

Cover sheet for submission of work for assessment



UNIT DETAILS

Unit name	Introduction to Artificial Intelligence	Class day/time	Tuesday/0830	Office use only	
Unit code	COS30019	Assignment no.	2B	Due date	25/5/2025
Name of lecturer/teacher	Prof. Bao Vo				
Tutor/marker's name	Hy Nguyen			Faculty or school date stamp	

STUDENT(S)

	Family Name(s)	Given Name(s)	Student ID Number(s)
(1)	Ly	Vien Minh Khoi	103844366
(2)	Hand	Gareth	104156787
(3)	Leong	Chun Wai	105239948
(4)	Bong	Christine Khai Ning	102787457
(5)			
(6)			

DECLARATION AND STATEMENT OF AUTHORSHIP

1. I/we have not impersonated, or allowed myself/ourselves to be impersonated by any person for the purposes of this assessment.
2. This assessment is my/our original work and no part of it has been copied from any other source except where due acknowledgement is made.
3. No part of this assessment has been written for me/us by any other person except where such collaboration has been authorised by the lecturer/teacher concerned.
4. I/we have not previously submitted this work for this or any other course/unit.
5. I/we give permission for my/our assessment response to be reproduced, communicated, compared and archived for plagiarism detection, benchmarking or educational purposes.

I/we understand that:

6. Plagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a form of cheating and is a very serious academic offence that may lead to exclusion from the University. Plagiarised material can be drawn from, and presented in, written, graphic and visual form, including electronic data and oral presentations. Plagiarism occurs when the origin of the material used is not appropriately cited.

Student signature/s

I/we declare that I/we have read and understood the declaration and statement of authorship.

(1)	KHOI	(4)	CHRISTINE
(2)	GARETH	(5)	
(3)	CHUN WAI	(6)	

Further information relating to the penalties for plagiarism, which range from a formal caution to expulsion from the University is contained on the Current Students website at www.swin.edu.au/student/

Copies of this form can be downloaded from the Student Forms web page at www.swinburne.edu.au/studentforms/

Table of Contents

Instructions	2
Introduction.....	6
Features/Bugs/Missing	7
Features Implemented	7
Bugs Found	8
Missing Features	9
Testing.....	10
Insights.....	13
Research	15
Conclusion	16
Acknowledgement/Resources	17
Workload Matrix	20

Instructions

The assignment consists of 3 executable scripts that are placed in the scripts directory in the zip folder. Each of these scripts has distinctive responsibility, with the order and mean of execution can be found in the following instructions:

1. Extract the zip folder to a desired directory.
2. Navigate to the extracted folder and open the terminal in the project's root directory by right clicking and choose Open in Terminal.
3. Check if all the data have been processed by navigating to the directory "data\processed", where precisely 139 Excel files should be found. Each of these file should represent a location in the raw data sheet. If there are less than the required number of files or if the directory is empty, process the raw data by executing the command:

a. `python -m scripts.process_data`

This will read the raw data from the directory "data\raw", process it, and export the processed data as well as the scaler used to the directory "data\processed" and "data\models" respectively. The filename of the exported files is just the name of the location, with the scaler have the "_scaler" added in the end. For example: the location "AUBURN_RD N of BURWOOD_RD" will have the processed data filename of "AUBURN_RD N of BURWOOD_RD.xlsx" and the scaler filename of "AUBURN_RD N of BURWOOD_RD_scaler.save".

4. Check if each implemented machine learning model (LSTM, GRU, and XGB) have been trained and exported on the data for each location by navigating to the directory "data\models", where precisely $139 \times 4 = 556$ files can be found. Out of these files, $139 \times 3 = 417$ files have the format of .pkl, indicating that they are the machine learning models that have been trained using the implemented models and processed data. These files have the naming format of "<location name>_<model name>.pkl", where <location name> and <model name> are the location in which the data is being used, as well as the name of the machine learning model being trained respectively. For instance, the trained model using GRU at the location "AUBURN_RD N of BURWOOD_RD" will have the filename of "AUBURN_RD N of BURWOOD_RD_gru.pkl". The remaining 139 file have the format of .save, indicating that they are the saved scaler, which are generated when executing step 3. If there are less than the required number of files or if the directory is empty, check step 3 and train the model by executing the command:

a. `python -m scripts.train_models`

This will use the implemented machine learning models, saved in the directory “src\models” to train one model at a time, using the processed data generated in step 3, and saved the trained model instance in the directory “data\models”.

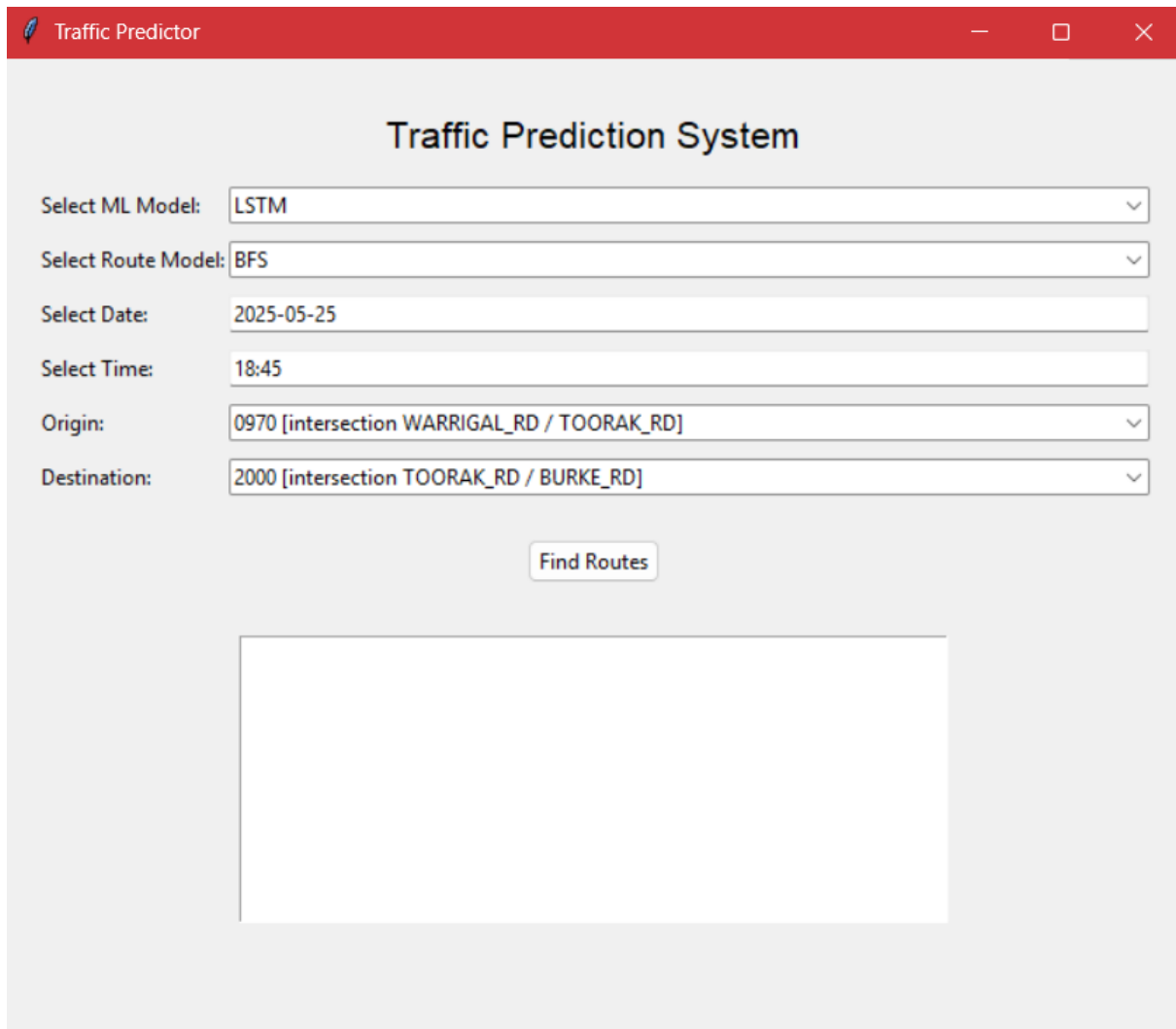
5. (Optional) If the trained models needed to be evaluated, execute the command:
 - a. `python -m scripts.evaluate_models <model name> <display>`

Where the <model name> can be the name of the trained model, such as “AUBURN_RD N of BURWOOD_RD_gru.pkl” if only a specific model is needed to be evaluated, or the flag “-a” if all model available in the directory “data\models” are required to be evaluated. The <display> option can be left empty if the only evaluation metric required is the model’s root mean squared error (RMSE), or the “-d” flag to also display a graph comparing the predicted and original values. Keep in mind that using both flags (“-a” and “-d”) at the same time is not recommended due each graph will be generated and show after each evaluation, which is extremely time consuming.

6. Open the interactive GUI and finding the most optimal route given the date, time, origin, destination, machine learning model, and path finding algorithm by executing the command:
 - a. `python -m scripts.run_app`

This will open the GUI, which can be seen in Figure 1. Traffic based Rout Guidance system GUI. Figure 1. Here, the “Select ML Model” field allow users to use one of the implemented machine learning models to predict the travel time between 2 scat sites (the cost of each edge between 2 nodes) at the date and time indicated in the “Select Date” and “Select Time” respectively. The current implementation supports any date and time from 2006-10-01 00:00 to current date, but the recommended date time is within October and maximum 1 day after November 2006 because any time afterward would increase the runtime significantly. The “Select Route Model” field allows users to choose a path finding algorithm, implemented in Assignment 2A (BFS, DFS, GBFS, AS, UCS, and IDA) to find the most optimal path between the origin and destination nodes, inserted in the “Origin” and “Destination” field respectively. If the chosen machine learning model, date, time, origin, and destination have never been chosen before, in other words, there are no graph file has been generated with those chosen parameters, the application will generate a new graph and exported in the directory “output” as a text file. The file has the naming format of “test_from_<origin>_to_<destination>_using_<model name>_at_<datetime>.txt”, with corresponding chosen parameters passed into the filename. This action of generating new graph file is very time consuming due to machine learning models needed to predict each location’s travel time, hence if the same parameters are

inputted, the application will find the file in the directory and use that file as the graph to find the path instead.

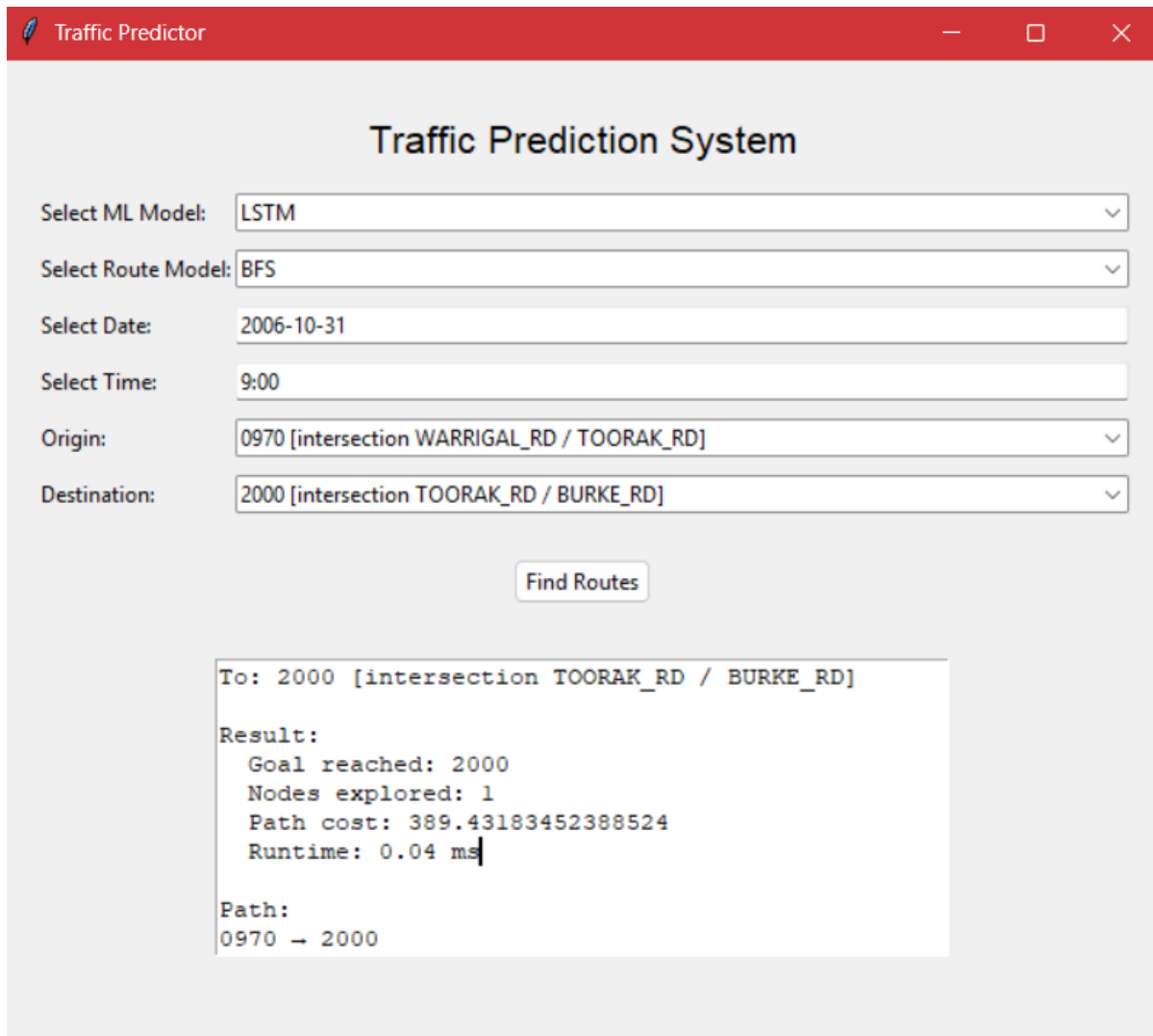


The screenshot shows a web application window titled "Traffic Predictor". The main heading is "Traffic Prediction System". Below the heading, there are several input fields and a button:

- Select ML Model:** A dropdown menu with "LSTM" selected.
- Select Route Model:** A dropdown menu with "BFS" selected.
- Select Date:** A text input field containing "2025-05-25".
- Select Time:** A text input field containing "18:45".
- Origin:** A dropdown menu with "0970 [intersection WARRIGAL_RD / TOORAK_RD]" selected.
- Destination:** A dropdown menu with "2000 [intersection TOORAK_RD / BURKE_RD]" selected.
- Find Routes:** A button located below the input fields.
- Map Area:** A large, empty rectangular box intended for displaying the route.

Figure 1. Traffic based Rout Guidance system GUI.

An example usage of the GUI can be seen in Figure 2.



The screenshot shows a window titled "Traffic Predictor" with a red header bar. The main content area is titled "Traffic Prediction System". It contains several input fields and a button:

- Select ML Model: LSTM
- Select Route Model: BFS
- Select Date: 2006-10-31
- Select Time: 9:00
- Origin: 0970 [intersection WARRIGAL_RD / TOORAK_RD]
- Destination: 2000 [intersection TOORAK_RD / BURKE_RD]

A "Find Routes" button is located below the input fields. Below the button, the output is displayed in a text box:

```
To: 2000 [intersection TOORAK_RD / BURKE_RD]
Result:
  Goal reached: 2000
  Nodes explored: 1
  Path cost: 389.43183452388524
  Runtime: 0.04 ms
Path:
0970 → 2000
```

Figure 2. A sample input and output of the GUI.

The application also output a map with all of the nodes and edges generated. The map highlights the found path in red for better visualization of the solution, as well as all the edges' cost annotated in a legend as can be seen in Figure 3.

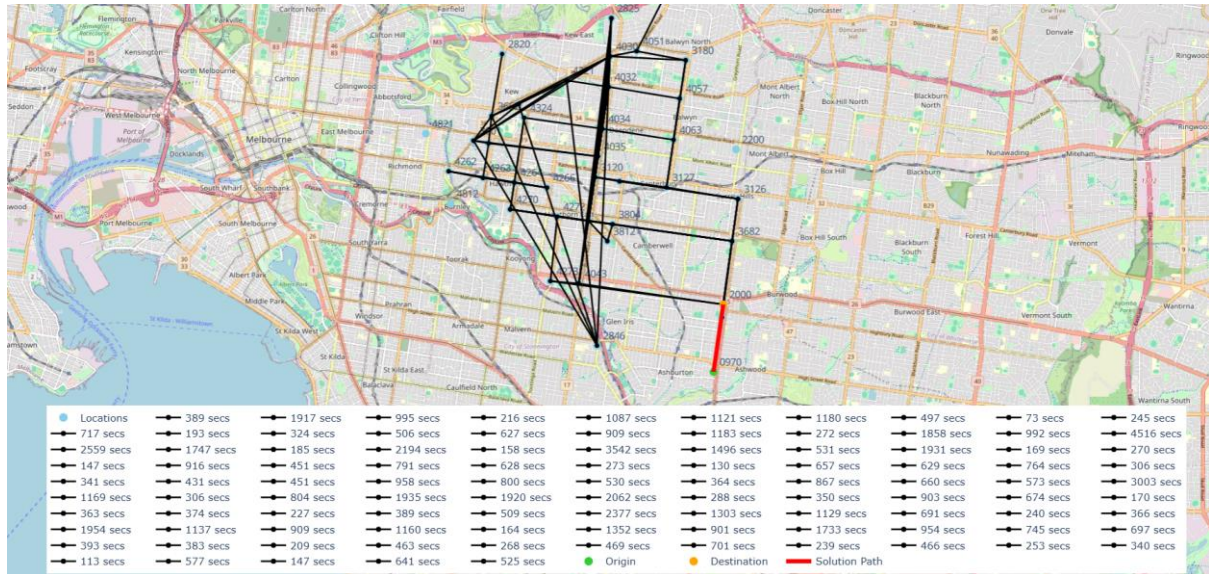


Figure 3. A sample graph map generated by the GUI.

Introduction

The Goal of this task was to create an efficient navigation system, The traffic-based route Guidance system address the challenge by utilizing machine learning models to predict traffic and guide users through the most efficient routes available at any given time.

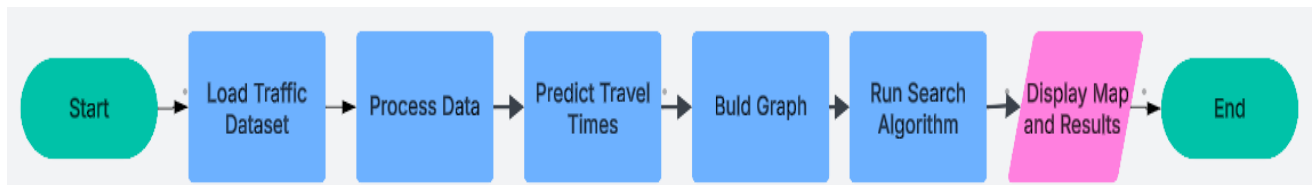
In this project we utilized the VicRoads 2006 October Boroondara traffic data. This dataset contained traffic volume which was collected every 15 minutes at every road. The goal was to use this data to predict future traffic condition and then use that to predict travel times between intersections. These predictions would give optimal routes between the origin and destination.

To Solve this, our system utilizes:

- **Machine Learning Models** to forecast times
 - LSTM (Long Short-Term Memory)
 - GRU (Gated Recurrent Unit)
 - XGBoost
- **Graph Based search algorithms** to find the optimal paths
 - DFS (Depth-First Search)
 - BFS (Breadth-First Search)
 - GBFS (Greedy Best-First Search)
 - A* (A-Star Search)
 - UCS / Dijkstra

- IDA (Iterative Deepening A-Star)

By combining these time-based traffic predictions with AI search techniques, it allows our system to its routing strategy and make predictions based on traffic volume. This is all displayed in the GUI which also gives visualisation of the traffic graph and display of the optimal route



Features/Bugs/Missing

Features Implemented

1. Machine Learning Traffic Prediction

The System integrates three machine learning models to predict/forecast traffic conditions (vehicle volume every 15 minutes). These predictions are then used to calculate the estimated time to travel between intersections to go from the origin to the destination and to update edge costs in the traffic network.

- **LSTM (Long short-term memory)**
 - a deep learning model that works well with time series data
 - it is integrated through the lstm_model.py
- **GRU (Gated Recurrent Unit)**
 - A lighter alternative to LSTM, offering faster training and fewer parameters
 - It is integrated through the gru_model.py
- **XGBoost (Extreme Gradient Boosting)**
 - A tree-based ensemble that works well for structured data like the traffic volumes data that we have been working with
 - Has high accuracy and is fast performing

All models follow the same structure by inheriting from a shared base class making them easy to manage and switch between, where they predict travel times for each road segment.

2. Search Algorithms for Route Calculation

We reused the six search algorithms developed in Assignment 2A and integrated them with the ML in part B. These algorithms help us find the optimal routes using the time travel times predicted by the ML models.

- DFS
- BFS
- GBFS
- A*
- UCS / Dijkstra
- IDA

Each search outputs the path, runtime and number of nodes explored to help you predict the travel time from origin to destination.

3. Data Processing

The data processor is used to clean, format and prepare the traffic data set for model training and testing

- The data is saved into structured files that are then ready for model input
- This speeds up the process considerably

4. Interactive Graph

A comprehensive GUI was built with tkinter and plotly which allows users to

- Display the full traffic graph using OpenStreetMap's tiles.
- Select origin and destination
- View the solution on the map

This includes

- Green nodes for origin
- Orange nodes for destination
- Red line for optimal path

Bugs Found

1. Lack of error handling

The system lacks error handling in certain areas; this can confuse users and make debugging the code harder when it lacks giving clear feedback .

2. Nodes Location

Certain node's locations Arnt 100% accurate this can create confusing lines in the map which makes it harder for the user to understand.

3. Infinite Future date prediction

There is no feature stopping users from guessing pass November 2006 which can cause the system to go far beyond the training range. This cause slow and unreliable results which can be meaningless while providing no warning.

Missing Features

1. Exporting Results

Currently there is no way to export results such as optimal paths, performance metrics or travel time predications. This limits the ability to document and compares different runs. Potentially adding this feature would improve analysis ability.

2. Prediction Range

Users can enter any future date or time even far beyond the range covered by the training data. This as stated before can be inaccurate and takes a while to process. A feature to restricting any timestamp beyond November would fix this issue.

3. Result Validation

The system provides the results for each test case however it does not confirm whether those results are correct. There isn't any cross checking, and any validation check must be done by yourself. This can make that there are errors that haven't yet been identified unless manually checked.

Testing

Evaluation of ML Models

LSTM

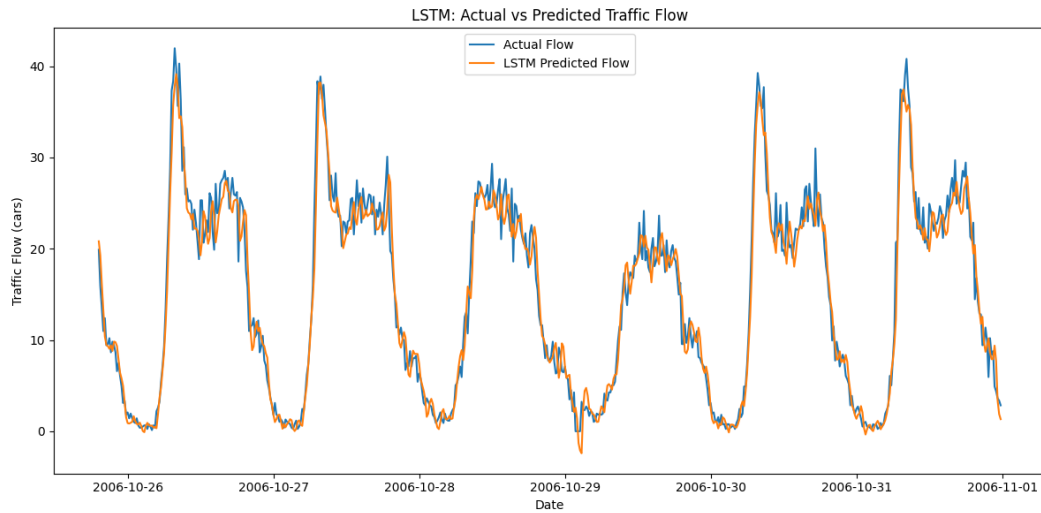


Figure 4: Evaluation of LSTM

Model: AUBURN_RD N of BURWOOD_RD_lstm.pkl

RMSE: 2.24

GRU

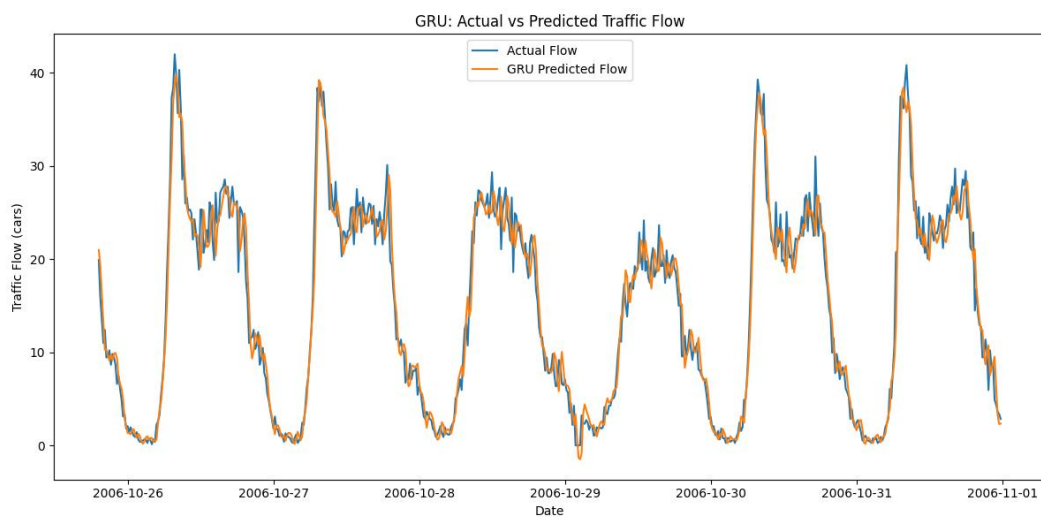


Figure 5: Evaluation of GRU

Model: AUBURN_RD N of BURWOOD_RD_gru.pkl

RMSE: 2.18

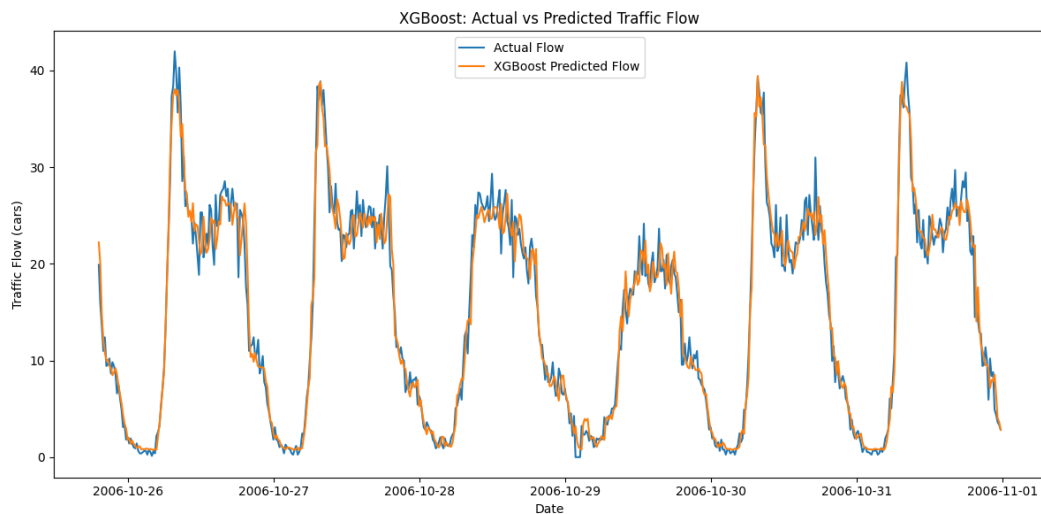
XGBoost

Figure 6: Evaluation of XGBoost

Model: AUBURN_RD N of BURWOOD_RD_xgb.pkl

RMSE: 2.02

The data for SCATS site AUBURN_RD N of BURWOOD_RD has been used to train the ML models. The data is separated into 80% for training and 20% for evaluation. The orange line is the model's predicted traffic flow which is compared with the blue line, which is the 20% unseen data from the original test set. Based on the results, XGBoost has the lowest Root Mean Squared Error (RMSE) then followed by GRU and lastly the LSTM which has the highest RMSE value. This result indicates that XGBoost has the best accuracy among these 3 ML models with a complete dataset given.

Software Testing

Origin	Destination	Date & Time	ML Model	Search Algorithm	Nodes explored	Travel time (seconds)	Runtime (seconds)
0970	2000	2006-10-01 09:00	LSTM	BFS	1	1180.36	100
0970	2000	2006-10-01 09:00	GRU	BFS	1	1081.86	108
0970	2000	2006-10-01 09:00	XGBoost	BFS	1	1038.94	58
2000	4063	2006-10-15 12:00	LSTM	IDA*	208	2746.22	107
2000	4063	2006-10-15 12:00	GRU	IDA*	212	2673.14	111
2000	4063	2006-10-15 12:00	XGBoost	IDA*	208	2878.96	57
3001	4324	2006-10-31 23:45	LSTM	A*	11	337.37	110
3001	4324	2006-10-31 23:45	GRU	A*	11	337.37	115
3001	4324	2006-10-31 23:45	XGBoost	A*	11	337.37	58
3001	4324	2006-11-01 23:45	LSTM	A*	15	1523.44	1080
3001	4324	2006-11-01 23:45	GRU	A*	7	914.24	1080
3001	4324	2006-11-01 23:45	XGBoost	A*	11	337.37	70

Table I: Traffic Prediction Results

12 test cases were executed using the software GUI. All test cases have a different combination of SCATS origin and destination, date, time, ML model and search algorithm. A new graph is generated with the predicted travel times which automatically opens up in your default browser. Then, the GUI will display the nodes explored, travel time (path cost) and the solution path in text format. There are 3 test cases where the date and time is out of range from the raw data provided to test each ML model predict function using the previous data.

For the first 9 test cases, the travel time for all 3 ML models is similar but XGBoost has the significant lower runtime compared to LSTM and GRU. This is because XGBoost uses decision tree instead of deep network which is much complicated to train. The difference in runtime gets even bigger when predicting traffic that is out of the given data range. In the last 3 test cases, both LSTM and GRU takes about 18 minutes (1080 seconds) for it to generate the graph for all edges but XGBoost only takes 1 minute and 10 seconds (70 seconds).

Insights

Model	RMSE
LSTM	2.24
GRU	2.18
XGBoost	2.02

Table II: ML Models' RMSE

Origin	Destination	Date & Time	ML Model	Search Algorithm	Nodes explored	Travel time (seconds)	Runtime (seconds)
3001	4324	2006-11-01 23:45	LSTM	A*	15	1523.44	1080
3001	4324	2006-11-01 23:45	GRU	A*	7	914.24	1080
3001	4324	2006-11-01 23:45	XGBoost	A*	11	337.37	70

Table III: Traffic Prediction on 1st November 2026

In the model evaluation for all three models, LSTM, GRU and XGBoost. XGBoost has the lowest RMSE (2.02) followed by GRU (2.18) and LSTM (2.24). Simply based on the RMSE scores, XGBoost has the best accuracy with the evaluation dataset. However, in the real-world application where these models were used to predict the traffic flow from SCATS site 3001 to 4324 on 1st November 2006, 23:45, XGBoost has an unrealistic travel time of 337.37 seconds. This happens because XGBoost is a tree-based regression model which lack of temporal sequence which means it does not handle the time-dependent patterns data.

XGBoost is good at learning the pattern. For example, it learns the pattern of the traffic flow in October with the 80% of training data and compared with the remaining 20%. Since both data is from the same dataset which has the similar pattern, that is why XGBoost has an outstanding performance when doing the evaluation. However, when predicting the future

traffic flow on 1st of November 2006, XGBoost perform poorly because there are some time-dependent patterns like peak hours or days which it could easily underestimate the traffic flow.

On the other hand, even though LSTM and GRU models also has a huge difference in the predicted travel time, but the travel time is much more realistic compared to XGBoost. This is because both LSTM and GRU are sequential models that could learn the time-dependent patterns which allow them to predict the future traffic flow on 1st November 2006 more accurately.

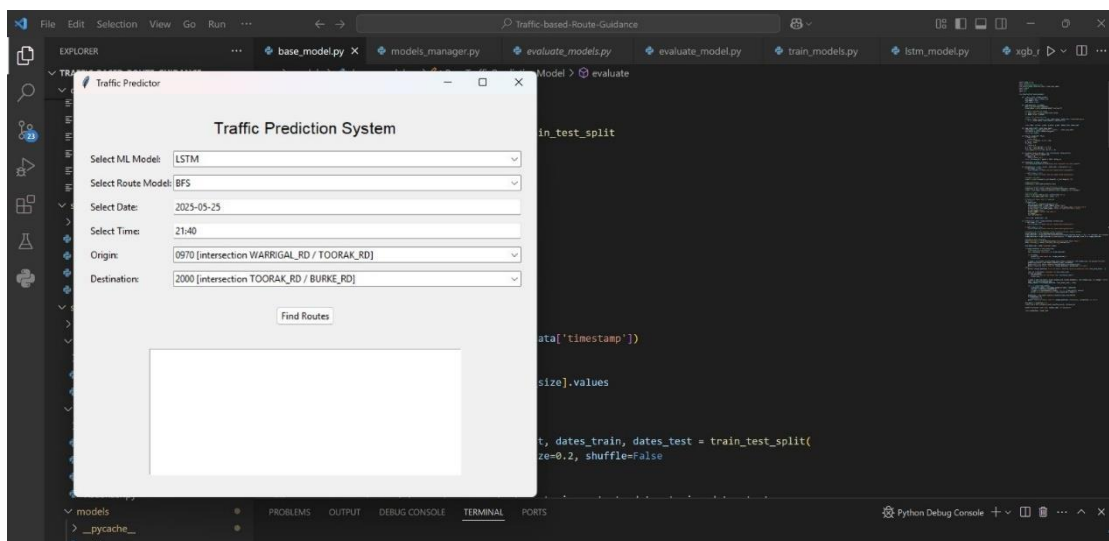
Despite all the ML models using a window size of 5, but non-sequence model and sequence model treat the inputs differently. XGBoost – non-sequence model only treats it as 5 independent values but LSTM and GRU – sequence model treats the values in order which allow it to learn how the input changes at certain time. Hence, sequence model is able to predict the future traffic flow more accurately based on the historical data trends even the date is out of range (1st November 2006) while non-sequence model underestimated the traffic flow and provide an inaccurate prediction. In conclusion, LSTM and GRU is more suitable for traffic flow prediction.

Research

GUI & Visualization Research

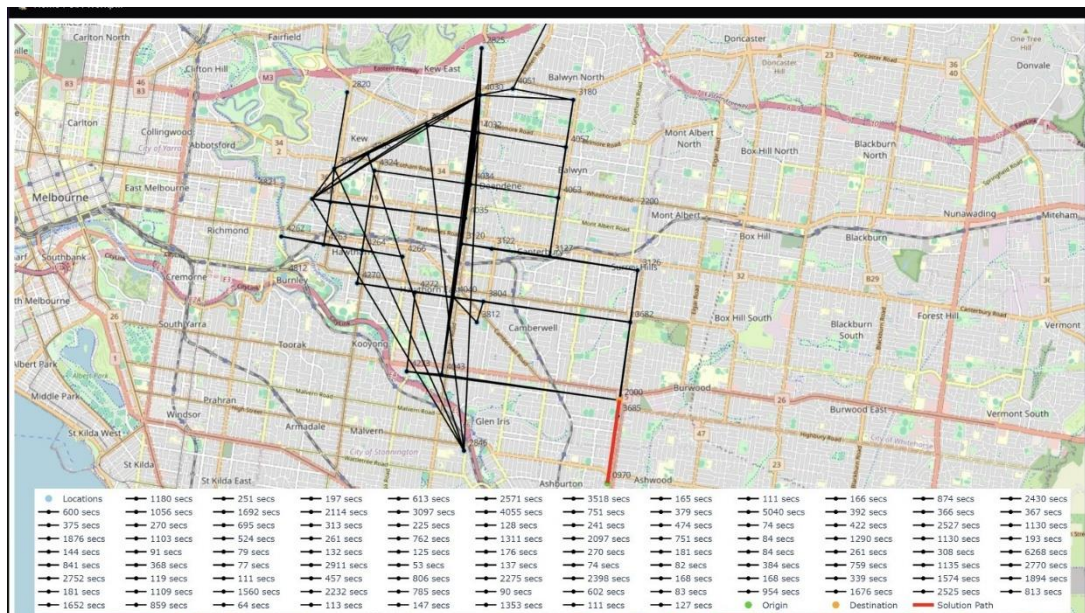
Built a GUI (using Tkinter) to:

- Let users pick locations/dates with dropdowns for all 139 SCATS sites and specify dates
- Show predictions as easy-to-read graphs comparing LSTM, GRU, and XGBoost performance
- Display the best routes on the map, providing an immediate visual understanding of recommended routes



Created automatic graphs that compare:

- Predicted vs actual traffic (LSTM vs GRU vs Actual data showing time-series patterns and daily traffic trends)
- Travel times for different routes (A* vs Dijkstra vs other algorithms) with performance metrics like nodes explored and processing time. Paths highlight on hover to show individual road travel times.



Technical Implementation:

Used Plotly for interactive graphs and Matplotlib for basic comparison charts. Added caching which made loading data much faster, cutting query times from minutes to just seconds.

Research Findings:

Comparisons showed that LSTM and GRU models were better at predicting future traffic than XGBoost, even though XGBoost had slightly lower error scores. This means understanding how traffic changes over time are more important than just getting a low error number.

Conclusion

We successfully built a smart traffic system that predicts traffic and finds the best routes using AI. In the system, we use three different models (LSTM, GRU, XGBoost) and includes a user-friendly map interface.

Key finding

XGBoost had the best test scores but gave unrealistic future predictions. LSTM and GRU had slightly worse scores but made much more realistic traffic predictions for future dates. This is because LSTM and GRU understand time patterns (like rushing hours) while XGBoost treats each data point separately.

Performance

103844366_104156787_102787457_105239948

XGBoost was the fastest (1 minute) but LSTM/GRU took up to 18 minutes for future predictions. However, the slower models gave more believable results.

Limitations

The system needs better error handling, location accuracy improvements, and the ability to save results.

Final Thoughts

It's proven that AI could solve real traffic problems. The main lesson: for time-based predictions, models that understand time patterns work better than those with just high accuracy scores. This system provides a solid foundation for advanced traffic management.

Acknowledgement/Resources

COS30019 Lectures and Tutorials

The Weekly lectures and lab sessions were essential in helping us understand the integration of AI algorithms. The tutorials were particularly useful in helping us

- Structuring graph based search problems
- Understanding how ML models work

LSTM (Long Short-Term Memory)

1. Machine learning Mastery – Time Series Prediction with LSTM in Python using keras

<https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/>

This article assists in helping us understand how to structure and shape input sequences for LSTM. It also expanded our knowledge on LSTM and gave us real python examples

2. GeeksForGeeks - Time Series Forecasting using Recurrent Neural Networks (RNN)

<https://www.geeksforgeeks.org/time-series-forecasting-using-recurrent-neural-networks-rnn-in-tensorflow/>

Provided us foundation guidance on how to prepare time series data and build the RNNs utilizing tools like TensorFlow/keras

3. GeeksForGeeks – Long Short-Term Memory (LSTM) RNN in TensorFlow

<https://www.geeksforgeeks.org/time-series-forecasting-using-recurrent-neural-networks-rnn-in-tensorflow/>

Supported in helping us implement the LSTM layers and overall configuration while using keras

XGBoost (Extreme Gradient Boosting)

1. Extreme Gradient Boosting with XGBoost – DataCamp Notebook Notes (GitHub)

<https://github.com/jadoonengr/DataCamp-Notes/blob/master/Extreme%20Gradient%20Boosting%20with%20XGBoost.ipynb>

Provided a step-by-step implementation of XGBoost, including model creation, fitting, and evaluation. Helped in understanding how to visualize model performance and interpret results.

2. GeeksForGeeks – Implementation of XGBoost (Extreme Gradient Boosting)

<https://www.geeksforgeeks.org/implementation-of-xgboost-extreme-gradient-boosting/>

Guided the basic setup of XGBoost using Python, especially the use of XGBRegressor, setting learning rate, and number of estimators.

3. Machine Learning Mastery – XGBoost Ensemble in Python

<https://machinelearningmastery.com/extreme-gradient-boosting-ensemble-in-python/>

Explained how to implement XGBoost from scratch, train and test the model, and evaluate it using performance metrics like RMSE.

GRU (Gated Recurrent Unit)

1. GeeksForGeeks – Introduction to Gated Recurrent Unit (GRU)

<https://www.geeksforgeeks.org/gated-recurrent-unit-networks/>

Helped us understand the architecture of GRUs and how they compare to LSTM models.

2. Towards Data Science – GRU Recurrent Neural Networks: A Smart Way to Predict Sequences in Python

<https://towardsdatascience.com/gru-recurrent-neural-networks-a-smart-way-to-predict-sequences-in-python-80864e4fe9f6/>

Provided practical implementation tips and visual explanations that helped us understand how GRUs work.

Haversine Formula

<https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/>

This reference helped us implement geographic distance calculations between SCATS site coordinates using the Haversine formula.

Plotly

<https://www.geeksforgeeks.org/python-plotly-tutorial/>

Helped us get started with Plotly's syntax and features. It was useful for understanding how to add traces, adjust layout, and integrate map visuals.

Workload Matrix

Task		Ly Vien Minh Khoi (103844366)	Gareth Hand (104156787)	Christine Khai Ning Bong (102787457)	Chun Wai Leong (105239948)
Implementation	Process Raw Data	✓			
	Implement LSTM Model	✓			
	Implement GRU Model				✓
	Implement XGB Model				✓
	Implement Models Manager (Train and Evaluate)	✓			
	Generate Graph File	✓	✓		
	Generate Map	✓			
	Implement GUI			✓	✓
	OOP Refactor	✓			
Testing	Test Epochs and Batch Size	✓			
	Generate Test Cases				✓
	Validate Test Cases				✓
	Debug	✓			✓
Report	Instructions	✓			
	Introduction		✓		
	Features/Bugs/Missing		✓		
	Testing				✓
	Insights				✓
	Research			✓	
	Conclusion			✓	