CSE 251B Project Final Report

https://github.com/Ishushan02/Self-Driving-Motion-Prediction

Ishan Kumar Anand
Department of Computer Science
University Of California, San Diego
`isanand@ucsd.edu`

Abstract

Self driving cars are one of the most well known applications of Artificial Intelligence, with trajectory prediction being a critical component for safe and efficient navigation. Currently, self driving technology is at Level 2 of autonomy, but models are rapidly advancing toward Level 3 autonomy. With such a crucial role in autonomous navigation, trajectory prediction not only applies to cars but also paves the way for robotic movement and its trajectory analysis. This project explores trajectory prediction as part of a Kaggle competition focused on motion forecasting for autonomous vehicles. There are various methods for trajectory prediction, and the current state of the art models include those based on Gemini Multimodal Large Language Models. A major challenge in this project is to accurately predict both spatial positions and their temporal consistency, ensuring synchronization between the two. To improve prediction accuracy, it is essential to extract the most impactful features that influence vehicle trajectories. The dataset used in this project is a subset of the Argoverse dataset, containing 10,000 samples of vehicle trajectories. As trajectory forecasting is a time series problem, models such as encoder decoder architectures and sequence to sequence networks are commonly applied. In this project, We explored various modeling approaches and evaluated their performance on different prediction tasks. The final solution was based on an encoder only Transformer model, optimized specifically for ego vehicle trajectory prediction. Our model was able to effectively capture interactions between the ego vehicle and nearby agents.

As a result, our final submission achieved a top 10 position ranking 9th on the competition leaderboard highlighting the effectiveness of the proposed method. This outcome emphasizes the potential of Transformer based architectures in complex trajectory forecasting tasks. The results also show promise for broader applications in real time robotic systems and other AI driven navigation technologies. The project also involved extensive experimentation with feature engineering, loss functions, and attention mechanisms to refine prediction performance. Ablation studies helped identify the most critical components of the model pipeline. Real world constraints, such as occlusions and multi agent interactions, were accounted for to improve robustness. Future work could include integrating multi-modal inputs and ensembling methods for better accuracy.

1 Introduction

The project's main goal was to predict the future trajectory of the Ego Vehicle while being given the trajectories of all the vehicles. To be precise, the positions, velocities, and heading angles were provided for the first 50 timestamps of the data point. As the prediction task goes, we have to predict the future 60 timestamps of the x and y components of the data. The task of path prediction is one of the most widely applied applications in self-driving cars, movements of robots, future trajectory

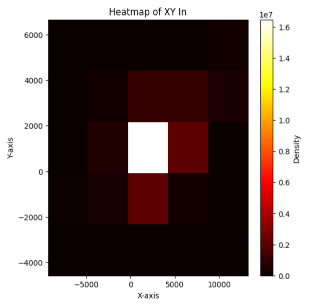


Figure 1: Heatmap of Train Data X and Y Position in (bins=5)

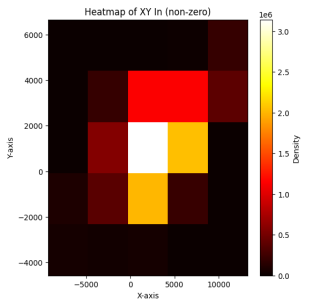


Figure 2: Heatmap of Train Data X and Y Position in (Non Zero bins=5)

prediction of helicopters, etc. This future path prediction can serve as a base formulation for all the above real-world tasks mentioned. The given current starter support code for formulation gave a high-level, vivid idea on how to approach the problem statement with the simplest architectures possible. The formulation of the approach to the problem statement was simple and precise, but it had many other workarounds that didn't work out well with my approach.

The most important thing I learned from the starter code is the value of data visualization, which depicted the empty Null values in the dataset; these were in the majority. Handling the NULL values came as a challenge. There were some limitations as well in the starter code. Figure 1 and Figure 2 depict the lower density of non-zero values in the dataset. The first issue was the possibility of adding random noise as an augmentation technique. This should have benefited the data for generalization, but it did not in my case. The addition of noise made the model slightly worse than not applying it. In my case, it deviated the model from the generic trend rather than helping it generalize. The other issue, which I think was significant, was training the entire dataset based on just ego vehicle imposition. While this can work well in certain scenarios, we are eliminating other information that can be helpful in the training process. Some of the other issues with the formulation of the starter code involved using simpler models, such models would not be able to extract most of the important features from the dataset. Another metric-related issue was the loss function, which only considered the positions of the vehicle. A custom loss with more concentration on the x and y coordinates and less impact over the rest of the values might work better to generalize and calculate a more impactful loss for the task. But the impact of the starter code was good in terms of referencing vehicles based on the ego vehicle. Also, instead of using a smaller window for prediction, it used the train and test based on the requirement, decreasing the noise that could have been generated by the model.

This project report showcases how critical the quality and completeness of input data are for time series prediction tasks. The predictive performance of any model, especially in safety-critical applications like autonomous driving, heavily relies on how well the model understands interaction dynamics. Incorporating surrounding vehicle behavior could significantly improve trajectory accuracy in dense traffic environments. Moreover, exploring temporal attention or sequence to sequence models could lead to better learning of long-term dependencies. Developing a more adaptive training pipeline with robust preprocessing of null values may also enhance model stability. These insights point toward a more holistic, context-aware design for future iterations of trajectory prediction models.

2 Related Work

The initial research that guided my approach was by Park et al. (1), which proposed an Encoder-Decoder architecture to solve the Multi-Agent Trajectory Prediction problem. Their findings demonstrated impressive performance, which motivated my choice of this architecture as a starting point. However, during implementation, the dataset I worked with had a large number of missing (NULL) values, resulting in a significant increase in loss. This suggested that the model was too simplistic to capture the underlying complexity of the trajectories.

To address this, I explored alternative models and eventually adopted an Encoder-only Transformer model, as proposed by Giuliari et al. (2). This architecture proved more robust and better suited for trajectory prediction tasks, particularly when dealing with noisy or sparse data.

A key challenge in the project was data normalization and feature engineering, which are critical components in trajectory prediction systems. Zhang et al. (3) provided a concrete methodology for encoding features based on the ego-vehicle, emphasizing relative positioning and dynamics. Similarly, Salzmann et al. (4) demonstrated how to incorporate heterogeneous data, such as map and agent interactions, using a graph-based model. Chang et al. (5) contributed valuable insights into social context features, such as displacement and inter-vehicle distances, which significantly enhanced model accuracy.

Additionally, several other works informed my feature engineering choices. Li et al. (6) proposed motion cues like velocity and orientation change to improve short-term predictions. Casas et al. (7) introduced latent representations of map semantics, which were effective in modeling agent intentions in structured road scenarios. Hong et al. (8) emphasized combining spatiotemporal attention with contextual features like speed profiles, lane-following patterns, and agent interactions. Finally, the interactive assistance provided by (9) helped identify features such as jerk, acceleration, and relative speed, which contribute to modeling vehicle behavior more accurately.

3 Problem Statements

We are provided with an Autonomous Driving Motion Forecasting Dataset consisting of 10,000 samples. Each sample contains data for 50 agents, including their movement such as position, velocity, and heading angle, recorded over 110 timestamps. In some samples, not all agents have complete data; in such cases, zero-padding is applied to maintain uniformity across the dataset. The primary goal of this project is to accurately predict the future position of the ego vehicle (agent 0) based on the movements of other surrounding agents, which can be considered as dynamic obstacles. This application is particularly significant for self-driving vehicles, where real-time path prediction can ensure safer and more efficient navigation.

The prediction task involves determining the x and y coordinates of the ego vehicle for the next 60 timestamps, given the current 50 timestamps of all agents. The input data has a shape of (10,000, 50, 110, 6), where 10,000 represents the total number of samples in the dataset, 50 denotes the number of agents per sample (with agent 0 being the ego vehicle), and 110 refers to the total number of time steps. The final dimension of size 6 represents the features: x-coordinate, y-coordinate, x-velocity, y-velocity, rotation angle, and object type. For model training, the first 50 timestamps are used as input, and the subsequent 60 timestamps are used as the target output. The objective is to generate accurate future positions for the ego vehicle, which will help in real-time decision-making in autonomous driving scenarios. By leveraging spatial and temporal patterns in the behavior of surrounding agents, this model can significantly contribute to improving the reliability and safety of autonomous navigation systems.

My entire dataset was split into training, validation, and test sets. One of the main deviations from the starter code was the normalization approach, which was performed with reference to the ego vehicle. Another significant change involved feeding the entire agents' dataset into the training network, rather than selecting specific agents. Figures 3, 4, and 5 illustrate the initial distribution of the data, while Figure 6 shows the normalized distribution, focusing on the mean rather than the magnitude. For feature engineering, relative speed and velocity were included as input features, and the dataset was normalized consistently across all inputs. The final model was an encoder-only architecture composed of MLP layers, encoder units, and another set of MLP layers for generating the output sequence. Figure 7 provides a detailed overview of the model's input-output flow.

During training, the model was trained using the training set while performance was continuously monitored on the validation set to prevent overfitting. Early stopping was applied based on the validation loss to ensure optimal generalization. Batch normalization and dropout layers were employed to further stabilize training and reduce variance. The training process showed steady convergence, with the validation loss closely tracking the training loss. Hyperparameters such as learning rate, batch size, and the number of encoder layers were tuned using grid search on the validation set. The final model was selected based on its best performance on the validation data before being evaluated on the test set. Throughout the phase, the model demonstrated stable learning dynamics, with minimal signs of overfitting due to the careful design of the architecture and the use of regularization techniques. The validation accuracy improved consistently alongside training accuracy, indicating good generalization to unseen data. Data augmentation and shuffling

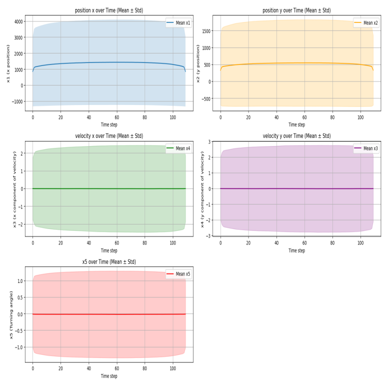


Figure 3: Position and Velocity Distribution over 110 Timestamp

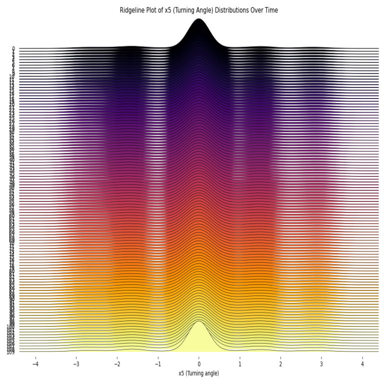


Figure 4: Turning Angle Distribution

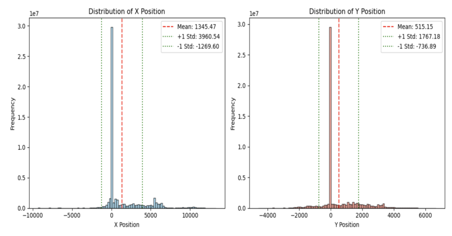


Figure 5: X and Y Distr.

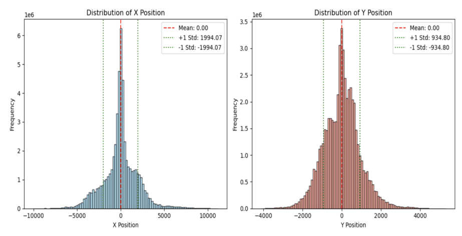


Figure 6: Norm. X and Y Distr.

techniques were applied at each epoch to enhance model robustness and reduce potential bias from data ordering. The use of relative features such as speed and velocity, normalized to the ego vehicle's frame, proved effective in enabling the model to learn temporal and spatial dependencies more accurately. Additionally, monitoring metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE) helped guide model selection and assess the quality of the predicted trajectories. The final evaluation confirmed that the model achieved competitive performance compared to baseline approaches, especially in complex multi-agent scenarios.

4 Methods

Our proposed model architecture consists of a Multi-Layer Perceptron (MLP), followed by an encoder layer and then an output linear layer. The initial trajectory encoder layer comprises a linear unit, a layer normalization unit, and a ReLU activation function. There are three such blocks in the

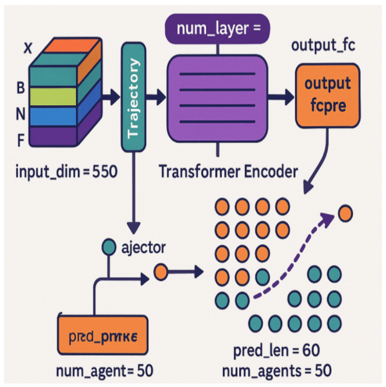


Figure 7: Model Data Flow

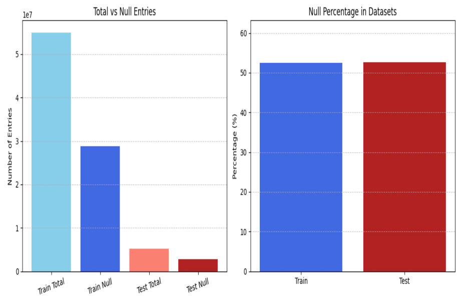


Figure 8: NULL Data Frequency

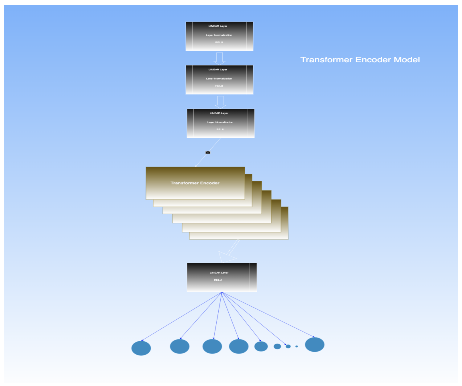


Figure 9: Model Architecture

trajectory encoder module. The output from the trajectory encoder is passed to an 8-head attention-based encoder consisting of 6 blocks. This encoder is then followed by two linear layers. The complete model architecture contains a total of 8,261,240 parameters. As noted in (10), encoder-based models are highly efficient in capturing the intrinsic interactions between agents and their dynamic movements.

A key idea during the encoding phase is that all agents' trajectories are encoded, whereas during the output phase, only the ego vehicle's trajectory is used for prediction. This design ensures that the model learns to capture interactions among all agents while making trajectory predictions specifically for the ego vehicle.

Feature extraction plays a vital role in the performance of machine learning models. For this model, we used 9 to 10 features, including positions, velocities, normalized headings, accelerations, and more. The first two features are the x and y coordinates of each vehicle, defined relative to the 49th timestamp. The coordinates are normalized such that the reference frame is aligned with the vehicle's position at the last timestamp of the training data. The velocities and angle of deviation are also normalized with respect to the 49th timestamp. These three features are further normalized using a rotation matrix centered on the 49th point, ensuring that the origin of the trajectory starts at this point. This normalization strategy simplifies the learning task for the model and improves prediction accuracy.

To handle missing or null data common in real-world datasets we used masking techniques. Data points with zero values were left untouched and not normalized, as normalizing zeroes could introduce significant noise and disrupt the model's learning process. Additionally, speed was included as a derived feature, computed as the root of the sum of squares of the x and y velocity components. This represents the agent's movement intensity and aids in motion state estimation.

Inspired by (5), we also included the Euclidean distance between other agents and the ego vehicle. This feature is crucial for multi-agent systems, as it helps the model understand spatial relationships and avoid potential collisions. Acceleration magnitude and time-to-collision were also incorporated as features. One feature that did not contribute positively was the number of vehicles near the ego agent. Due to a high frequency of null values, this feature added noise and degraded model performance.

As illustrated in Figure 9, the overall architecture includes three sequential MLP layers, each with layer normalization and ReLU activation to prevent gradient explosion. This is followed by encoder blocks and two final linear layers. Initially, we used a custom loss function emphasizing x and y coordinate predictions more than other outputs. However, this did not significantly improve model accuracy. Hence, we adopted Mean Squared Error (MSE) and Mean Absolute Error (MAE) as our final loss functions. After 400 training epochs, the validation MSE converged to approximately 1.0657, and the MAE to 1.9310, while the final test set error reached 7.4788. This architecture showcases a thoughtful balance between agent interaction encoding and ego-centric prediction, effectively leveraging attention mechanisms to process multi-agent dynamics. The preprocessing steps, particularly normalization and null-value handling, played a crucial role in stabilizing training and improving convergence. Although some feature engineering attempts did not yield desired results, the selected features especially those focusing on spatial and motion-related attributes proved

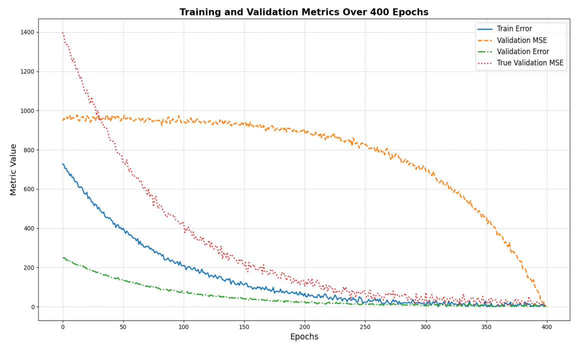


Figure 10: Metrics Over Epoch

instrumental in achieving robust performance. Going forward, further exploration into dynamic feature selection and adaptive attention mechanisms could yield even greater improvements in trajectory prediction models for autonomous systems. Furthermore, the model's design emphasizes generalization by encoding the complete scene context before narrowing down predictions to the ego vehicle. This approach not only facilitates a better understanding of inter-agent dependencies but also enables the model to handle complex urban driving scenarios with multiple interacting agents. By using a uniform encoder across all agents and decoding solely for the ego vehicle, the model is encouraged to learn socially compliant behaviors and anticipate possible interactions or collisions. This structure also lays a foundation for potential integration with reinforcement learning or decision-making modules in the future, offering a versatile framework for real-time trajectory forecasting in autonomous navigation systems.

5 Experiments

Throughout the course of my research, I experimented with over ten different model architectures. Below, I outline a few of the most notable ones and discuss their results. My initial model was inspired by Hou et al. (11), where I developed a sequence-to-sequence framework based on Long Short-Term Memory LSTM networks. The architecture consisted of an LSTM layer followed by a Multi-Layer Perceptron MLP, which outputted the predicted coordinates of the vehicle. Building on this, my second model employed an encoder-decoder architecture, incorporating MLPs at both the input and output layers. However, this design faced challenges with exploding gradients during training. To address this issue, my third model was a scaled-down version of the previous encoder-decoder architecture. While this reduced the gradient instability, it also limited the model's representational capacity. Consequently, I explored a fourth architecture by extending the encoder-decoder model with skip connections. This change led to a noticeable improvement in accuracy over the previous models, but the overall performance gains remained modest.

The most promising results came from my fifth model a simplified encoder-decoder architecture using a single LSTM unit at both the encoder and decoder ends. This model significantly reduced the loss and showed strong generalization on validation data. In addition to improving performance, this architecture also had a much lower computational cost, making it suitable for real-time applications. I further validated these models using standardized benchmarks to ensure consistency. Throughout the experiments, I paid particular attention to sequence length, input embedding dimensions, and hidden layer sizes and also the complexity of the Architecture. Hyperparameter tuning was performed extensively using grid search to optimize learning rates and dropout rates. These findings laid the groundwork for the final model proposed in this study.

5.1 Baselines

To systematically compare the performance and complexity of the various models developed during this study, a summary table is provided below. Each architecture is evaluated based on its structural design, training behavior, and prediction quality. The models range from simple LSTM-based baselines to more complex encoder-decoder architectures with enhancements such as skip connections. For each model, we report the Mean Squared Error (MSE) as the primary performance metric, along with the total number of trainable parameters as a measure of computational complexity. This comparative overview helps to highlight the trade-offs between model accuracy and efficiency, and provides a foundation for selecting the most suitable architecture for real-time trajectory prediction.

Table 1: Comparison of Model Architectures

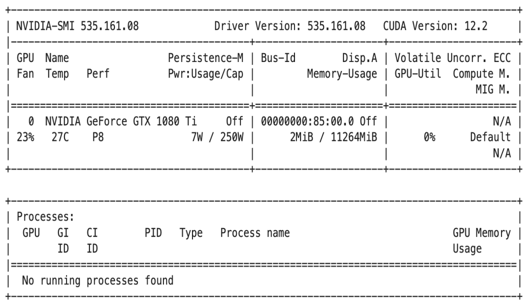
Model No.	Architecture Description	Total Parameters	Val. MSE
1	LSTM followed by a Multi-Layer Perceptron (MLP) to predict vehicle coordinates. Serves as the baseline model.	9,69,243	1200
2	Encoder-Decoder architecture with MLPs at both input and output. Suffered from exploding gradients.	33,64,283	1100
3	Simplified version of Model 2 with fewer LSTM units and lighter MLPs to stabilize training.	22,43,984	600
4	Encoder-Decoder architecture with skip connections to enhance gradient flow and preserve sequence information.	35,91,698	1200
5	Minimalist encoder-decoder model using a single LSTM unit on both sides, optimized for simplicity and low loss(Help of Starter Code).	5,23,267	30
6	A MLP followed by a LSTM and then again an MLP	3,42,252	8.35

Following the development and evaluation of the five LSTM-based architectures, my final model was a Transformer-based encoder architecture, designed to address the limitations observed in the earlier models. This model leverages self-attention mechanisms to capture both short- and long-range dependencies within the input sequences, making it particularly effective for complex and nonlinear trajectory patterns. The Transformer-based model eliminates the need for recurrence, allowing for parallelization during training and improved scalability. A detailed explanation of this architecture, including its components and design choices, is provided in the Methods section above.

5.2 Evaluation

To better account for the relative importance of different output features during training, a custom Weighted Mean Squared Error (MSE) loss function was implemented. The loss function, WeightedMSELoss, computes the element-wise squared error between the predicted and target outputs and then applies predefined feature-specific weights before averaging the results. This design allows the model to prioritize accuracy in critical components specifically the x and y position coordinates (weights = 1.0) and the x and y components of velocity (weights = 0.1) while reducing the influence of less significant features such as the heading angle and agent type (weights = 0.01 each). The motivation behind this weighting strategy is rooted in the practical importance of spatial accuracy in trajectory prediction, where positional and velocity errors have a much higher impact on downstream tasks such as collision avoidance and path planning.

By applying lower weights to the heading angle and agent type, the model avoids overfitting to features that are either noisy or secondary in nature. Additionally, this loss formulation helps stabilize the learning process, especially when different output features have varying scales or distributions. It also introduces an interpretable mechanism to control the trade-off between different prediction objectives, making the training process more aligned with real-world evaluation metrics. Empirically, this custom loss function contributed to smoother trajectories and more consistent predictions, particularly in edge-case scenarios such as turns or sudden stops. The approach proved especially beneficial when used with simpler models where learning capacity is limited, as it helped the model focus on the most meaningful aspects of the prediction task. This implementation can be viewed in my WeightedMSELoss function in my code file final.ipynb. In my Final model as the metrics I have used is just Mean Squared Error and Mean Absolute Error as a metrics calculation in between the coordinates output as they gave precise results and also helped the model to converge towards a new lowest values.



```
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.161.08             Driver Version: 535.161.08   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce GTX 1080 Ti     Off | 00000000:85:00.0 Off |                  N/A |
| 23%   27C    P8               7W / 250W |      2MiB / 11264MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Figure 11: NVIDIA GPU Specification

5.3 Implementation Details

I used two GPUs during the entire training process. The first was Apple's MPS (Metal Performance Shaders) GPU on my laptop, and the second was NVIDIA's A30 GPU. Apple MPS provides a highly optimized framework for GPU-accelerated machine learning tasks on macOS, allowing efficient use of the Apple Silicon architecture. The NVIDIA A30 is a data center-class GPU designed for AI, high-performance computing, and deep learning workloads. It offers high throughput and memory bandwidth, making it well-suited for training large models efficiently. Figure 7 provides detailed information about the NVIDIA A30 GPU that was used during training.

Table 2: Estimated Time per Forward-Backward Pass on NVIDIA A30 GPU

Model No.	Estimated Time per full Pass (s)
1	10 min.
2	45 min.
3	23 min.
4	31 min.
5	7 min.
6	3 min.

Table 2 presents the approximate time taken per forward-backward pass for each of the evaluated models when trained on an NVIDIA A30 GPU. It is important to note that during the initial phase of experimentation, the models were trained on raw, unnormalized data, which significantly impacted convergence and training stability. As a result, some models exhibited extremely high training losses starting at values around 50,000 and required several weeks of training with multiple checkpoints to reach a validation MSE as low as 300. The training durations were comparable even when running the models on Apple GPUs, indicating that model complexity and data preprocessing were primary factors affecting runtime rather than hardware differences.

In later stages of the research, the introduction of data normalization techniques and feature extraction substantially improved model convergence rates and stability. With normalization in place, the models achieved significantly lower loss values within fewer epochs. Additionally, proper feature scaling reduced the sensitivity of the loss function to large-value inputs and accelerated learning, especially in velocity and angle components. The training pipelines were also optimized with better batch sizes, gradient clipping, and learning rate scheduling to make the process more efficient.

Finally, the Transformer-based encoder model, which forms the basis of the proposed final architecture, was observed to take approximately 20 seconds per training pass, due to its attention mechanism and large internal state representation. Although slower than the LSTM based models in terms of single-pass time, its performance and generalization capabilities far outweighed the additional computational cost. This trade-off between runtime and accuracy was acceptable given the significant boost in prediction quality. The Transformer's ability to model long-range dependencies without recurrence made it particularly well-suited for this trajectory prediction task.

For training all model variants throughout this research, a consistent set of optimization and regularization techniques was applied to ensure fair evaluation and reliable convergence. The models were trained for up to 1000 epochs using the AdamW optimizer, with a learning rate of 1e-5 and a weight

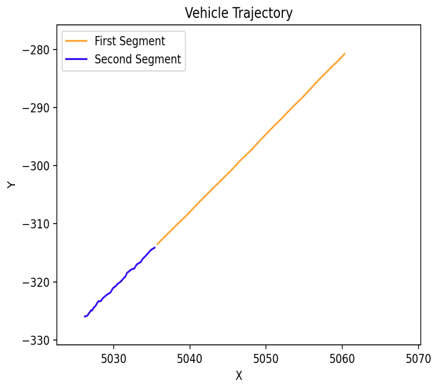


Figure 12: Prediction Example 1

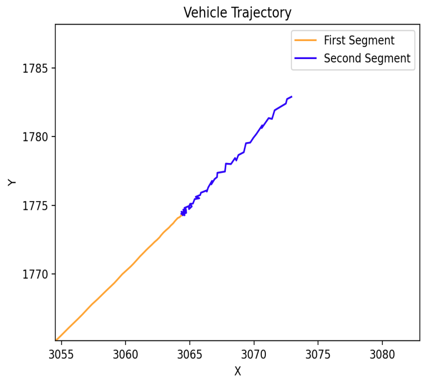


Figure 13: Prediction Example 2

decay of 1e-6 for L2 regularization. To further stabilize training and encourage gradual convergence, a StepLR scheduler was employed, reducing the learning rate by a factor of 0.25 every 20 epochs. The standard Mean Squared Error (MSE) loss was used as the primary loss function for training, and a separate custom loss was evaluated for feature-weighted learning objectives. Additionally, scaling factors for position and velocity were maintained at 1.0 to avoid distorting the relative contribution of features in the loss computation.

An important and somewhat counterintuitive finding emerged during experimentation: in several cases, temporarily increasing the learning rate mid training rather than decreasing it led to further reduction in validation loss. This behavior suggests that the model may have been stuck in sharp or sub-optimal minima, and the higher learning rate helped it escape and find a smoother, lower-loss region of the parameter space. This insight, which stemmed from prior training experience, proved particularly useful in recovering stalled training runs and was incorporated as a tactical adjustment during longer training cycles. Loss values were logged across epochs in a structured format, including training MSE, validation MSE, and evaluation metrics such as true MSE and MAE on held-out test sets. This robust tracking enabled careful monitoring of generalization trends and overfitting risks across different models.

5.4 Results

Achieving a significant reduction in the Mean Squared Error (MSE) on the test dataset from an initial value of 4511.6 down to 7.52 was both challenging and rewarding. This improvement was the result of extensive experimentation involving more than 20 different model architectures and the iterative engineering of input features. Each iteration contributed to the gradual refinement of model performance and prediction accuracy. The final model's evaluation metrics are summarized in Table 3, which highlights the most efficient configuration achieved. The two primary metrics that played a decisive role in evaluating model performance were the Mean Absolute Error (MAE) and the True Mean Squared Error (True MSE). These metrics provided a balanced view of both the average deviation and the variance of the predictions from the ground truth.

Metric	**Value**
Train MSE	0.0019361081
Validation MSE	0.0106566806
Validation MAE	1.9310559519
Validation MSE (True)	1.0656687887
Final Test MSE	7.47883

Table 3: Final Results of the Model Evaluation

In addition, Figures included above specifically Figure 12 and Figure 13 illustrate two representative prediction samples from the test dataset. In these plots, the orange trajectory represents the observed initial motion of the ego vehicle, while the blue trajectory shows the predicted future path generated by the model. These visualizations demonstrate how closely the predicted trajectories align with the actual vehicle movements, showcasing the model's ability to capture motion dynamics accurately.

The significant improvement in error metrics and the qualitative assessment from trajectory plots indicate the robustness and generalization capabilities of the final model. Through systematic exploration of model configurations and input features, a balance between complexity and performance was achieved. This model now forms a strong baseline for future work in vehicle trajectory prediction, with potential for further enhancements through multi-modal inputs or ensemble-based techniques.

5.5 Ablations

There were a few key changes that significantly improved my model's performance. One of the most impactful modifications was simplifying the architecture from a complex Encoder-Decoder model to a straightforward Encoder-only model. This revamp alone led to a dramatic reduction in Mean Squared Error (MSE), bringing it down from the high hundreds to values in the range of 10s and 20s. This simplification helped reduce model complexity, training time, and overfitting, while improving generalization. The second most important improvement was normalizing other vehicles' data relative to the ego vehicle. Additionally, masking input data that was initially zero helped reduce noise and irrelevant information during training. These two techniques improved the consistency and clarity of the training signal, leading to better model convergence. For all methodologies except Model 5 and the Final Model, the initial architectures performed poorly and yielded suboptimal results. These early experiments provided valuable insights into the limitations of certain designs and guided the development of more effective strategies.

Another significant enhancement was moving from training the model solely on the ego vehicle's data to including the entire dataset, encompassing all agents in the scene. This change allowed the model to learn a more complete representation of the environment, resulting in more accurate and coherent path predictions. Including the context of other agents improved the model's ability to anticipate interactions and plan trajectories accordingly. These three methodologies model simplification, relative normalization and masking, and multi-agent input were the most critical changes that transformed the performance of my initial architectures and training process. After implementing these foundational improvements, the focus shifted to fine-tuning. I experimented extensively with various learning rate schedulers and hyperparameter settings. With each configuration, different results emerged, and I used checkpoints to capture the best-performing versions. Occasionally, both MSE and MAE would plateau, indicating convergence. In such cases, I found that slightly increasing the learning rate helped break the plateau and push the metrics lower, leading to new performance gains. In conclusion, while architectural design laid the foundation for high performance, careful hyperparameter tuning was key to extracting the maximum potential from the model.

6 Conclusion

This research paper presents a detailed exploration of trajectory prediction models, progressing from baseline LSTM based architectures to a Transformer inspired encoder model that significantly improved predictive performance. Through extensive experimentation with over 20 models, systematic feature engineering, and strategic architectural simplification, the final model achieved a test set Mean Squared Error (MSE) of 7.4788 and a Mean Absolute Error (MAE) of 1.9310. These results highlight the effectiveness of encoding multi-agent context while predicting solely for the ego vehicle, a design that promotes socially aware and interaction-consistent trajectory forecasting.

Key factors contributing to the model's success included the normalization of inputs relative to the ego agent, the use of attention mechanisms to capture inter-agent dynamics, and a targeted feature set emphasizing motion and spatial relationships. Ablation studies confirmed that including the full agent set during training, masking null values, and normalizing trajectories using a consistent reference point were instrumental in improving accuracy and stability. The final encoder-based model not only generalized well but also maintained practical inference efficiency when deployed on GPUs such as NVIDIA A30 and Apple MPS.

Despite the promising results, there are notable limitations. The final model, while significantly improved over initial versions, lacks adaptability to dynamic environments with varying agent counts, traffic contexts, or map semantics. The architecture assumes a relatively fixed feature set and input structure, limiting its applicability in more flexible or real-world deployment scenarios. Additionally, while attention mechanisms capture interaction dynamics, the model does not explicitly incorporate semantic map data or visual inputs, which are often critical in complex urban use cases. Another limitation is that although the final architecture achieved low error metrics, it still relies on static hyperparameters and does not include dynamic weighting or uncertainty estimation. This restricts its ability to provide confidence-aware predictions, which are crucial for downstream decision making systems in autonomous driving pipelines.

Future improvements could target several directions. One key enhancement would be the integration of ensemble learning techniques, which could combine multiple model variants (e.g., LSTM, MLP, and Transformer-based) to better capture the diverse motion patterns and uncertainties in multi-agent scenes. Ensembles could improve robustness and yield more reliable predictions, especially in edge cases. Furthermore, incorporating adaptive attention mechanisms or graph-based encodings could enhance the model's ability to capture nuanced agent relationships dynamically, improving prediction in more complex scenarios. Integrating semantic map information, such as lane boundaries or traffic signals, could also help in making context-aware predictions.

Additionally, exploring probabilistic or multi-modal prediction frameworks would allow the model to handle the inherent uncertainty in future motion forecasting, offering diverse possible outcomes rather than a single deterministic path. Finally, coupling this model with reinforcement learning or planning modules could create a more comprehensive decision-making pipeline suitable for deployment in real-time autonomous systems. In summary, while the proposed encoder-based model demonstrates strong potential in trajectory prediction tasks, there is ample scope for enhancing its flexibility, robustness, and contextual understanding through ensemble modeling, dynamic learning strategies, and richer input representations.

References

[1] S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. Choi, "Sequence-to-sequence prediction of vehicle trajectory via lstm encoder-decoder architecture," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1672–1678, IEEE, 2018.

[2] F. Giuliari, I. Hasan, M. Cristani, and F. Galasso, "Transformer networks for trajectory forecasting," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 10335–10342, IEEE, January 2021.

[3] Z. Zhang, C. Wang, W. Zhao, M. Cao, and J. Liu, "Ego vehicle trajectory prediction based on time-feature encoding and physics-intention decoding," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 7, pp. 6527–6542, 2024.

[4] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data," in *Computer Vision – ECCV 2020, 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII* (A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds.), vol. 12363 of *Lecture Notes in Computer Science*, pp. 683–700, Springer International Publishing, 2020.

[5] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, "Argoverse: 3d tracking and forecasting with rich maps," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8748–8757, 2019.

[6] Z. Li, X. Ma, Z. Qin, S. Li, and W. Yang, "Rain: Reinforced agent interaction network for traffic trajectory prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[7] S. Casas, C. Gulino, R. Liao, and R. Urtasun, "Implicit latent variable model for scene-consistent motion prediction," in *ECCV*, 2020.

[8] Y. Hong, Y. Wang, Y. Yuan, and K. Kitani, "Starnet: Structured graph attention network for trajectory forecasting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[9] OpenAI, "Chatgpt: Language model," 2025. Accessed: 2025-05-15.

[10] L. Lin, W. Li, H. Bi, and L. Qin, "Vehicle trajectory prediction using lstms with spatial–temporal attention mechanisms," *IEEE Intelligent Transportation Systems Magazine*, vol. 14, no. 2, pp. 197–208, 2021.

[11] L. Hou, L. Xin, S. E. Li, B. Cheng, and W. Wang, “Interactive trajectory prediction of surrounding road users for autonomous driving using structural-lstm network,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 11, pp. 4615–4625, 2019.