on the SUBMIT DEMAND & TRIPS button. Figure B.4 shows the control used for uploading the demand and trip files.

Figure B.4: Background traffic dashboard control

B.4 Set service vehicles

In this section, users can provide the service vehicles plan. The service vehicle is a MOD vehicle that will be monitored by the application and its trajectories will be stored in the database. The input should be formatted as JSON. The data format is described in Table B.3.

The service vehicles are designed to have departure and destinations according to Table B.4. Also, their departure time is specified in the design.

The VSP dashboard and API, accept JSON inputs. So, after converting the service vehicles plan to a JSON formatted string, we used the dashboard to set the service vehicles and start the simulation. Next, we should set the beginning and end times for simulations. These times are in seconds. For example, for the simulation of a scenario between 10:00 AM and 11:00 AM, the values

Table B.3: Service vehicle plan json data format

Property Name	Data Type	Description
vehicle_id	String	unique identification of the vehicle
sequence_number	Integer	the sequence number of the segment for a vehicle plan
departure	String	departure address
destination	String	destination address
begin_time	Integer	the time in seconds that vehicle starts the segment

are 36000 and 39600. Once we set all the configurations, we can start the simulation by clicking on the START SIMULATION button. The application processes the inputs and generates configuration files for the SUMO simulation software and while the SUMO is running the application monitors and stores the required parameters of the service vehicles. Figure B.5 shows the control section that we can use for setting up the times and service vehicles.

Figure B.5: simulation start dashboard control

According to these steps, we explored the map using the dashboard and specified an area of the city of Montreal. Then, based on the available demographic data, we set the population and density. These values affect the amount of activity on the simulation map. Then, we chose two random points on the map for a specific vehicle. Our expected result is as follows:

- Vehicle GPS traces should be logged to the database (trajectories)

Table B.4: Test case service vehicles

Vehicle ID	Departure	Destination	Departure Time
1	”1905 Bassins, Montreal”	”1201 rue Guy, Montreal ”	30100
2	”1280 Rue Saint-Jacques, Montreal ”	”2206 Rue Quesnel, Montreal”	30300
3	”1944 Av. Lionel-Groulx, Montreal”	”2150 Tupper, Montreal”	30400
4	”3019 Sherbrooke W, Montreal”	”1402 René-Lévesque Ouest, Montreal”	30500
5	”2530 Rue Saint-Antoine Ouest, Montreal”	”151 Rue du Séminaire, Montreal”	30600
6	”1487 Argyle, Montreal”	”282 Rue de la Montagne, Montreal”	40100
7	”1905 Bassins, Montreal”	”282 Rue de la Montagne, Montreal”	40200
8	”1944 Av. Lionel-Groulx, Montreal”	”151 Rue du Séminaire, Montreal”	40500
9	”1201 rue Guy, Montreal ”	”151 Rue du Séminaire, Montreal”	40600
10	”2206 Rue Quesnel, Montreal”	”2060 Tupper, Montreal”	40800
11	”1402 René-Lévesque Ouest, montreal”	”151 Rue du Séminaire, Montreal”	41000

- Vehicle should begin to move at the specified time and find a route to reach its destination
- The travel time should be stored as a result of the simulation.

Once we started the simulation we could identify the vehicle on the map B.6. At each simulation step, the VSP core stored the GPS location of the vehicle in the database. Furthermore, we could access the vehicle's travel time once it reached its destination. I used the VSP APIs to access results and received the JSON formatted response.

Table B.5 demonstrates the results of the simulation regarding the travel time of each vehicle. The distance between every departure and destination pair is the Route length which is presented in Meters. The travel time shows the amount of time it took for the vehicle to reach the designated destination with traffic conditions that have been configured in the initialization step. The test case inputs could be changed to investigate different traffic scenarios with various service vehicle plans.

Figure B.6: SUMO Running Scenario

Table B.5: Simulation Scenario Results

Vehicle ID	Route Length (m)	Travel Time (s)
1	692.36	106
2	1851.87	281
3	2000.50	207
4	2137.27	277
5	3152.24	332
6	1426.04	252
7	1192.14	153
8	984.73	134
9	1332.18	189
10	933.42	126
11	2139.62	219

References

Abohashima, H. S., Gheith, M., & Eltawil, A. (2019). Solving the urban traffic lights scheduling problem using discrete event simulation and design of experiments. In *Proc. int. conf. comput. ind. eng. cie* (Vol. 2019).

Afeche, P., Liu, Z., & Maglaras, C. (2018). Ride-hailing networks with strategic drivers: The impact of platform control capabilities on performance. *Rotman School of Management Working Paper*(3120544), 18–19.

Ashkrof, P., de Almeida Correia, G. H., Cats, O., & van Arem, B. (2020). Understanding ride-sourcing drivers' behaviour and preferences: Insights from focus groups analysis. *Research in Transportation Business & Management*, *37*, 100516.

Bando, M., Hasebe, K., Nakayama, A., Shibata, A., & Sugiyama, Y. (1995). Dynamical model of traffic congestion and numerical simulation. *Physical review E*, *51*(2), 1035.

Barata, J. C., Lisboa, D., Bastos, L. C., & Neto, A. (2022). Agile requirements engineering practices: a survey in brazilian software development companies. *arXiv preprint arXiv:2202.12956*.

Behrisch, M., Bieker, L., Erdmann, J., & Krajzewicz, D. (2011). Sumo–simulation of urban mobility: an overview.

Berger, T., Chen, C., & Frey, C. B. (2018). Drivers of disruption? estimating the uber effect. *European Economic Review*, *110*, 197–210.

Bi, H., Mao, T., Wang, Z., & Deng, Z. (2019). A deep learning-based framework for intersectional traffic simulation and editing. *IEEE Transactions on Visualization and Computer Graphics*, *26*(7), 2335–2348.

Bischoff, J., Maciejewski, M., & Nagel, K. (2017). City-wide shared taxis: A simulation study in berlin. In *2017 ieee 20th international conference on intelligent transportation systems (itsc)* (pp. 275–280).

Bokányi, E., & Hannák, A. (2020). Understanding inequalities in ride-hailing services through simulations. *Scientific reports*, *10*(1), 1–11.

Chao, Q., Bi, H., Li, W., Mao, T., Wang, Z., Lin, M. C., & Deng, Z. (2020). A survey on visual traffic simulation: Models, evaluations, and applications in autonomous driving. In *Computer graphics forum* (Vol. 39, pp. 287–308).

Chao, Q., Deng, Z., Ren, J., Ye, Q., & Jin, X. (2017). Realistic data-driven traffic flow animation using texture synthesis. *IEEE transactions on visualization and computer graphics*, *24*(2), 1167–1178.

Chen, L., Mislove, A., & Wilson, C. (2015). Peeking beneath the hood of uber. In *Proceedings of the 2015 internet measurement conference* (pp. 495–508).

Chen, X. M., Zheng, H., Ke, J., & Yang, H. (2020). Dynamic optimization strategies for on-demand ride services platform: Surge pricing, commission rate, and incentives. *Transportation Research Part B: Methodological*, *138*, 23–45.

Chen, Y., Lv, Y., Wang, X., & Wang, F.-Y. (2018). Traffic flow prediction with parallel data. , 614–619.

Codecá, L., Erdmann, J., Cahill, V., & Harri, J. (2020). Saga: An activity-based multi-modal mobility scenario generator for sumo. In *Sumo user conference 2020, virtual conference.*

Codeca, L., Frank, R., & Engel, T. (2015). Luxembourg sumo traffic (lust) scenario: 24 hours of mobility for vehicular networking research. In *2015 ieee vehicular networking conference (vnc)* (pp. 1–8).

Cui, Y., Makhija, R. S. M. S., Chen, R. B., He, Q., & Khani, A. (2021). Understanding and modeling the social preferences for riders in rideshare matching. *Transportation*, *48*(4), 1809–1835.

Holler, J., Vuorio, R., Qin, Z., Tang, X., Jiao, Y., Jin, T., ... Ye, J. (2019). Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. In *2019 ieee international conference on data mining (icdm)* (pp. 1090–1095).

Hoogendoorn, S. P., & Bovy, P. H. (2001). Generic gas-kinetic traffic systems modeling with

applications to vehicular traffic flow. *Transportation Research Part B: Methodological*, *35*(4), 317–336.

Kogler, C., Rauch, P., et al. (2018). Discrete event simulation of multimodal and unimodal transportation in the wood supply chain: a literature review. *Silva Fenn*, *52*(4), 29.

Kucharski, R., & Cats, O. (2020a). Exact matching of attractive shared rides (exmas) for system-wide strategic evaluations. *Transportation Research Part B: Methodological*, *139*, 285–310.

Kucharski, R., & Cats, O. (2020b). Maassim–agent-based two-sided mobility platform simulator. *arXiv preprint arXiv:2011.12827*.

Kulkarni, V., & Garbinato, B. (2017). Generating synthetic mobility traffic using rnns. , 1–4.

Li, W., Wolinski, D., & Lin, M. C. (2017). City-scale traffic animation using statistical learning and metamodel-based optimization. *ACM Transactions on Graphics (TOG)*, *36*(6), 1–12.

Lighthill, M. J., & Whitham, G. B. (1955). On kinematic waves ii. a theory of traffic flow on long crowded roads. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, *229*(1178), 317–345.

Liu, Y., Xie, J., & Chen, N. (2021). Offline-online approximate dynamic programming for stochastic carsharing systems with relocation incentives. *Available at SSRN 3882532*.

Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., ... Wießner, E. (2018). Microscopic traffic simulation using sumo. In *The 21st ieee international conference on intelligent transportation systems*. IEEE. Retrieved from https://elib.dlr.de/124092/

Mitchell, W. J., Borroni-Bird, C. E., & Burns, L. D. (2010). *Reinventing the automobile: Personal urban mobility for the 21st century*. MIT press.

Montreal trajectories. (2018). City of Montreal. Retrieved from https://donnees.montreal.ca/ville-de-montreal/mtl-trajet#territories

Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., & Damas, L. (2013). Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, *14*(3), 1393–1402.

Mumford, E. (1985). Defining system requirements to meet business needs: a case study example. *The Computer Journal*, *28*(2), 97–104.

Nourinejad, M., & Ramezani, M. (2020). Ride-sourcing modeling and pricing in non-equilibrium two-sided markets. *Transportation Research Part B: Methodological*, *132*, 340–357.

Pavone, M. (2015). Autonomous mobility-on-demand systems for future urban mobility. In *Autonomes fahren* (pp. 399–416). Springer.

Pelekis, N., Sideridis, S., Tampakis, P., & Theodoridis, Y. (2015). Hermoupolis: a semantic trajectory generator in the data science era. *SIGSPATIAL Special*, *7*(1), 19–26.

Richards, P. I. (1956). Shock waves on the highway. *Operations research*, *4*(1), 42–51.

Sapre, V., Kalambur, S., Sitaram, D., & Bastian, R. (2018). Synthetic generation of traffic data for urban mobility. , 2151–2157.

Sewall, J., Wilkie, D., & Lin, M. C. (2011). Interactive hybrid simulation of large-scale traffic. In *Proceedings of the 2011 siggraph asia conference* (pp. 1–12).

Sewall, J., Wilkie, D., Merrell, P., & Lin, M. C. (2010). Continuum traffic simulation. In *Computer graphics forum* (Vol. 29, pp. 439–448).

Shaheen, S., Cohen, A., Yelchuru, B., Sarkhili, S., Hamilton, B. A., et al. (2017). Mobility on demand operational concept report.

Shen, J., & Jin, X. (2012). Detailed traffic animation for urban road networks. *Graphical Models*, *74*(5), 265–282.

Song, H., & Min, O. (2018). Statistical traffic generation methods for urban traffic simulation. In *2018 20th international conference on advanced communication technology (icact)* (pp. 247–250).

Statistics Canada, . C. o. P. (2022). Statistics canada. 2022. (table). census profile, 2021 census of population, statistics canada catalog no. 98-316-x2021001. ottawa.

Treiber, M., & Helbing, D. (2001). Microsimulations of freeway traffic including control measures.

Un population projection. (2018). United Nations. Retrieved from https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html

Varga, A. (2001). Discrete event simulation system. , 1–7.

Wang, X., Liu, W., Yang, H., Wang, D., & Ye, J. (2019). Customer behavioural modelling of order cancellation in coupled ride-sourcing and taxi markets. *Transportation Research Procedia*,

38, 853–873.

Wang, Y., Lin, X., Wei, H., Wo, T., Huang, Z., Zhang, Y., & Xu, J. (2019). A unified framework with multi-source data for predicting passenger demands of ride services. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *13*(6), 1–24.

Xie, H., Tanin, E., Ramamohanarao, K., Karunasekera, S., Kulik, L., Zhang, R., & Qi, J. (2019). Generating traffic data for any city using smarts simulator. *SIGSPATIAL Special*, *11*(1), 22–28.

Xu, Z., Yin, Y., & Ye, J. (2020). On the supply curve of ride-hailing systems. *Transportation Research Part B: Methodological*, *132*, 29–43.

Zambrano, J. L., Calafate, C. T., Soler, D., Cano, J.-C., & Manzoni, P. (2016). Using real traffic data for its simulation: Procedure and validation. In *2016 intl ieee conferences on ubiquitous intelligence & computing, advanced and trusted computing, scalable computing and communications, cloud and big data computing, internet of people, and smart world congress (uic/atc/scalcom/cbdcom/iop/smartworld)* (pp. 161–170).

Zardini, G., Lanzetti, N., Pavone, M., & Frazzoli, E. (2022). Analysis and control of autonomous mobility-on-demand systems. *Annual Review of Control, Robotics, and Autonomous Systems*, *5*, 633–658.

Zha, L., Yin, Y., & Du, Y. (2017). Surge pricing and labor supply in the ride-sourcing market. *Transportation Research Procedia*, *23*, 2–21.

Zhang, H., Feng, S., Liu, C., Ding, Y., Zhu, Y., Zhou, Z., … Li, Z. (2019). Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. , 3620–3624.

Zuniga-Garcia, N., Tec, M., Scott, J. G., Ruiz-Juri, N., & Machemehl, R. B. (2020). Evaluation of ride-sourcing search frictions and driver productivity: A spatial denoising approach. *Transportation Research Part C: Emerging Technologies*, *110*, 346–367.