

**LAST-MILE ROUTE OPTIMISATION WITH MACHINE LEARNING**

By

CHAN TZE KEET

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

**BACHELOR OF COMPUTER SCIENCE (HONOURS)**

Faculty of Information and Communication Technology  
(Kampar Campus)

JUNE 2024

## REPORT STATUS DECLARATION FORM

Title: LAST-MILE ROUTE OPTIMISATION  
USING MACHINE LEARNING

Academic Session: June 2024

I CHAN TZE KEET  
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in  
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.



(Author's signature)

Verified by,



(Supervisor's signature)

Address:

58, Jalan Pasir Kuning,

Taman Kaya Shatin,

31650 Ipoh, Perak

Ms Tseu Kwan Lee

Supervisor's name

Date: 11/09/2024

Date: 11/09/2024

<b>Universiti Tunku Abdul Rahman</b>			
Form Title : <b>Sample of Submission Sheet for FYP/Dissertation/Thesis</b>			
Form Number: <b>FM-IAD-004</b>	Rev No.: <b>0</b>	Effective Date: <b>21 JUNE 2011</b>	Page No.: <b>1 of 1</b>

**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

**UNIVERSITI TUNKU ABDUL RAHMAN**

Date: 11/09/2024

**SUBMISSION OF FINAL YEAR PROJECT**

It is hereby certified that CHAN TZE KEET (ID No: 20ACB04193 ) has completed this final year project entitled “ LAST-MILE ROUTE OPTIMISATION WITH MACHINE LEARNING ” under the supervision of MS. TSEU KWAN LEE (Supervisor) from the Department of Computer Science , Faculty of Information And Communication Technology.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.


Yours truly,



\_\_\_\_\_  
(CHAN TZE KEET)

## DECLARATION OF ORIGINALITY

I declare that this report entitled “**LAST-MILE ROUTE OPTIMISATION WITH MACHINE LEARNING**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : 

Name : CHAN TZE KEET

Date : 11-09-2024

## **ACKNOWLEDGEMENTS**

I would like to express my sincere thanks and appreciation to my supervisor, Ms. Tseu Kwan Lee, who has given me this bright opportunity to engage in a route optimisation project, utilising machine learning techniques. Thank you for the guidance, patience, and understandings throughout the project. This project provides me the opportunity to conduct extensive research on existing real-world application of machines learning techniques, to be specific, logistics field. A million thanks to you. Besides, I also would like to thank my parents and my family for their love, support, and continuous encouragement throughout the course.

## **ABSTRACT**

Ever since COVID-19 pandemic, online shopping had been skyrocketed. To handle the enormous volume of deliveries, last-mile delivery route planning and optimization had become more significant than ever for logistics services. Last-mile logistics are referring to the final stage of the delivery process, where goods are transported from a distribution hub to the end destination, typically a residential or commercial address. Last-mile logistics had always been the costliest part in the overall supply chain. Numerous last-mile route optimization models/frameworks are proposed and been practiced in logistics services, to reduce operation costs while attempt to fulfill customers' satisfaction. However, existing pure optimization frameworks often overlooked that in real-world practices, the prescribed routes may be not followed by delivery drivers, as they may prioritize personal knowledges and experiences. Deviation of prescribed delivery routes by delivery drivers may be due to various underlying reasons, including but not limited to traffics conditions, and customers' preferences. In this project, we proposed a Simple R-NN model to uncover the underlying relationship/pattern between customers' acceptable delivery time windows and deviations of prescribed delivery routes by drivers. The proposed model, Simple R-NN model aims to predicts possible delivery routes by drivers, then output an optimized delivery route that seems acceptable for the drivers to actual adapts in actual delivery operation.

# TABLE OF CONTENTS

<b>TITLE PAGE</b>	<b>i</b>
<b>REPORT STATUS DECLARATION FORM</b>	<b>ii</b>
<b>DECLARATION OF ORIGINALITY</b>	<b>iv</b>
<b>ACKNOWLEDGEMENTS</b>	<b>v</b>
<b>ABSTRACT</b>	<b>vi</b>
<b>TABLE OF CONTENTS</b>	<b>vii</b>
<b>LIST OF FIGURES</b>	<b>x</b>
<b>LIST OF TABLES</b>	<b>xii</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xiii</b>
<b>CHAPTER 1 PROJECT BACKGROUND</b>	<b>1</b>
1.1 Introduction	1
1.2 Problem Statement and Motivation	2
1.3 Research Objectives	3
1.4 Project Scope	4
1.5 Contributions	4
1.6 Report Organisation	4
<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>5</b>
2.1 Previous Works	5
2.1.1 Delivery route prediction using machine learning model	5
2.1.2 Delivery route optimization	10
2.2 Results	11
2.3 Summarization of findings	13
2.4 Proposed Method	15
<b>CHAPTER 3 SYSTEM METHODOLOGY</b>	<b>16</b>
3.1 Sequence-to-sequence (seq2seq) modelling framework	16
3.2 Simple RNN Encoder-Decoder (Simple RNN E-D) Model	16

3.2.1	Simple RNN encoder	17
3.2.2	Simple RNN decoder	18
3.3	LSTM Encoder-Decoder with Pair-Wise Attention Model	19
3.3.1	LSTM encoder	20
3.3.2	LSTM decoder	20
3.3.3	Pair-Wise Attention Layer	21
<b>CHAPTER 4 EXPERIMENTAL SETUP</b>		<b>23</b>
4.1	System Requirements	23
4.1.1	Hardware	23
4.1.2	Software	23
4.1.3	Data Source	24
4.2	System Design	29
4.2.1	Data Analysis	29
4.2.2	Data Preprocessing	30
4.2.3	Model Building and Training	31
4.2.4	Model Evaluation	31
4.3	Timeline	32
<b>CHAPTER 5 SYSTEM IMPLEMENTATION</b>		<b>35</b>
5.1	Data Findings	35
5.2	Data Preprocessing	46
5.3	Data Transformation	55
5.3.1	Data Transformation	55
5.3.2	Padding	57
5.4	Model Building	60
5.4.1	Simple RNN Encoder-Decoder	60
5.4.2	LSTM Encoder-Decoder with Attention	64
5.5	Model Training	72
5.6	Model Evaluation	78
5.7	Result Visualization	88
5.8	Implementation Issues and Challenges	97



<b>CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION</b>	<b>98</b>
6.1 Driver Behaviours in Last Mile Delivery	98
6.2 Model Performance	100
6.2.1 Performance metrics	100
6.2.2 Train Result Evaluation	102
6.2.3 Cross-Validation Results	104
6.2.4 Test Result Evaluation	106
6.2.5 Benchmarking	108
6.3 Objective Evaluation	109
<b>CHAPTER 7 CONCLUSION</b>	<b>110</b>
<b>REFERENCES</b>	<b>111</b>
<b>APPENDIX</b>	
A1 Weekly Report	A-1
A2 Poster	A-7
<b>PLAGARISM CHECK RESULT</b>	
<b>CHECK LISTS</b>	

## LIST OF FIGURES

<b>Figure Number</b>	<b>Title</b>	<b>Page</b>
Figure 2.1.1	Overall framework of the route prediction model by [3].	6
Figure 2.1.2	Tailored chain-reaction-based algorithm, proposed by [3].	6
Figure 2.1.3	Overall architecture of attention-based pointer NN, proposed by [5].	7
Figure 2.1.4	Route Sequence Inference algorithm, by [5].	8
Figure 2.1.5	Features applied in inter-zone sequence prediction, by [4].	8
Figure 2.1.6	Architecture model of feedforward NN model, proposed by [4].	9
Figure 2.1.7	Best Hyperparameters found for NN model, using ASHA algorithm.	9
Figure 2.2.1	Pair-wise Attention-based NN model Performance Table.	12
Figure 3.2	Overall architecture of Simple RNN E-D Model	17
Figure 4.1.1	ERD for Amazon Last-Mile Routing Research Challenge Dataset .	25
Figure 4.1.2	Sample Delivery Route, in Irvine, California.	26
Figure 4.2.1	Overall project framework.	29
Figure 4.2.2	Steps involved in Data Preprocessing.	30
Figure 4.3.1	Gantt Chart for Project 1.	33
Figure 4.3.2	Gantt Chart for Project 2.	34
Figure 5.1.1	Range of delivery package without time windows for each route.	37
Figure 5.1.2	Distribution of Number of Packages with Time Window per Route	38
Figure 5.1.3	Range of delivery package for each route.	38
Figure 5.1.4	Distribution of Number of Stops per Route.	39
Figure 5.1.5	Distribution of Number of Zone per Route.	40

<b>Figure Number</b>	<b>Title</b>	<b>Page</b>
Figure 5.1.6	Function for Extraction Zone Data Features.	41
Figure 5.1.7	Zone ID Order sequence, aggregated from stop sequence for Route1	42
Figure 5.1.8	Zone ID Order sequence, aggregated from stop sequence for Route2	43
Figure 5.1.9	Zone ID Order sequence for Route0	44
Figure 5.2.1	Plot Graph for predicted vs actual at Route 821	89
Figure 5.2.2	Plot Graph for predicted vs actual at Route 27	89
Figure 5.2.3	Actual Route0 shown using Leaflet	93
Figure 5.2.4	Predicted Route0 shown using Leaflet	94
Figure 5.2.5	Actual Route20 shown using Leaflet	95
Figure 5.2.6	Predicted Route20 shown using Leaflet	96
Figure 6.1	Disparity Score Distribution of Simple RNN E-D	103
Figure 6.2	Disparity Score Distribution of LSTM E-D with Attention	104
Figure 6.3	Disparity Score Distribution of Simple RNN E-D	107
Figure 6.4	Disparity Score Distribution of LSTM E-D with Attention	107

## LIST OF TABLES

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
Table 2.3.1	Summary of Route Prediction and Optimization Studies.	13
Table 4.1.1	Specifications of laptop.	23
Table 4.1.2	Data Description on provided Amazon Last-Mile Routing Research Challenge Dataset	27
Table 5.1	Zone Order sequence for Route0, arranged by cluster	45
Table 6.1	Model Performance on Training Data	102
Table 6.2	Simple RNN E-D Cross Validation Results	105
Table 6.3	LSTM E-D with Attention Cross Validation Results	105
Table 6.4	Model Performance on Test Data	106
Table 6.5	Model Performance with Benchmark model	108

## LIST OF ABBREVIATIONS

<i>TSP</i>	Travelling Salesman Problem
<i>VRP</i>	Vehicle Routing Problem
<i>TSPTW</i>	Travelling Salesman Problem with Time Windows
<i>TSPTW-Dev</i>	Travelling Salesman Problem with Time Windows and Deviation
<i>VNS</i>	Variable Neighborhood Search
<i>ML</i>	Machine Learning
<i>NLP</i>	Natural Language Processing
<i>NN</i>	Neural Network
<i>ReLU</i>	Rectified Linear
<i>ALMRRC</i>	Amazon Last-Mile Routing Research Challenge
<i>API</i>	Application Programming Interface
<i>US</i>	United States
<i>ERD</i>	Entity Relationship Diagram
<i>LSTM</i>	Long Short-Term Memory
<i>GIGO</i>	Garbage In Garbage Out
<i>FYP</i>	Final Year Project

# CHAPTER 1

## Project Background

### 1.1 Introduction

In the intricate dance of supply chains, last-mile delivery is where the grand finale unfolds. It's the moment when a product or goods, routed through factories, warehouses, and highways, finally reaches to their end users. While remaining as the cost driver of the overall supply chain, a continuous rise demanding for last-mile delivery operations are observed. [1] suggested that the global parcel volume is projected to hit a staggering 200 billion by 2025. Moreover, last-mile delivery also accountable to customer satisfactory towards both the business and partnering logistics company. Nowadays, most online customers are expecting swifter yet reliable deliveries. In short, the study of optimization on last-mile logistics are pivotal for the growth of business and then overall economy.

Travelling Salesman Problem (TSP) is a classic combinatorial optimization problem in mathematics and computer science, where given a scenario of a salesperson are tasked to visit a set of cities (location points) and returns to the starting point while covering the shortest possible route. In the context of last-mile delivery, this translates to finding the most efficient path for a delivery agent to serve multiple customers and return to the delivery centre. For a long time, TSP models had been widely practiced optimizing traditional logistics criteria like overall travel time [2].

Despite that, [3] points out that, real-world delivery route optimization goes beyond only identifying the shortest delivery route. Real-world factors, including but not limited only to traffics, parking, as well as customer delivery preferences should be considered in last-mile delivery route optimization. [3], [4] also mentioned that by leveraging drivers' delivery routes patterns, it may have a positive influence on real-world delivery route optimization.

### 1.2 Problem Statement and Motivation

Inefficient deliveries in the last mile of the supply chain had grown significant concern over time for businesses across the world, as possible leads to collapsed operations, increased in delivery costs, etc.

In recent years, extensive research on last-mile delivery route optimization had been done, covering approaches such as TSP or vehicle routing problem (VRP), in regards of both computational costs and solution quality [5]. Most of the corresponding solutions prioritize optimization on traditional logistics factors such as depot. However, the studies often overlook the driver on-road knowledge and behavior, the often-underestimated attributes in the optimization equation [4]. This is supported by [3], [4], [5], where the studies mentioned that, in practice, most of the drivers tend to deviate from provided optimal routes, prioritizing to drivers' personal knowledge and daily experience gained on the delivery area. When considering of on-road factors, such as knowledge on temporal traffic/road conditions, drivers' deviations may lead to possible profit gain in operation. On the contrary, given drivers' actual routes inferior to suggested routes may lead to operating loss. According to a study on daily commuting habits of drivers in both Japan and the Philippines conducted by [6], they concluded that drivers tend to deviate from recommendations routes by the navigation system in favor of familiar routes. Still, drivers' deviation of prescribed route had raised uncertainties in context of last-mile route optimization.

In the meantime, to provide quality-of-life improvements to their customers, logistics providers nowadays do provide preferred delivery time window options for customers. However, having delivery time preferences add complexity into last-mile route optimization. To demonstrate, given a delivery zone, there will be various delivery time preferences by the customers. In a business standpoint, other than minimizing operational costs, maximizing customers satisfactions are key in organizational success.

In [3], [4], [5], using machine learning models approach, by learning drivers' delivery route pattern, to achieve last-mile delivery optimization while allowing drivers' deviations on prescribed route, to some extent. However, with further studies, on the effects of real-world factor (customer preferences/acceptable delivery time

window) on delivery drivers' deviations on prescribed route, may provide useful insights for stakeholders, improve state-of-the-art last-mile route optimization system.

The project aims to introduce a novel model for driver's delivery route predictions and optimizations used in last-mile delivery, using machine learning technique. The motivation of this projects is to continuously optimize the as-is last-mile delivery frameworks and increase efficiency of overall supply chain. To achieve such goal, the expected outcome in this project is to output a drivers' delivery routes prediction and optimization model, based on the driving pattern from historical delivery data. From the model's output, then imply the underlying factor, specifically customers' acceptable delivery time windows, and its effects on drivers' deviation of prescribed routes.

### **1.3 Research Objectives**

The aim of the thesis is to explore the effects of real-world factors on possible delivery routes deviations by drivers. The ultimate intention of the thesis is to enable effective minimizing operation costs in real-world last-mile logistics, while maximizing business profit and growth from achieving high customer satisfactory.

The specific research objectives of last-mile route optimization model are:

- (i) To study and derives the possible factors affecting deviation of delivery routes by drivers (i.e., customer acceptable delivery time windows) from the actual drivers' delivery routes patterns.
- (ii) To proposed Simple Recurrent Neutral Network (Simple-RNN) for this project that able to identify the drivers' delivery routes pattern and performs routes prediction and optimization tasks.
- (iii) To evaluate the proposed solution in terms of disparity performance metrics.



### **1.4 Project Scope**

The model proposed will be developed in Python programming language on Google Collaboratory. In this project, machine learning techniques will be utilised to learn drivers' delivery pattern from historical real-world data, to output a prediction on drivers' possible delivery routes for last-mile delivery. From the model's output, we then study and identify relationships between customer acceptable delivery time windows and the drivers' delivery routes patterns. By having better understanding on the effects of real-world factors towards deviations of prescribed delivery routes, could improve last-mile route optimization model and being practiced in real world scenarios.

### **1.5 Contributions**

Our project aims to explicitly study and understand the effects of customers' acceptable delivery time windows on drivers' deviation on prescribed routes, by utilizing machine learning model to unfold the relationship between them. It will provide useful insights for real-world last-mile route optimization as identifying and realizing how customers preferences can impacts the overall actual delivery route by drivers. This project also aims to perform predictions on potential delivery routes by drivers and utilizing optimization approach used in operation research to optimize the best delivery routes for drivers. By learning drivers' delivery patterns, it allows actual adaptations of prescribed routes by drivers in real-world delivery.

### **1.6 Report Organisation**

The remainder of this report is structured as follows. In Chapter 2, we reviewed previous research publications, methodologies proposed related to this paper. In Chapter 3, we covered our proposed methodology, including a detailed discussion of the model architecture. In Chapter 4, we present the project setup and flow. In Chapter 5, we discuss the system implementation in detail. Chapter 6 provides the model evaluation and discussion of the results. Finally, Chapter 7 concludes the report with a summary of the entire project.

## CHAPTER 2

### Literature Review

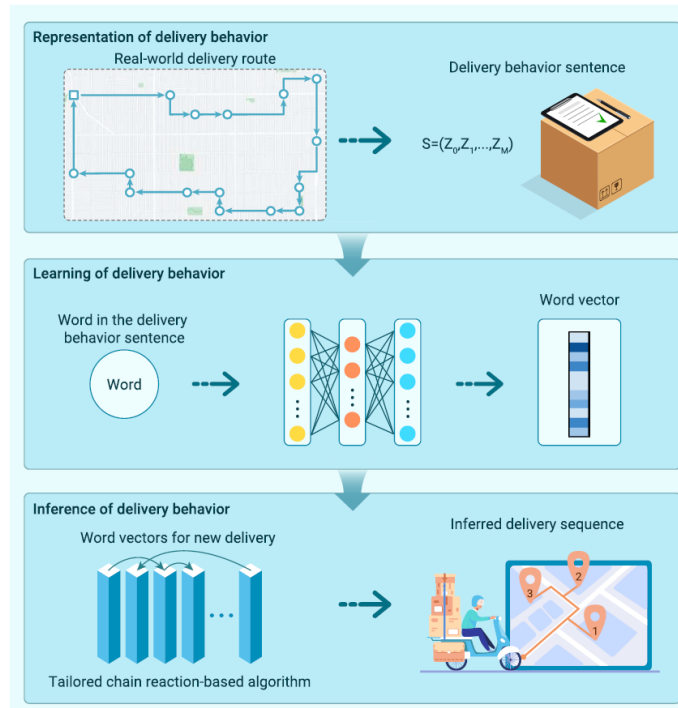
#### 2.1 Previous Work

##### 2.1.1 Delivery route prediction using machine learning models

As mentioned in Chapter 1, capturing tacit knowledges are crucial for future last-mile route optimisation system. This is due to traditional last-mile routes optimisation systems simply does not take in accounts of real-world factors faced by delivery drivers (e.g. temporal traffic conditions, drivers' preferences). In recent years, several research, including, but not limited only to [3], [4], [5], by feeding in machine learning models with real-world delivery routes sequences, in order to capture and predicts possible delivery route being practiced by drivers.

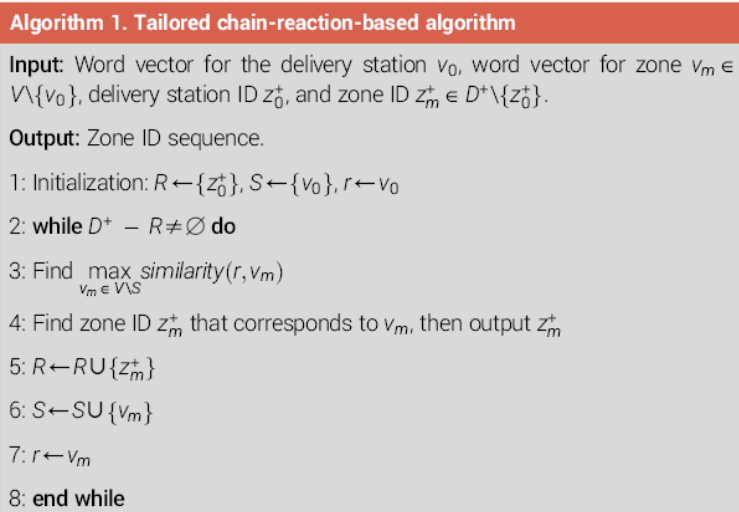
[3] proposed, by transforming historical delivery routes data into a natural language sentence, (i.e., in a delivery route, each delivery stops are represented as a "word" element, and "word" element are arranged in exact order based on the actual delivery route order.), following with the use of Word2Vec approach in natural language processing (NLP) to learn vector representations of "words" in delivery behaviour sentence, and finally the real-world delivery route are inferred from the output word vector (from previous step), utilising a tailored chain-reaction-based algorithm. Figure 2.1.1 demonstrates the overall framework of the proposed model.

The idea of treating every zone or delivery station in a delivery route as an element found in sentence, and word ordering in sentences are similar to the drivers' travel trajectories, are proposed by [3]. Then, by utilizing Word2Vec algorithms, found commonly in natural language processing (NLP), to learn the vector representation of 'word' elements in a delivery behaviour 'sentences'. After obtaining the word vector, inference on delivery behaviour are done based on a tailored chain-reaction-based algorithm.



**Figure 2.1.1** Overall framework of the route prediction model by [3].

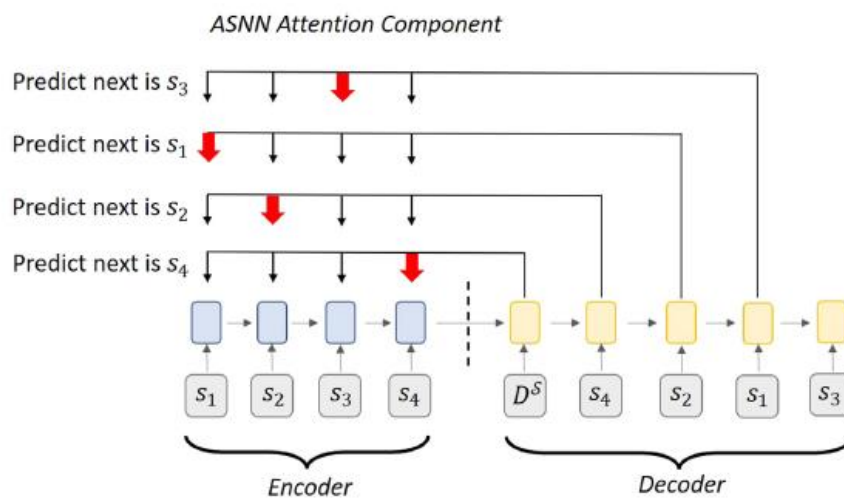
The algorithm proposed by [3], elaborate in Figure 2.1.2, took inspiration from basic stages of chain reaction, namely initiation, propagation, and termination. Given an unsorted zone sequence, as an initiation, delivery station are always the starting node. From there, propagation phase, where the algorithm, paired with word vectors obtained from earlier stages, it will iteratively find the next delivery zones. Finally, iterative search of next delivery zones come to termination when all zones are covered.



**Figure 2.1.2**  
Tailored chain-reaction-based algorithm, proposed by [3].

In [5], authors use combinational of seq2seq modelling and pair-wise attention-based pointer neural network (NN) that learn both local (ASNN Attention-based Spatial NN) and global (Encoder-Decoder LSTM) relationship between delivery stops. Then, using greedy algorithm to generate possible delivery sequences with different initial stops (to improve model's accuracy), and select the one with the lowest operational cost [5]. It is worth noting that [3], [5] approaches are to tackle interzone sequencing only and assuming drivers always take optimal routes in intrazonal level due to principle of local optimality.

For model training, [5] introduced a seq2seq modelling framework, for an arbitrarily ordered sequence as an input, the seq2seq model, with a recurrent neural network, computes the conditional probabilities of actual route trajectories ( $c_1, \dots, c_n$ ) given  $S$  (all training routes), and  $\theta$  (parameters learnt by empirical risk minimization). Then, [5] uses two LSTM layers, encoder-decoder combo, having time step = 1, obtaining a vector representation by reading input sequence, then extracts the output sequence. Inherently, LSTM encoder-decoder, embeds input sequence to hidden vectorization, are powerful algorithm to obtain the global patterns of the input data. In [5], they proposed by adding attention technique (i.e., pair-wise), masking over the input sequence then make predictions with LSTM encoder-decoder. Attention techniques proposed are targeted to obtain the local view of the input sequence, such as relationship between two nodes. Then, by feeding in input sequence, the pair-wise attention-based pointer NN, will output a learnt parameter,  $\theta$ .



**Figure 2.1.3** Overall architecture of attention-based pointer NN, proposed by [5].

At sequence inference stage, using greedy algorithm, [5] generate a list of sequences with different initial stops, and the route sequence with the lowest operational cost are selected, as an output route sequence. The route sequence inference, by [5], are elaborated in Figure 2.1.4.

---

**Algorithm 1** Sequence generation
 

---

**Input:** Trained model,  $S$ 
**Output:** Predicted stop sequence

 1: **for**  $s$  in  $S$  **do**

 2: Let the first predicted stop be  $\hat{s}_{(1)} = s$ 

 3: Predict the following stop sequence  $(\hat{s}_{(2)}, \dots, \hat{s}_{(n)})$  using the greedy algorithm. Denote the predicted sequence as  $P_s$ .

 4: Calculate the total operation cost of the whole sequence (including depot), denoted as  $OC_s$ .

**return**  $P_{s^*}$ , where  $s^* = \operatorname{argmin}_{s \in S} OC_s$ 


---

**Figure 2.1.4** Route Sequence Inference algorithm, by [5].

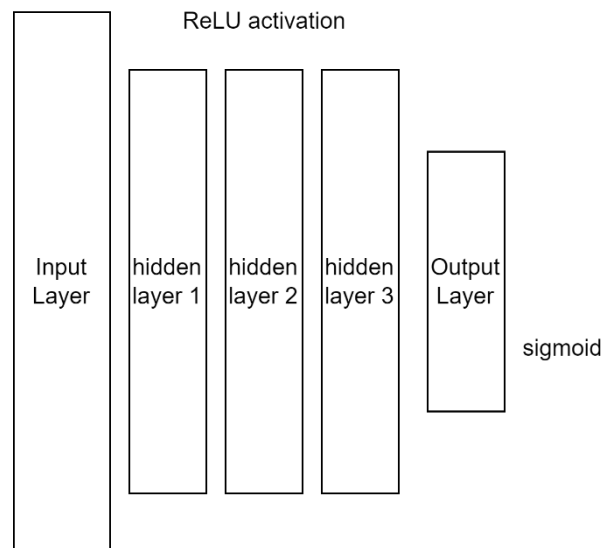
On the other hand, [4] suggested that, for any machine learning model that are capable to make discrete classification, can be employed, given the nature of presented approach (i.e., dataset provided by [7]). Using a feedforward NN as prediction architecture, [4]'s prediction model are divided into two phases, interzone phase, and intrazonal phase. The sequence of delivery zones is predicted, then, within each delivery zone, the sequence of delivery stops is only then predicted. Figure 2.1.5 presents the features applied in predicting the inter-zone sequence, proposed by [4].

Features to predict next cluster chosen by driver .

No.	Feature
1	Average travel time and geographical distance from current cluster to candidate cluster
2	Average travel time and geographical distance from the candidate cluster to not yet visited clusters
3	Average travel time and geographical distance from current cluster to the depot
4	One-hot encoded vector representing opening hours
5	Percentage of customers already visited

**Figure 2.1.5** Features applied in inter-zone sequence prediction, by [4].

After data preprocessing, dataset is feed into a feed-forward neural network, with three hidden layers, and rectified linear (ReLU) activation functions. The loss function selected by [4] for the NN model is Binary Cross Entropy loss. Figure 2.1.6 presents the architecture of the feedforward NN model proposed by [4]. Then, using ASHA algorithm, hyperparameter tuning for the NN model are done, results seen in Figure 2.1.7.



**Figure 2.1.6** Architecture model of feedforward NN model, proposed by [4].

Hyperparameters of feedforward neural network .

Phase	Hidden layer size one	Hidden layer size two	Hidden layer size three	Learning rate	Batch size
Cluster	128	64	16	0.00113	32
Customer	64	32	8	0.000589	4

**Figure 2.1.7** Best Hyperparameters found for NN model, using ASHA algorithm.

### 2.1.2 Delivery route optimization

TSP is a classic NP-hard problem, and have seen in various real-world application, such as logistics, circuit design and even DNA sequencing. Over the years, the problem had been studied extensively, resulting in various TSP variants, solving specific concern, as products.

In last-mile delivery applications, some packages can be time-sensitive, leading to the introduction of time window constraints into the delivery process, which is TSP with time window (TSPTW). In solving TSP problems, it is generally divided into exact approaches and approximate approaches. For small scales TSP problems (i.e., up to 50 nodes), exact approaches, including algorithms like branch-and-bound [8], branch-and-cut [9], are utilised and able to tackle the problem optimally. To solve large scales TSP problems (>200 nodes), approximate approaches, heuristics including local search, insertion, simulated annealing, etc. are used instead.

In [4], authors extend the TSPTW formulation by adding an upper bound of allowed deviation between actual tour,  $T$  and predicted tour,  $T'$ , as constraint, presenting formulation of TSPTW and deviation (TSPTW-Dev). In optimizing the intrazonal level delivery routes, [4] draw in Variable Neighbourhood Search (VNS), employing three neighbourhood structures and 2-opt local search operators, to improve the predicted tour solutions, output by their ML model.

## 2.2 Results

The evaluation metrics used in [5] are both disparity score, and prediction accuracy.

Disparity score, introduced by Amazon Last Mile Routing Research Challenge, are used by Amazon to evaluate the quality of predicted delivery trajectories, output by models. The disparity score, is a metrics that portrays how well the output delivery sequences able to mimics the delivery route, preferred by experienced delivery driver. (The lower the disparity score, the better) Aside from disparity score, prediction accuracy of first four zones for every routes are also evaluated by [5].

Below defines the mathematical equation of the disparity score: (For detailed equation explanation, see [5]).

$$R(A, B) = \frac{S D (A, B) \cdot ERP_{norm}(A, B)}{ERP_e(A, B)} \quad (1)$$

Below defines the mathematical equation of the prediction accuracy: (For detailed equation explanation, see [5]).

$$Prediction\ accuracy_i = \frac{\sum_{m=1}^M \mathbb{1}_{\{A_i^{(m)}=B_i^{(m)}\}}}{M} \quad (2)$$

The results obtained from [5]'s proposed model, shows positive results. In terms of disparity score, the model obtained a score of 0.0369, comparing to all other traditional operation research solver, [5] outperforms them. When compared to Amazon Last-Mile Routing Research Challenge (ALMRRC) winning teams solution, the pair-wise attention-based pointer NN model, is behind than the first-place team, with score of 0.0198. In terms of prediction accuracy scores, the model outperformed all other traditional operation research solver, in every first four zone, yielding higher accuracy score. Figure 2.2.1 shows the performance table of proposed model by [5], and other benchmarking model.



Model performance.

Sequence generation	Model	Disparity score		Prediction accuracy			
		Mean	Std. Dev	1st zone	2nd zone	3rd zone	4th zone
–	Tour TSP	0.0443	0.0289	0.207	0.185	0.163	0.168
–	Open-tour TSP	0.0430	0.0302	0.270	0.244	0.227	0.232
Greedy	ASNN	0.0470	0.0289	0.150	0.141	0.119	0.123
	LSTM-E-D	0.0503	0.0313	0.207	0.183	0.161	0.166
	Pnt Net	0.0460	0.0309	0.224	0.204	0.186	0.165
	<b>Ours</b>	<b>0.0417</b>	<b>0.0306</b>	<b>0.241</b>	<b>0.231</b>	<b>0.224</b>	<b>0.221</b>
Algorithm 1	ASNN	0.0429	0.0299	0.221	0.213	0.203	0.195
	LSTM-E-D	0.0501	0.0305	0.182	0.156	0.142	0.149
	Pnt Net	0.0382	0.0301	0.286	0.273	0.262	0.274
	<b>Ours</b>	<b>0.0369</b>	<b>0.0301</b>	<b>0.320</b>	<b>0.310</b>	<b>0.303</b>	<b>0.314</b>
Amazon Last-Mile	<a href="#">Cook et al. (2022)</a>	0.0198	N.A.	N.A.	N.A.	N.A.	N.A.
Routing Research Challenge	<a href="#">Guo et al. (2023)</a>	0.0381	N.A.	N.A.	N.A.	N.A.	N.A.
Winning Teams Solutions	<a href="#">Arslan and Abay (2021)</a>	0.0367	N.A.	N.A.	N.A.	N.A.	N.A.

**Figure 2.2.1** Pair-wise Attention-based NN model Performance Table. Sourced from [5].

### 2.3 Summarization of findings

Table 2.3.1 summarizes the key differences of some of the existing model for last-mile delivery route prediction and optimization.

**Table 2.3.1** Summary of Route Prediction and Optimization Studies

Study	Model	Significances	Limitations
[3]	NLP + tailored chain-reaction-based algorithm	<ul style="list-style-type: none"> <li>- Extracting tacit driver knowledge, by converting historical delivery routes into natural language sentences and feeding into NLPs.</li> <li>- Relatively low computational time. compared to traditional TSP solutions.</li> <li>- High adaptability.</li> </ul>	<ul style="list-style-type: none"> <li>- Average error value increases as length of targeted sequence prediction increases.</li> <li>- Due to weak correlations between inputs and outputs in longer sequences.</li> </ul>
[5]	seq2seq model (a deep learning model) + pair-wise attention-based pointer NN	<ul style="list-style-type: none"> <li>- Predicts possible stop sequences similar to high quality TSP solutions (replacing TSP optimizations model).</li> <li>- Utilizing LSTM encoder to capture global view of input (i.e., overall tour sequence pattern).</li> </ul>	<ul style="list-style-type: none"> <li>- Relatively higher computational time due to system network complexity.</li> <li>- Disparity score can be improved by incorporating local search rules</li> </ul>

		<ul style="list-style-type: none"> <li>- Utilizing ASNN Attention-based Spatial NN to capture local view of input (i.e., relationship between 2 distinct stops in tour sequence).</li> </ul>	
[4]	Feedforward NN + VNS	<ul style="list-style-type: none"> <li>- Allow decision maker to alter level of deviation and the penalized effect of time window constraint based on preferences.</li> <li>- Prediction and optimization on both interzone and intrazonal level routes.</li> </ul>	<ul style="list-style-type: none"> <li>- Sequence deviation measures (Jaro &amp; LCSS) does not consider geographical distance between two stop nodes.</li> <li>- May results in significant changes in suggested delivery routes when swapping to-be customer node, especially when two stop nodes are further away from each other.</li> <li>- Lack of proper model benchmarking on both machine learning model and optimization approach (due to the focus on proposing novel hybrid framework)</li> </ul>

### 2.4 Proposed Method

In this project, our main objective is to infer the possible factors that might cause possible deviations of pre-planned routes by delivery drivers. Before inferencing possible factors affecting delivery trajectories, we should be able to predict delivery routes using neural network.

For model selection, the idea of utilizing NLP, by [3] faces possible errors as the targeted prediction sequence increases, due to as the sequence spans, correlations between inputs and outputs are increasingly lower. Next, although the pair-wise attention-based pointer NN model are the best performing state-of-art machine learning solution for delivery trajectories, the computation complexity of the model and higher computation resources required, are not feasible.

After considering on the existing works and the objectives of this project, in this paper, we are proposing a simple RNN model to learn the delivery route trajectories, backed by suggestion from [4]. In the following chapter, the proposed methodology is elaborated.

## CHAPTER 3

### System Methodology

This section first introduced the high level seq2seq modeling framework and followed by elaboration of the architecture of (1) proposing model, Simple RNN Encoder-Decoder (Simple RNN E-D), as well as (2) modified LSTM Encoder-Decoder with Pair-Wise Attention (LSTM E-D with Attention) adapted from [5].

#### 3.1 Sequence-to-sequence (seq2seq) modeling framework

Given an arbitrary-ordered input route sequence,  $(s_1, \dots, s_n) \in S$ . Let the predicted route sequence as  $(\hat{s}_{(1)}, \dots, \hat{s}_{(n)})$ , and  $c_i$  be the positional index of stop  $\hat{s}_{(i)}$  corresponding to the input sequence (where  $c_i \in \{1, \dots, n\}$ ) [5]. In this seq2seq model framework, by utilizing Recurrent-NN, the conditional probability  $P(c_1, \dots, c_n | S; \theta)$ , with parameter  $\theta$ , can be calculated as followed:

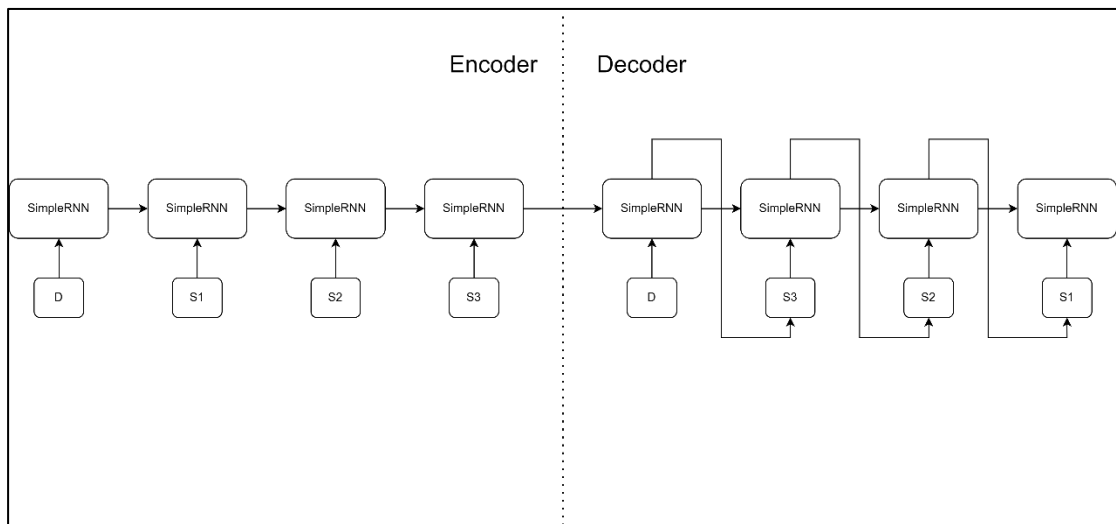
$$\begin{aligned} P(c_1, \dots, c_n | S, X^S; \theta) \\ = P(c_1 | S, X^S; \theta) \cdot \prod_{i=2}^n P(c_i | c_1 \dots, c_{i-1}, S, X^S; \theta) \end{aligned} \quad (3) [5]$$

where  $X^S$  is the features of stops in  $S$ . The calculation of  $P(c_1, \dots, c_n | S, X^S; \theta)$  for the two model (i.e., Simple RNN E-D, and LSTM E-D with Attention) are documented later in this chapter.

#### 3.2 Simple RNN Encoder-Decoder (Simple RNN E-D) Model

A Simple Recurrent Neural Network (RNN) is a type of neural network with internal memory that captures temporal dependencies between inputs, allowing them to retain

information from previous sequential inputs. **Figure 3.2** shows the architecture of LSTM E-D with Attention model, presented at four-step. The entire prediction process uses a SimpleRNN encoder to extract feature vectors, followed by a SimpleRNN decoder to process the last visited stop's feature vector. The decoder incorporates a simplified attention mechanism by averaging the encoder outputs, which helps provide global context when predicting the next stop in the sequence. The model is designed to leverage the SimpleRNN encoder-decoder framework to capture the overall sequence pattern of the route, while the context mechanism focuses on combining both global and local sequence relationships between the last visited stop and potential next stops.



**Figure 3.2** Overall architecture of Simple RNN E-D Model

### 3.2.1 Simple RNN encoder

The role of the Simple RNN encoder in the Simple RNN E-D model is to gather and aggregated each stop information. The input for encoder model is features of the stop  $s_i$ ,  $x_i \in \mathbb{R}^K$  in a given arbitrary stop sequence  $(s_1, \dots, s_n)$ , where  $x_i$  may include geographical information and package information of the stop  $s_i$ .  $K$  is the number of features. The output of the encoder model will be a sequence of encoder output vectors  $(e_1, \dots, e_n)$  through calculation, expressed in:

$$h^E_i, e_i = \text{SimpleRNN}(x_i, h^E_{i-1}; \theta^E) \quad \forall i = 1, \dots, n \quad (4) [5]$$

where (1) the encoder hidden vector,  $h^E_i \in \mathbb{R}^K$ , with  $h^E_0 := 0$ ;

(2) the encoder output vector,  $e_i \in \mathbb{R}^{K_e}$ ;

(3) the corresponding vector dimensions are defined as  $K^E_h$  and  $K_e$

The final step hidden vector  $h^E_n$  is used for input of Simple RNN decoder, consisting global features of entire input route sequence.

### 3.2.2 Simple RNN decoder

The role of the Simple RNN decoder is to predict the next stop for every timestep, later forming a route sequence. In the proposing Simple RNN decoder model, aggregation of both local information (previous visited stop features  $x_{(i)}$ , previous RNN hidden state,  $h^D_{(i)}$ ) and global information (encoder outputs,  $e$ ).

$$h^D_{(i+1)}, d_{(i)} = \text{SimpleRNN}(x_{(i)}, h^D_{(i)}; \theta^D) \quad \forall i = 0, 1, \dots, n \quad (5)$$

where (1) the decoder hidden vector,  $h^D_{(i)} \in \mathbb{R}^{K^D_h}$ , with  $h^D_{(0)} := h^E_n$ ;

(2) the decoder output vector,  $d_{(i)} \in \mathbb{R}^{K_d}$ ;

(3) the corresponding vector dimensions are defined as  $K^D_h$  and  $K_d$ ;

(4) the features of last visited stops,  $x_{(i)}$ ;

For depot station case,  $x_{(0)} = x_{(D)}$  and  $d_{(0)} = d_{(D)}$ .

$$\text{logits} = fc(\text{concat}(h^D_{(i+1)}, \mu_e)) \quad \forall i = 0, 1, \dots, n \quad (6)$$

where (1) the decoder hidden vector at current timestep  $(i + 1)$ ,  $h^D_{(i+1)}$ ;

(2) the mean of encoder outputs,  $e$  across all timesteps,  $\mu_e$ ;

The model performs prediction on all candidate stops based on conditional probabilities. The conditional probabilities for next possible stops are calculated, and prediction  $\hat{s}_{(i+1)}$ , expressed in:

$$\begin{aligned} P(c_{i+1} = j \mid c_1 \dots, c_i, S, X^S; \theta) &= \text{Softmax}(\text{logits}) \\ \forall i &= 0, 1, \dots, n \\ j &= 1, \dots, n \end{aligned} \quad (7)$$

$$\begin{aligned} \hat{s}_{(i+1)} &= \underset{s_j \in S \setminus S^V_{(i)}}{\text{argmax}} P(c_{i+1} = j \mid c_1 \dots, c_i, S, X^S; \theta) \\ \forall i &= 0, 1, \dots, n \\ j &= 1, \dots, n \end{aligned} \quad (8)$$

where (1) the set of visited/predicted stops until decoder step  $i$ ,  $S^V_{(i)} = \{\hat{s}_{(1)}, \dots, \hat{s}_{(i)}\}$

### 3.3 LSTM Encoder-Decoder with Pair-Wise Attention Model

The LSTM with Attention model architecture is adapted from [5] with slight modification. **Figure 2.1.3** shows the architecture of LSTM E-D with Attention model, presented at four-step. The entire prediction processes use an LSTM encoder to extract feature vectors, an LSTM decoder to extract last visited feature vector, then incorporates pairwise attention mechanism to the predict the next stop sequence. The idea is to utilize LSTM encoder-decoder framework to captures global perspective of



the overall sequence pattern, while the Attention mechanism focuses on local relationships between two stop pairs (i.e., last visited stop, and candidate stops).

### 3.3.1 LSTM encoder

The role of the LSTM encoder in the LSTM E-D with Attention model is similar to Simple RNN encoder model, which is to gather and aggregated each stop information. The primary difference between LSTM and Simple RNN is that LSTM are more capable to retain long term dependencies (overcome vanishing gradients), which makes LSTM a superior choice for complex tasks (long route sequence). The input for encoder model is features of the stop  $s_i$ ,  $x_i \in \mathbb{R}^K$  in a given arbitrary stop sequence  $(s_1, \dots, s_n)$ , where  $x_i$  may include geographical information and package information of the stop  $s_i$ .  $K$  is the number of features. The output of the encoder model will be a sequence of encoder output vectors  $(e_1, \dots, e_n)$  through calculation, expressed in:

$$h^E_i, e_i = LSTM(x_i, h^E_{i-1}; \theta^E) \quad \forall i = 1, \dots, n \quad (9) [5]$$

where (1) the encoder hidden vector,  $h^E_i \in \mathbb{R}^K$ , with  $h^E_0 := 0$ ;

(2) the encoder output vector,  $e_i \in \mathbb{R}^{K_e}$ ;

(3) the corresponding vector dimensions are defined as  $K^E_h$  and  $K_e$

The final step hidden vector  $h^E_n$  is used for input of Simple RNN decoder, consisting global features of entire input route sequence.

### 3.3.2 LSTM decoder

Following [5], the role of the LSTM decoder in the LSTME-D with Attention model is to produce last visit stop vectors, which are used for the attention mechanism to predict next zone sequence. Denote the output route sequence  $(\hat{s}_{(1)}, \dots, \hat{s}_{(n)})$ . Given features of the stop  $\hat{s}_{(i)}$ ,  $x_{(i)}$ . At each decoder timestep,  $i$ , the process can be expressed in:

$$h^D_{(i+1)}, d_{(i)} = LSTM(\text{concat}(x_{(i)}, \omega_{(i)}), h^D_{(i)}; \theta^D) \quad \forall i = 0, 1, \dots, n \quad (10) [5]$$

where (1) the decoder hidden vector,  $h^D_{(i)} \in \mathbb{R}^{K^D_h}$ , with  $h^D_{(0)} := h^E_n$ ;

(2) the decoder output vector,  $d_{(i)} \in \mathbb{R}^{K_d}$ ;

(3) the corresponding vector dimensions are defined as  $K^D_h$  and  $K_d$ ;

(4) the context vector is computed from attention layer,  $\omega_{(i)}$ ;

For depot station case,  $x_{(0)} = x_{(D)}$  and  $d_{(0)} = d_{(D)}$ .

### 3.3.3 Pair-Wise Attention Layer

The role of pair wise attention layer is predicting next possible stop by aggregating both global and local information in a given sequence of stops ( $s_1, \dots, s_n$ ). The mechanism works as at each decoder time step  $i \in \{0, \dots, n\}$ , after identifying the last visited stop,  $\hat{s}_{(i)}$ , the model perform prediction on  $\hat{s}_{(i+1)}$  from all candidates stops (all valid, unvisited zone),  $\hat{s}_{(j)} \in S$ . The input for attention layer, denoted as  $v^j_{(i)}$ , consists information of the stop pair  $\hat{s}_{(i)}$  and  $s_j$ , which can be expressed by:

$$v^j_{(i)} = \text{concat}(t^j_{(i)}, d_{(i)}, e_j) \quad (11)$$

where (1) the travel time features between stop pair  $\hat{s}_{(i)}$  and  $s_j$ ,  $t^j_{(i)}$ ;

(2) decoder output vector,  $d_{(i)}$ ;

(3) encoder output vector,  $e_j$ ;

The attention of stop pairs are calculated, following the equation:

$$u^j_{(i)} = MLP(v^j_{(i)}; \theta^A) \quad \forall i, j = 1, \dots, n \quad (12) [5]$$

$$a^j_{(i)} = Softmax(u^j_{(i)}) \quad \forall i, j = 1, \dots, n \quad (13) [5]$$

where the attention of stop  $\hat{s}_{(i)}$  to  $s_j$ ,  $a^j_{(i)} \in \mathbb{R}$ ;

Then, conditional probability for each stop pairs are calculated, and prediction  $\hat{s}_{(i+1)}$ , expressed in:

$$\hat{s}_{(i+1)} = \underset{s_j \in S \setminus S^V_{(i)}}{\operatorname{argmax}} a^j_{(i)} \quad \forall i = 0, 1, \dots, n \quad (14) [5]$$

where (1) the set of visited/predicted stops until decoder step  $i$ ,  $S^V_{(i)} = \{\hat{s}_{(1)}, \dots, \hat{s}_{(i)}\}$

In addition, context vector,  $\omega_{(i)}$  which is a weighted sum of all encoder output vectors with attention as weights.  $\omega_{(i)}$  is introduced to leverages the attention information as the decoder input for timestep  $i + 1$ . The formulation of context vector can be expressed:

$$\omega_{(i)} = \sum_{j=1}^n a^j_{(i)} \cdot e_j \quad (15) [5]$$

## CHAPTER 4

### Experimental Setup

#### 4.1 System Requirements

##### 4.1.1 Hardware

The hardware involved in this project is personal computer. Table 3.1.1 shows the detailed specification of the hardware involved.

**Table 4.1.1** Specifications of laptop.

Description	Specifications
Model	LENOVO 81WD
Processor	Intel® Core™ i5-1035G4 CPU @ 1.10GHz 1.50 GHz
Operating System	Windows 10 Home Single Language 64-bit
Graphic	Intel® Iris® Plus Graphics
Memory	12.0 GB (11.7 GB usable) DDR4 RAM
Storage	477 GB SSD ROM

##### 4.1.2 Software

The software involved in this project are listed as below:

- (i) Google Collaboratory (Google Colab).

Google Colab is a cloud based Jupyter Notebook service that provides free access to computing resources, including GPUs and TPUs. With Google Colab, users do not require to install libraries, and packages on personal devices, and running codes does not consume resources of the working system (hardware).

(ii) Google Drive.

Google Drive is a cloud storage service provided by Google, that allows users to store, synchronise, share files across the platform. Google Drive also integrates with Google Colab, allows data storage and retrieval from the drive.

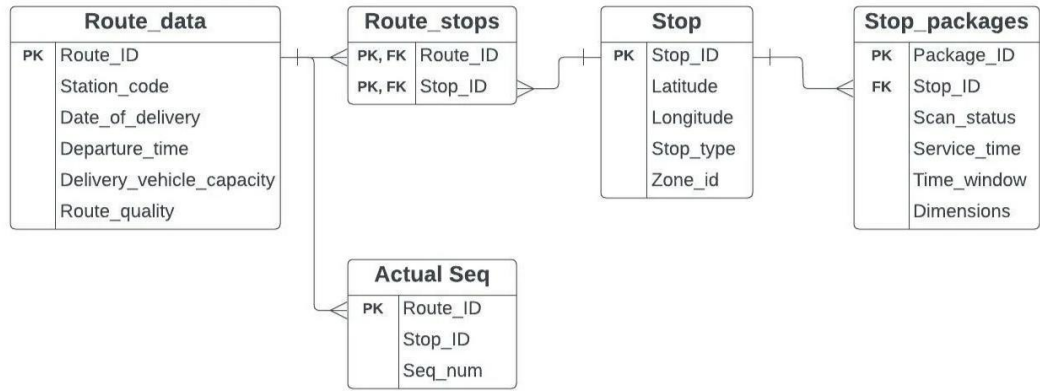
(iii) Tensorflow with Keras.

Tensorflow is an open-source deep learning framework provided by Google. With Keras, a python-based high-level neural network Application Programming Interface (API), running on top of Tensorflow, the combo provides users libraries, and tools to conduct experimentation with neural networks.

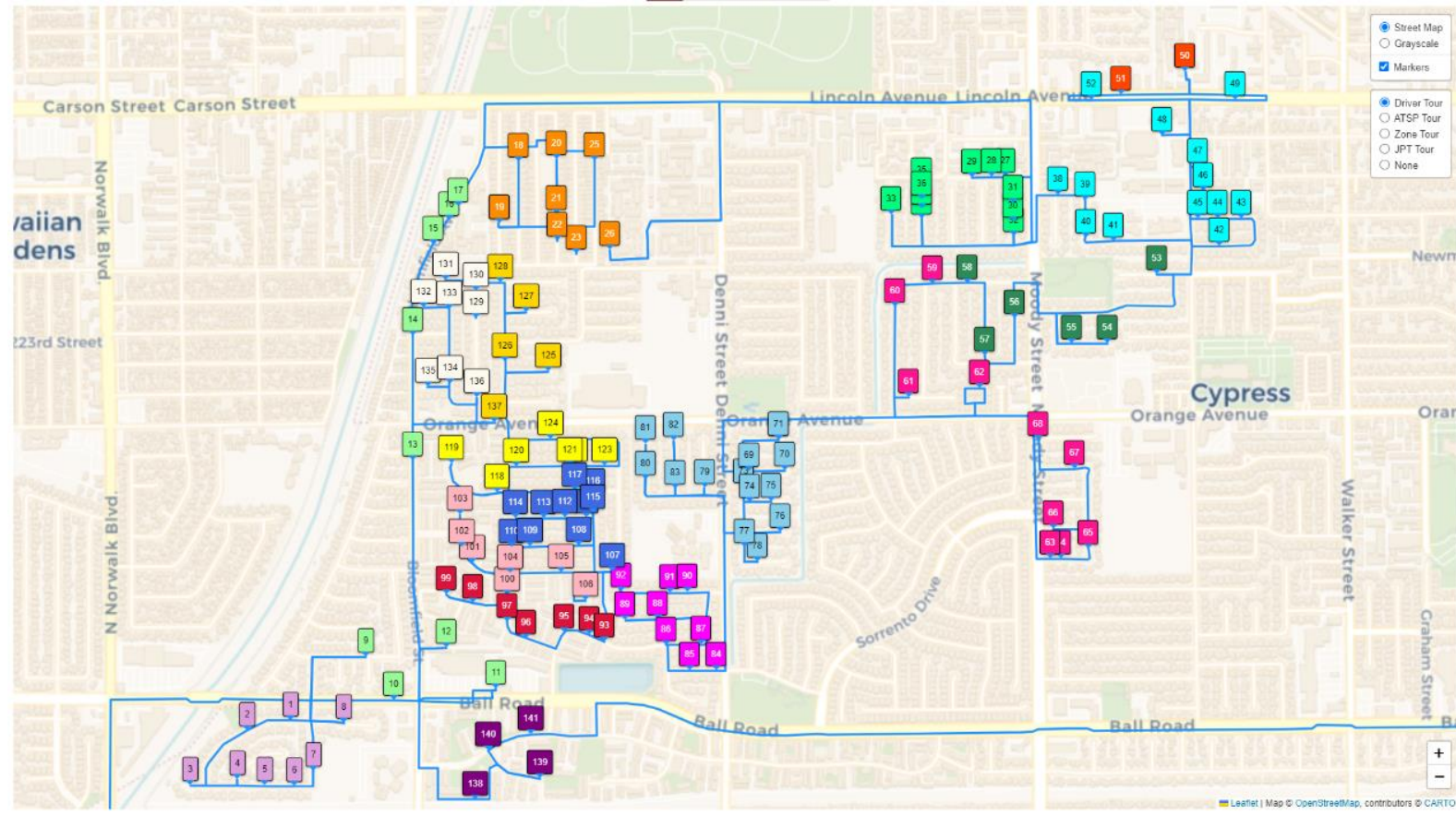
### 4.1.3 Data Source

The project will be using real-world data provided by Amazon Last-Mile Routing Research Challenge [7]. The dataset consists of 6112 historical drivers last mile delivery routes, which was collected between July and August 2018 in five metropolitan areas of United States (U.S.), namely Austin, Boston, Chicago, Los Angeles, and Seattle [5]. Each route is characterized by a variety of route-level, stop-level, package-level features, route quality attributes, delivery defects, driver experience, customer satisfaction, and productivity; dataset is summarised, and explained in Table 3.1.2 [7]. Besides, each route is labelled according to its perceived route quality (i.e., low, medium, and high). The entire dataset can be visualised and represented in the form of an Entity relationship diagram (ERD), seen in Figure 3.1.1.

## CHAPTER 4



**Figure 4.1.1** ERD for Amazon Last-Mile Routing Research Challenge Dataset.  
Sourced from [10].



**Figure 4.1.2** Sample Delivery Route, in Irvine, California.

Sourced from [11].

**Table 4.1.2** Data Description on provided Amazon Last-Mile Routing Research Challenge Dataset. Sourced from [7].

Data Field	Description	Unit/Format
Route information		
Route ID	Unique and anonymized identifier of each route.	-
Station code	Unique identifier for a depot station.	(alphanumeric string)
Date	Date of route execution.	YYYY-MM-DD
Departure time	Time when vehicle leaves depot.	-
Executor capacity	Volumetric capacity of vehicle.	cm <sup>3</sup>
Stops	A list of each stop in route.	-
Observed sequence	Actual sequence in which stops were visited.	-
Route score	Quality of the observed sequence.	Categorical (i.e., high, medium, or low)
Stop information		
Stop ID	Unique identifier of each stop on a route.	-
Latitude/Longitude	Obfuscated coordinates of each stop.	-
Type	Type of stop.	Categorical (i.e., station or drop-off)
Zone ID	Geographical planning area in which the stop falls.	-

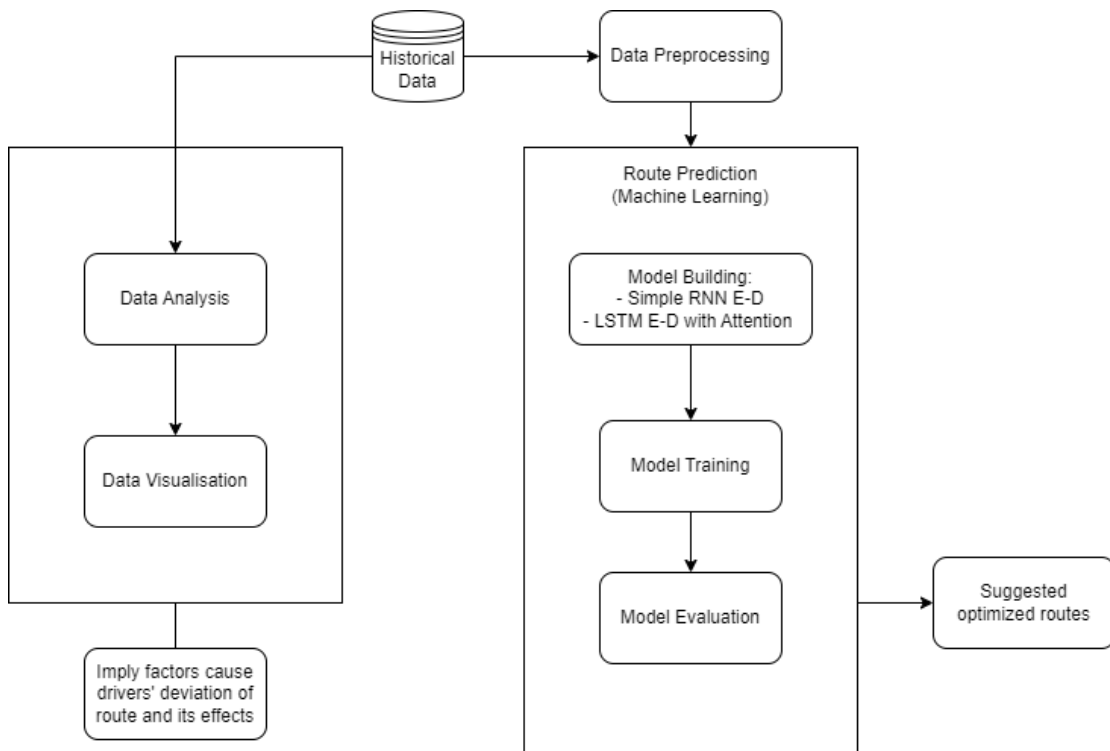


## CHAPTER 4

Packages	A list of packages to be delivered at each stop.	-
Transit time	Estimated transit time to every other stop on route	Seconds
<hr/> <b>Package information</b> <hr/>		
Package ID	Unique and anonymized identifier of each package.	-
Status	Delivery status of package.	Categorical (i.e., DELIVERED, DELIVERY_ATTEMPTED, or REJECTED) (if not specified, fields are filled with value 'NaN')
Time window	Start and end time window, when applicable.	
Planned service time	Time that serving the package is expected to require.	Seconds
Dimensions	Length, width, and height of package.	cm

## 4.2 System Design

The proposed framework is outlined (see Figure 4.2.1), with the input data of real-world historical delivery routes data and output of optimized delivery routes, which similar to real-world drivers' preferences.



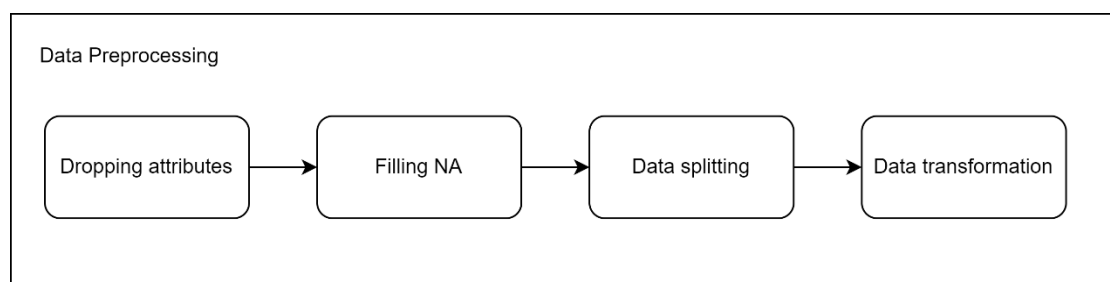
**Figure 4.2.1** Overall project framework. Adapted from [4].

### 4.2.1 Data Analysis

After obtaining the historical data (more data information, refer Chapter 4.1.3), data analysis and visualisation is performed as project initial step. Basic data analysis was performed on the dataset, allowing us to easily grasp complex information, identify outliers, and communicate findings to others. Built-in library such as seaborn and matplotlib are utilized to provide graph visualisation. By performing data analysis, we could imply factors that cause drivers deviate from the pre-planned route (i.e., time-sensitive packages.)

### 4.2.2 Data Preprocessing

Then, data preprocessing will be performed before feeding into the machine learning model. Data preprocessing steps include feature selection, data cleaning (i.e., filling missing values), data splitting, data transformation, and data reduction. Data preprocessing are essential steps, as data quality defined the output of machine learning model. With high quality data input, machine learning models are able to converge effectively and produce accurate predictions. Figure 4.2.2 shows the steps taken in Data Preprocessing process.



**Figure 4.2.2** Steps involved in Data Preprocessing.

In this project, we chose to focus only on the 2718 routes with ‘High’ route scorings. This is because, in both validation and testing datasets, all routes are rated ‘High’. Besides, dropping ‘Low’ tiered routes are due to routes rated ‘Low’ only contribute around 1.6%, having only 102 routes out of the overall datasets (6112 routes), which might cause biased outcomes. Then, we dropped all packaged-level features, includes ‘pack\_ID’, ‘time\_window\_start’, ‘time\_window\_end’. The reason behind the action, is in this project, we would like to focus on delivery zone level sequencing, following [3], stating drivers tends to stick to the shortest path within intra-zonal delivery. Furthermore, the findings also display only around 7% of the parcels are time sensitive, filling missing values with a random value might produce possible biased outcomes. Moreover, we dropped time-related features (i.e., date, and departure time) as departure time for all routes are within an hour period, 9.00 – 10 a.m. period.

For data cleaning steps, we filled up all stops with null are filled with value ‘NA’. Then, for all station type records, indicate by sequence id = 0, the field, zone id

with value '@-0.0@', as initial zone. Moreover, for every route, with missing or invalid zone\_id, are filled with zone\_id value, by filling with nearest zone id, determined by travel distances between valid zones and target (stops with missing or invalid zone id).

After data cleaning steps, dataset is now split into train set X and train set y. The motivation behind action splitting data into train and test set, is that the provided dataset from AWS is divided into train set (model\_build), test set (eval\_model\_apply) respectively, where test set data are completely independent from train set.

### 4.2.2 Model Building and Training

After data preprocessing, two machine learning model (i.e., Simple RNN E-D Model, and LSTM E-D with Attention Model) are built to fulfil the project task, (i.e., predict the zone sequence Id for every route). Both Simple RNN E-D Model, and LSTM E-D with Attention Model are built with Keras library, which are an established library for deep learning model building and training operations. Detailed model methodology, as well as architecture are discussed in **Chapter 3**. Then, pre-processed data (historical delivery routes) are fed into the machine learning model for pattern convergence. For detailed model building and model training are documented in **Chapter 5**.

### 4.2.3 Model Evaluation

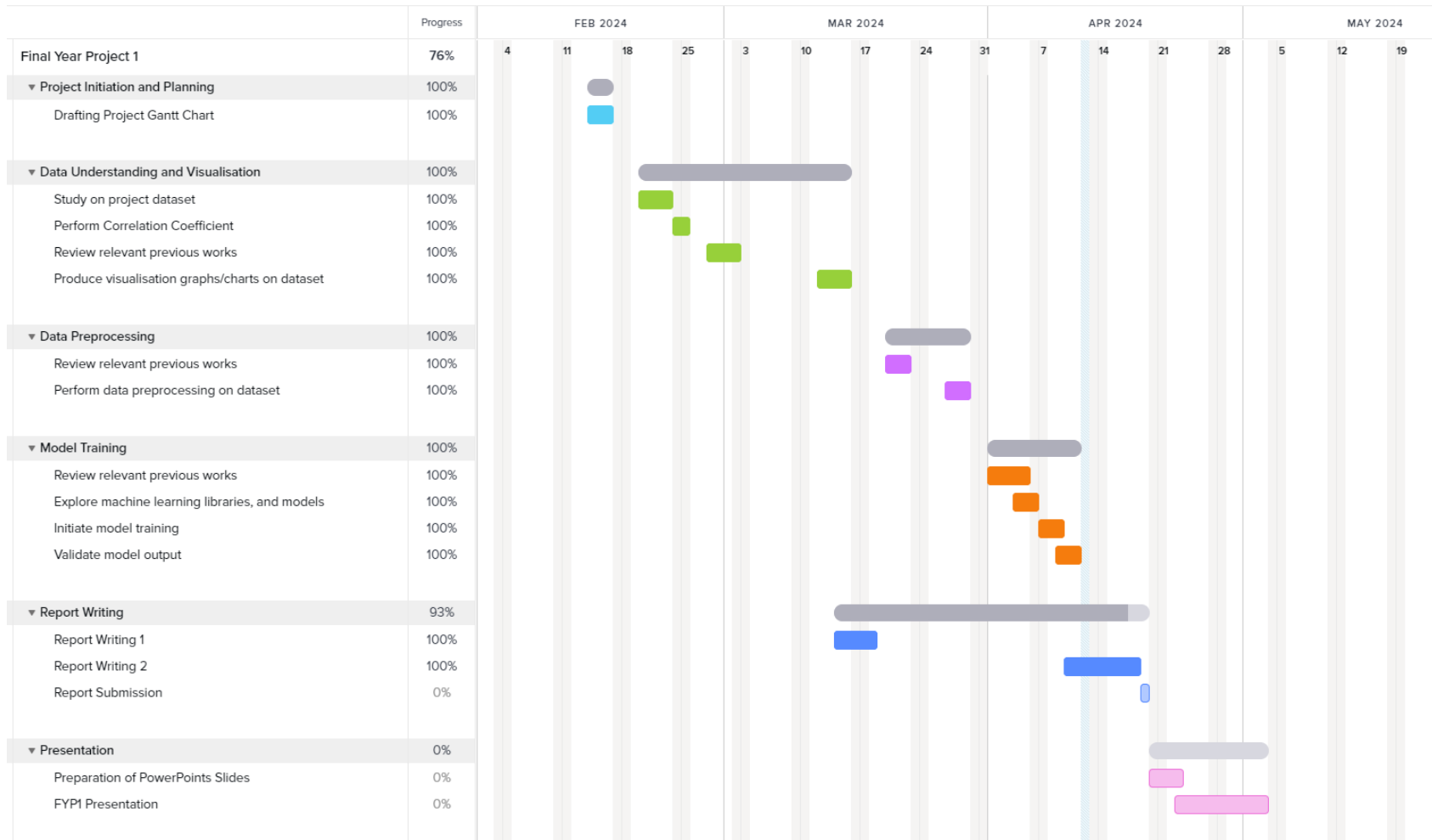
Model evaluation, including cross validation on model built are performed, to ensure proposed model able to predict good routes, aligning the project objectives. For performance metrics, [5] mentioned that accuracy metric does not differentiate “how wrong an erroneous prediction is”, in contrast, disparity score does not negatively impact too much when predicted stop,  $s_p$  are geographically closed towards the actual stop,  $s_a$ . The output of the model (quality of predicted delivery route) is then evaluated. All results and evaluation are discussed in **Chapter 6**.

### **4.3 Timeline**

Figure 4.3.1 presents the Gantt chart for all the works done with the corresponding timeline in Final Year Project 1. Works including, but not limited to, performing project initiation and planning, data understanding and visualisation, data preprocessing, model training, report writing, and FYP presentation preparation.

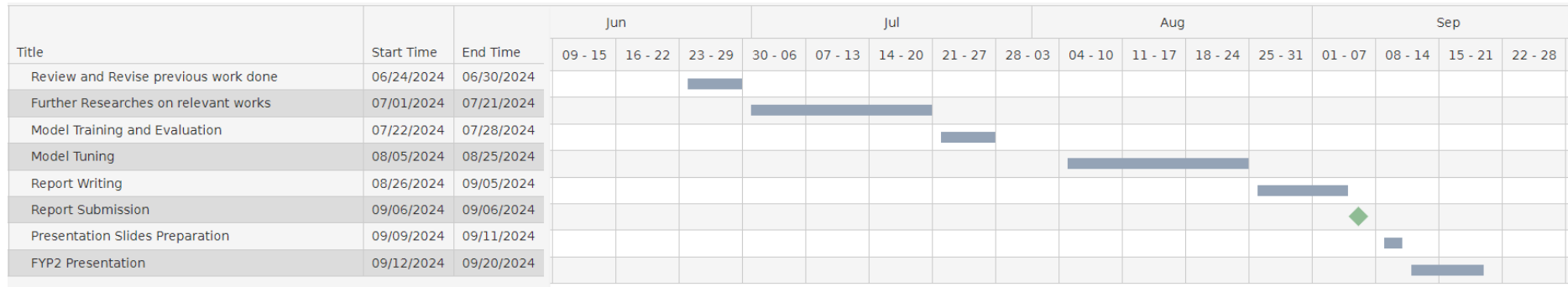
In Figure 4.3.2, Gantt chart for Final Year Project 2 is roughly drafted and outlined, with works to be done with corresponding timeline during upcoming trimester. Works including, but not limited to, review and revise previous work done, further research on relevant papers or works, model training and evaluation, model tuning, report writing and FYP presentation preparation.

# CHAPTER 4



**Figure 4.3.1** Gantt Chart for Project 1.

CHAPTER 4



**Figure 4.3.2** Gantt Chart for Project 2

## CHAPTER 5

### System Implementation

#### 5.1 Data Findings

##### *Route Score*

Below code, from line 1 to 6, obtained the number of routes for each scoring (i.e., Low, Medium, and High). Line 7 obtained all number of routes, by summing up the number of routes, for each class. Finally, line 8, output the percentage of routes with low scorings over all routes.

```
Line
01     num_low_route = rt.filter(rt['route_score'] ==
02         'Low').select('route_id').unique().height
03     num_high_route = rt.filter(rt['route_score'] ==
04         'High').select('route_id').unique().height
05     num_medium_route = rt.filter(rt['route_score'] ==
06         'Medium').select('route_id').unique().height
07     num_route = num_low_route + num_high_route + num_medium_route
08     print('Percentage of low scoring routes: ', (num_low_route /
09         num_route)*100 , '%')
```

- Routes with scoring 'Low' take up only 1.6% of overall routes.



Below code obtained the value of unique route with high scorings.

Line

```
01      rt.filter(rt['route_score'] == 'High').select('route_id').unique().height
```

- There are total of 2718 unique routes with 'High' scorings.

### *Time Windows Sensitivity*

Below code, line 1 obtained the number of packages in all 6,142 routes, whereas line 2 obtained the number of packages without time window stated from all 6,142 routes.

Then, line 4 and 5-6, output the total number of packages and the total number of packages without time window, respectively. Finally, line 7-8, output the percentage of packages without time window.

Line

```
01      num_package= rt.filter(pl.col('pack_ID').is_not_null()).height
02      num_package_wo_tw = rt.filter(pl.col('type').eq('Dropoff')).filter(
03          pl.col('time_window_start').is_null()).height
04      print("Total number of packages: ", num_package)
05      print("Total number of packages without time window: ",
06          num_package_wo_tw)
07      print("Percentage of packages without time window: ",
08          (num_package_wo_tw / num_package)*100 , '%')
```

- Out of 1,457,175 packages, 1,343,182 packages do not have specified delivery time windows.
- Over 92.27% of delivery packages are considered time window insensitive.

Below code, sort the dataframe, `rt`, for every unique route, calculate the number of packages without specified time window, and then sorted the dataframe based on number of packages without time window, in ascending order. The range for number of delivery packages without specified time window per route are [92, 302].

Line

```
01 rt.filter(pl.col('type').eq('Dropoff')).groupby('route_id').agg(num_pkgwotw
02     = pl.col('time_window_end').is_null().sum()).sort(
03     by = 'num_pkgwotw')
```

Output:

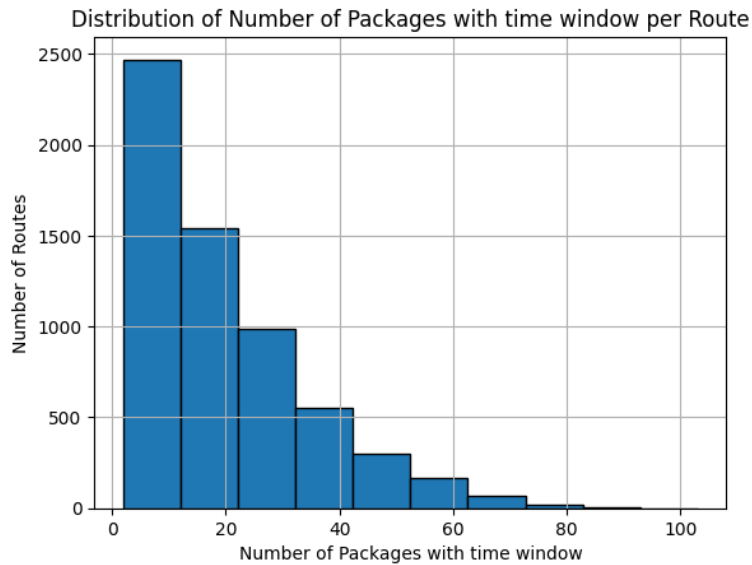
route_id	num_null_zones	route_id	num_null_zones
str	u32	str	u32
"RouteID_a5d547...	92	"RouteID_edfdbc...	295
"RouteID_e7737d...	106	"RouteID_abdfea...	296
"RouteID_13ad9c...	107	"RouteID_9ec69b...	296
"RouteID_934623...	113	"RouteID_4e9d93...	296
"RouteID_6d8f34...	113	"RouteID_aa4088...	297
"RouteID_ffd16e...	115	"RouteID_acab12...	298
		"RouteID_e1a4bf...	302

**Figure 5.1.1** Range of delivery package without time windows for each route.

**Figure 5.1.2** shows histogram of the distribution of the number of packages with a time window per route. It suggests that majority of routes, around 2500, have between 0 and 10 packages with time windows only.

Inference:

- The limited number of packages with time windows in most routes suggests that the overall influence of time-sensitive deliveries on driver behavior may be minimal for most routes.



**Figure 5.1.2** Distribution of Number of Packages with Time Window per Route

### *Route*

Below code, sort the dataframe, `rt`, for every unique route, calculate the number of packages, and then sorted the dataframe based on number of packages, in ascending order. The range for number of delivery packages per route are [151, 305].

Line

```
01     rt.groupby('route_id').agg(num_package = pl.count()).sort(by =
02                                     'num_package')
```

Output:

route_id	num_package
"RouteID_fd7c75...	300
"RouteID_6db9f2...	300
"RouteID_6d4c14...	151
"RouteID_3896ac...	300
"RouteID_13ad9c...	151
"RouteID_0e7eb7...	300
"RouteID_b02475...	151
"RouteID_d0483a...	300
"RouteID_e96d63...	152
"RouteID_c9e851...	300
"RouteID_6d8f34...	152
"RouteID_e1a4bf...	305

**Figure 5.1.3** Range of delivery package for each route.

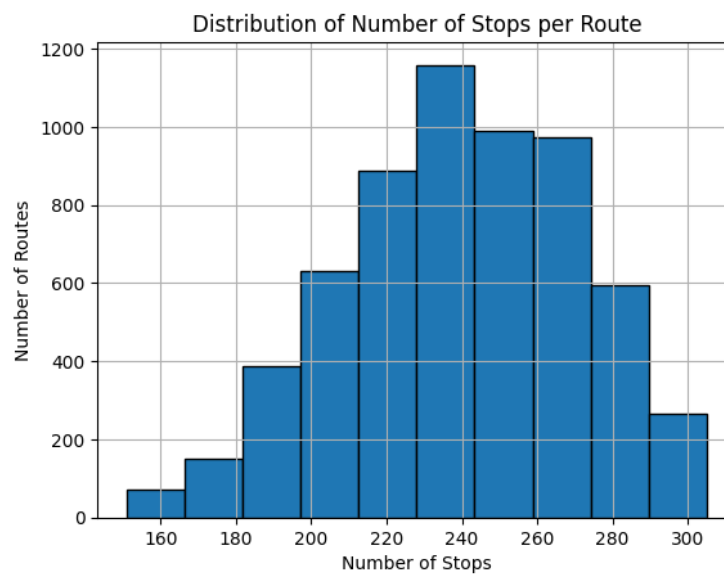
## CHAPTER 5

Below code, at line 1-2, number of stops for each route are obtained and sorted. Then, from line 3-8, histogram of the 'Distribution of Number of Stops per Route' are plotted, accessing the matplotlib library. Matplotlib is a plotting library for Python language.

Line

```
01 num_stop_df = rt.groupby('route_id').agg(num_stop = (pl.col('type') ==  
02         'Dropoff').count()).sort(by = 'num_stop')  
03 plt.hist(num_stop_df["num_stop"], bins=10, edgecolor="black")  
04 plt.xlabel("Number of Stops")  
05 plt.ylabel("Number of Routes")  
06 plt.title("Distribution of Number of Stops per Route")  
07 plt.grid(True)  
08 plt.show()
```

Output:



**Figure 5.1.4** Distribution of Number of Stops per Route.

Inference:

- Most of the routes have around 230 – 260 number of stops.

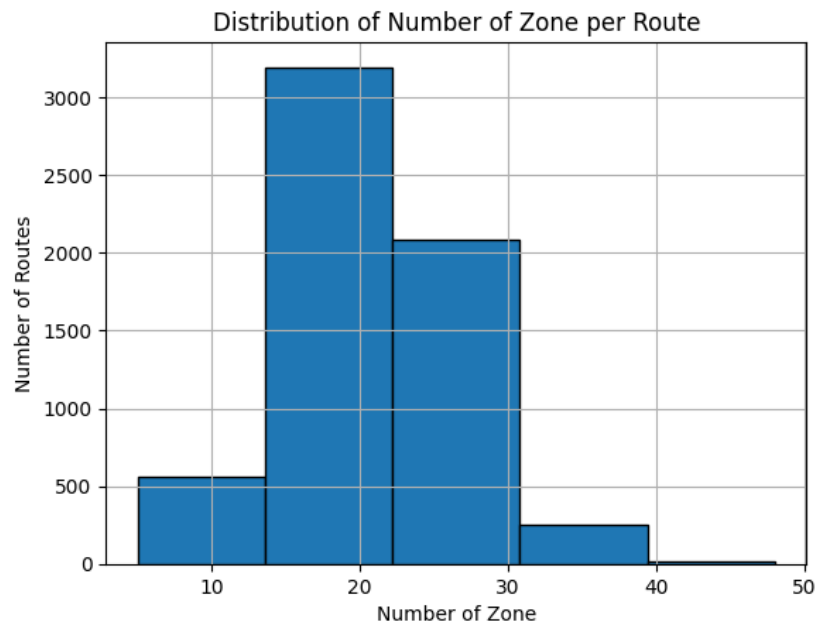
## CHAPTER 5

Below code, at line 1-2, number of zones for each route are obtained and sorted. Then, from line 3-8, histogram of the 'Distribution of Number of Zone per Route' are plotted, again utilizing the matplotlib library.

Line

```
01 num_zone_df = rt.groupby('route_id').agg(num_zone =
02         pl.col('zone_id').n_unique()).sort(by = 'num_zone')
03 plt.hist(num_zone_df["num_zone"], bins=5, edgecolor="black")
04 plt.xlabel("Number of Zone")
05 plt.ylabel("Number of Routes")
06 plt.title("Distribution of Number of Zone per Route")
07 plt.grid(True)
08 plt.show()
```

Output:



**Figure 5.1.5** Distribution of Number of Zone per Route.

Inference:

- Most of the routes serves about 20 zones per route.

**Zones**

The function below extracts zone data features (i.e., zone ID and stop sequences) to visualize the zone IDs and the overall route sequence.

```
def preprocess_route_simple(route_id, all_route_data, all_actual_seq):

    route_data = all_route_data[route_id]['stops']
    actual_seq = all_actual_seq[route_id]['actual']

    zone_features = {}

    for stop_id, stop_data in route_data.items():
        zone_id = stop_data['zone_id']
        stop_seq = actual_seq.get(stop_id, 1e6)

        if zone_id in zone_features:
            zone_features[zone_id].append(stop_seq)
        else:
            zone_features[zone_id] = [stop_seq]

    for zone_id in zone_features:
        zone_features[zone_id].sort()

    formatted_zone_features = {
        zone_id: {'stop_seq': stop_seq_list}
        for zone_id, stop_seq_list in zone_features.items()
    }

    return formatted_zone_features
```

**Figure 5.1.6** Function for Extraction Zone Data Features

After extracting the zone IDs and their corresponding sequences for each route, we inspected the data visually to understand how drivers move through zones. The code below used for this inspection sorts the zone IDs based on the stop sequences, groups the stops by their associated zone IDs, and then aggregates the sequences accordingly. We are able to visualize how stops within a particular zone are served in order, and how the sequence progresses as the driver moves from one zone to the next.

Line

```

01     zone_data = [{"zone_id": zone_id, "stop_seq": stop_seq}
02                 for zone_id, zone_info in data["zone_features"].items()
03                 for stop_seq in zone_info["stop_seq"]]
04     df = pd.DataFrame(zone_data)
05     pd.set_option('display.max_rows', None)
06     df_sorted = df.sort_values(by="stop_seq")
07     df_aggregated = df_sorted.groupby('zone_id').agg({'stop_seq':
08                 list}).reset_index()
09     df_aggregated['first_stop_seq'] = df_aggregated['stop_seq'].apply(
10                 lambda x: x[0])
11     df_aggregated = df_aggregated.sort_values(by='first_stop_seq').drop(
12                 columns='first_stop_seq')
13     print(df_aggregated)

```

Output:

```

zone_id                                stop_seq
C-3.2B                                [1, 2, 3, 4, 5, 6, 7, 8, 9]
C-3.1B                                [10, 11, 12, 13, 14, 15, 16, 17, 18]
C-3.1C  [19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]
C-3.2C                                [34, 35, 36, 37, 38, 39, 40, 41, 42, 43]
C-3.3C                                [44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54]
C-3.3D  [55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69]
C-3.2D  [70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84]
C-3.1D  [85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98]
C-3.1E  [99, 100, 101, 102, 103, 104, 105, 106, 107, 108]
C-3.2E                                [109, 110, 111, 112, 113, 114]
C-3.3E  [115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128]
C-3.3G  [129, 130, 131, 132, 133, 134, 135, 136, 137]
C-3.2G  [138, 139, 140, 141, 142, 143, 144, 145, 146]

```

**Figure 5.1.7** Zone ID Order sequence, aggregated from stop sequence for Route1

zone_id	stop_seq
D-3.1H	[1, 2, 3, 4, 5, 6, 7]
D-3.2H	[8, 9, 10, 11, 12]
D-3.3H	[13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 2...]
D-3.3J	[26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 3...]
D-3.2J	[49, 50, 51, 52, 53, 54]
D-3.1J	[55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65]
D-2.1J	[66, 67, 68, 69, 70]
D-2.2J	[71, 72, 73, 74, 75, 76, 77, 78]
D-2.3J	[79, 80, 81, 82, 83]
D-2.3H	[84, 85, 86, 87]
D-2.2H	[88, 89, 90, 91, 92, 93, 94, 95, 96]
D-2.1H	[97, 98, 99, 100, 101, 102]
D-2.1G	[103, 104, 105, 106, 107, 108, 109, 110]
D-2.2G	[111, 112, 113]
D-2.3G	[114, 115, 116, 117, 118]
D-2.3E	[119, 120, 121]
D-2.1E	[122, 123, 124, 125]
D-2.1D	[126, 127, 128, 129, 130, 131, 132, 133]
D-2.2D	[134, 135, 136, 137, 138, 139, 140]
D-2.3D	[141, 142, 143, 144, 145, 146, 147, 148, 149]
D-2.3C	[150, 151]
D-2.2C	[152, 153, 154, 155, 156, 157, 158, 159, 160]
D-2.1C	[161, 162, 163, 164, 165, 167, 168, 169, 170, ...]
D-2.1B	[172, 173, 174, 175, 176, 177, 178]

**Figure 5.1.8** Zone ID Order sequence, aggregated from stop sequence for Route2

Based on multiple route cases shown in **Figures 5.1.7** and **5.1.8**, we observed the stop sequences associated with various zone IDs. For example, zone ‘C-3.2B’ includes the stops in the sequence [1, 2, 3, ..., 9], while zone ‘C-3.1B’ covers stops [10, 11, 12, ..., 18]. This pattern is repeated across all zones in a structured manner, suggesting that within each zone, stops are served sequentially before moving on to the next zone. This pattern suggesting that drivers tend to focus on completing all stops within a given zone before moving to the next zone.

Inference:

- Driver behavior can be inferred based on the stop sequence data within each zone.
- Drivers tend to complete all stops within the current zone before moving on to the next zone, suggesting a zone-based delivery pattern.



***Relationship of Zone IDs***

Based on previous insights (i.e., drivers serve all stops within the current zone before moving to next zone), we conducted further inspections of zone IDs in the route sequence. Below shows the Zone ID Order sequence for Route 0. Note that '@-0.0@' represent Depot zone.

Outputs:

zone_id	zone_seq
@-0.0@	0
D-18.2J	1
D-18.3J	2
D-18.3H	3
D-18.2H	4
D-18.1H	5
D-18.1G	6
D-18.2G	7
D-18.3G	8
D-18.1E	9
D-18.1D	10
D-18.2D	11
D-18.3D	12
D-18.3C	13
D-18.2C	14
D-18.1C	15
D-18.1B	16

**Figure 5.1.9** Zone ID Order sequence for Route0

Based on the output, we noticed that zone IDs can be broken down into four parts: (i) super-super clusters, (ii) super clusters, (iii) clusters, and (iv) zones. For example, in the zone ID 'D-18.2J' the starting letter 'D' represent the super-super cluster. When paired with next integer value, in this case 'D-18' formed a super cluster. The ending letter 'J' represent the cluster, making 'D-18.-J'. Finally, the decimal value '.2' completes the zone ID, with 'D-18.J' represent the full zone ID.

**Table 5.1** Zone Order sequence for Route0, arranged by cluster

Cluster	Zone 1	Zone 2	Zone 3
J		D-18.2J	D-18.3J
H	D-18.3H	D-18.2H	D-18.1H
G	D-18.1G	D-18.2G	D-18.3G
E	D-18.1E		
D	D-18.1D	D-18.2D	D-18.3D
C	D-18.3C	D-18.2C	D-18.1C
B	D-18.1B		

Next, for a given route, the sequence pattern of zone IDs can be identified. Based on **Table 5.1**, we observed that the zone IDs within each cluster follow either an ascending or descending order, depending on the sequence of the previous cluster.

Inference:

- The relationship of zone IDs within a given route are identified.
- Each zone ID is structured in a hierarchical manner, starting with the super-super cluster, followed by the super cluster, cluster, and zone.
- The pattern of zone IDs within a route appears to follow a specific pattern (either ascending or descending) within each cluster, which may indicate a systematic approach to how delivery zones are being clustered.

## 5.2 Data Preprocessing

In this project, we focus on zone level sequence prediction, justified by [3], explained in previous chapter. For the route data, we select only routes with a score rated as 'High,' prioritizing quality data for better pattern learning from model. To prepare zone level data, relevant features are selected and aggregated from stop-level data. Stop-level features, including, latitude, longitude, stop type (indicating Depot or Dropoff), travel times between stop pairs, and corresponding actual route sequence (at stop level) are aggregated by zone IDs. Package-level features, such as time windows, planned service times, and dimensions, are excluded. This decision is based on the justification provided in previous sub chapter, stating that over 92.27% packages are considered time windows insensitive.

To begin, all route files are classified and separated based on their route score. Function *classify\_route* reads the *'route\_data.json'* file and returns three distinct lists: *'low\_routes'*, *'medium\_routes'*, and *'high\_routes'*, which contain the corresponding route IDs based on their score. Next, all relevant route data are sorted into their respective directories using the function *move\_files*.

Function *classify\_route*

```
def classify_routes(input_file):
    with open(input_file, 'r') as f:
        data = json.load(f)
        for route_id, route_info in data.items():
            route_score = route_info.get("route_score", "Uncategorized")
            if route_score == "Low":
                low_routes.append(route_id)
            elif route_score == "Medium":
                medium_routes.append(route_id)
            elif route_score == "High":
                high_routes.append(route_id)
```

Function *move\_files*

```
def move_files(input_directory, output_directory):
    for filename in os.listdir(input_directory):
        if filename.endswith(".json"):
            shutil.move(os.path.join(input_directory, filename),
                        os.path.join(output_directory, filename))
```

The function preprocesses all routes data are defined as *preprocess\_route()*, with seven defined functions, namely *parse\_zone\_id*, *is\_valid\_zone\_id*, *assign\_zone\_id\_for\_dropoffs*, *encode\_zone\_id*, *extract\_zone\_features*, *calculate\_mean\_travel\_times*, *convert\_stop\_sequence\_to\_zone\_sequence*. The preprocessing function takes in lists: (1) *route\_id*, (2) *all\_route\_data*, (3) *all\_travel\_times*, (4) *all\_package\_data*, and (5) *all\_actual\_seq*, and return lists; (i) *zone\_features*, and (ii) *mean\_travel\_times*.

Function *preprocess\_route*

```
def preprocess_route(route_id, all_route_data, all_travel_times, all_package_data,
                    all_actual_seq):
    route_data = all_route_data[route_id]['stops']
    travel_times = all_travel_times[route_id]
    package_data = all_package_data[route_id]
    actual_seq = all_actual_seq[route_id]['actual']
    route_data = assign_zone_id_for_dropoffs(route_data, travel_times)
    encoded_zones, station_zone = encode_zone_id(route_data)
    zone_features = extract_zone_features(encoded_zones, station_zone,
                                         package_data)
    mean_travel_times = calculate_mean_travel_times(zone_features, travel_times,
                                                    encoded_zones)
    zone_seq_map = convert_stop_sequence_to_zone_sequence(actual_seq,
                                                         route_data)
    for zone_id in zone_features.keys():
```

```

if zone_id == station_zone: zone_features[zone_id]['zone_seq'] = 0
else: zone_features[zone_id]['zone_seq'] = zone_seq_map.get(zone_id, 1e6)
return zone_features, mean_travel_times

```

The preprocessing pipeline is structured around the *preprocess\_route* function, which coordinates the entire data preparation process for each route. It begins by handling missing zone IDs using the *assign\_zone\_id\_for\_dropoffs* function. This step assigns the nearest valid zone ID to dropoff stops without a valid zone ID based on travel times, ensuring that all stops have valid zone data. The assignment of nearest zone can be expressed in the following mathematical expression:

Let:

- (1) The dropoff stop with a missing or invalid zone ID,  $s_m \in S$ , a set of stops in given route.
- (2) A valid stop with a known zone ID,  $z_i \in S$ .
- (3) The travel time between stop  $s_m$  and  $s_v$ ,  $t(s_m, z_i)$ .

$$Z_{nearest} = \arg \min_i t(s_m, z_i) \quad (16)$$

where  $Z_{nearest}$  is the zone ID of the nearest stop, which is identified by finding the stop  $z_i$  that yields the minimum travel time  $t(s_m, z_i)$ .

The function *is\_valid\_zone\_id* function is called to validate the *zone\_id* format. A valid zone id is defined in format: `^[A-Z]-\d{1,2}\.\d[A-Z]$`.

Function *assign\_zone\_id\_for\_dropoffs*

```

def assign_zone_id_for_dropoffs(route_data, travel_times):
    for stop_id, stop_data in route_data.items():
        zone_id = stop_data.get('zone_id')
        if zone_id is None or not is_valid_zone_id(zone_id):
            if stop_data['type'] == 'Dropoff':

```

```

nearest_zone = None
min_travel_time = float('inf')

# Find the nearest stop with a valid zone_id
for neighbor_stop_id, travel_time in travel_times.get(stop_id, {}).items():
    neighbor_zone_id = route_data[neighbor_stop_id].get('zone_id')

    if neighbor_zone_id and is_valid_zone_id(neighbor_zone_id):
        if travel_time < min_travel_time:
            min_travel_time = travel_time
            nearest_zone = neighbor_zone_id

# Assign the nearest zone_id to the dropoff stop
if nearest_zone:
    route_data[stop_id]['zone_id'] = nearest_zone
else:
    # If no valid zone_id is found, assign a default
    route_data[stop_id]['zone_id'] = "[-9.9]"

return route_data

```

Function *is\_valid\_zone\_id*

```

def is_valid_zone_id(zone_id):
    if not isinstance(zone_id, str):
        return False # If zone_id is not a string, it's invalid
    pattern = re.compile(r'^[A-Z]-\d{1,2}\.\d[A-Z]$')
    # If the zone_id does not match the pattern, it is invalid
    if not pattern.match(zone_id):
        return False
    return True

```

Next, the *encode\_zone\_id* function processes the zone IDs for every stop, by calling function *parse\_zone\_id*, to split the zone ids into four components (super-super cluster,

super cluster, cluster, zone) and obtain each stop features such as latitude, longitude, and whether the stop is a station. For station stop, default zone ID '@-0.0@' is assigned. This step creates a structured representation of each stop's zone information.

Function *encode\_zone\_id*

```
def encode_zone_id(route_data):
    default_station_zone = "@-0.0@"
    dropoff_with_missing_zone = []
    encoded_zones = {}
    station_zone = None
    for stop_id, stop_data in route_data.items():
        zone_id = stop_data.get('zone_id')

        # Check for NaN values in various forms
        if zone_id is None or zone_id in ['NaN', 'nan'] or (isinstance(zone_id, float) and
            np.isnan(zone_id)):
            if stop_data['type'] == 'Station':
                zone_id = default_station_zone
                station_zone = default_station_zone # Mark the station zone
            parsed_zone = parse_zone_id(zone_id)
            # Add the encoded zone and other features to the dictionary
            encoded_zones[stop_id] = {
                'zone_id': zone_id,
                'super_super_cluster': parsed_zone['super_super_cluster'],
                'integer_part': parsed_zone['integer_part'],
                'decimal_part': parsed_zone['decimal_part'],
                'super_cluster': parsed_zone['super_cluster'],
                'is_station': 1 if stop_data['type'] == 'Station' else 0,
                'lat': stop_data['lat'],
                'lng': stop_data['lng']
            }
    return encoded_zones, station_zone
```

Function *parse\_zone\_id*

```

def parse_zone_id(zone_id):
    temp = zone_id.split('-')
    super_super_cluster = ord(temp[0]) - 64
    temp2 = temp[1].split('.')
    integer_part = int(temp2[0])
    decimal_part = int(temp2[1][:-1])
    super_cluster = ord(temp2[1][-1]) - 64

    return {
        'super_super_cluster': super_super_cluster,
        'integer_part': integer_part,
        'decimal_part': decimal_part,
        'super_cluster': super_cluster
    }

```

The pipeline then proceeds to the *extract\_zone\_features* function, which aggregates stop-level data at the zone level. This includes counting the number of stops in each zone, calculating the minimum, maximum, and mean latitudes and longitudes, and determining the number of packages handled in each zone. Station zones are marked as '1' for feature *'is\_station'*.

Function *extract\_zone\_features*

```

def extract_zone_features(encoded_zones, station_zone, package_data):
    zone_features = defaultdict(lambda: {
        'super_super_cluster': None,
        'super_cluster': None,
        'integer_part': None,
        'decimal_part': None,
        'is_station': 0,
        'num_stops': 0,

```



```

    'latitudes': [],
    'longitudes': [],
    'num_of_packages': 0
})

for stop_id, stop_data in encoded_zones.items():
    zone_id = stop_data['zone_id']

    # Aggregate zone_id features
    zone_features[zone_id]['super_super_cluster'] =
        stop_data['super_super_cluster']
    zone_features[zone_id]['super_cluster'] = stop_data['super_cluster']
    zone_features[zone_id]['integer_part'] = stop_data['integer_part']
    zone_features[zone_id]['decimal_part'] = stop_data['decimal_part']

    # Count the number of stops and collect latitudes/longitudes
    zone_features[zone_id]['num_stops'] += 1
    zone_features[zone_id]['latitudes'].append(stop_data['lat'])
    zone_features[zone_id]['longitudes'].append(stop_data['lng'])

    # Mark if the stop is a station
    if stop_data['is_station']:
        zone_features[zone_id]['is_station'] = 1

    # Count the number of packages at each stop and aggregate at the zone level
    if stop_id in package_data:
        zone_features[zone_id]['num_of_packages'] += len(package_data[stop_id])

    # Compute min, mean, max lat/lng for each zone
    for zone_id, features in zone_features.items():
        features['min_lat'] = min(features['latitudes'])
        features['max_lat'] = max(features['latitudes'])
        features['mean_lat'] = np.mean(features['latitudes'])

```

```

features['min_lng'] = min(features['longitudes'])
features['max_lng'] = max(features['longitudes'])
features['mean_lng'] = np.mean(features['longitudes'])

return zone_features

```

After the zone features are aggregated, the *calculate\_mean\_travel\_times* function computes the mean travel times between different zones based on the travel times between individual stops in different zones. The mean travel times between two zones can be expressed in the below mathematical equation:

Given:

- (1) The travel time between stop  $i$  in zone A and stop  $j$  in zone B,  $T_{iAjB}$ .
- (2) The total number of stop pairs between zone A and zone B,  $N$ .

$$\mu_{AB} = \frac{1}{N} \sum_{i=1}^N T_{iAjB} \quad (17)$$

where  $\mu_{AB}$  is the mean travel time between zone A and zone B.  $N$  is the number of travel times between the stop pairs of zone A and zone B.  $T_{iAjB}$  represents the individual travel time between stop  $i$  (in zone A) and stop  $j$  (in zone B).

Function *calculate\_mean\_travel\_times*

```

def calculate_mean_travel_times(zone_features, travel_times, encoded_zones):
    zone_travel_times = defaultdict(lambda: defaultdict(list))

    for from_stop, to_stops in travel_times.items():
        from_zone = encoded_zones[from_stop]['zone_id']
        for to_stop, travel_time in to_stops.items():
            to_zone = encoded_zones[to_stop]['zone_id']

```

```

    if from_zone != to_zone:
        zone_travel_times[from_zone][to_zone].append(travel_time)

mean_travel_times = {}
for from_zone, to_zones in zone_travel_times.items():
    mean_travel_times[from_zone] = {}
    for to_zone, times in to_zones.items():
        mean_travel_times[from_zone][to_zone] = np.mean(times)

return mean_travel_times

```

Finally, the *convert\_stop\_sequence\_to\_zone\_sequence* function translates the stop-level route sequence into a zone-level sequence, ensuring that the correct order of zones is captured. The zone-level sequence is then added to the zone features.

Function *convert\_stop\_sequence\_to\_zone\_sequence*

```

def convert_stop_sequence_to_zone_sequence(actual_seq, stop_data):
    # Sort the actual sequence based on the stop positions
    sorted_stops = sorted(actual_seq, key=actual_seq.get)
    seen_zones = set() # Track zones we've already encountered
    zone_seq_map = {} # Map to store zone_id and its sequence index
    seq_index = 0    # Sequence index starts at 0

    for stop_id in sorted_stops:
        # Get the zone_id for the current stop
        zone_id = stop_data[stop_id]['zone_id']

        # Add the zone to the sequence if it's the first time we encounter it
        if zone_id not in seen_zones:
            zone_seq_map[zone_id] = seq_index
            seen_zones.add(zone_id)
            seq_index += 1

    return zone_seq_map

```

### 5.3 Data Transformation and Padding

#### 5.3.1 Data Transformation

The *extract\_zone\_features\_and\_sequences* function is responsible for transforming route data into training data (*X\_train* and *Y\_train*) for use in zone-level sequence prediction models. This function takes in all preprocessed data, from previous step and constructs feature vectors (*X\_train*) for each zone, consisting of (1) super-super cluster, (2) super cluster, (3) cluster, (4) zone, (5) number of stops in the zone, (6) number of packages in the zone, (7) mean latitude of zone, (8) mean longitude of zone, (9) all interzonal travel times in the route. During the transformation, the depot station zone is explicitly placed at the first zone in every route. For *Y\_train*, the sequence index of each zone is stored, indicating the order in which the zones are visited. By the end of the process, *X\_train* contains feature vectors for each zone, including the travel times, and *Y\_train* holds the sequence indices corresponding to the visit order of the zones.

Function *extract\_zone\_features\_and\_sequences*

```
def extract_zone_features_and_sequences(route_data):
    X_train = []
    Y_train = []
    travel_times = route_data['mean_travel_times']
    depot_zone = None
    non_depot_zones = []
    final_zone_order = []

    # Separate the depot and non-depot zones
    for zone_id, zone_data in route_data['zone_features'].items():
        # Create feature vector for the zone
        feature_vector = [
            zone_data['super_super_cluster'],
            zone_data['super_cluster'],
            zone_data['integer_part'],
            zone_data['decimal_part'],
```

```

zone_data['num_stops'],
zone_data['num_of_packages'],
zone_data['mean_lat'],
zone_data['mean_lng']
]

# Check if it's a Depot (is_station == 1)
if zone_data['is_station'] == 1:
    depot_zone = (zone_id, feature_vector, zone_data['zone_seq'])
else:
    non_depot_zones.append((zone_id, feature_vector, zone_data['zone_seq']))

# Ensure Depot is the first input and output
if depot_zone:
    final_zone_order.append(depot_zone[0])
    X_train.append(depot_zone[1])
    Y_train.append(depot_zone[2])

# Append the non-depot zones after the depot in the original input order
for zone in non_depot_zones:
    final_zone_order.append(zone[0])
    X_train.append(zone[1])
    Y_train.append(zone[2])

# Now that final_zone_order is established, add travel times for each zone based
on this order
for idx, zone_id in enumerate(final_zone_order):
    travel_distances = []
    for dest_zone_id in final_zone_order:
        if zone_id == dest_zone_id:
            travel_distances.append(0.0) # Travel time to self is 0
        elif zone_id in travel_times and dest_zone_id in travel_times[zone_id]:
            travel_distances.append(travel_times[zone_id][dest_zone_id])

```

```

#else:
    #travel_distances.append(9999) # Use default if travel time is missing

# Add travel distances to the feature vector
X_train[idx].extend(travel_distances)

return X_train, Y_train

```

### 5.3.2 Padding

Padding is essential for handling input data (X) with varying lengths, ensuring consistent input dimensions for model feeding in the later process. The target variable (Y) is also padded to maintain a consistent shape. The *pad\_route\_features* function is used to perform padding on X data at two levels: (1) zone-level, (2) feature-level. This ensures that each route in the dataset has a consistent number of zones by padding routes that have fewer zones than the maximum number of zones observed across all routes. In our case, the maximum number of zones across all routes are identified with code below, which is 48.

```
max_zones = max([len(route) for route in X_train_all])
```

The function takes the data (X) and the value of the maximum number of zones as input. For each route, X is padded at the feature level by iterating over the route features, which represent the features for each zone within a route. Each zone's feature set is divided into two parts: the first 8 elements are static features (such as zone ID, number of stops, etc.), and the remaining elements represent the travel distances between zones. If the number of travel distances for a zone is less than *max\_zones* (the maximum number of zones across all routes), the travel distances are padded with zeros using `np.pad`. The zeros ensure that the length of travel distances matches *max\_zones*, ensuring consistency in feature dimensions. After padding the travel distances, the static features and padded travel distances are concatenated to form a complete padded feature vector for that zone.

Once all zones in a route are processed, the function performs padding at the zone level. For routes that contain fewer than 48 zones, additional zero-filled feature vectors are appended to the route until the total number of zones equals *max\_zones*. This ensures that even routes with fewer zones are padded to match the maximum zone length, creating a consistent data structure across all routes. As a result, X will have a consistent dimension of (None, 48, 56).

Function *pad\_route\_features*

```
def pad_route_features(route_features, max_zones):
    padded_routes = []
    # Pad existing zones with actual features
    for features in route_features:
        static_features = features[:8]
        travel_distances = features[8:]
        # Pad travel distances to match the max_zones
        padded_travel_distances = np.pad(travel_distances,
                                         (0, max_zones - len(travel_distances)),
                                         mode='constant', constant_values=0)
        # Combine static features and padded travel distances
        padded_features = np.concatenate((static_features, padded_travel_distances))
        padded_routes.append(padded_features)

    # If the route has fewer zones than max_zones, pad the remaining zones with zeros
    num_existing_zones = len(route_features)
    if num_existing_zones < max_zones:
        # Create zero-filled feature vectors for the non-existent zones
        zero_padding = [np.zeros(len(padded_routes[0]))] *
                       (max_zones - num_existing_zones)
        padded_routes.extend(zero_padding)

    return padded_routes
```

## CHAPTER 5

For data Y, np.pad is used, where shorter route sequences are padded with -1 at the end, making every sequence the same length. This padding ensures the data is ready for model input, with -1 marking the padded, non-existent zones.

```
padded_Y = [np.pad(seq, (0, max_zones - len(seq)),  
                  mode='constant', constant_values=-1) for seq in Y_all]
```



## 5.4 Model Building

### 5.4.1 Simple RNN Encoder-Decoder

For Simple RNN Encoder-Decoder model implementation, the model structure is constructed following methodology proposed in **Chapter 3**. For both the Simple RNN encoder and decoder, the hidden unit sizes are set to 128.

The Simple RNN Encoder consists of four embedding layers for zone id features (super-super cluster, super cluster, cluster, and zone), followed by a SimpleRNN layer that outputs both the encoder output and the hidden state. The expected input for Simple RNN Encoder is zones features (split into i. super-super cluster, ii. cluster, iii. super cluster, iv. zone, v. other continuous zone features) and a valid zone mask, that indicates valid zones in given route input. All the encoder inputs are concatenate before feeding into the Simple RNN cells.

Class Simple RNN Encoder Model

```
class SimpleRNNEncoder(tf.keras.Model):
    def __init__(self, hidden_size):
        super(SimpleRNNEncoder, self).__init__()
        self.embedding_1 = tf.keras.layers.Embedding(input_dim=28, output_dim=4,
                                                    mask_zero=True)
        self.embedding_2 = tf.keras.layers.Embedding(input_dim=28, output_dim=10,
                                                    mask_zero=True)
        self.embedding_3 = tf.keras.layers.Embedding(input_dim=101,
                                                    output_dim=10, mask_zero=True)
        self.embedding_4 = tf.keras.layers.Embedding(input_dim=11, output_dim=6,
                                                    mask_zero=True)

        self.simple_rnn = tf.keras.layers.SimpleRNN(hidden_size,
                                                    return_sequences=True, return_state=True, name="encoder_rnn")
```

```

def call(self, input_feature_1, input_feature_2, input_feature_3, input_feature_4,
         input_continuous, masked_input):

    embedding_1 = self.embedding_1(input_feature_1)
    embedding_2 = self.embedding_2(input_feature_2)
    embedding_3 = self.embedding_3(input_feature_3)
    embedding_4 = self.embedding_4(input_feature_4)

    concat_embeddings = tf.keras.layers.Concatenate(axis=-1)([embedding_1,
                                                             embedding_2, embedding_3, embedding_4])
    final_input = tf.keras.layers.Concatenate(axis=-1)([concat_embeddings,
                                                         input_continuous])

    masked_input = tf.keras.layers.Masking(mask_value=0.0)(final_input)

    encoder_output, encoder_state = self.simple_rnn(masked_input)

    return encoder_output, encoder_state

```

### Building Simple RNN Encoder Model

```

def build_encoder(hidden_size):

    input_feature_1 = tf.keras.Input(shape=(None,), name="feature_1_input")
    input_feature_2 = tf.keras.Input(shape=(None,), name="feature_2_input")
    input_feature_3 = tf.keras.Input(shape=(None,), name="feature_3_input")
    input_feature_4 = tf.keras.Input(shape=(None,), name="feature_4_input")
    input_continuous = tf.keras.Input(shape=(None, 52), name="continuous_input")
    mask_input = tf.keras.Input(shape=(None,), name="mask_input")

    encoder = SimpleRNNEncoder(hidden_size)

    encoder_output, encoder_state = encoder(

```

```

        input_feature_1, input_feature_2, input_feature_3, input_feature_4,
        input_continuous, mask_input)

encoder_model = tf.keras.Model(
    inputs=[input_feature_1, input_feature_2, input_feature_3, input_feature_4,
           input_continuous, mask_input],
    outputs=[encoder_output, encoder_state]
)

return encoder_model

SimpleRNN_encoder = build_encoder(hidden_size)

```

The Simple RNN Decoder takes in the last visited zone features, previous decoder state, encoder outputs and a valid zone mask, that indicates valid and unvisited zones in given route input, as input. The last visited zone features with previous decoder state is processed through a SimpleRNN layer and combine the output with the mean of the encoder outputs to produce logits. These logits are passed through a dense layer with a vocabulary size of 48, followed by masking. Finally, a softmax layer is applied on the logits produced, to predict the next output in the sequence.

#### Class Simple RNN Decoder Model

```

class SimpleRNNDecoder(tf.keras.Model):
    def __init__(self, hidden_size, vocab_size):
        super(SimpleRNNDecoder, self).__init__()
        self.hidden_size = hidden_size
        self.simple_rnn = tf.keras.layers.SimpleRNN(hidden_size, return_state=True)
        self.fc = tf.keras.layers.Dense(vocab_size)

    def call(self, decoder_input, decoder_state, encoder_outputs, mask):

```

```

decoder_input = tf.expand_dims(decoder_input, 1)

rnn_output, decoder_state = self.simple_rnn(decoder_input,
                                             initial_state=decoder_state)

combined_output = tf.concat([rnn_output, tf.reduce_mean(encoder_outputs,
                                                         axis=1)], axis=-1)

logits = self.fc(combined_output)

if mask is not None:
    logits += (mask * -1e9)

probabilities = Softmax()(logits)
predictions = tf.argmax(probabilities, axis=-1)
return logits, predictions, decoder_state

```

### Building Simple RNN Decoder Model

```

def build_decoder(hidden_size, vocab_size):

    decoder_input = tf.keras.Input(shape=(52,), name="decoder_input")
    decoder_state_input = tf.keras.Input(shape=(hidden_size,),
                                         name="decoder_state_input")
    encoder_outputs_input = tf.keras.Input(shape=(None, hidden_size),
                                           name="encoder_outputs_input")
    mask_input = tf.keras.Input(shape=(None,), name="mask_input")

    decoder = SimpleRNNDecoder(hidden_size, vocab_size)

    logits, predictions, decoder_state = decoder(
        decoder_input,
        decoder_state_input,
        encoder_outputs_input,
        mask_input

```

```

)

decoder_model = tf.keras.Model(
    inputs=[decoder_input, decoder_state_input, encoder_outputs_input,
           mask_input],
    outputs=[logits, predictions, decoder_state]
)

return decoder_model

SimpleRNN_decoder = build_decoder(hidden_size, vocab_size)

```

#### 5.4.2 LSTM Encoder-Decoder with Attention

For LSTM Encoder-Decoder with Attention model implementation, the model structure is constructed proposed in Chapter 3. For both the LSTM encoder and decoder with Attention, the hidden unit sizes are set to 128.

The LSTM Encoder consists of four embedding layers for zone id features (super-super cluster, super cluster, cluster, and zone), followed by a LSTM layer that outputs both the encoder output and the hidden state. The expected input for Simple RNN Encoder is zones features (split into i. super-super cluster, ii. cluster, iii. super cluster, iv. zone, v. other continuous zone features) and a valid zone mask, that indicates valid zones in given route input. All the encoder inputs are concatenate before feeding into the LSTM cells.

##### Class LSTM Encoder Model

```

class Encoder(tf.keras.Model):
    def __init__(self, hidden_size):
        super(Encoder, self).__init__()
        self.embedding_1 = tf.keras.layers.Embedding(input_dim=28, output_dim=4,

```

```

                                                    mask_zero=True)
self.embedding_2 = tf.keras.layers.Embedding(input_dim=28, output_dim=10,
                                                    mask_zero=True)
self.embedding_3 = tf.keras.layers.Embedding(input_dim=101, output_dim=8,
                                                    mask_zero=True)
self.embedding_4 = tf.keras.layers.Embedding(input_dim=11, output_dim=6,
                                                    mask_zero=True)

self.lstm = tf.keras.layers.LSTM(hidden_size, return_sequences=True,
                                return_state=True, name="encoder_lstm")

def call(self, input_feature_1, input_feature_2, input_feature_3, input_feature_4,
        input_continuous, masked_input):
    embedding_1 = self.embedding_1(input_feature_1)
    embedding_2 = self.embedding_2(input_feature_2)
    embedding_3 = self.embedding_3(input_feature_3)
    embedding_4 = self.embedding_4(input_feature_4)

    concat_embeddings = tf.keras.layers.Concatenate(axis=-1)([embedding_1,
                                                            embedding_2, embedding_3, embedding_4])
    final_input = tf.keras.layers.Concatenate(axis=-1)([concat_embeddings,
                                                        input_continuous])

    masked_input = tf.keras.layers.Masking(mask_value=0.0)(final_input)

    encoder_output, encoder_hidden, encoder_cell = self.lstm(masked_input)

    return encoder_output, encoder_hidden, encoder_cell

```

### Building LSTM Encoder Model

```
def build_encoder(hidden_size):
```

```

input_feature_1 = tf.keras.Input(shape=(None,), name="feature_1_input")
input_feature_2 = tf.keras.Input(shape=(None,), name="feature_2_input")
input_feature_3 = tf.keras.Input(shape=(None,), name="feature_3_input")
input_feature_4 = tf.keras.Input(shape=(None,), name="feature_4_input")
input_continuous = tf.keras.Input(shape=(None, 52),
                                   name="continuous_input") # Continuous features input
mask_input = tf.keras.Input(shape=(None,), name="mask_input")

encoder = Encoder(hidden_size)

encoder_output, encoder_hidden, encoder_cell = encoder(
    input_feature_1, input_feature_2, input_feature_3, input_feature_4,
    input_continuous, mask_input
)

encoder_model = tf.keras.Model(
    inputs=[input_feature_1, input_feature_2, input_feature_3, input_feature_4,
           input_continuous, mask_input],
    outputs=[encoder_output, encoder_hidden, encoder_cell]
)

return encoder_model

APNN_encoder = build_encoder(hidden_size)

```

The LSTM Decoder with Attention takes in last visited zone index, last visited zone features, previous decoder state, encoder outputs, a valid zone mask, that indicates valid and unvisited zones in given route input, and timestep index, as input.

At timestep 0, context vector,  $\omega$  (see **Chapter 3**) and attention index are initialized with the value of 0. The next zone features (here referring to depot zone features) with initial context vector,  $\omega_{(0)} = 0$ , is processed through a LSTM layer to

produce hidden decoder state, and logits. These logits are passed through a dense layer with a vocabulary size of 48.

At subsequent timestep, context vector,  $\omega_{(i)}$  (see **Chapter 3**) and attention index are determined from the attention model. The next zone features (determined by attention index) with the corresponding context vector,  $\omega_{(i)}$ , is processed through a LSTM layer to produce hidden decoder state, and logits.

For next zone predictions, the attention layer, with hidden layer of 128 produced context vectors,  $\omega_{(j)}$ , along with attention weights,  $\alpha^j_{(i)}$  for every candidate stops,  $s_j$ . For calculation of attention weights and context vector, see Chapter 3. The next zone prediction is determined by the highest attention weights of given candidate stops,  $s_j$ .

#### Class LSTM Decoder with Attention Model

```
class DecoderWithAttention(tf.keras.Model):
    def __init__(self, hidden_size, vocab_size):
        super(DecoderWithAttention, self).__init__()
        self.hidden_size = hidden_size
        self.lstm = tf.keras.layers.LSTM(hidden_size, return_sequences=False,
                                         return_state=True)
        self.attention = PointerAttention(hidden_size)
        self.fc = tf.keras.layers.Dense(vocab_size)

    def call(self, last_visited_idx, decoder_input, decoder_hidden, decoder_output,
            encoder_outputs, zone_features, mask, timestep):
        batch_size = tf.shape(encoder_outputs)[0]
        seq_len = tf.shape(encoder_outputs)[1]

        decoder_initial_state = decoder_hidden

    def timestep_zero():
        context_vector = tf.zeros([batch_size, self.hidden_size])
        attention_idx = tf.zeros([batch_size], dtype=tf.int64)
```



```

selected_zone_features = tf.gather(zone_features, attention_idx,
                                   batch_dims=1)
decoder_input_with_context = tf.concat([tf.expand_dims(context_vector, 1),
                                       tf.expand_dims(selected_zone_features, 1)], axis=-1)
lstm_output, decoder_hidden, decoder_cell = self.lstm(
    decoder_input_with_context, initial_state=decoder_initial_state)
logits = self.fc(lstm_output)
return lstm_output, decoder_hidden, decoder_cell, logits, attention_idx

def timestep_n():
    context_vector, attention_weights = self.attention(last_visited_idx,
                                                    encoder_outputs, decoder_output, zone_features, mask)
    attention_idx = tf.argmax(attention_weights, axis=1)
    selected_zone_features = tf.gather(zone_features, attention_idx,
                                       batch_dims=1)
    decoder_input_with_context = tf.concat([tf.expand_dims(context_vector, 1),
                                           tf.expand_dims(selected_zone_features, 1)], axis=-1)
    lstm_output, decoder_hidden, decoder_cell = self.lstm(
        decoder_input_with_context, initial_state=decoder_initial_state)
    logits = self.fc(lstm_output)
    return lstm_output, decoder_hidden, decoder_cell, logits, attention_idx

lstm_output, decoder_hidden_out, decoder_cell, logits, attention_idx = tf.cond(
    tf.equal(timestep, 0),
    timestep_zero,
    timestep_n
)
return lstm_output, decoder_hidden_out, decoder_cell, logits, attention_idx

```

### Class Pointer Attention

```

class PointerAttention(layers.Layer):
    def __init__(self, hidden_size):

```

```

super(PointerAttention, self).__init__()
self.hidden_size = hidden_size
self.mlp = layers.Dense(1)

def call(self, last_visited_idx, encoder_outputs, decoder_hidden, zone_features,
        mask):
    seq_len = tf.shape(encoder_outputs)[1]

    travel_time_index_w = last_visited_idx + 4

    seq_indices = tf.range(seq_len)[tf.newaxis, :]

    travel_time_index_w_tiled = tf.tile(travel_time_index_w[:, tf.newaxis], [1,
                                                                              seq_len])
    travel_time_indices = tf.stack([tf.zeros_like(seq_indices), seq_indices,
                                   travel_time_index_w_tiled], axis=-1)
    travel_times = tf.gather_nd(zone_features, travel_time_indices)
    travel_times = tf.expand_dims(travel_times, -1)
    decoder_hidden_with_time_axis = tf.expand_dims(decoder_hidden, 1)
    decoder_hidden_tiled = tf.tile(decoder_hidden_with_time_axis, [1, seq_len, 1])
    v_j = tf.concat([travel_times, decoder_hidden_tiled, encoder_outputs], axis=-1)

    u_j = self.mlp(v_j)
    u_j = tf.squeeze(u_j, -1)

    if mask is not None:
        u_j += (mask * -1e9)

    a_j = tf.nn.softmax(u_j, axis=1)

    context_vector = tf.reduce_sum(a_j[:, :, tf.newaxis] * encoder_outputs, axis=1)

    return context_vector, a_j

```

## Building LSTM Decoder with Attention Model

```

def build_decoder(hidden_size, attention_layer):

    last_visited_idx = tf.keras.Input(shape=(), dtype=tf.int32,
                                       name="last_visited_idx_input")
    decoder_input = tf.keras.Input(shape=(52,), name="decoder_input")
    decoder_hidden_input = tf.keras.Input(shape=(hidden_size,),
                                           name="decoder_hidden_input")
    decoder_cell_input = tf.keras.Input(shape=(hidden_size,),
                                         name="decoder_cell_input")
    encoder_output_input = tf.keras.Input(shape=(None, hidden_size),
                                           name="encoder_output_input")
    continuous_input = tf.keras.Input(shape=(None, 52), name="continuous_input")
    mask_input = tf.keras.Input(shape=(None,), name="mask_input")
    t_input = tf.keras.Input(shape=(), dtype=tf.int32, name="timestep_input")
    last_decoder_output = tf.keras.Input(shape=(hidden_size,),
                                         name="decoder_output")

    decoder = DecoderWithAttention(hidden_size, attention_layer)

    decoder_output, decoder_hidden, decoder_cell, logits, predictions = decoder(
        last_visited_idx,
        decoder_input,
        [decoder_hidden_input, decoder_cell_input],
        last_decoder_output,
        encoder_output_input,
        continuous_input,
        mask_input,
        t_input,
    )

    decoder_model = tf.keras.Model(

```

```
inputs=[last_visited_idx, decoder_input, decoder_hidden_input,  
        decoder_cell_input, last_decoder_output, encoder_output_input,  
        continuous_input, mask_input, t_input],  
outputs=[decoder_output, decoder_hidden, decoder_cell, logits, predictions]  
)  
  
return decoder_model  
  
APNN_decoder = build_decoder(hidden_size, vocab_size)
```

## 5.5 Model Training

For Model Training, custom training loops are defined, with main training process defined at Function *train\_model*, and for each timestep, process are defined in Function *train\_steps*. For the detailed process, including process at every timestep at decoder, inputs, are defined at **Chapter 3**.

Function *train\_model*

```
def train_model(encoder_model, decoder_model, X_train, Y_train, optimizer,
epochs):

    for epoch in range(epochs):
        epoch_loss = 0.0
        epoch_accuracy = 0.0

        # Shuffle the training data at the start of each epoch
        X_train_shuffled, Y_train_shuffled = shuffle_data(X_train, Y_train)

        for route_idx in range(len(X_train_shuffled)):
            inputs = X_train_shuffled[route_idx]
            targets = Y_train_shuffled[route_idx]
            targets = transpose_target_Y(targets)
            inputs = np.array(inputs)
            targets = np.array(targets)
            avg_loss, predicted_Y = train_step(inputs, targets, encoder_model,
                                                decoder_model, optimizer)
            route_accuracy = evaluate_predictions(predicted_Y[tf.newaxis, ...],
                                                targets[tf.newaxis, ...])

            epoch_loss += avg_loss.numpy()
            epoch_accuracy += route_accuracy
        epoch_loss /= len(X_train_shuffled)
        epoch_accuracy /= len(X_train_shuffled)
```

```
print(f'\nEpoch {epoch + 1}/{epochs}, Loss: {epoch_loss}, Accuracy:
[{epoch_accuracy}])
```

Function *train\_step*

```
def train_step(inputs, targets, encoder_model, decoder_model, optimizer):
    loss = 0
    predicted_Y = []

    with tf.GradientTape() as tape:
        # Prepare inputs for the encoder
        input_feature_1 = inputs[:, 0]
        input_feature_2 = inputs[:, 1]
        input_feature_3 = inputs[:, 2]
        input_feature_4 = inputs[:, 3]
        input_continuous = inputs[:, 4:]

        x_mask = create_x_mask(inputs[tf.newaxis, ...])
        y_mask = create_y_mask(targets[tf.newaxis, ...])

        # Run the encoder
        encoder_outputs, encoder_hidden, _ = encoder_model(
            [input_feature_1[tf.newaxis, ...],
             input_feature_2[tf.newaxis, ...],
             input_feature_3[tf.newaxis, ...],
             input_feature_4[tf.newaxis, ...],
             input_continuous[tf.newaxis, ...],
             x_mask]
        )

        # Initialize decoder state
        decoder_input = input_continuous[0] # Start with the first location
        decoder_hidden = encoder_hidden
```

```

last_visited_idx = tf.constant(0, dtype=tf.int32)
decoder_cell = tf.zeros_like(encoder_hidden)
decoder_output = tf.zeros_like(encoder_hidden)
visited_mask = tf.zeros_like(x_mask[0])
check_mask = tf.ones_like(x_mask[0])

seq_len = tf.shape(inputs)[0]

for t in range(seq_len):
    combined_mask = combine_masks(x_mask, visited_mask)
    if tf.reduce_all(tf.equal(combined_mask, check_mask)):
        break
    decoder_output, decoder_hidden, decoder_cell, logits, predictions =
    decoder_model(
        [last_visited_idx[tf.newaxis, ...],
         decoder_input[tf.newaxis, ...],
         decoder_hidden,
         decoder_cell,
         decoder_output,
         encoder_outputs,
         input_continuous[tf.newaxis, ...],
         combined_mask[tf.newaxis, ...],
         tf.constant(t, dtype=tf.int32)]
    )

    predicted_Y.append(predictions)
    loss += tf.keras.losses.sparse_categorical_crossentropy(targets[t:t+1], logits,
                                                           from_logits=True)

    decoder_input = input_continuous[t]
    last_visited_idx = targets[t]
    visited_mask = visited_mask + tf.one_hot(last_visited_idx, depth=seq_len)

avg_loss = loss / tf.reduce_sum(y_mask)

```

```

gradients = tape.gradient(avg_loss, encoder_model.trainable_variables +
                           decoder_model.trainable_variables)
optimizer.apply_gradients(zip(gradients, encoder_model.trainable_variables +
                              decoder_model.trainable_variables))

predicted_Y = tf.stack(predicted_Y)
predicted_Y = tf.squeeze(predicted_Y)

return avg_loss, predicted_Y

```

For function `create_x_mask` and `create_y_mask`, is to create masking for input X and Y. For input X, masking are done at both zones and features level, for padding values with '0'. On the other hand, for input Y, masking is done at sequence, marked as '-1'. Masking is to ensure exclusion padding values during model training, evaluation.

Function `create_x_mask`

```

def create_x_mask(X):
    _, num_zones, num_features = X.shape
    mask = np.zeros((num_zones))
    for i in range(num_zones):
        if np.all(X[0][i] == 0):
            mask[i] = 1
    return tf.convert_to_tensor(mask, dtype=tf.float32)

```

Function `create_y_mask`

```

def create_y_mask(Y):
    return tf.cast(Y != -1, tf.float32)

```



## CHAPTER 5

For function `combine_masks`, it is used to create an updated valid zone mask (valid and unvisited zone) at every timestep. The function combine `x_mask` and `visited_mask` together.

Function `combine_masks`

```
def combine_masks(x_mask, visited_mask):
    combined_mask = tf.maximum(x_mask, visited_mask)
    return combined_mask
```

Function `transpose_target_Y` is to rearrange the target sequence `target_Y` such that each zone's position is mapped to its corresponding index in `X`. Essentially, it "transposes" the target sequence so that the indices of the zones in the sequence align with the order of features in `X`.

Function `transpose_target_Y`

```
def transpose_target_Y(target_Y):
    transposed_Y = np.zeros_like(target_Y)
    valid_positions = target_Y[target_Y != -1]
    for idx, zone in enumerate(valid_positions):
        transposed_Y[zone] = idx
    return transposed_Y[target_Y != -1]
```

## CHAPTER 5

Function `shuffle_data` are used to shuffle the sequence of input training data, to ensure that the model are not learning the input sequential pattern.

Function `shuffle_data`

```
def shuffle_data(X, Y):  
    """Shuffle the X and Y data while maintaining correspondence."""  
    indices = np.arange(len(X))  
    shuffled_indices = shuffle(indices)  
    return [X[i] for i in shuffled_indices], [Y[i] for i in shuffled_indices]
```

Function `evaluate_predictions` is to evaluate the average accuracy of the predicted route compared to the actual sequence (target).

Function `evaluate_predictions`

```
def evaluate_predictions(predicted_Y, targets):  
    accuracy = 0  
    assert len(predicted_Y) == len(targets), "Predicted and target sequences must have  
        the same length."  
    correct_predictions = tf.reduce_sum(tf.cast(tf.equal(predicted_Y, targets),  
        dtype=tf.int32))  
    total_elements = tf.size(targets)  
    accuracy = correct_predictions / tf.cast(total_elements, dtype=tf.int32)  
    return accuracy
```

## 5.6 Model Evaluation

For Model Evaluation, custom evaluation loops are defined, align with training loops. The custom model evaluation function return information includes average loss, average accuracy, first four zone accuracy, average disparity score, standard deviations of disparity score, list of disparity scores, with the list of all predictions, all targets, all X, uses for visualization later on.

Function evaluate model

```
def evaluate_model(encoder_model, decoder_model, X_data, Y_data):  
    total_loss = 0.0  
    total_accuracy = 0.0  
    total_disparity_score = 0.0  
    total_routes = len(X_data)  
    all_predictions = []  
    all_targets = []  
    all_X = []  
  
    good_predictions = []  
    good_targets = []  
    good_X = []  
  
    bad_predictions = []  
    bad_targets = []  
    bad_X = []  
  
    disparity_scores = []  
    first_correct = 0  
  
    X_data_shuffled, Y_data_shuffled = shuffle_data(X_data, Y_data)  
  
    for route_idx in range(total_routes):
```

```

inputs = X_data_shuffled[route_idx]
targets = Y_data_shuffled[route_idx]
targets = transpose_target_Y(targets)

inputs = np.array(inputs)
targets = np.array(targets)
# Prepare inputs for the encoder
input_feature_1 = inputs[:, 0]
input_feature_2 = inputs[:, 1]
input_feature_3 = inputs[:, 2]
input_feature_4 = inputs[:, 3]
input_continuous = inputs[:, 4:]

x_mask = create_x_mask(inputs[tf.newaxis, ...])
y_mask = create_y_mask(targets[tf.newaxis, ...])

# Run the encoder
encoder_outputs, encoder_hidden, _ = encoder_model(
    [input_feature_1[tf.newaxis, ...],
     input_feature_2[tf.newaxis, ...],
     input_feature_3[tf.newaxis, ...],
     input_feature_4[tf.newaxis, ...],
     input_continuous[tf.newaxis, ...],
     x_mask[tf.newaxis, ...]], training=False
)

# Initialize decoder state
decoder_input = input_continuous[0]
decoder_hidden = encoder_hidden
last_visited_idx = tf.constant(0, dtype=tf.int32)
decoder_cell = tf.zeros_like(encoder_hidden)
decoder_output = tf.zeros_like(encoder_hidden)

```

```

visited_mask = tf.zeros_like(x_mask[0])
check_mask = tf.ones_like(x_mask[0])

seq_len = tf.shape(inputs)[0]
predicted_Y = []
route_loss = 0

for t in range(seq_len):
    combined_mask = combine_masks(x_mask, visited_mask)

    if tf.reduce_all(tf.equal(combined_mask, check_mask)):
        break

    decoder_output, decoder_hidden, decoder_cell, logits, predictions =
    decoder_model(
        [last_visited_idx[tf.newaxis, ...],
        decoder_input[tf.newaxis, ...],
        decoder_hidden,
        decoder_cell,
        decoder_output,
        encoder_outputs,
        input_continuous[tf.newaxis, ...],
        combined_mask[tf.newaxis, ...],
        tf.constant(t, dtype=tf.int32)[tf.newaxis, ...]]
    )

    predicted_Y.append(predictions)
    route_loss += tf.keras.losses.sparse_categorical_crossentropy(targets[t:t+1],
                                                                    logits, from_logits=True)

    pred_idx = predictions[-1].numpy()

    decoder_input = input_continuous[pred_idx]

```

```

last_visited_idx = pred_idx
visited_mask = visited_mask + tf.one_hot(last_visited_idx, depth=seq_len)

predicted_Y = tf.stack(predicted_Y)
predicted_Y = tf.squeeze(predicted_Y)

route_accuracy = evaluate_predictions(predicted_Y[tf.newaxis, ...],
                                     targets[tf.newaxis, ...])

if predicted_Y[1] == targets[1]:
    first_correct += 1

time_matrix = np.zeros((len(targets), len(targets)))
for i in range(len(targets)):
    for j in range(len(targets)):
        time_matrix[i, j] = input_continuous[i][j+4]

disparity_score = calculate_disparity_score(targets, predicted_Y.numpy(),
                                           time_matrix)

total_loss += route_loss / tf.reduce_sum(y_mask)
total_accuracy += route_accuracy
total_disparity_score += disparity_score
disparity_scores.append(disparity_score)

# Store predictions and targets
all_predictions.append(predicted_Y.numpy())
all_targets.append(targets)
all_X.append(inputs)

if route_accuracy > 0.80:
    good_predictions.append(predicted_Y.numpy())
    good_targets.append(targets)

```

```

        good_X.append(inputs)
    elif route_accuracy < 0.5:
        bad_predictions.append(predicted_Y.numpy())
        bad_targets.append(targets)
        bad_X.append(inputs)

    progress = ((route_idx + 1) / total_routes) * 100
    print(f'\rProgress: {progress:.2f}% - Route {route_idx + 1}/{total_routes}, '
          f'Loss: {(route_loss / tf.reduce_sum(y_mask)).numpy()}, '
          f'Accuracy: {route_accuracy:.4f}, '
          f'Disparity Score: {disparity_score:.4f}', end=")

    avg_loss = total_loss / total_routes
    avg_accuracy = total_accuracy / total_routes
    avg_disparity_score = total_disparity_score / total_routes
    disparity_score_std = np.std(disparity_scores)
    first_zone_accuracy = first_correct / total_routes

    print(f'\nEvaluation complete. Average Loss: {avg_loss:}, '
          f'Average Accuracy: {avg_accuracy:}, '
          f'Average Disparity Score: {avg_disparity_score:}, '
          f'Disparity Score Std: {disparity_score_std:}')

    return (avg_loss, avg_accuracy, first_zone_accuracy,
            avg_disparity_score, disparity_score_std, disparity_scores,
            all_predictions, all_targets, all_X,
            good_predictions, good_targets, good_X,
            bad_predictions, bad_targets, bad_X)

```

## CHAPTER 5

For function `calculate_disparity_score` is to calculate the disparity score given target sequence and predicted sequence. The mathematical formula is defined in **Chapter 6.2.1**.

Function `calculate_disparity_score`

```
def calculate_disparity_score(actual_transposed, predicted, time_matrix):
    sd = calculate_sequence_deviation(actual_transposed, predicted)
    erp_norm = calculate_erp_norm(actual_transposed, predicted, time_matrix)
    erp_e = calculate_erp_e(actual_transposed, predicted)
    if erp_e == 0: # Perfect prediction
        return 0
    return (sd * erp_norm) / erp_e
```

Function `calculate_sequence_deviation`

```
def calculate_sequence_deviation(actual, predicted):
    n = len(actual)
    c = {stop: idx for idx, stop in enumerate(actual)}
    sd = 0
    for i in range(1, n):
        sd += abs(c[predicted[i]] - c[predicted[i-1]]) - 1
    return (2 / (n * (n - 1))) * sd
```

Function `calculate_erp_norm`

```
def calculate_erp_norm(actual, predicted, time_matrix):
    if len(actual) == 1 or len(predicted) == 1:
        return 0
    first_actual, first_predicted = actual[0], predicted[0]
    time_norm = time_matrix[first_actual, first_predicted] / np.sum(
        time_matrix[first_actual, :])
    return calculate_erp_norm(actual[1:], predicted[1:], time_matrix) + time_norm
```



## CHAPTER 5

### Function calculate\_erp\_e

```
def calculate_erp_e(actual, predicted):
    m, n = len(actual), len(predicted)
    dp = [[0] * (n + 1) for _ in range(m + 1)]
    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if actual[i-1] == predicted[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])
    return dp[m][n]
```

Function `k_fold_cross_validation` are custom k fold cross validation which call custom training loops and evaluation functions.

### Function `k_fold_cross_validation`

```
def k_fold_cross_validation(X, Y, k=5, hidden_size=128, vocab_size=48,
    epochs=10, learning_rate=0.0001,
    patience=3, min_delta=0.001):
    kf = KFold(n_splits=k, shuffle=True, random_state=42)
    fold_results = []

    sequence_indices = np.arange(len(X))

    for fold, (train_index, val_index) in enumerate(kf.split(sequence_indices), 1):
        print(f"\nFold {fold}/{k}")
```

```

# Split the data
X_train, X_val = [X[i] for i in train_index], [X[i] for i in val_index]
Y_train, Y_val = [Y[i] for i in train_index], [Y[i] for i in val_index]

# Build and compile the model
encoder = build_encoder(hidden_size)
decoder = build_decoder(hidden_size, vocab_size)
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

best_val_loss = float('inf')
best_encoder = None
best_decoder = None
patience_counter = 0

for epoch in range(epochs):
    print(f"Epoch {epoch + 1}/{epochs}")

    train_model(encoder, decoder, X_train, Y_train, optimizer, 1)

# Evaluate on validation set
val_results = evaluate_model(encoder, decoder, X_val, Y_val)
val_loss, val_accuracy, val_first_zone_accuracy = val_results[0],
                                                    val_results[1], val_results[2]
val_disparity, val_disparity_std = val_results[3], val_results[4]

print(f"Validation Set - Loss: {val_loss}, Accuracy: {val_accuracy}, "
      f"First zone Accuracy: {val_first_zone_accuracy}, "
      f"Disparity Score: {val_disparity}, Disparity Std: {val_disparity_std}")

# Check if this is the best model so far
if val_loss < best_val_loss - min_delta:
    best_val_loss = val_loss
    best_encoder = tf.keras.models.clone_model(encoder)

```

```

        best_encoder.set_weights(encoder.get_weights())
        best_decoder = tf.keras.models.clone_model(decoder)
        best_decoder.set_weights(decoder.get_weights())
        patience_counter = 0
    else:
        patience_counter += 1

    # Check if we should stop training
    if patience_counter >= patience:
        print(f"Early stopping triggered at epoch {epoch + 1}")
        break

final_val_results = evaluate_model(best_encoder, best_decoder, X_val, Y_val)

(val_loss, val_accuracy, val_first_zone_accuracy,
 val_disparity, val_disparity_std, val_disparity_scores,
 val_predictions, val_targets, val_X,
 good_val_predictions, good_val_targets, good_val_X,
 bad_val_predictions, bad_val_targets, bad_val_X) = final_val_results

print(f"Final Validation Set - Loss: {val_loss}, Accuracy: {val_accuracy}, "
      f"Disparity Score: {val_disparity}, Disparity Std: {val_disparity_std}")

fold_results.append({
    'loss': val_loss,
    'accuracy': val_accuracy,
    'first_zone_accuracy': val_first_zone_accuracy,
    'disparity': val_disparity,
    'disparity_std': val_disparity_std
})

# Calculate average metrics across all folds
avg_loss = np.mean([fold['loss'] for fold in fold_results])

```

```
avg_accuracy = np.mean([fold['accuracy'] for fold in fold_results])
avg_disparity = np.mean([fold['disparity'] for fold in fold_results])
avg_disparity_std = np.mean([fold['disparity_std'] for fold in fold_results])

print("\nAverage results across all folds:")
print(f"Loss: {avg_loss}")
print(f"Accuracy: {avg_accuracy}")
print(f"Disparity Score: {avg_disparity}")
print(f"Disparity Score Std: {avg_disparity_std}")

return fold_results
```

## 5.7 Result Visualization

Finally, predicted route and target route are used to visualize how the predicted route compared to target route.

### *Visualization with Plot Graph*

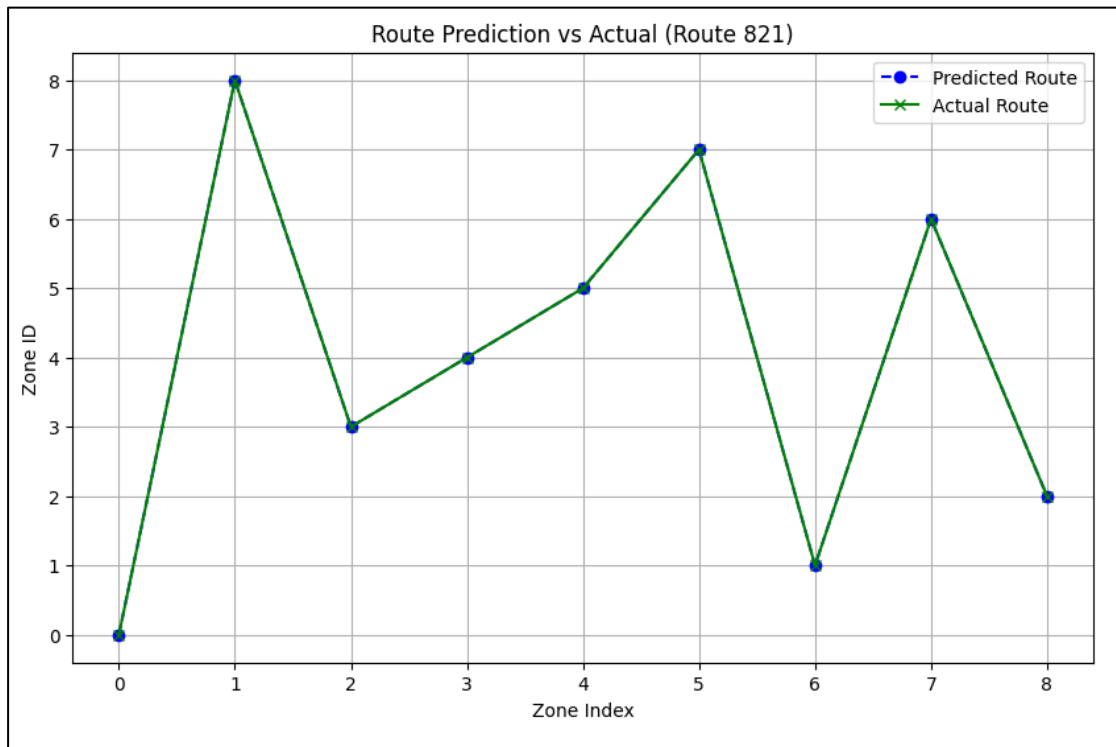
Function `plot_route_comparison`

```
def plot_route_comparison(route_idx, predicted_sequences, actual_sequences):
    predicted_route = predicted_sequences[route_idx]
    actual_route = actual_sequences[route_idx]

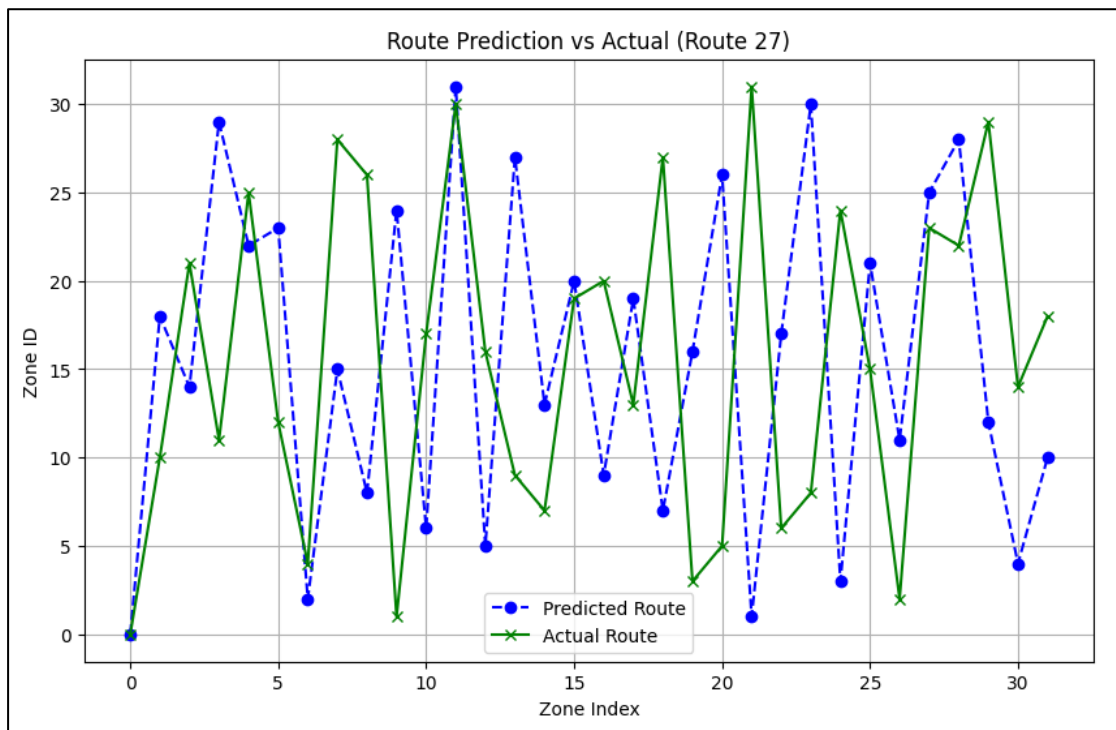
    # Filter out padded zones (-1)
    predicted_route = [zone for zone in predicted_route if zone != -1]
    actual_route = [zone for zone in actual_route if zone != -1]

    # Plot the sequences
    plt.figure(figsize=(10, 6))
    plt.plot(predicted_route, label="Predicted Route", marker='o', linestyle='--',
             color='blue')
    plt.plot(actual_route, label="Actual Route", marker='x', linestyle='-',
             color='green')
    plt.title(f"Route Prediction vs Actual (Route {route_idx})")
    plt.xlabel("Zone Index")
    plt.ylabel("Zone ID")
    plt.legend()
    plt.grid(True)
    plt.show()
```

Example:



**Figure 5.2.1** Plot Graph for predicted vs actual at Route 821



**Figure 5.2.2** Plot Graph for predicted vs actual at Route 27

*Route Visualization with Leaflet*Function `create_map_with_sequence`

```
def create_map_with_sequence(X, Y):
    stops_sequence = [X[idx] for idx in Y]
    avg_lat = sum([stop[6] for stop in stops_sequence]) / len(stops_sequence)
    avg_lng = sum([stop[7] for stop in stops_sequence]) / len(stops_sequence)
    folium_map = folium.Map(location=[avg_lat, avg_lng], zoom_start=10)
    stop_coordinates = []
    for stop_idx, stop in enumerate(stops_sequence):
        lat = stop[6]
        lng = stop[7]

        # Construct the stop name
        feature_1 = chr(int(stop[0]) + 64)
        feature_2 = chr(int(stop[1]) + 64)
        feature_3 = str(int(stop[2]))
        feature_4 = str(int(stop[3]))
        name = f"{feature_1}-{feature_3}.{feature_4}{feature_2}"

        color = generate_random_color()

        folium.map.Marker(
            [lat, lng],
            icon=folium.DivIcon(
                icon_size=(50, 60),
                html=f"""
                <div style="
                    background-color: {color};
                    border-radius: 10px;
                    padding: 2px;
                    text-align: center;
```

```

        font-size: 8pt;
        color: white;
        width: 50px;">
        <b>{stop_idx}</b><br>{name}
    </div>""
    )
).add_to(folium_map)

stop_coordinates.append((lat, lng))

for i in range(len(stop_coordinates) - 1):
    start_lat, start_lng = stop_coordinates[i]
    end_lat, end_lng = stop_coordinates[i + 1]

    route = get_osrm_route(start_lat, start_lng, end_lat, end_lng)

    folium.PolyLine(locations=route, color='blue', weight=2.5,
                    opacity=0.7).add_to(folium_map)

start_lat, start_lng = stop_coordinates[-1]
end_lat, end_lng = stop_coordinates[0]
return_route = get_osrm_route(start_lat, start_lng, end_lat, end_lng)

folium.PolyLine(locations=return_route, color='blue', weight=2.5,
                opacity=0.7).add_to(folium_map)

return folium_map

```



## CHAPTER 5

### Function get\_osrm\_route

```
def get_osrm_route(start_lat, start_lng, end_lat, end_lng):
    osrm_url = f"http://router.project-osrm.org/route/v1/driving/
                {start_lng},{start_lat};{end_lng},{end_lat}
                ?overview=full&geometries=geojson"
    response = requests.get(osrm_url)
    if response.status_code == 200:
        data = response.json()
        route = data['routes'][0]['geometry']['coordinates']
        route = [(lat, lng) for lng, lat in route] # Convert to [(lat, lng)] format
        return route
    else:
        print(f"Error fetching route: {response.status_code}")
        return []
```

### Function generate\_random\_color

```
def generate_random_color():
    return "#{:02x}{:02x}{:02x}".format(random.randint(0, 255),
                                       random.randint(0, 255), random.randint(0, 255))
```

CHAPTER 5

Example:

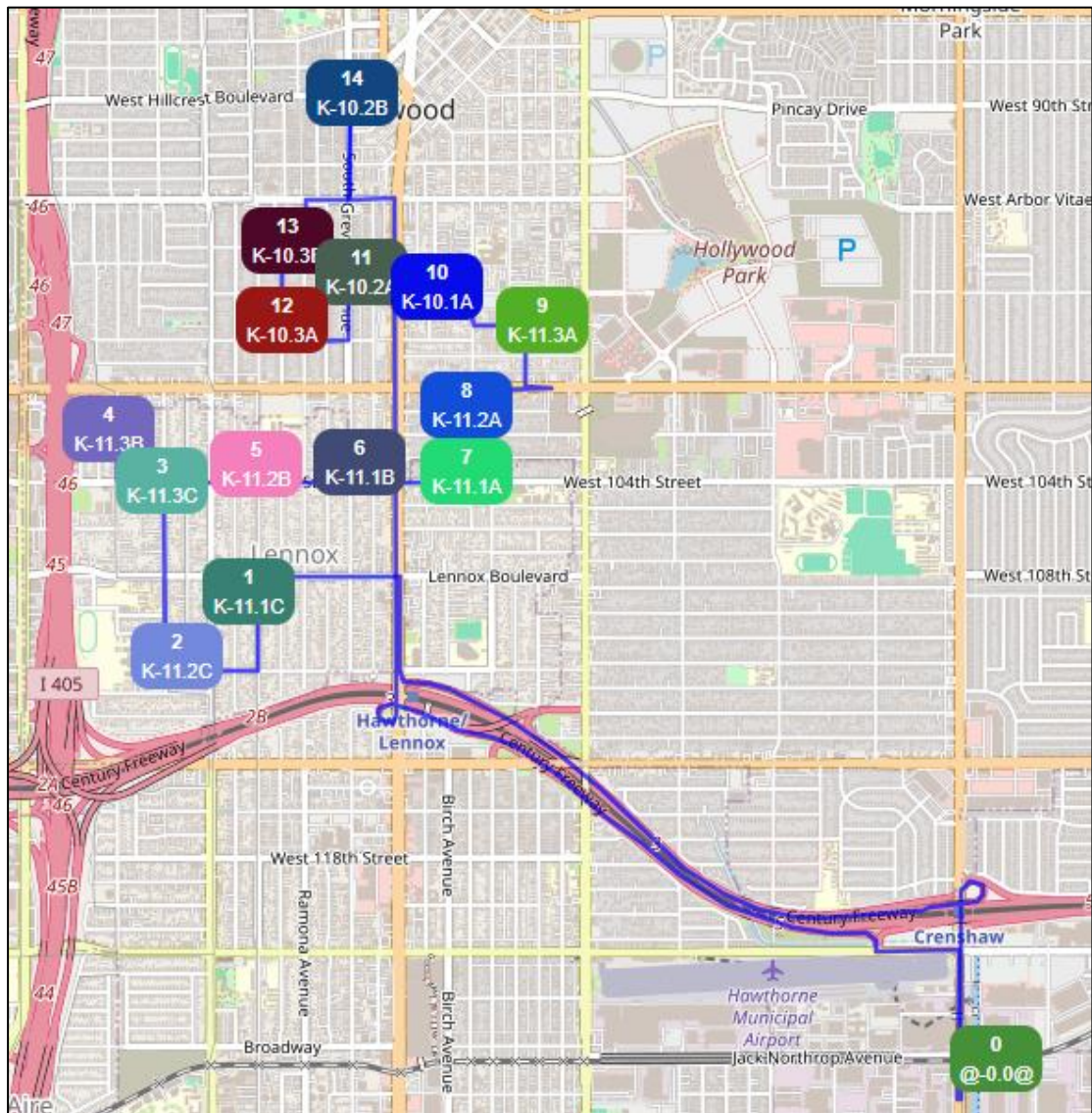
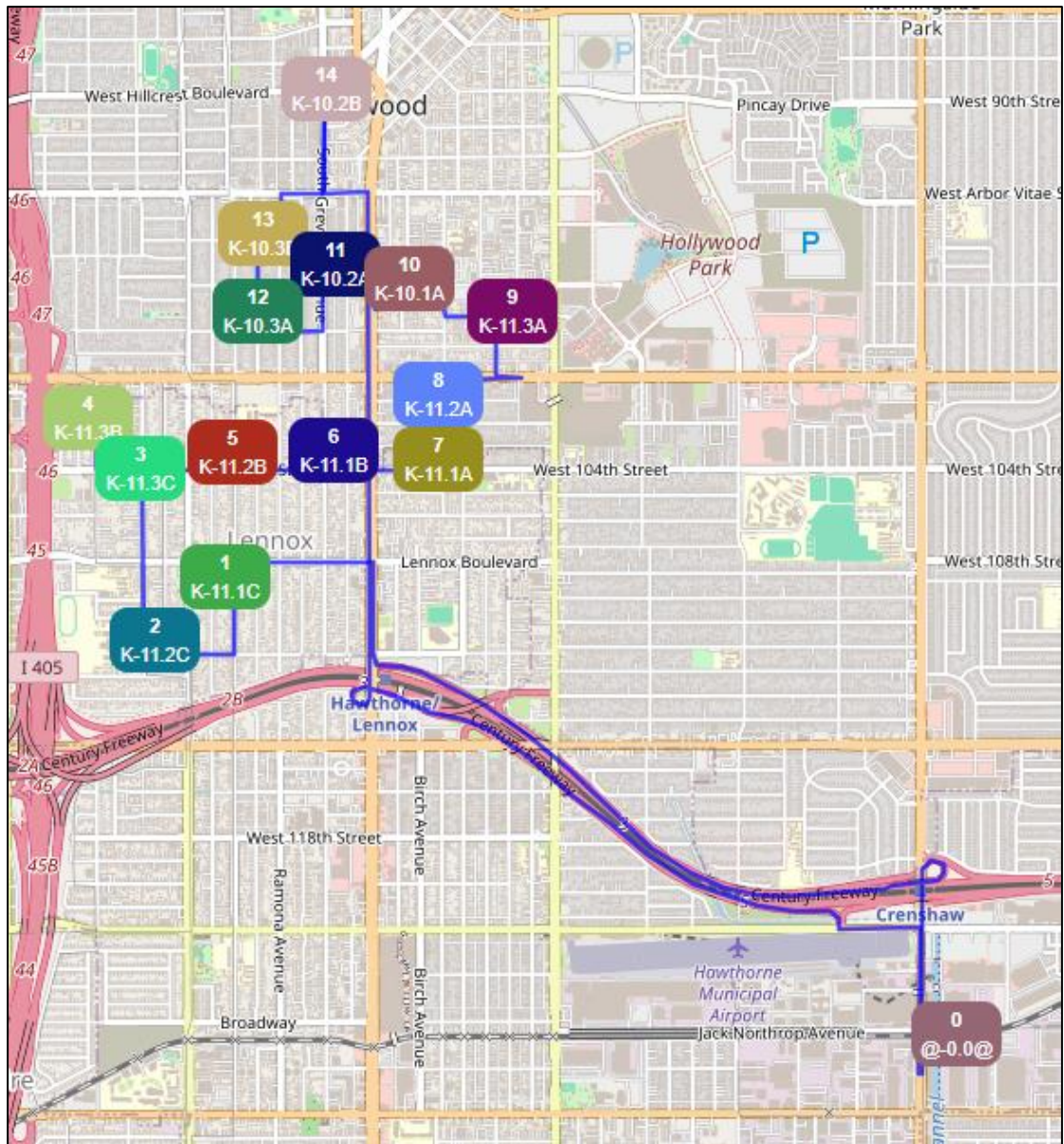
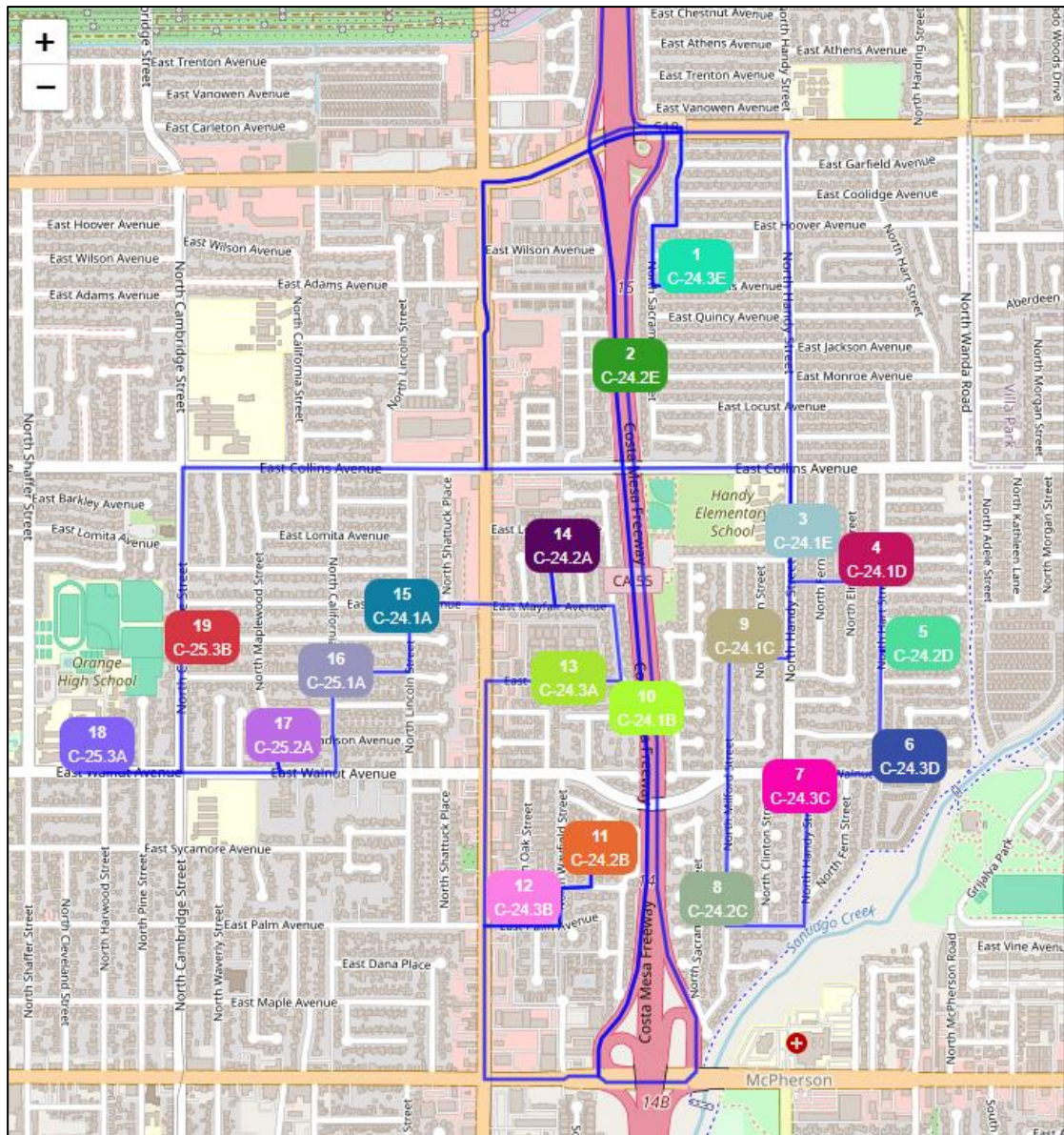


Figure 5.2.3 Actual Route0 shown using Leaflet



**Figure 5.2.4** Predicted Route0 shown using Leaflet



**Figure 5.2.5** Actual Route20 shown using Leaflet

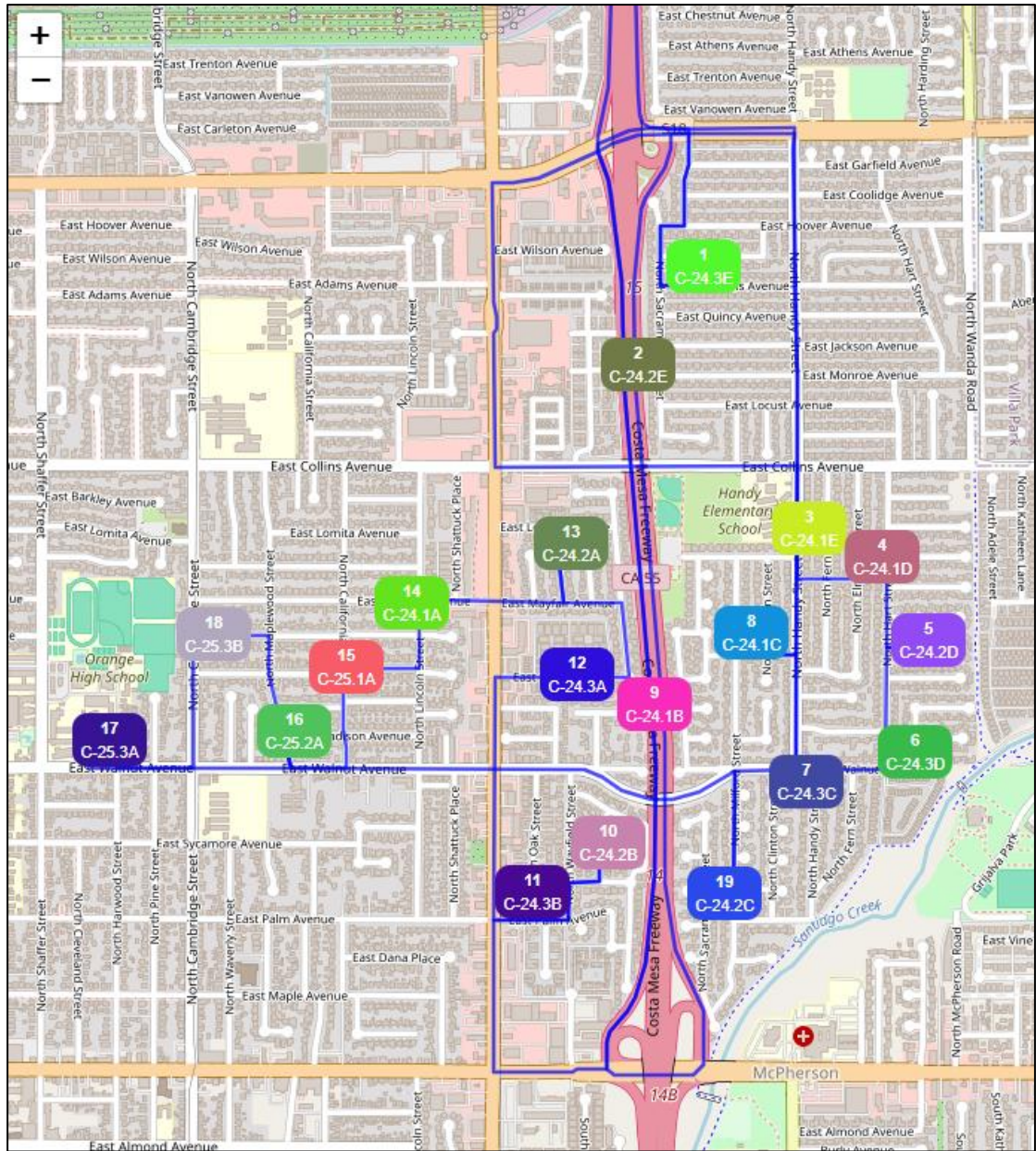


Figure 5.2.6 Predicted Route20 shown using Leaflet

## 5.8 Implementation Issues and Challenges

### *Cloud-Based Computing Resources*

The project is implemented on Google Colab, a cloud-based Jupyter Notebook. Due to the nature of cloud-based computing, computing process are strongly reliant on internet connections. Poor Internet connection or internet outbreaks hinders project progress. Moreover, Google Colab also comes with idle time constraints given a period of time. When running time consuming codes, such as model training, and cross validation, Google Colab may disconnect due to reaching limit of idle time, causing session to be halted. Therefore, it might be potential bottleneck for the project.

## System Evaluation and Discussion

### 6.1 Driver Behaviours in Last-Mile Delivery

In this project, data analysis was conducted using the dataset from the Amazon Last Mile Routing Research Challenge [7]. The dataset consists of over 6,112 actual last-mile delivery routes, providing valuable insights into how drivers navigate their routes, particularly in zone-based deliveries. The analysis reveals that within delivery zones, drivers show a strong preference for completing all stops in a single zone before moving on to the next. This sequential pattern suggests that driver behavior is heavily influenced by the structure of the zones rather than by individual package constraints, such as preferred delivery time windows. For instance, in zone ‘C-3.2B,’ drivers follow a fixed sequence of stops [1, 2, 3, ..., 9], while zone ‘C-3.1B’ covers stops [10, 11, 12, ..., 18] (refer to **Figure 5.1.7**). This consistent zone-focused approach implies a systematic method for reducing route deviations by organizing deliveries within specific geographic clusters.

Further analysis into how routes are planned uncovered a hierarchical structure in the zone IDs used in delivery routes. Each zone ID can be broken down into four components: the super-super cluster, the super cluster, the cluster, and the individual zone. For example, in the zone ID ‘D-18.2J,’ the letter ‘D’ represents the super-super cluster, while ‘D-18’ forms the super cluster, and ‘J’ represents the cluster. The final decimal value ‘.2’ identifies the specific delivery zone. This hierarchical structure plays a crucial role in determining the order in which drivers approach their deliveries. The data indicates that the zone IDs tend to follow either an ascending or descending order, pointing to a systematic clustering approach that drivers likely follow. This structured pattern helps minimize deviations by ensuring drivers adhere to an organized route across multiple clusters and zones.

Another key finding from the analysis is that over 92.27% of the delivery packages are time-window insensitive. Out of the 1,457,175 packages, 1,343,182

lacked specified delivery time windows. This overwhelming percentage suggests that time-sensitive deliveries are not a major factor influencing most routes. Consequently, the impact of time-sensitive delivery constraints on driver decisions is minimal, allowing drivers to focus on other factors such as route efficiency, traffic patterns, and overall distance traveled.

In conclusion, the findings emphasize the minimal role that time-sensitive deliveries play in influencing driver behavior. Instead, driver behavior is primarily driven by zone-based delivery patterns, where stops within a given zone are completed sequentially before moving to the next. The hierarchical structure of zone IDs further supports a systematic approach to delivery route planning, indicating that geographic clustering of stops is the primary factor affecting route deviations. By understanding the factors that may cause drivers to deviate from pre-planned delivery routes and last-mile delivery behavior, we can better select features and preprocess data for model training. This, in turn, will lead to more accurate and reliable delivery predictions.



## 6.2 Model Performance

### 6.2.1 Performance metrics

#### *Disparity score metrics*

For this route prediction problem, the quality of predicted zone sequences is evaluated using ‘disparity score’, aligning with the Amazon Last Mile Routing Research Challenge, provided by [5], with the mathematical formula of:

$$R(A, B) = \frac{SD(A, B) \cdot ERP_{\text{norm}}(A, B)}{ERP_e(A, B)} \quad (1)$$

where (i) the disparity score for the actual sequence  $A$  and predicted sequence  $B$ , denoted by  $R(A, B)$ ; and

(ii) the sequence deviation between actual sequence  $A$  and predicted sequence  $B$ , denoted as  $SD(A, B)$  are expressed in following:

$$SD(A, B) = \frac{2}{n(n-1)} \sum_{i=2}^n (|c_{[B_i]} - c_{[B_{i-1}]}| - 1) \quad (18)$$

where (i) total number of zones found in a given route, denoted by  $n$ ;

(ii) the  $i$ th zone of sequence  $B$ , denoted by  $B_i$ ;

(iii) the index of zone  $B_i$  in the actual sequence  $A$ , denoted by  $c_{[B_i]}$ .

For every perfect predicted case, where sequence  $A$  is completely equals to sequence  $B$ ,  $SD(A, B)$  return value of 0.

Next, the recursive function, Edited Distance with Real Penalty (ERP), denoted by  $ERP_{\text{norm}}(A, B)$  are used to calculate the penalty score for every deviated zone, compared to the actual sequences. It can be expressed as:

$$ERP_{\text{norm}}(A, B) = ERP_{\text{norm}}(A_{2:|A|}, B_{2:|B|}) + \text{TIME}_{\text{norm}}(A_1, B_1) \quad (19)$$

where the normalized travel time between zone  $z_i$  and  $z_j$ , expressed in:

$$\text{TIME}_{\text{norm}}(z_i, z_j) = \frac{\text{TIME}(z_i, z_j)}{\sum_{j' \in \{1, \dots, n\}} \text{TIME}(z_i, z_{j'})} \quad (20)$$

$\text{ERPe}(A, B)$  provide the number of modify operations (i.e., insertions, substitutions, deletions) needed to convert sequence A to sequence B while computing the recursive function,  $\text{ERP}_{\text{norm}}(A, B)$  [5]. Given that the ratio of  $\frac{\text{ERP}_{\text{norm}}(A, B)}{\text{ERPe}(A, B)}$  provides the average  $\text{TIME}_{\text{norm}}(z_i, z_j)$  involved in every ERP modification operation. Note that a score of zero indicates a perfect prediction, hence the lower the metrics tells positive model performance.

The motivation for choosing the disparity score over traditional accuracy metrics as the primary evaluation metric lies in its ability to provide a more meaningful assessment of route prediction performance. While accuracy metrics focus solely on how well the predicted route matches the actual sequence, they do not account for the quality or efficiency of the predictions. In contrast, the disparity score evaluates the predicted route in terms of its deviation from optimality, penalizing errors based on the travel distance between the actual and predicted zones at each step. This approach offers a clearer understanding of "how suboptimal" a predicted route is, emphasizing the importance of minimizing travel distance and improving overall route efficiency.

#### *Prediction accuracy*

Moreover, along with the disparity score, we assess the prediction accuracy of the first four zones in each given route as every route contains at least four zones. Let the predicted sequence for the  $m$ th route be  $A^{(m)}$  and the actual sequence be  $B^{(m)}$ . The prediction accuracy of the  $i$ 'th zone is defined as:

$$\text{Prediction accuracy}_i = \frac{\sum_{m=1}^M \mathbb{1}_{\{A_i^{(m)} = B_i^{(m)}\}}}{M} \quad (2)$$

where  $M$  is the total number of testing samples, and  $\mathbb{1}_{\{A_i^{(m)}=B_i^{(m)}\}}$  is an indicator function that returns 1 when the condition is met and 0 otherwise.

### 6.2.2 Train Result Evaluation

After model training, the performance of both the Simple RNN E-D and LSTM E-D with Attention models is evaluated on the training data using two primary metrics: *Disparity Score* and *Prediction Accuracy* for the first four zones in the route. As explained the disparity score assesses how far the predicted route deviates from the optimal route, while the prediction accuracy measures the model's ability to correctly predict the sequence of the first four zones. The evaluation results are obtained and tabulated, shown in **Table 6.1**. As shown in **Table 6.1**, the LSTM E-D with Attention model outperformed the Simple RNN E-D with a significantly lower mean disparity score of 0.0091 compared to 0.0209. The smaller standard deviation (0.0059) in the LSTM E-D with Attention model's disparity score also suggests more consistent predictions across different routes.

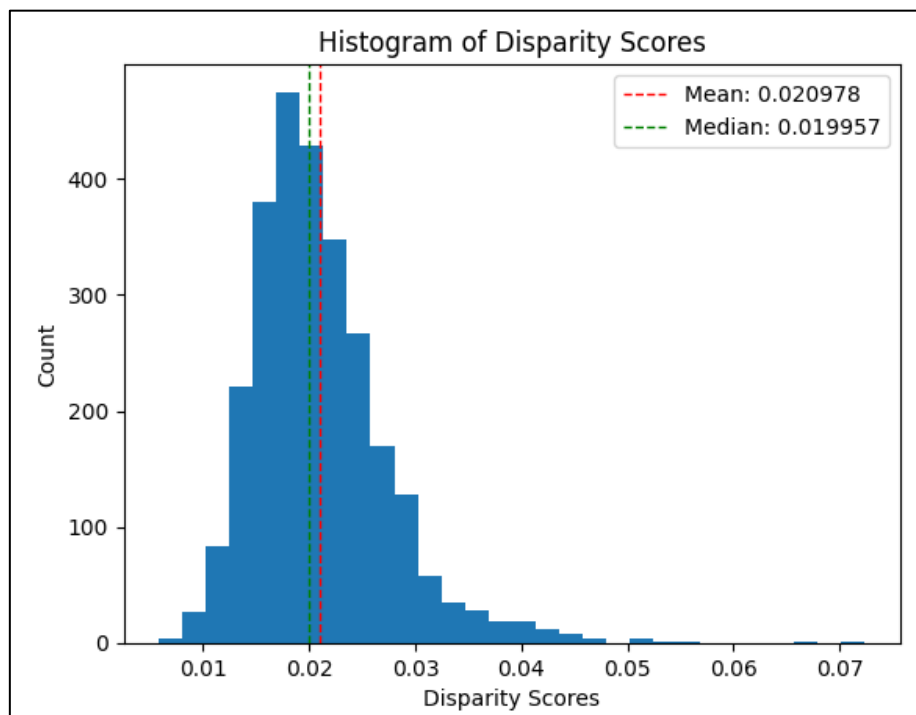
Moreover, according to **Table 6.1**, the LSTM E-D with Attention model demonstrated a higher prediction accuracy across all four zones, particularly achieving 0.190 accuracy for the first zone, but gradually decrease to 0.117 for the fourth zone. The Simple RNN model, on the other hand, had lower accuracy across the first four zones, with its best performance being 0.069 for the first zone and declining to 0.057 for the fourth zone. This demonstrates that the LSTM with Attention model is better at predicting the early part of the route. Based on the result obtained, it suggests that the LSTM E-D with Attention model offers both lower disparity scores and higher prediction accuracies, making it a better fit for this route prediction problem on the training data.

**Table 6.1** Model Performance on Training Data

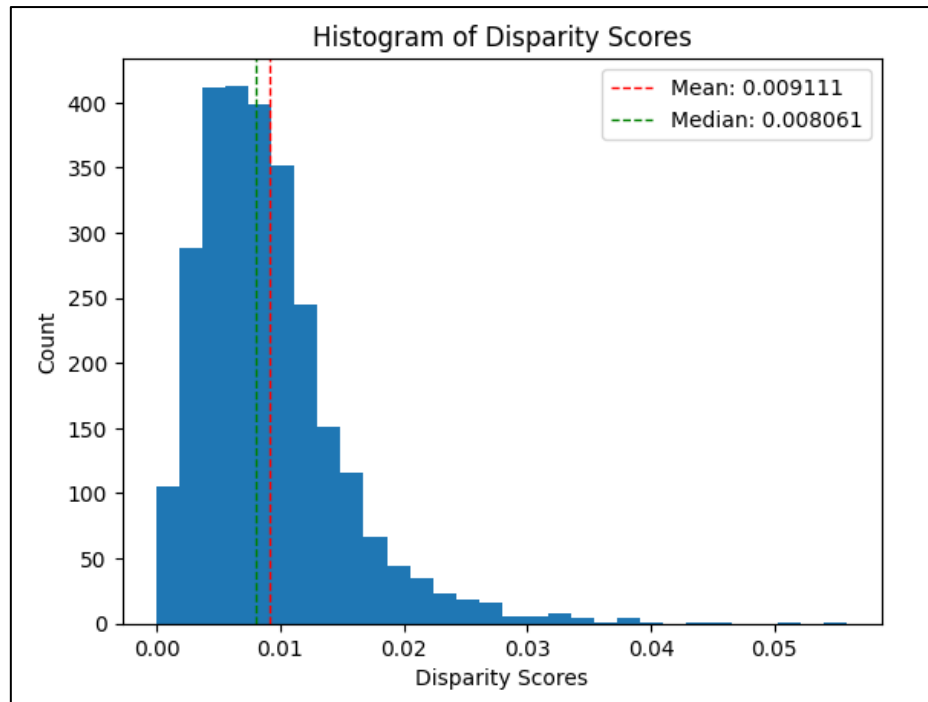
Model	Disparity Score		Prediction accuracy (zone)			
	Mean	Std. Dev	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
Simple RNN E-D	0.0209	0.0063	0.069	0.055	0.065	0.057
LSTM E-D with Attention	0.0091	0.0059	0.190	0.153	0.136	0.117

**Figure 6.1** and **Figure 6.2** shows the disparity score distribution of Simple RNN E-D Model and LSTM E-D with Attention Model, respectively. Based on the disparity score distribution for LSTM E-D with Attention Model histogram (shown in **Figure 6.2**), it shows that most disparity scores are concentrated between 0.005 and 0.015, with the mean disparity score of 0.0091 and a median of 0.0081. This indicates that the LSTM E-D with Attention model produces relatively low disparity scores for most of the routes, implying that the predicted routes are close to the actual routes. The tight distribution around the mean shows consistency in the model's performance across different routes.

On the other hand, the disparity score distribution for Simple RNN E-D Model histogram (shown in **Figure 6.1**) displays a wider spread of disparity scores, with most routes concentrated between 0.01 and 0.03, with the mean disparity score being 0.021 and a median of 0.020. As comparison, the larger spread in the disparity scores, along with a higher mean and median compared to the LSTM E-D model, it indicates that the Simple RNN E-D model is less consistent in its route prediction and more prone to produces routes that deviate significantly from the optimal route. In addition, the tail of the distribution extends further, with several routes showing disparity scores greater than 0.03, which highlights more frequent suboptimal predictions.



**Figure 6.1** Disparity Score Distribution of Simple RNN E-D



**Figure 6.2** Disparity Score Distribution of LSTM E-D with Attention

### 6.2.3 Cross-Validation Results

**Table 6.2** and **Table 6.3** shows the cross-validation results for both the Simple RNN E-D and LSTM E-D with Attention, respectively. In this evaluation, we used 5-fold cross-validation, where the training data was split into 5 subsets (folds), and each fold was used as the validation set while the remaining folds were used for training. The mean and standard deviation of the disparity score across all folds are reported.

Based on **Table 6.2**, it suggests that the Simple RNN E-D model performs fairly consistent across all the 5 folds. The mean disparity score across multiple folds ranges from 0.023017 to 0.023764, with the standard deviation around 0.006 across all folds. The results from the cross-validation confirm the earlier training results, with consistent disparity scores across all folds. The mean disparity score is around 0.023, indicating a moderate deviation from the actual route.

**Table 6.2** Simple RNN E-D Cross Validation Results

Model: Simple RNN	Disparity Score	
	Mean	Std. Dev
Fold 1	0.023460	0.006667
Fold 2	0.023764	0.006907
Fold 3	0.023075	0.006357
Fold 4	0.023017	0.006159
Fold 5	0.023648	0.006748

On the other hand, the LSTM E-D with Attention model shows stronger performance across the 5 folds, with lower disparity scores compared to the Simple RNN, shown in **Table 6.3**. However, at Fold 3, the model produces a higher disparity score (0.032001) than the others, which could be due to outliers or more difficult routes in that fold. Despite the anomaly in Fold 3, the LSTM E-D with Attention model outperforms the Simple RNN E-D overall, with an average disparity score around 0.009 across most folds. This supports the conclusion that the LSTM model produces more accurate and efficient route predictions, particularly in comparison to the Simple RNN.

**Table 6.3** LSTM E-D with Attention Cross Validation Results

Model: LSTM with Attention	Disparity Score	
	Mean	Std. Dev
Fold 1	0.009096	0.005643
Fold 2	0.009280	0.005772
Fold 3	0.032001	0.008083
Fold 4	0.009828	0.005921
Fold 5	0.009233	0.005630

In short, both models show consistency in performance across different folds, with the LSTM E-D with Attention model having a significant advantage over the Simple RNN E-D in terms of lower disparity scores.

### 6.2.4 Test Result Evaluation

Finally, both models (i.e, Simple RNN E-D and LSTM E-D with Attention) were evaluated on a test set consisting of 906 unseen routes. This evaluation aims to assess the model's generalization capability by measuring its performance on completely new data that was not part of the training process. The evaluation results are obtained and tabulated, shown in **Table 6.4**. As shown in **Table 6.4**, Simple RNN E-D model achieved a mean disparity score of 0.0220 with a standard deviation of 0.0070. This suggests that while the model can make predictions, the deviation from the optimal route is moderate. LSTM E-D with Attention model, on the other hand, performed significantly better, with a mean disparity score of 0.0091 and a standard deviation of 0.0061, indicating that the LSTM E-D with Attention model's predictions on the unseen test data are much closer to the optimal route compared to the Simple RNN E-D model.

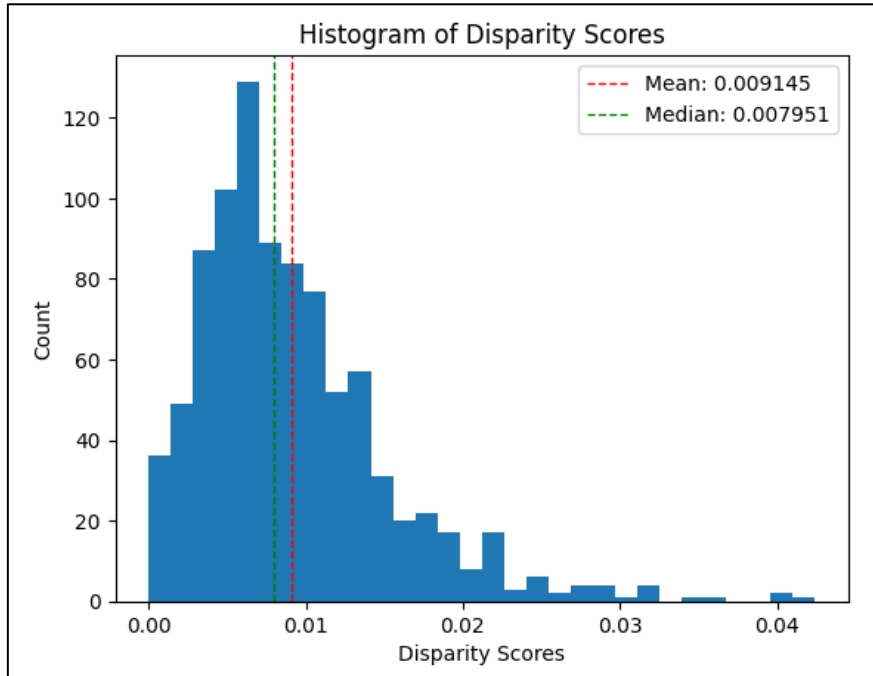
In terms of prediction accuracy, the Simple RNN E-D model showed a prediction accuracy of only 0.066 for the first zone, and steadily declined to 0.050 by the fourth zone. The decline in accuracy highlights the model's difficulty in making accurate predictions for the later parts of the route. As for LSTM E-D with Attention model, the model demonstrated a much stronger performance, with prediction accuracy starting at 0.194 for the first zone and decreasing more gradually to 0.128 by the fourth zone.

**Table 6.4** Model Performance on Test Data

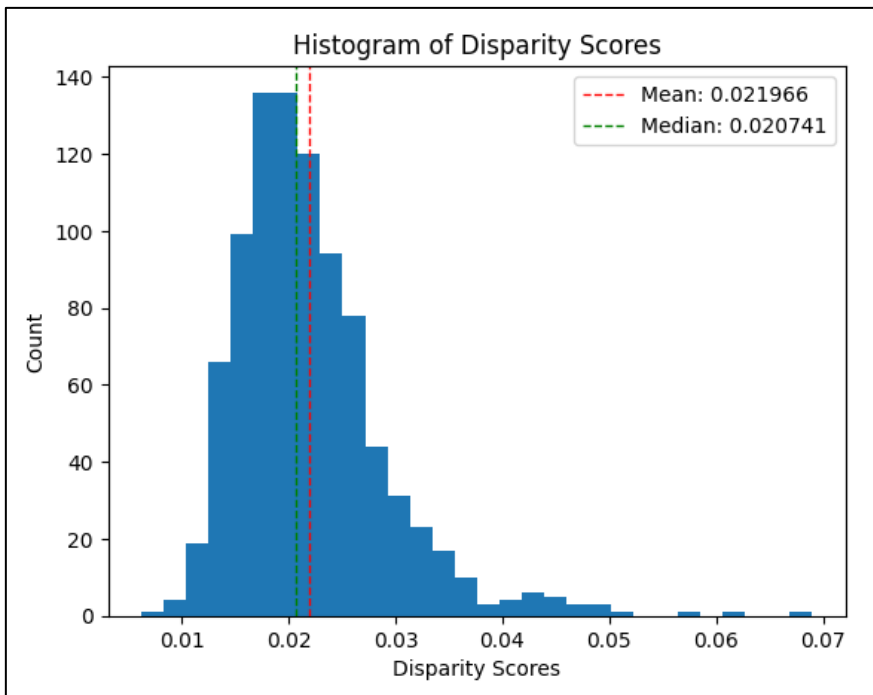
Model	Disparity Score		Prediction accuracy (zone)			
	Mean	Std. Dev	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
Simple RNN	0.0220	0.0070	0.066	0.057	0.061	0.050
LSTM with Attention	0.0091	0.0061	0.194	0.152	0.151	0.128

**Figure 6.3** and **Figure 6.4** shows the disparity score distribution of Simple RNN E-D Model and LSTM E-D with Attention Model, respectively. Based on the disparity score distribution for LSTM E-D with Attention Model histogram (shown in **Figure 6.4**), most disparity scores are concentrated between 0.005 and 0.015, with a mean disparity score of 0.0091 and a median of 0.0079. It indicates a relatively symmetrical

distribution with a slight rightward skew. Most of the routes exhibit low disparity, with very few routes deviating significantly from the optimal route. This distribution further demonstrates the LSTM model's efficiency, making predictions close to the actual route and producing only minor deviations.



**Figure 6.3** Disparity Score Distribution of Simple RNN E-D



**Figure 6.4** Disparity Score Distribution of LSTM E-D with Attention



### 6.2.5 Benchmarking

In the benchmark comparison, the LSTM E-D with Attention model shows respectable, though not competitive, results when compared to other state-of-the-art models. The prediction accuracy for the LSTM E-D with Attention model starts at  $0.194$  for the first zone and decreases to  $0.128$  for the fourth zone. In contrast, the model proposed by [5] using self-proposed Algorithm achieves a notably higher performance, with a consistent prediction accuracy starting at  $0.320$  for the first zone and  $0.314$  at the fourth zone.

It is important to note that the disparity scores for the LSTM E-D with Attention and Simple RNN E-D models in this project are evaluated at the zone level, while the benchmark models' disparity scores are evaluated at the stop level. This difference in evaluation granularity indicates that our models focus on optimizing predictions across broader zones, whereas the benchmark models assess performance on a more detailed, stop-by-stop basis. As a result, direct comparison of disparity scores between the models is excluded to avoid misleading conclusions due to this difference in evaluation criteria.

**Table 6.5** Model Performance with Benchmark model

Model	Disparity Score		Prediction accuracy (zone)			
	Mean	Std. Dev	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>
[5] Tour TSP	0.044	0.0289	0.207	0.185	0.163	0.168
[5] Open-tour TSP	0.043	0.0302	0.270	0.244	0.227	0.232
[12]	0.0198	N/A	N/A	N/A	N/A	N/A
[5] using Greedy Algorithm	0.0417	0.0306	0.241	0.231	0.224	0.221
[5] using Algorithm proposed	0.0369	0.0301	0.320	0.310	0.303	0.314
Simple RNN	0.0220	0.0070	0.066	0.057	0.061	0.050
LSTM with Attention	0.0091	0.0061	0.194	0.152	0.151	0.128

### 6.3 Objective Evaluation

Through extensive data analysis, we discovered that time-sensitive deliveries have minimal influence on driver behavior. Instead, the primary factor influencing delivery routes is zone-based delivery patterns. Drivers tend to complete all stops within a designated delivery zone sequentially before moving to the next, highlighting the geographic clustering of stops as a possible factor. The hierarchical structure of zone IDs further supports this behavior. This insight into the last-mile delivery behavior and deviations allows us to refine feature selection and data preprocessing for improved model accuracy and reliability in predicting delivery routes.

Then, we developed and implemented a Simple RNN encoder-decoder model to predict delivery routes. While the model was able to capture some of the delivery route patterns, its performance in terms of route optimization was moderate. The model achieved a mean disparity score of 0.0220 with a standard deviation of 0.0070, indicating some deviation from the optimal route. The prediction accuracy for the first zone was 0.066, but it steadily declined to 0.050 by the fourth zone, illustrating the model's limitations in retaining long sequential data and making accurate predictions for the later stages of the route.

After evaluating the Simple RNN model, we identified its shortcomings, particularly in maintaining prediction accuracy for longer routes. As a result, we explored an LSTM encoder-decoder model with an attention mechanism. This model significantly outperformed the Simple RNN, achieving a mean disparity score of 0.0091 with a standard deviation of 0.0061. The LSTM model's prediction accuracy started at 0.194 for the first zone, with a more gradual decline to 0.128 by the fourth zone.

## CHAPTER 7

### Conclusion

In conclusion, in the entire supply chain, last-mile logistics are the finale character to deliver the final goods to the end users. Although the modern logistics industry utilizes operation research tools to minimize operation costs, last-mile logistics remains as the most expensive part of the entire supply chain. However, ever since pandemic, e-commerce had taken off, so as the demands for last-mile logistics remain soaring. In topics of last-mile optimization, delivery driver, with their tacit experience, could provide valuable real-life on-the-road knowledge to the table. The objective of the project was to study and derives possible factors causing deviation of pre-planned delivery routes by drivers. The motivation behind this project was to develop a machine learning model that able to capture drivers' tacit knowledges from historical delivery routes, thereafter, continuously optimize the as-is last-mile delivery frameworks and increase efficiency of the overall supply chain. In this project, we proposed a Simple R-NN model to output possible delivery routes, preferable by delivery drivers, learning from a set of historical delivery routes provided by Amazon. However, the project process faced challenges as the limitation of cloud computing unit and resources (i.e., idle time constraint), that hinders project progress when executing time consuming tasks. For future work, we will include focusing on fine-tuning (parameter tuning), as well as applied greedy algorithm, adapted from [5], into the proposed model in this project, continuous research effort on models that allows making inference on possible factors affects drivers' deviations from pre-planned routes.

**REFERENCES**

- [1] S. Srivatsa Srinivas and M. S. Gajanand, “Vehicle routing problem and driver behaviour: a review and framework for analysis,” *Transp Rev*, vol. 37, no. 5, pp. 590–611, Sep. 2017, doi: 10.1080/01441647.2016.1273276.
- [2] J. W. Ohlmann and B. W. Thomas, “A Compressed-Annealing Heuristic for the Traveling Salesman Problem with Time Windows,” *INFORMS J Comput*, vol. 19, no. 1, pp. 80–90, Feb. 2007, doi: 10.1287/ijoc.1050.0145.
- [3] Y. Liu, F. Wu, Z. Liu, K. Wang, F. Wang, and X. Qu, “Can language models be used for real-world urban-delivery route optimization?,” *Innovation*, vol. 4, no. 6, Nov. 2023, doi: 10.1016/j.xinn.2023.100520.
- [4] P. Dieter, M. Caron, and G. Schryen, “Integrating driver behavior into last-mile delivery routing: Combining machine learning and optimization in a hybrid decision support framework,” *Eur J Oper Res*, vol. 311, no. 1, pp. 283–300, Nov. 2023, doi: 10.1016/j.ejor.2023.04.043.
- [5] B. Mo, Q. Wang, X. Guo, M. Winkenbach, and J. Zhao, “Predicting drivers’ route trajectories in last-mile delivery using a pair-wise attention-based pointer neural network,” *Transp Res E Logist Transp Rev*, vol. 175, Jul. 2023, doi: 10.1016/j.tre.2023.103168.
- [6] B. P. V. Samson and Y. Sumi, “Exploring Factors that Influence Connected Drivers to (Not) Use or Follow Recommended Optimal Routes,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, New York, NY, USA: ACM, May 2019, pp. 1–14. doi: 10.1145/3290605.3300601.
- [7] D. Merchán *et al.*, “2021 Amazon Last Mile Routing Research Challenge: Data Set,” *Transportation Science*, vol. 58, no. 1, pp. 8–11, Jan. 2024, doi: 10.1287/trsc.2022.1173.
- [8] R. Salman, F. Ekstedt, and P. Damaschke, “Branch-and-bound for the Precedence Constrained Generalized Traveling Salesman Problem,” *Operations Research Letters*, vol. 48, no. 2, pp. 163–166, Mar. 2020, doi: 10.1016/j.orl.2020.01.009.

## REFERENCES

- [9] Y. Yuan, D. Cattaruzza, M. Ogier, and F. Semet, “A branch-and-cut algorithm for the generalized traveling salesman problem with time windows,” *Eur J Oper Res*, vol. 286, no. 3, pp. 849–866, Nov. 2020, doi: 10.1016/j.ejor.2020.04.024.
- [10] S. Nagula, “Last Mile Routing Research Challenge,” Medium.
- [11] William Cook, Stephan Held, and Keld Helsgaun, “Just Passing Through | Routes.”
- [12] William Cook, Stephan Held, and Keld Helsgaun, “Just Passing Through | Routes.”

## APPENDIX

### FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 2</b>
<b>Student Name &amp; ID: CHAN TZE KEET, 2004193</b>	
<b>Supervisor: Ms. Tseu Kwan Lee</b>	
<b>Project Title: LAST-MILE ROUTE OPTIMISATION WITH MACHINE LEARNING</b>	

#### 1. WORK DONE

[Please write the details of the work done in the last fortnight.]  
Code tidying and review from FYP I.

#### 2. WORK TO BE DONE

Getting more study materials for further research.

#### 3. PROBLEMS ENCOUNTERED

No problem encountered.

#### 4. SELF EVALUATION OF THE PROGRESS

Lack of knowledge to proceed, for model improvement.



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 4</b>
<b>Student Name &amp; ID: CHAN TZE KEET, 2004193</b>	
<b>Supervisor: Ms. Tseu Kwan Lee</b>	
<b>Project Title: LAST-MILE ROUTE OPTIMISATION WITH MACHINE LEARNING</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Successfully retrieved important feature (i.e., travel times between stops) from data file.

## 2. WORK TO BE DONE

Perform data cleaning and transformation with newest feature for model feeding.

## 3. PROBLEMS ENCOUNTERED

No problem encountered.

## 4. SELF EVALUATION OF THE PROGRESS

More attention required for better project progression.



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 6</b>
<b>Student Name &amp; ID: CHAN TZE KEET, 2004193</b>	
<b>Supervisor: Ms. Tseu Kwan Lee</b>	
<b>Project Title: LAST-MILE ROUTE OPTIMISATION WITH MACHINE LEARNING</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Done data processing and transformation.

## 2. WORK TO BE DONE

Build model.

## 3. PROBLEMS ENCOUNTERED

No problem encountered.

## 4. SELF EVALUATION OF THE PROGRESS

More attention required for better project progression.



Supervisor's signature



Student's signature



**FINAL YEAR PROJECT WEEKLY REPORT***(Project II)*

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 8</b>
<b>Student Name &amp; ID: CHAN TZE KEET, 2004193</b>	
<b>Supervisor: Ms. Tseu Kwan Lee</b>	
<b>Project Title: LAST-MILE ROUTE OPTIMISATION WITH MACHINE LEARNING</b>	

**1. WORK DONE**

[Please write the details of the work done in the last fortnight.]

Model Building.

**2. WORK TO BE DONE**

Model Training and Evaluation.

**3. PROBLEMS ENCOUNTERED**

Limited Cloud Computing Unit from Google Colab, which hinder the project progression.

**4. SELF EVALUATION OF THE PROGRESS**

More attention required for better project progression.



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 10</b>
<b>Student Name &amp; ID: CHAN TZE KEET, 2004193</b>	
<b>Supervisor: Ms. Tseu Kwan Lee</b>	
<b>Project Title: LAST-MILE ROUTE OPTIMISATION WITH MACHINE LEARNING</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Model Evaluation.

## 2. WORK TO BE DONE

Research on model. Design better model architecture.

## 3. PROBLEMS ENCOUNTERED

Masking problem in model leading to bad route prediction.

## 4. SELF EVALUATION OF THE PROGRESS

More attention required for better project progression.



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Trimester 3, Year 3</b>	<b>Study week no.: 12</b>
<b>Student Name &amp; ID: CHAN TZE KEET, 2004193</b>	
<b>Supervisor: Ms. Tseu Kwan Lee</b>	
<b>Project Title: LAST-MILE ROUTE OPTIMISATION WITH MACHINE LEARNING</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Design additional model for better prediction.

## 2. WORK TO BE DONE

Model Evaluation and Report writing.

## 3. PROBLEMS ENCOUNTERED

No problem encountered.

## 4. SELF EVALUATION OF THE PROGRESS

On track.



Supervisor's signature



Student's signature

POSTER



Wholly owned by UTAR Education Foundation  
(Co. No. 578227-M)  
DU012(A)

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Done by: Chan Tze Keet  
Supervisor: Ms. Tseu Kwan Lee

LAST-MILE ROUTE OPTIMISATION WITH MACHINE LEARNING

INTRODUCTION

Last-mile logistics ensures goods are delivered to every doorstep of end users.

Remains most expensive operation in entire supply chain logistics.

Tacit knowledge of drivers are often overlooked.

OBJECTIVES

To study and derives possible factors affecting drivers deviate from the pre-planned delivery routes.

To proposed Simple-RNN for route prediction.

To evaluate the proposed solution in terms of accuracy performance metrics.

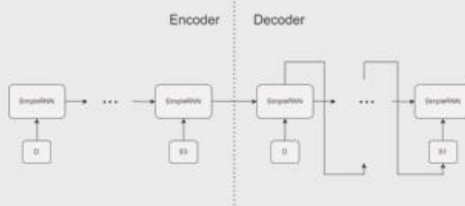
MOTIVATION

To develop a machine learning model (i.e., Simple R-NN model) that able to capture delivery drivers' tacit knowledge from historical routes.

FUTURE WORK

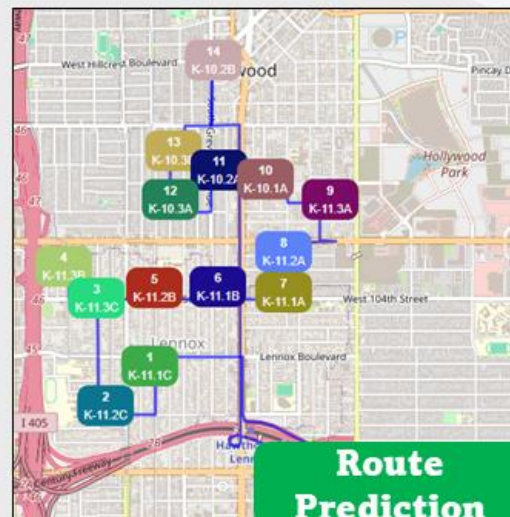
To fine-tuned proposed model as well as implement greedy algorithms.

METHODOLOGY:



CHALLENGES:

- Limitation of cloud computing resources.



## PLAGARISM CHECK RESULT

FYP2\_20ACB04193

ORIGINALITY REPORT

<b>15%</b> SIMILARITY INDEX	<b>11%</b> INTERNET SOURCES	<b>9%</b> PUBLICATIONS	<b>8%</b> STUDENT PAPERS
--------------------------------	--------------------------------	---------------------------	-----------------------------

PRIMARY SOURCES

<b>1</b>	<b>www.mit.edu</b> Internet Source	<b>2%</b>
<b>2</b>	<b>Submitted to Universiti Tunku Abdul Rahman</b> Student Paper	<b>1%</b>
<b>3</b>	<b>Baichuan Mo, Qingyi Wang, Xiaotong Guo, Matthias Winkenbach, Jinhua Zhao.</b> "Predicting drivers' route trajectories in last-mile delivery using a pair-wise attention-based pointer neural network", Transportation Research Part E: Logistics and Transportation Review, 2023 Publication	<b>1%</b>
<b>4</b>	<b>github.com</b> Internet Source	<b>1%</b>
<b>5</b>	<b>www.coursehero.com</b> Internet Source	<b>1%</b>
<b>6</b>	<b>Submitted to University of North Texas</b> Student Paper	<b>1%</b>
<b>7</b>	<b>pubsonline.informs.org</b> Internet Source	<b>1%</b>

## PLAGARISM CHECK RESULT

8	Submitted to Leiden University Student Paper	<1%
9	Yang Liu, Fanyou Wu, Zhiyuan Liu, Kai Wang, Feiyue Wang, Xiaobo Qu. "Can language models be used for real-world urban-delivery route optimization?", The Innovation, 2023 Publication	<1%
10	dokumen.pub Internet Source	<1%
11	Peter Dieter, Matthew Caron, Guido Schryen. "Integrating driver behavior into last-mile delivery routing: Combining machine learning and optimization in a hybrid decision support framework", European Journal of Operational Research, 2023 Publication	<1%
12	huggingface.co Internet Source	<1%
13	programtalk.com Internet Source	<1%
14	Poornachandra Sarang. "Artificial Neural Networks with TensorFlow 2", Springer Science and Business Media LLC, 2021 Publication	<1%
15	Submitted to Instituto de Empress S.L. Student Paper	<1%

## PLAGARISM CHECK RESULT

16	Submitted to King's College Student Paper	<1%
17	Submitted to University of Hertfordshire Student Paper	<1%
18	Submitted to University of Glasgow Student Paper	<1%
19	Submitted to UCL Student Paper	<1%
20	Pedro Zattoni Scroccaro, Piet van Beek, Peyman Mohajerin Esfahani, Bilge Atasoy. "Inverse Optimization for Routing Problems", Transportation Science, 2024 Publication	<1%
21	Submitted to Harrisburg University of Science and Technology Student Paper	<1%
22	Submitted to University of Surrey Student Paper	<1%
23	Submitted to American Public University System Student Paper	<1%
24	Submitted to Bilkent University Student Paper	<1%
25	Submitted to Monash University Student Paper	<1%

## PLAGARISM CHECK RESULT

26	<a href="http://ela.kpi.ua">ela.kpi.ua</a> Internet Source	<1%
27	Submitted to Macquarie University Student Paper	<1%
28	Submitted to University of Adelaide Student Paper	<1%
29	<a href="http://tensorflow.google.cn">tensorflow.google.cn</a> Internet Source	<1%
30	<a href="http://panamahitek.com">panamahitek.com</a> Internet Source	<1%
31	<a href="http://www.analyticsvidhya.com">www.analyticsvidhya.com</a> Internet Source	<1%
32	Submitted to Sikkim Manipal University Student Paper	<1%
33	<a href="http://e-jamet.org">e-jamet.org</a> Internet Source	<1%
34	<a href="http://www.ncbi.nlm.nih.gov">www.ncbi.nlm.nih.gov</a> Internet Source	<1%
35	<a href="http://tensorflow.classcat.com">tensorflow.classcat.com</a> Internet Source	<1%
36	<a href="http://www.appypie.com">www.appypie.com</a> Internet Source	<1%
37	Submitted to The Robert Gordon University Student Paper	<1%



## PLAGARISM CHECK RESULT

38	Submitted to University of East London Student Paper	<1 %
39	Submitted to University of Nebraska at Omaha Student Paper	<1 %
40	hdl.handle.net Internet Source	<1 %
41	Submitted to University of Glamorgan Student Paper	<1 %
42	qspace.library.queensu.ca Internet Source	<1 %
43	rdr.io Internet Source	<1 %
44	Yonghua Zhu, Weilin Zhang, Yihai Chen, Honghao Gao. "A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment", EURASIP Journal on Wireless Communications and Networking, 2019 Publication	<1 %
45	www.isl21.org Internet Source	<1 %
46	Submitted to Associatie K.U.Leuven Student Paper	<1 %
47	issuehub.io Internet Source	

## PLAGARISM CHECK RESULT

		<1%
48	<a href="https://sandipanweb.wordpress.com">sandipanweb.wordpress.com</a> Internet Source	<1%
49	Submitted to Middle East Technical University Student Paper	<1%
50	Santanu Pattanayak. "Chapter 4 Natural Language Processing", Springer Science and Business Media LLC, 2023 Publication	<1%
51	Submitted to The University of Manchester Student Paper	<1%
52	Submitted to UT, Dallas Student Paper	<1%
53	Submitted to University of Alabama at Birmingham Student Paper	<1%
54	Submitted to University of Southampton Student Paper	<1%
55	Xiao Yang, Ramesh Bist, Bidur Paneru, Lilong Chai. "Monitoring Activity Index and Behaviors of Cage-free Hens with Advanced Deep Learning Technologies", Poultry Science, 2024 Publication	<1%

## PLAGARISM CHECK RESULT

56	<a href="https://aiforsocialgood.ca">aiforsocialgood.ca</a> Internet Source	<1%
57	<a href="https://kandi.openweaver.com">kandi.openweaver.com</a> Internet Source	<1%
58	<a href="https://mafiadoc.com">mafiadoc.com</a> Internet Source	<1%
59	Daniel Merchán, Jatin Arora, Julian Pachon, Karthik Konduri, Matthias Winkenbach, Steven Parks, Joseph Noszek. "2021 Amazon Last Mile Routing Research Challenge: Data Set", Transportation Science, 2024 Publication	<1%
60	Submitted to University of Warwick Student Paper	<1%
61	Submitted to Brunel University Student Paper	<1%
62	<a href="https://blog.csdn.net">blog.csdn.net</a> Internet Source	<1%
63	<a href="https://tind-customer-uchicago.s3.amazonaws.com">tind-customer-uchicago.s3.amazonaws.com</a> Internet Source	<1%
64	<a href="https://www.kluniversity.in">www.kluniversity.in</a> Internet Source	<1%
65	Submitted to Katholieke Universiteit Leuven Student Paper	<1%

## PLAGARISM CHECK RESULT

66	<b>businessdocbox.com</b> Internet Source	<1 %
67	<b>hybrid-analysis.com</b> Internet Source	<1 %
68	<b>kipdf.com</b> Internet Source	<1 %
69	<b>ucf.digital.flvc.org</b> Internet Source	<1 %
70	<b>forum.posit.co</b> Internet Source	<1 %
71	<b>ir.uitm.edu.my</b> Internet Source	<1 %
72	<b>"CS2-PC-24_HR.pdf", ActEd</b> Publication	<1 %
73	<b>Jyotsna Kumar Mandal, Sanjay, Jyoti Sekhar Banerjee, Somen Nayak. "Applications of Machine Intelligence in Engineering - Proceedings of 2 Global Conference on Artificial Intelligence and Applications IGCAIA, 2021), September 8-10, 2021, Jaipur, India", Routledge, 2022</b> Publication	<1 %
74	<b>Lei Wang, Chanying Li, Michael Z. Q. Chen, Qing-Guo Wang, Fei Tao. "Connectivity-Based Accessibility for Public Bicycle Sharing</b>	<1 %

## PLAGARISM CHECK RESULT

### Systems", IEEE Transactions on Automation Science and Engineering, 2018

Publication

---

**75** Mohamed Abdel-Basset, Hossam Hawash,  
Laila Abdel-Fatah. "Artificial Intelligence and  
Internet of Things in Smart Farming", CRC  
Press, 2024

Publication

<1%

---

**76** Sougata Sheet, Ranjan Ghosh, Anupam  
Ghosh. "Recognition of cancer mediating  
genes using MLP-SDAE model", Systems and  
Soft Computing, 2024

Publication

<1%

---

**77** [assets.publishing.service.gov.uk](https://assets.publishing.service.gov.uk)

Internet Source

<1%

---

**78** [debuggercafe.com](https://debuggercafe.com)

Internet Source

<1%

---

**79** [dl.lib.mrt.ac.lk](https://dl.lib.mrt.ac.lk)

Internet Source

<1%

---

**80** [forums.mcafee.com](https://forums.mcafee.com)

Internet Source

<1%

---

**81** [meridian.allenpress.com](https://meridian.allenpress.com)

Internet Source

<1%

---

**82** [pythonlang.dev](https://pythonlang.dev)

Internet Source

<1%

---

[www.mdpi.com](https://www.mdpi.com)

## PLAGARISM CHECK RESULT

83	Internet Source	<1%
84	<a href="http://www.themesnap.com">www.themesnap.com</a> Internet Source	<1%
85	Mehdi Ghayoumi. "Generative Adversarial Networks in Practice", CRC Press, 2023 Publication	<1%
86	Mohamed Abdel-Basset, Nour Moustafa, Hossam Hawash, Zahir Tari. "Responsible Graph Neural Networks", CRC Press, 2023 Publication	<1%
87	Titus A. Beu. "Introduction to Numerical Programming - A Practical Guide for Scientists and Engineers Using Python and C/C++", CRC Press, 2019 Publication	<1%
88	"Cognitive Informatics and Soft Computing", Springer Science and Business Media LLC, 2021 Publication	<1%
89	S. Putman. "Integrated Urban Models Volume 1: Policy Analysis of Transportation and Land Use (RLE: The City)", Routledge, 2013 Publication	<1%

## PLAGARISM CHECK RESULT

Exclude quotes Off  
Exclude bibliography Off

Exclude matches Off

PLAGARISM CHECK RESULT

<b>Form Title: Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)</b>			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

<b>Full Name(s) of Candidate(s)</b>	CHAN TZE KEET
<b>ID Number(s)</b>	2004193
<b>Programme / Course</b>	BACHELOR OF COMPUTER SCIENCE (HONOURS)
<b>Title of Final Year Project</b>	LAST-MILE ROUTE OPTIMISATION WITH MACHINE LEARNING

<b>Similarity</b>	<b>Supervisor's Comments (Compulsory if parameters of originality exceed the limits approved by UTAR)</b>
<b>Overall similarity index: <u>15</u> %</b>  <b>Similarity by source</b>  Internet Sources: <u>11</u> % Publications: <u>9</u> % Student Papers: <u>8</u> %	
<b>Number of individual sources listed of more than 3% similarity: <u>0</u></b>	
<b>Parameters of originality required, and limits approved by UTAR are as Follows:</b> (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note: Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

***Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.***



\_\_\_\_\_  
 Signature of Supervisor  
  
 Name: Ms. Tseu Kwan Lee  
  
 Date: 11/09/2024

\_\_\_\_\_  
 Signature of Co-Supervisor  
  
 Name: \_\_\_\_\_  
  
 Date: \_\_\_\_\_



## FYP 2 CHECKLIST



### UNIVERSITI TUNKU ABDUL RAHMAN

#### FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS) CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB04193
Student Name	CHAN TZE KEET
Supervisor Name	MS. TSEU KWAN LEE

TICK (✓)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
✓	Title Page
✓	Signed Report Status Declaration Form
✓	Signed FYP Thesis Submission Form
✓	Signed form of the Declaration of Originality
✓	Acknowledgement
✓	Abstract
✓	Table of Contents
✓	List of Figures (if applicable)
✓	List of Tables (if applicable)
✓	List of Symbols (if applicable)
✓	List of Abbreviations (if applicable)
✓	Chapters / Content
✓	Bibliography (or References)
✓	All references in bibliography are cited in the thesis, especially in the chapter of literature review
✓	Appendices (if applicable)
✓	Weekly Log
✓	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
✓	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

\*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date: 11/09/2024