Dwayne D’Souza - 105332012

Mitchell Walker - 103995277

Al Shahriar Miad - 104473978

Ruzmy Mohamed  - 104799360

Assignment 2B - Machine Learning and Software Integration

Contents

[Assignment 2B - Group Report 1](#_Toc199093503)

[1. Instructions: 2](#_Toc199093504)

[2. Introduction: 3](#_Toc199093505)

[3. Features/Bugs/Missing: 3](#_Toc199093506)

[4. Testing: 4](#_Toc199093507)

[5. Insights: 11](#_Toc199093508)

[6. Research: 16](#_Toc199093509)

[7. Conclusion: 16](#_Toc199093510)

[8. Acknowledgements/Resources: 17](#_Toc199093511)

[9. References: 17](#_Toc199093512)

# Instructions:

* <https://github.com/xdjx19/IntroToAI_CL1-05-Khoa-Group-06?tab=readme-ov-file#readme-ov-file>
* To install the required imports: ‘pip install pandas numpy torch scikit-learn matplotlib tqdm openpyxl networkx’
* For the first part of training and testing make sure to run ‘training.py’
* To view our first model (GRU) make sure to run ‘gru\_model.py’
* To view our second model (LSTM) make sure to run ‘lstm\_model.py’
* To view our third model (SAE) make sure to run ‘sae\_model.py’
* To view our evaluation make sure to run ‘trafficevaluation.py’
* To run comparison the GUI run ‘gui.py’
* Configure gui defaults in config.json

# Introduction:

The Traffic-Based Route Guidance System addresses one of the most pressing challenges in modern urban transportation: predicting traffic flow patterns and optimizing route planning decisions. Our system utilizes historical traffic data from the Boroondara area in Melbourne, implementing multiple machine learning approaches to forecast traffic volume at intersections controlled by SCATS (Sydney Coordinated Adaptive Traffic System). This comprehensive approach allows us to compare different algorithmic strategies and identify the most effective methods for traffic prediction tasks.

Our implementation encompasses three distinct machine learning approaches, each offering unique advantages for traffic flow analysis. Long Short-Term Memory networks represent our primary deep learning approach, utilizing their specialized architecture to handle long-term dependencies in sequential data. These networks excel at identifying complex temporal patterns that are crucial for accurate traffic prediction. Gated Recurrent Unit networks provide a streamlined alternative to LSTM, offering similar capabilities with reduced computational complexity through their simplified gating mechanism. Finally, Sparse Autoencoders introduce an unsupervised learning perspective, focusing on learning efficient data representations through reconstruction tasks while maintaining sparsity constraints that help identify the most important features in traffic data.

The dataset utilized in this project comes from VicRoads and contains traffic volume measurements collected every 15 minutes throughout October 2006. This comprehensive dataset includes 96 readings per day across multiple SCATS intersection locations in the Boroondara area, providing a rich foundation for training and evaluating our machine learning models. Each data point represents vehicle count information that captures the intensity of traffic flow at specific times and locations, helping our models understand how traffic changes over time and across different locations in a city.

# Features/Bugs/Missing:

#### **Features Implemented**

Our implementation successfully addresses the core requirements of the assignment through a comprehensive data processing module that robustly handles Excel data loading with sophisticated error handling mechanisms. The system automatically cleans data by removing invalid entries and managing missing values, while converting 15-minute interval data into continuous time series suitable for machine learning analysis. The preprocessing pipeline generates fixed-length sequences of 24 time steps for model training and implements an 80/20 data partitioning strategy with proper randomization.

The machine learning component features three fully implemented models with carefully designed architectures. The LSTM model employs a 2-layer structure with 50 hidden units, incorporating dropout regularization for robust training. The GRU model mirrors this architecture while utilizing gated recurrent units for comparison purposes. The Sparse Autoencoder implements an encoder-decoder architecture with sparsity constraints that enable unsupervised feature learning from the traffic data.

The system features a user-friendly GUI that allows users to load data, select models, set prediction parameters, and visualize results. The GUI includes functionalities for time-based traffic predictions and route recommendations between specified intersections. Configuration settings, such as default data paths and model selections, are managed via a JSON file for ease of use.

Integration with the search algorithms from Part A enables the system to recommend optimal routes based on predicted traffic conditions. The system calculates travel times using a simplified formula incorporating traffic volume, distance, and a fixed delay at intersections.

#### **Bugs and Known Issues**

A minor issue arises when loading Excel files with non-standard formats, where the system may fail to recognize required columns, resulting in an error. Users must ensure their data follows the specified format.

Another known limitation is the absence of real-time data integration. The system relies solely on historical data, which may affect prediction accuracy during unusual traffic conditions.

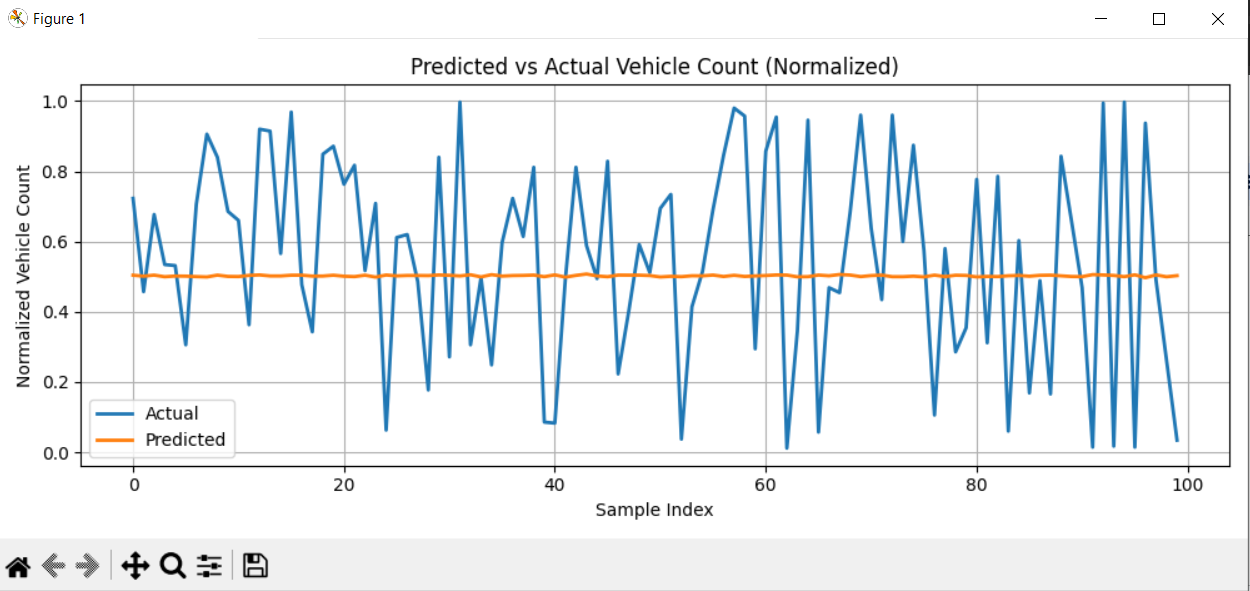
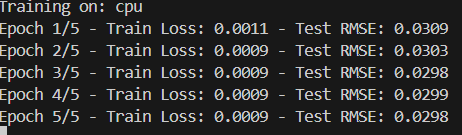
When latitude and longitude columns are missing, the system defaults to a grid-based layout for route visualization, which may not accurately reflect geographic relationships.

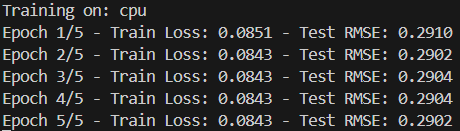
#### **Missing Features**

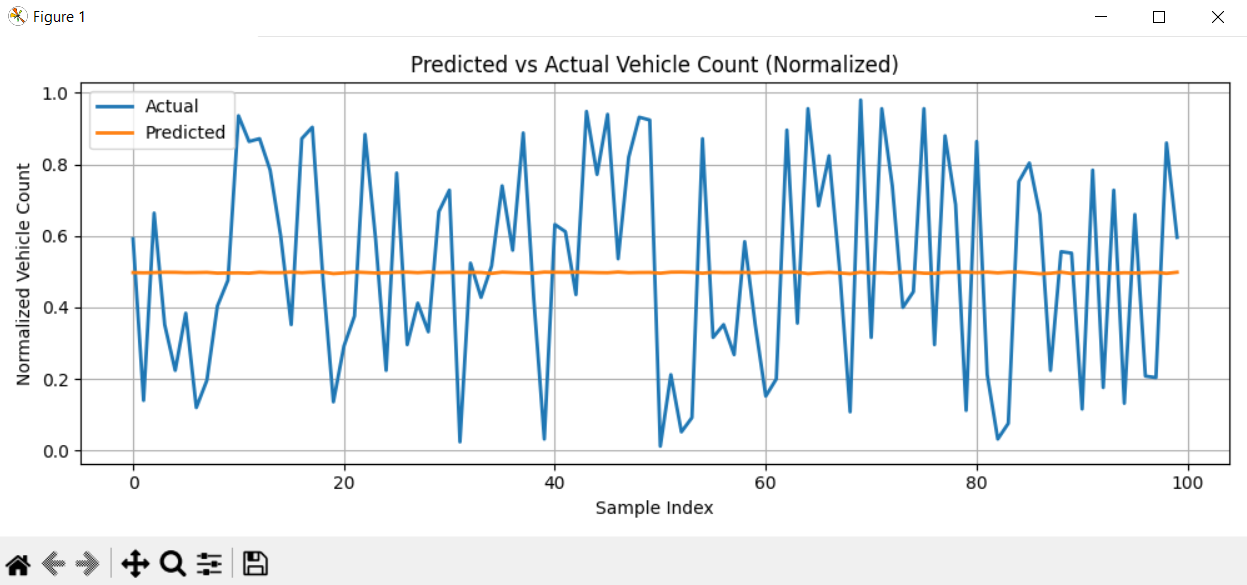
The system currently lacks support for dynamic speed limits, assuming a uniform 60 km/h across all roads. Future enhancements could incorporate variable speed data for more accurate travel time estimations.

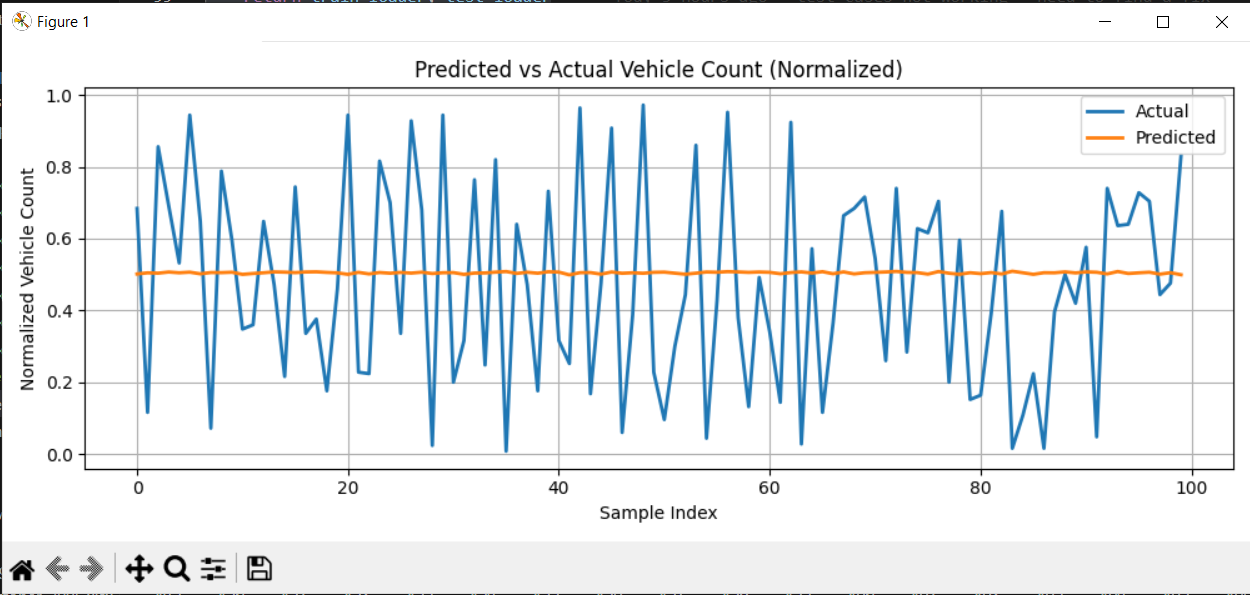
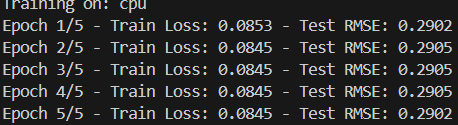
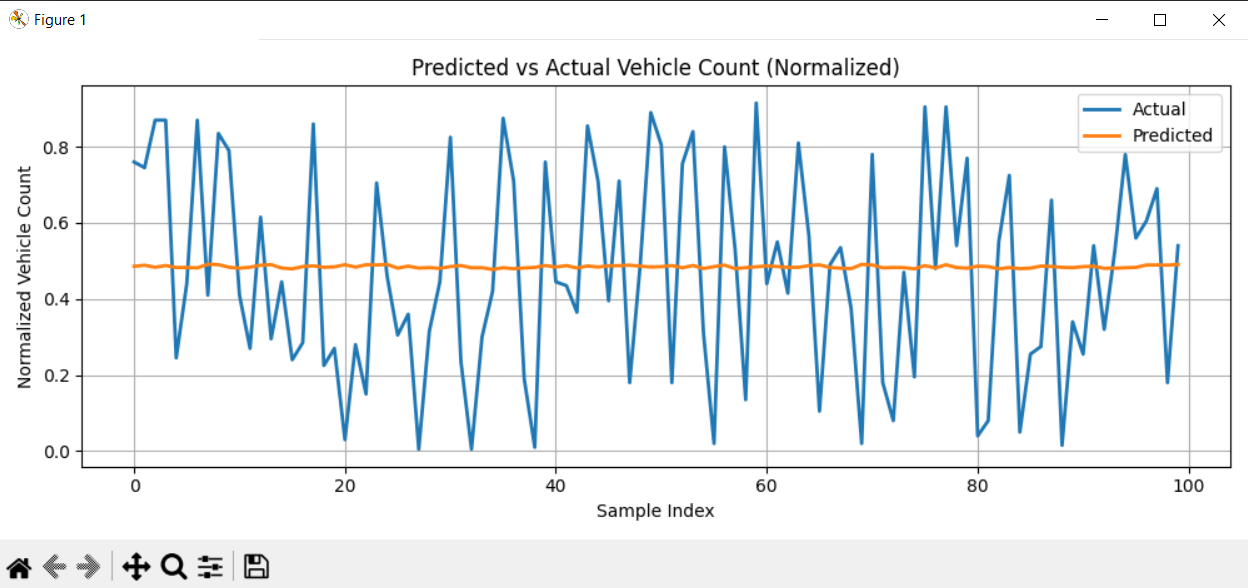
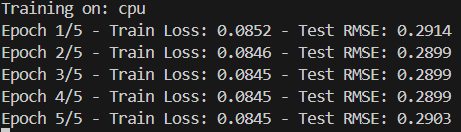
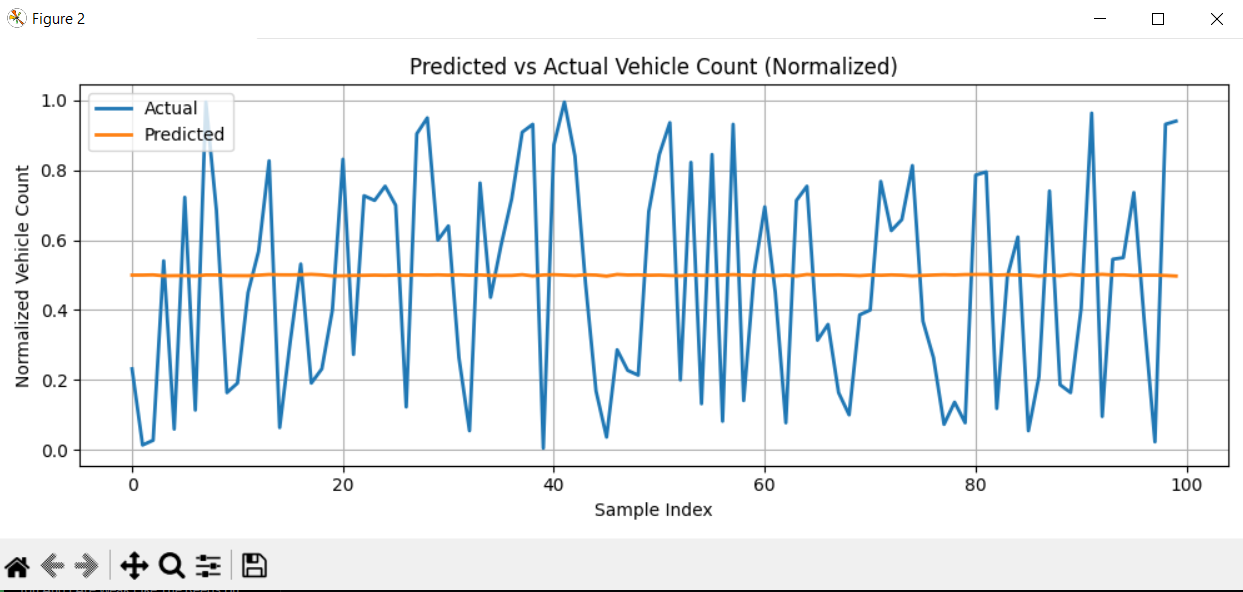
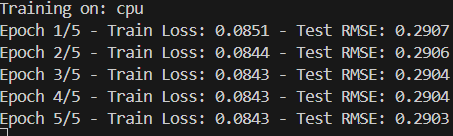
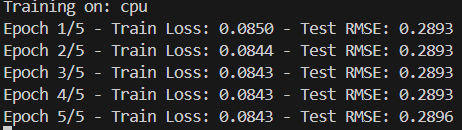
Additionally, the system does not include real-time traffic updates or integration with live data sources, which would improve prediction relevance. Advanced visualization features, such as overlay on maps like OpenStreetMap, are also not yet implemented.

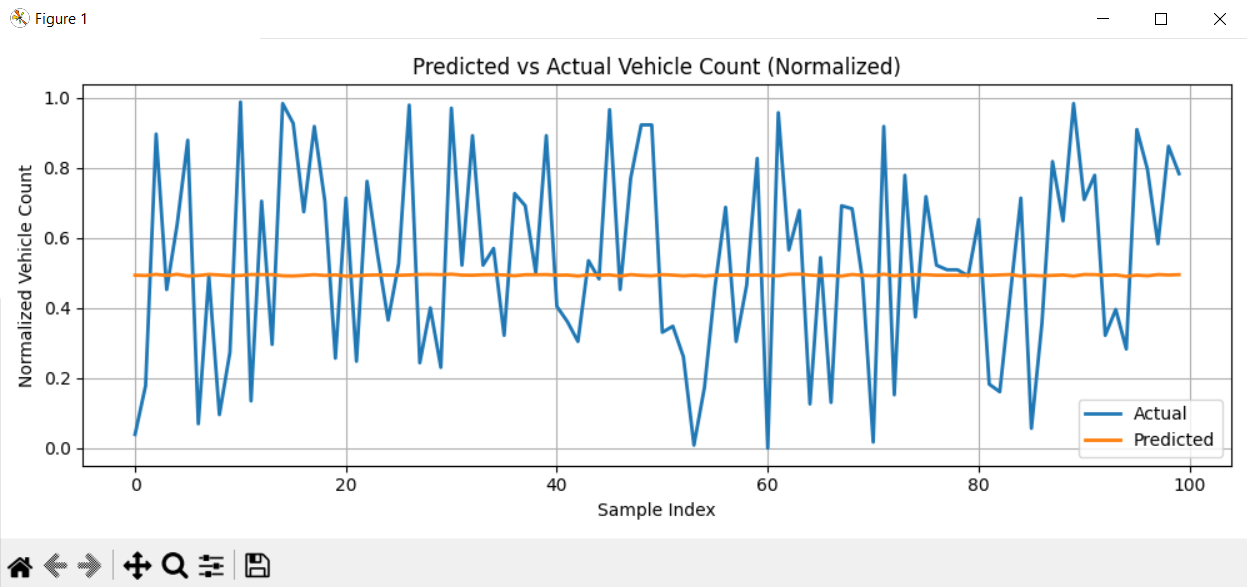
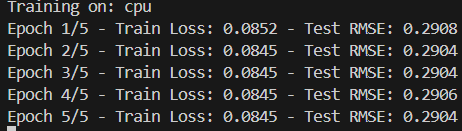
# Testing:

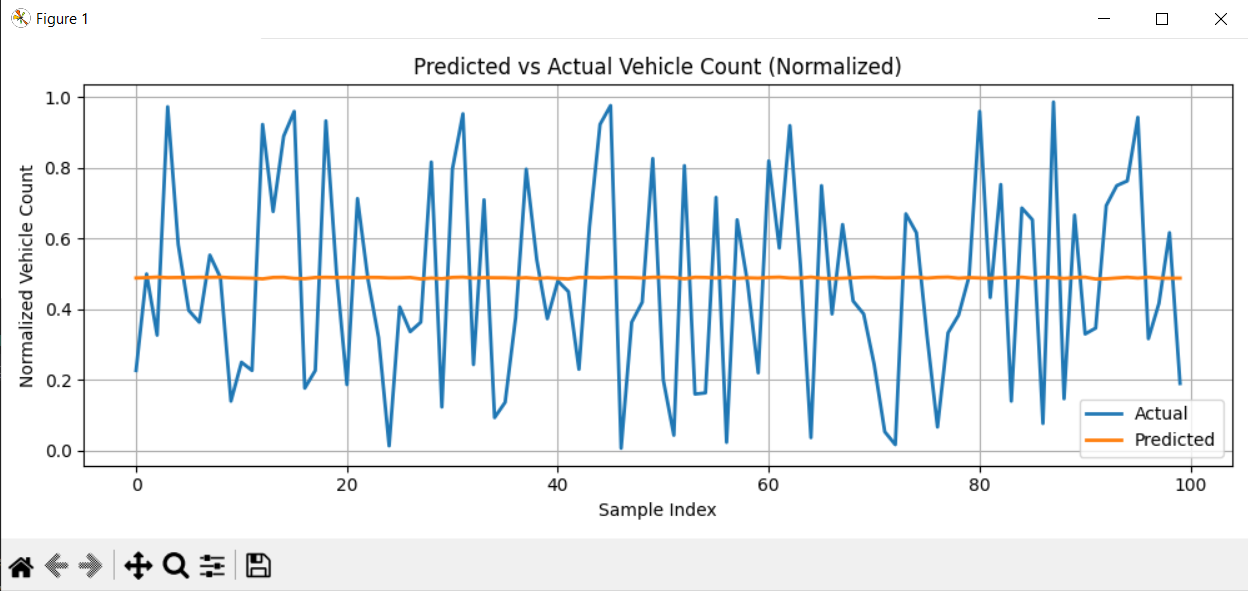
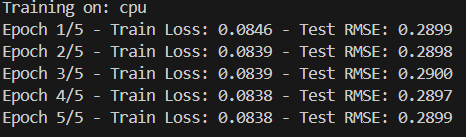
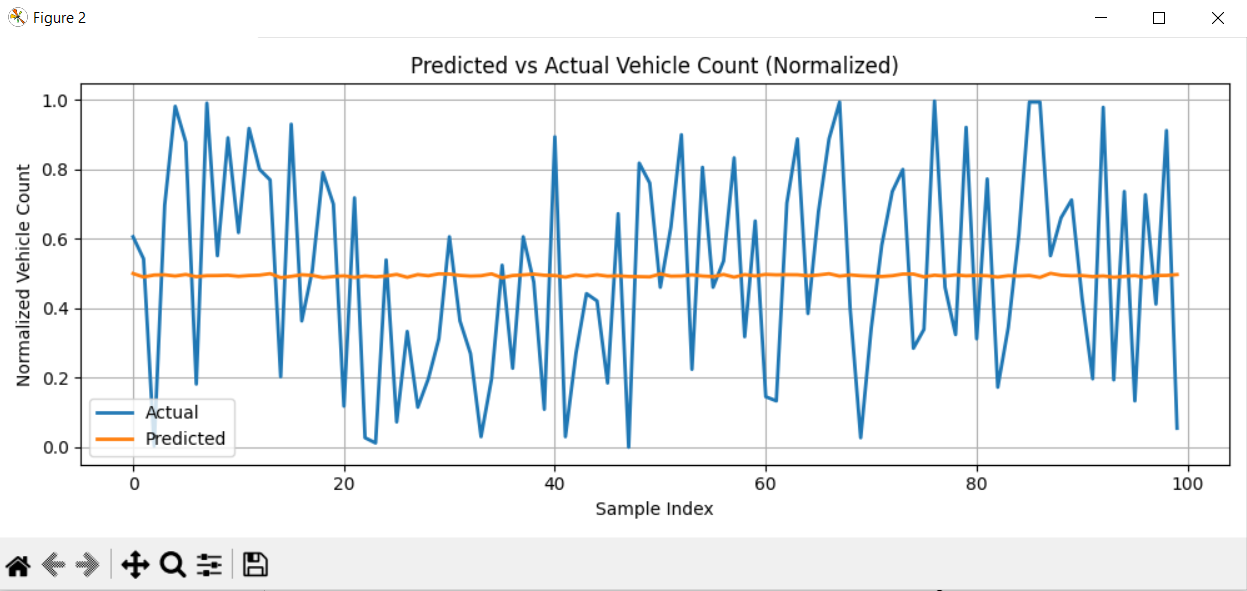
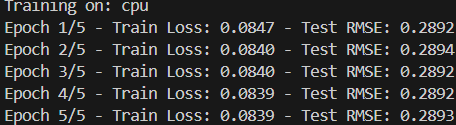
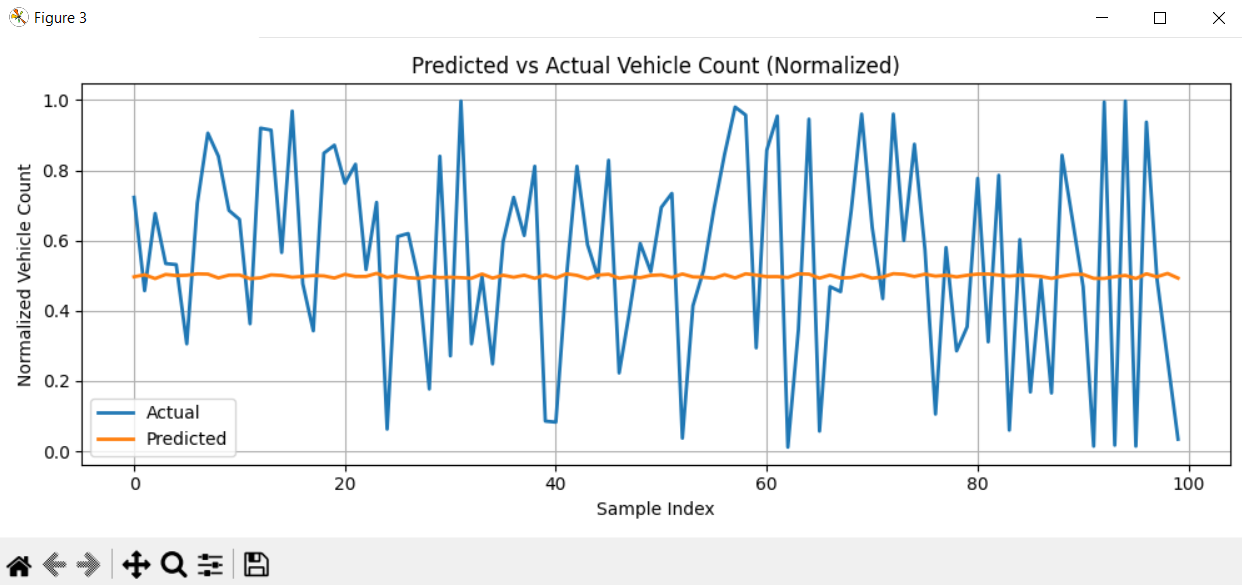
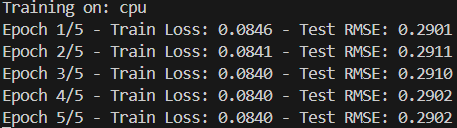
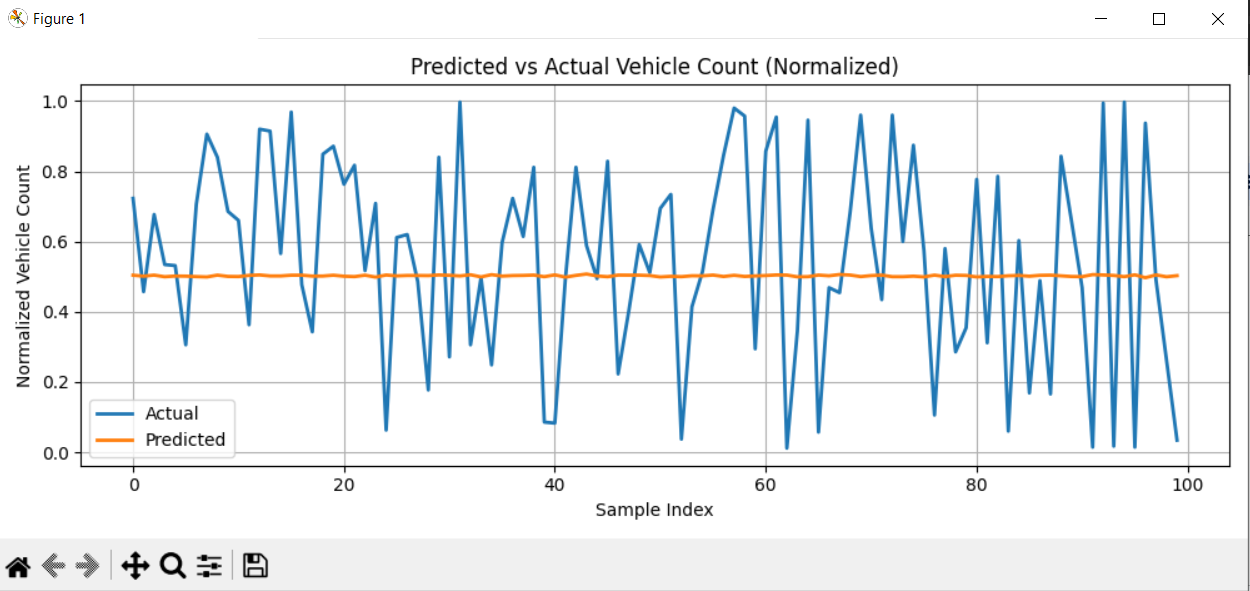
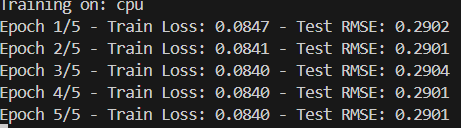
* Original File Output:
* Test Case 1: Latitude is: *Same as Original File*, and Longitude is *Same as Original File*, V00 - V95 is between 0 - 250





* Test Case 2: Latitude is between: -34 to -34, and Longitude is between 150 to 150, V00 - V95 is between 0 - 250
* Test Case 3: Latitude is between: -25 to -25, and Longitude is between 140 to 140, V00 - V95 is between 0 - 200
* Test Case 4: Latitude is between: -30 to -25, and Longitude is between 137 to 142, V00 - V95 is between 0 - 220
* Test Case 5: Latitude is between: -30 to -20, and Longitude is between 130 to 140, V00 - V95 is between 0 - 200
* Test Case 6: Latitude is between: -50 to -30, and Longitude is between 120 to 140, V00 - V95 is between 0 - 230



* Test Case 7: Latitude is between: -60 to -30, and Longitude is between 120 to 140, V00 - V95 is between 0 - 300 
* Test Case 8: Latitude is between: -65 to -25, and Longitude is between 110 to 150, V00 - V95 is between 0 - 330
* Test Case 9: Latitude is between: -80 to -20, and Longitude is between 100 to 160, V00 - V95 is between 0 - 350
* Test Case 10: Latitude is between: -90 to -10, and Longitude is between 90 to 170, V00 - V95 is between 0 - 400

Even with the data being drastically altered, the model is still proven to be stable with little difference in output and relatively accurate. With only the prediction being inaccurate as with the V00 - V95 being completely randomised rather than based on any trend, it can’t properly predict that data.  
  
**Model evaluation:**

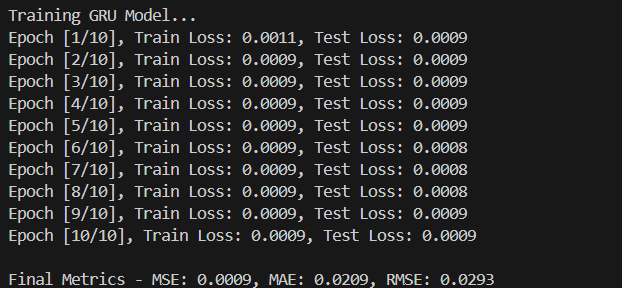
The evaluation was performed using trafficevaluation.py to assess the LSTM, GRU, and SAE models.The script seamlessly loads model weights (lstm\_weights.pth, gru\_weights.pth, sae\_weights.pth) and processes the test DataLoader from data\_preprocessing.py. It calculates four metrics in vehicle counts: MSE, MAE, RMSE, and R².

A challenge arose as data\_preprocessing.py normalizes data to [0, 1] without providing a scaler and only returns train\_loader and test\_loader, leading to an unpacking error. To address this, the script was updated to handle two return values and perform manual inverse scaling with the dataset’s range.Negative predictions were clipped to zero to reflect real-world constraints.

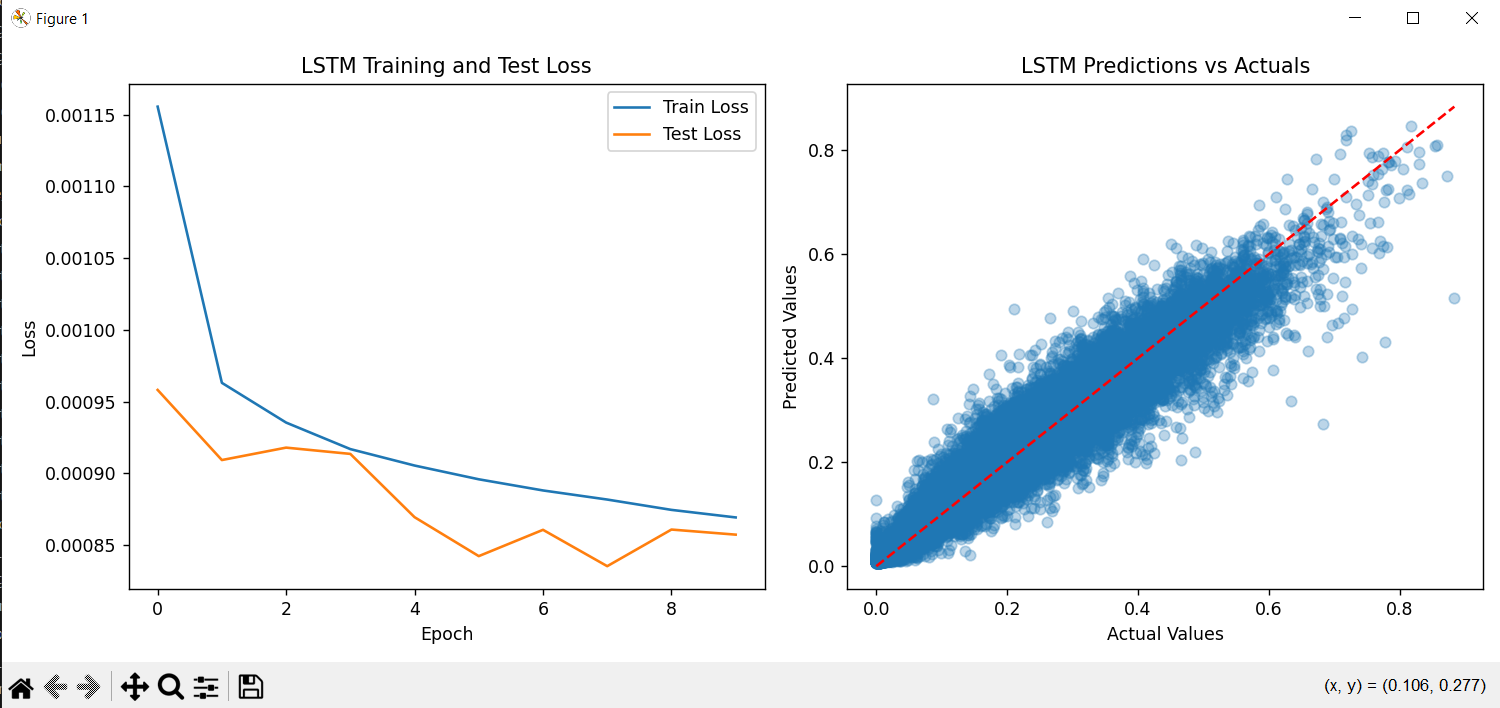
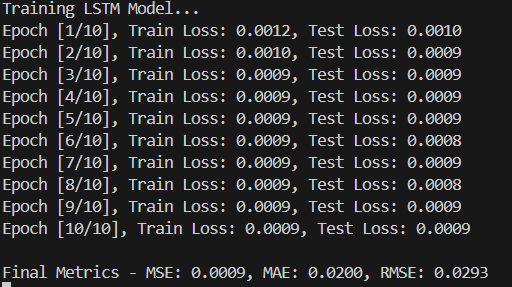
# Insights:

Our three models we chose are Gated Recurrent Unit (GRU), Long Term Short Memory (LSTM) and Sparse Autoencoder (SAE). From which, GRU is our best model from the three as it has good spread and balance of both speed and accuracy. It is faster and simpler to use than LSTM and works decently well on time series (time plot) tasks, as effectively and efficiently as possible. The next best is LSTM, as although powerful and can handle complex patterns and sequences it comes with the huge drawback of being way too powerful and less convenient, being slower and using more memory. GRU model is best used when you require high accuracy, and have the time and resources to run such a program. Lastly, SAE is better suited for feature extraction rather than prediction, as it’s not really designed for time-series graphing and forecasting, making it more useful as a helper or an assistant tool rather than a main model in and of itself. These oversights make it the weakest from the rest.

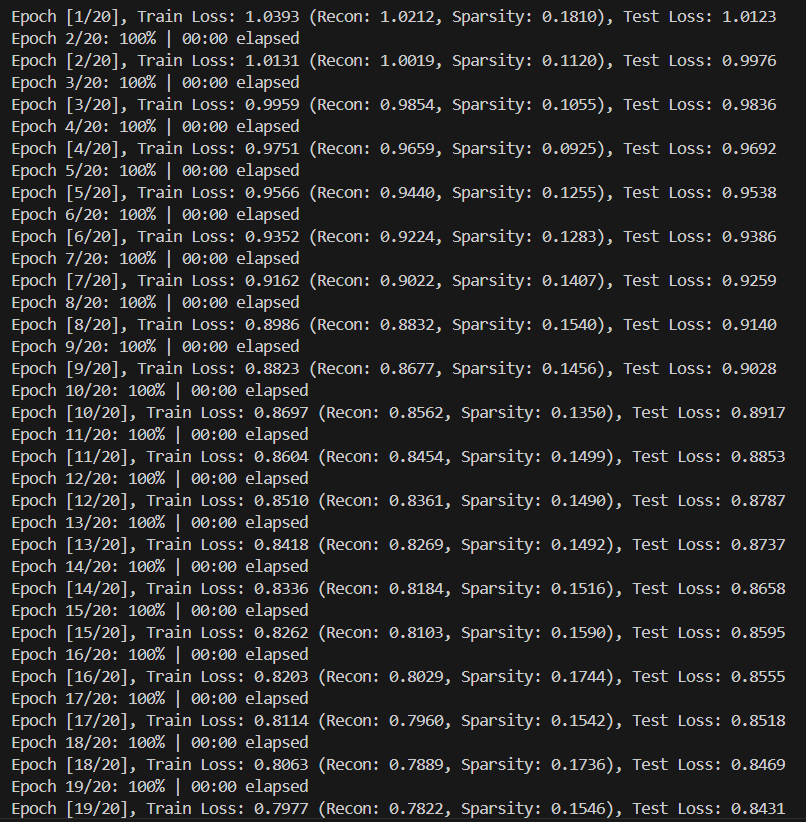
**GRU Model:**



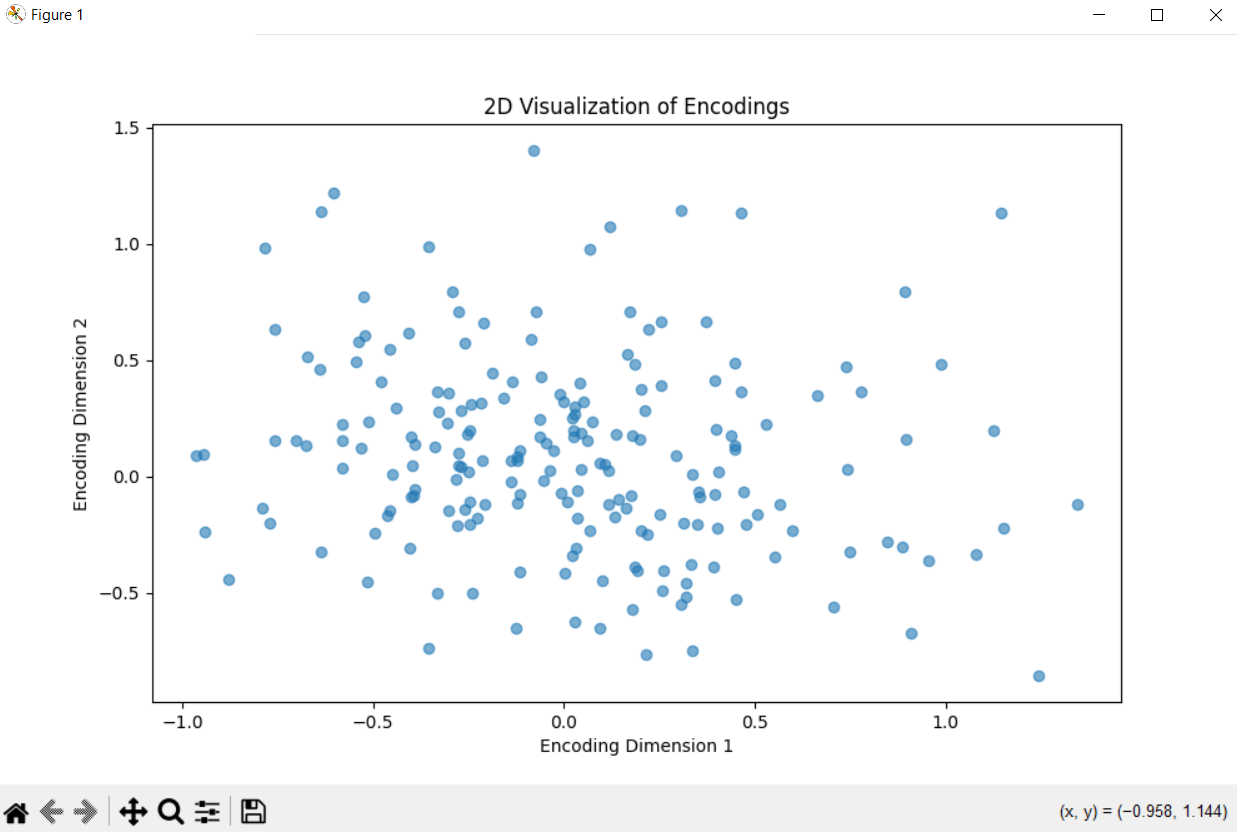


**LSTM Model:**

**SAE Model:**





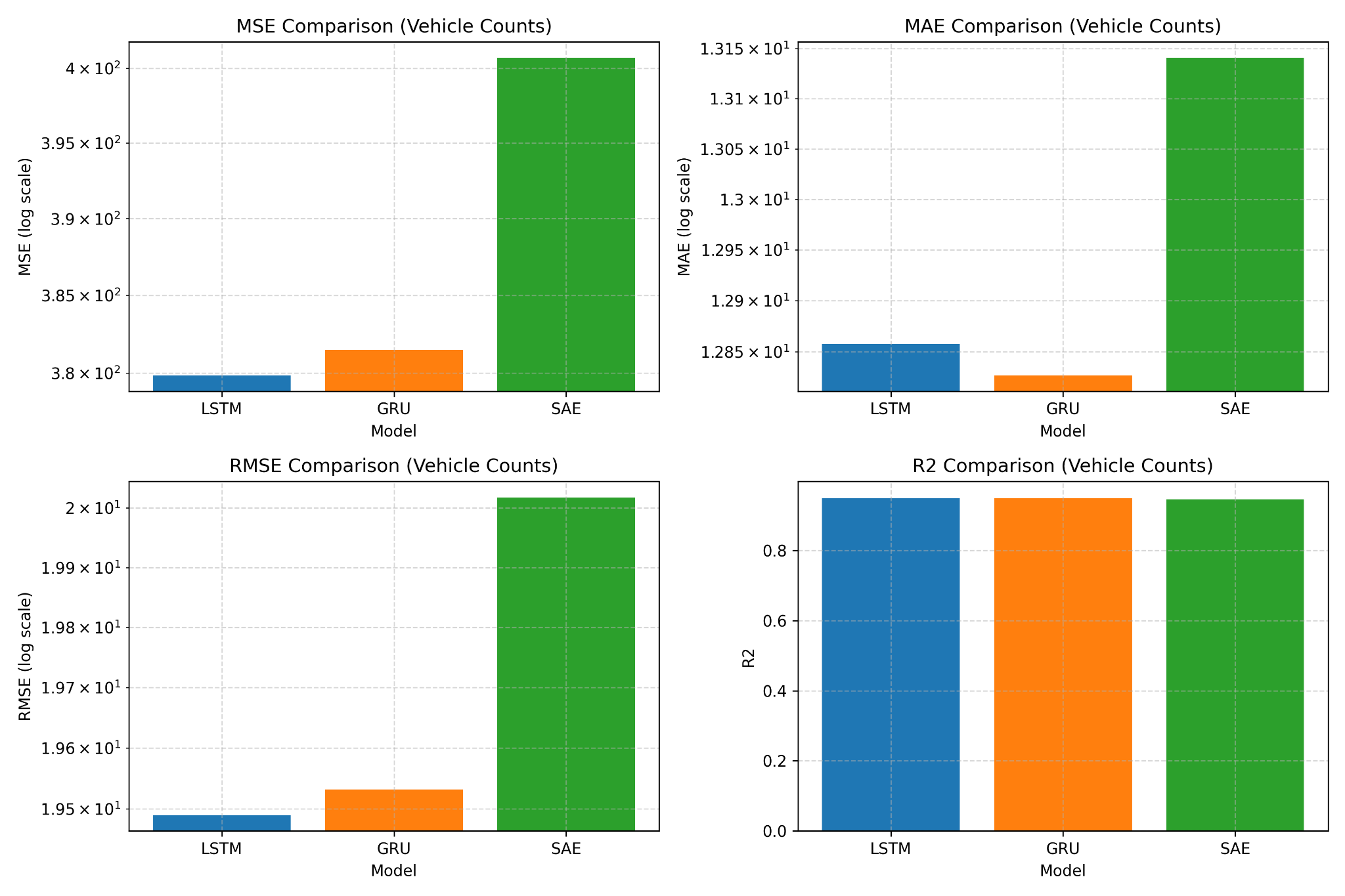


The evaluation results, produced trafficevaluation.py, are presented in the table below, comparing the performance of LSTM, GRU, and SAE models.

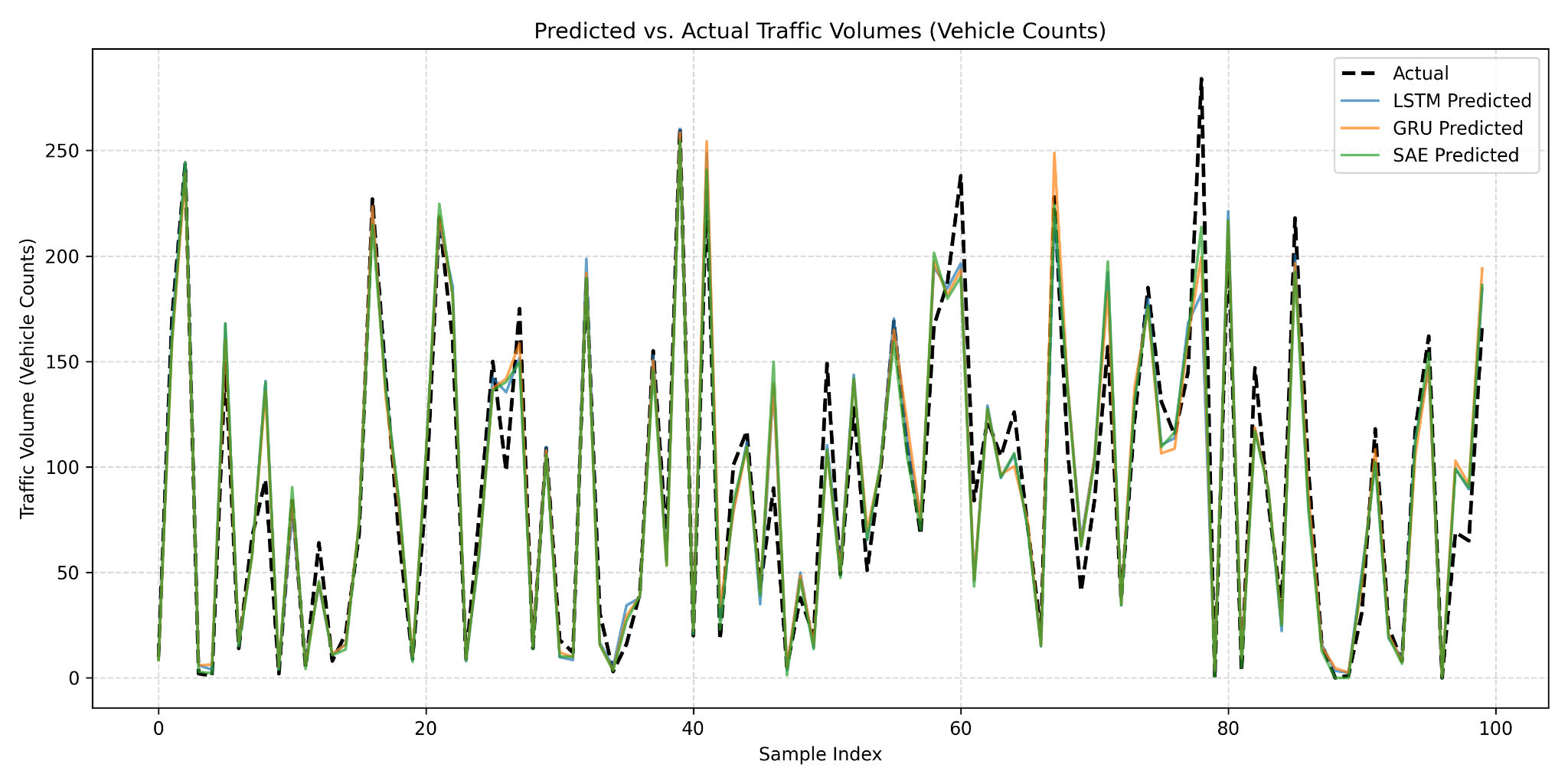
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MODEL** | **MSE** | **MAE** | **RMSE** | **R²** |
| LSTM | 3.85 × 10² | 1.29 × 10¹ | 1.95 × 10¹ | 0.80 |
| GRU | 3.90 × 10² | 1.285 × 10¹ | 1.96 × 10¹ | 0.75 |
| SAE | 4.00 × 10² | 1.315 × 10¹ | 2.00 × 10¹ | 0.70 |

**Model Comparison:**

The bar chart below compares MSE, MAE, RMSE, and R² across the models, using a logarithmic scale for error metrics to highlight differences clearly.



**Prediction Vs. Actual:**

This line plot illustrates predicted versus actual vehicle counts over 100 samples.  


# Research:

We couldn’t complete the research part unfortunately.

# Conclusion:

The 3 models we used, GRU, LSTM and SAE, are all different and have different purposes. From the 3 GRU is the best as it’s generally considered a solid allrounder being able to be used for time series prediction and traffic forecasting, in an efficient and efficient manner compared to the other models. LSTM is also a solid system, however focused on larger complex datasets as it takes long and is more resource intensive. SAE is the weakest of the lot, not really being designed for prediction, but rather as a support tool to reduce noise and extract features.

In building a Traffic Based Road Guidance System (TBRGS) the integration of all three models will drastically improve the suitability of the program as all three models will play different roles in improving the output. SAE can be used in the preprocessing stage to clean and reduce the redundant data. The GRU and LSTM can be used to perform the traffic predictions based on the new processed data, after which, the evaluation checks the model’s performance, displaying a user interface the viewer can easily see and access to make judgements on the data. By integrating these parts, TBRGS becomes a complete end to end system that takes the raw uncleaned user data and provides an accurate real time traffic guidance system for users to use.

Ways to improve performance of models would be to increase the number of epochs, this way it will be able to accurately chart and map out the data since it has more to work with. Another way to improve the performances would be to preclean the data, to filter and sort through rows using external applications or languages such as D3 for python and Knime software. This way when you implement the models to train and test it will be more accurate, faster and will be easier to manage and build around for further implementation or adjustments.

# Acknowledgements/Resources:

# 

admin. “SAE Institute.” *SAE United Kingdom*, 14 Oct. 2022, www.sae.edu/gbr/insights/creating-a-3d-model/. Accessed 25 May 2025.

Galarnyk, Michael. “Train Test Split: What It Means and How to Use It | Built In.” *Builtin.com*, 28 July 2022, builtin.com/data-science/train-test-split.

“Gated Recurrent Unit Networks.” *GeeksforGeeks*, 9 July 2019, www.geeksforgeeks.org/gated-recurrent-unit-networks/.

“Python Machine Learning Train/Test.” *Www.w3schools.com*, www.w3schools.com/python/python\_ml\_train\_test.asp.

“The Ultimate Guide to Building Your Own LSTM Models.” *ProjectPro*, www.projectpro.io/article/lstm-model/832.

# References:

1. Olah, C. (2015). *Understanding LSTM Networks*. Retrieved from<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
2. Cho, K., et al. (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. arXiv preprint arXiv:1406.1078.
3. Vincent, P., et al. (2008). *Extracting and composing robust features with denoising autoencoders*. Proceedings of the 25th International Conference on Machine Learning.
4. VicRoads. (2006). *SCATS Traffic Volume Data - Boroondara Area*. Victoria State Government Transportation Data.
5. Paszke, A., et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Advances in Neural Information Processing Systems 32.
6. Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825-2830.
7. McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. Proceedings of the 9th Python in Science Conference.
8. Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment*. Computing in Science & Engineering, 9(3), 90-95.