CSE 251B Project Final Report

Haochen Wang
Department of Electrical and Computer Engineering
University of California, San Diego
San Diego, CA, 92111
haw057@ucsd.edu

Yuankai Tao
Rady School of Management
University of California, San Diego
San Diego, CA, 92122
yut033@ucsd.edu

Abstract

We present a deep learning approach to the Argoverse motion forecasting task using an **autoregressive LSTM model**, which only autoregressively predicts the next step each time. The model takes a 245-dimensional input vector of past motion features – displacement, velocity, and heading – for 49 frames and predicts the agent's future trajectory over 60 frames. At each step, the LSTM outputs 5 motion deltas (Δx, Δy, Δv_x, Δv_y, Δyaw), which are rolled into a sliding input window to form the next prediction, enabling sequential autoregressive inference. Our design incorporates dropout regularization, learnable initial states, and achieves an MSE of **8.58** on the hold-out test set.

1 Introduction

1.1 Description about the Task

Trajectory forecasting is a fundamental task in autonomous driving and intelligent transportation systems. The goal is to predict the future positions of dynamic agents (such as vehicles or pedestrians) based on their past motion history. Accurate motion prediction enables safer path planning and collision avoidance in real-world scenarios. Deep learning methods, particularly recurrent models, have become a key approach to address this problem due to their ability to model temporal dependencies and complex motion patterns. Real-world applications include:

- **Autonomous Driving:** Predicting vehicle and pedestrian trajectories is essential for safe motion planning and collision avoidance in self-driving cars.
- **Advanced Driver-Assistance Systems (ADAS):** Used in emergency braking, lane keeping, and turn prediction for enhanced road safety.
- **Robotics and Crowd Navigation:** Delivery and service robots rely on forecasting to safely maneuver through human environments.
- **Smart Surveillance:** Motion prediction helps detect abnormal behavior in public areas for security applications.
- **Aerial Drone Planning:** Drones use motion forecasting to avoid dynamic obstacles and coordinate in fleet missions.

1.2 Explanation about Starter Code

The provided starter code implements a basic LSTM-based model that performs direct prediction over future trajectories using fixed-length input sequences. However, it suffers from several limitations:

- The starter code does not incorporate any modeling of inter-agent interactions. In reality, the future trajectory of an agent often depends not only on its own motion but the behavior

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

of surrounding agents (e.g., merging, yielding, or braking in response to pedestrians or vehicles)

- The model is not autoregressive, thus failing to utilize its own predictions during inference
- The data augmentation relies on random flipping and rotation, which we found that will not greatly improve robustness against small-scale trajectory noise
- The coordinate frame is not normalized, introducing unnecessary variation across scenes

1.3 Main Contributions

- **Contribution A:** We implement a sequential prediction mechanism that recursively feeds the model's output back into its input over 60 future steps, enabling effective multi-step forecasting with a single-step LSTM model.
- **Contribution B:** We align all agent trajectories to a canonical reference frame based on the initial yaw angle, eliminating orientation bias and improving model generalization across scenes.
- **Contribution C:** We replace the original random flipping and rotation augmentation with a targeted Gaussian noise strategy to increase robustness against micro-level trajectory jitter.

2 Related Work

Vehicle trajectory prediction has attracted increasing attention due to its importance in autonomous driving and intelligent transportation systems. Existing approaches generally fall into physics-based models, learning-based methods, and socially/context-aware frameworks.

Physics-based and Rule-based Models. Traditional models such as constant velocity (CV) and constant acceleration (CA) assume simple motion patterns and are limited in complex or interactive scenarios. Kalman filters and motion primitives offer some flexibility, but they lack predictive richness.

Learning-based Models. With the rise of deep learning, RNN-based models, especially Long Short-Term Memory (LSTM) networks, have become widely used for sequential motion modeling Alahi et al. [2016]. Encoder-decoder architectures further improved their predictive power, especially when combined with attention mechanisms.

Social and Interaction-aware Models. Social LSTM Alahi et al. [2016] introduced the concept of social pooling to capture interactions among agents. Follow-up works leveraged convolutional pooling Deo and Trivedi [2018] and graph-based attention networks to more effectively model spatial and temporal inter-agent relations.

Scene and Map Context Integration. Incorporating map features such as lanes, crosswalks, and drivable areas has been shown to significantly enhance prediction accuracy Chai et al. [2019]. Transformer-based models like Trajectron++ and mmTransformer Liu et al. [2021] use structured scene information to produce multimodal predictions.

Rotation-Invariance and Local Frames. Several works have shown that transforming inputs into local agent-centric frames or designing rotation-equivariant architectures improves model generalization and convergence. Ablation studies consistently reveal that ignoring heading alignment leads to unstable learning or poor performance in diverse scenes.

Multimodal and Uncertainty-aware Prediction. Probabilistic models such as CVAE Lee et al. [2017] and mixture density networks (MDN) are popular for handling the inherent uncertainty in motion forecasting, especially in scenarios with multiple plausible futures.

3 Problem Statement

3.1 Task Description

The goal of this project is to forecast the future motion of an agent using its past trajectory. The task is framed as a sequence prediction problem based on the Argoverse motion forecasting dataset.

Specifically, given the past 49 frames of an agent's motion, the model is required to predict its motion over the next 60 frames. We focus on single-agent prediction and do not utilize map or multi-agent context information.

3.2 Dataset Descrption

The dataset is structured as a 4-dimensional tensor of shape $(10000, 50, 110, 6)$. Each scene contains 50 agents over 110 time steps. At each timestep, six features are provided: absolute x and y positions, x and y velocities, heading angle (yaw), and agent type ID. For our task, we extract only the ego agent's trajectory from each scene, and use the first 5 features only.

3.3 Input and Output Definition

From the data, we extracted three main input feature types: (1) position displacements (Δpos), (2) velocity deltas (Δvel), and (3) yaw angle differences (Δyaw), each over the past 49 frames. These were flattened and concatenated into a single 245-dimensional input vector per scene.

The model input consists of the ego agent's motion over the first 49 frames. Specifically, we compute the following:

$$\text{Position displacements: } \Delta p_t = p_t - p_{t-1}$$
$$\text{Velocity differences: } \Delta v_t = v_t - v_{t-1}$$
$$\text{Yaw angle differences: } \Delta\text{yaw}_t = \text{yaw}_t - \text{yaw}_{t-1}$$

These differences are stacked and flattened to form a 245-dimensional input vector:

$$x_0 = [\Delta p_{1:49},\ \Delta v_{1:49},\ \Delta\text{yaw}_{1:49}] \in \mathbb{R}^{245}$$

The output is the predicted delta motion for the next 60 frames:

$$y = [\Delta p_{50:109},\ \Delta v_{50:109},\ \Delta\text{yaw}_{50:109}] \in \mathbb{R}^{60\times 5}$$

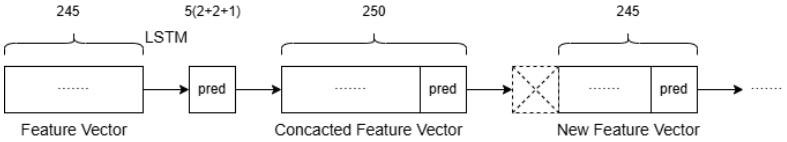

Figure 1: Rolling feature window

And specificly, in the part of prediction, the model generates the full 60-step trajectory autoregressively: at each timestep, the latest prediction, size of which equals to $5 = (2\Delta_{Pos} + 2\Delta_{Vel} + 1\Delta_{Yaw})$, is appended to a sliding input window, for the size of which equals to 245 initially, replacing the oldest feature slice, (size = 5 as well). This rolling mechanism, illustrated in Figure 2 forms the next input vector, which is then passed to the same LSTM to generate the subsequent step.

3.4 Data Split

We split the dataset into 90% training and 10% validation using PyTorch's `random_split` function. During training, samples are shuffled within each batch to promote stochasticity and prevent overfitting. The validation set remains unseen during training and is used for early stopping and model selection.

4 Methods

4.1 Data Prepocessing

4.1.1 Rotation Alignment

We focused on modeling the ego agent's motion by computing relative and rotated motion features. First, we aligned the ego's trajectory to a canonical orientation by rotating all positions and velocities

according to the initial heading angle. This alignment ensures that model learning is invariant to global scene orientation. Without this step, the model would need to implicitly learn rotational invariance from data, which increases learning complexity and data requirements. By conduct the regularization the model can focus on learning motion patterns rather than memorizing scene-specific orientations.

4.1.2 Gaussian Noise Regularization

Secondly, For training data, we applied Gaussian noise regularization to the input vector to improve generalization. This regularization step was motivated by our early-stage visualization of test trajectories, which revealed that some trajectories exhibited small-amplitude jitter or oscillations, especially in curved or low-speed segments. We observed that such micro-fluctuations caused the model to overfit to noise patterns, leading to noticeable directional drifts in predictions. By injecting mild Gaussian noise during training, we aimed to increase the model's tolerance to these perturbations and reduce its sensitivity to minor fluctuations, thus improving trajectory stability and robustness.

4.1.3 Further Augmentation

Additionally, we found that the original starter code included a data augmentation strategy which applied random coordinate rotation and flipping with a 50% probability. However, after conducting our own experiments, we found that these augmentation methods did not yield significant improvements in model performance. As a result, we removed both techniques and instead introduced a replacement during the preprocessing stage: adding random "angular noise" to the heading direction. This approach served as a more targeted solution to the aforementioned jitter issue and led to a slight improvement in prediction accuracy

4.2 Deep Learning Model Construction

To predict future agent trajectories, we design a custom LSTM-based model named `OneStepLSTM`, which performs sequential one-step-ahead prediction in an autoregressive manner. The model receives the flattened 245-dimensional input vector composed of displacement, velocity, and heading change features over 49 past timesteps. This vector is first processed by a linear projection and layer normalization block to stabilize training dynamics.

The core of the architecture is a two-layer LSTM module with a hidden size of 256, followed by layer normalization and dropout regularization. A fully connected output head maps the LSTM's output to a 5-dimensional motion delta vector: $(\Delta x, \Delta y, \Delta v_x, \Delta v_y, \Delta\text{yaw})$. The initial hidden and cell states of the LSTM are liabilistic parameters, allowing the model to adapt its initialization over the course of training.

As illustrated in Figure 1, this recursive feedback structure allows the LSTM to model long-term temporal dependencies while maintaining low memory cost. This architecture effectively balances model capacity and stability, and integrates seamlessly with the rotation-invariant features constructed during preprocessing.

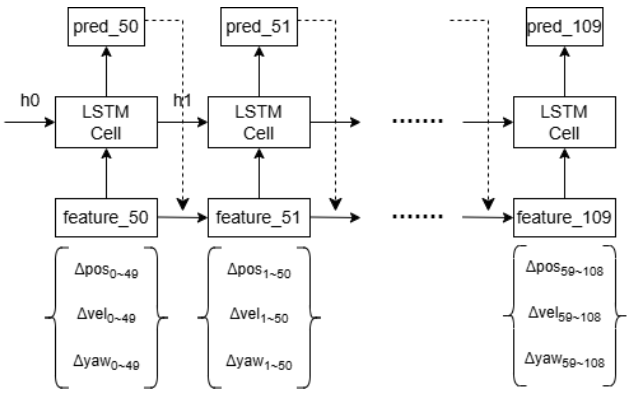

Figure 2: Autoregressive architecture with rolling feature input

4.3 Training Strategies and Evaluation

In training part of the task, We trained the model using the custom autoregressive loop designed to unroll future predictions step-by-step from $t = 50$ to $t = 109$. At each timestep, the LSTM receives a 245-dimensional input vector and outputs five motion deltas $(\Delta x, \Delta y, \Delta v_x, \Delta v_y, \Delta\text{yaw})$. These are appended into a rolling window, replacing the oldest frame in the feature vector to construct the next input—thereby implementing autoregressive inference across 60 future steps. During inference, the initial hidden and cell states are set using learnable parameters, and recurrent state is updated across time.

For training, we used the standard mean squared error (MSE) loss to compute supervision between the predicted and ground truth deltas in position, velocity, and yaw. Additionally, we evaluated the model using the original (non-rotated) coordinates by applying an inverse transformation, and reported a separate MSE based on absolute position error. Optimization was performed using the Adam optimizer with a learning rate of 1×10^{-3} and weight decay of 1×10^{-3}. We adopted a `ReduceLROnPlateau` scheduler, which is an effective scheduler that lowers the learning rate when validation loss stops improving, with a patience of 4 epochs and decay factor of 0.15 to reduce the learning rate adaptively based on validation loss.

The model was trained for up to 100 epochs with early stopping applied if no improvement was observed in 10 consecutive epochs. We tracked two types of validation loss: one based on predicted deltas (in the rotated frame) and another based on reconstructed absolute positions (in the global frame). The best-performing model was selected based on the latter. For fair comparison, data loaders were shuffled and validation was conducted on a hold-out 10% of the training set unless full data training mode was activated.

5 Experiments

5.1 Baselines

We use two baselines as benchmarks to compare our LSTM model against:

- **Constant Acceleration (CA)**: This baseline assumes the ego agent has a constant acceleration during the last 60 time steps and calculates the position at each step from the motion formula
$$\boldsymbol{p}(t) = \boldsymbol{p}_0 + \boldsymbol{v}_0 t + \frac{1}{2}\boldsymbol{a}t^2,$$
where $\boldsymbol{p}_0$ and $\boldsymbol{v}_0$ are position and velocity at the 50_{th} step, $\boldsymbol{a}$ is the acceleration from the 30_{th} step to the 50_{th} step, and t ranges from 1 to 60.
- **eXtreme Gradient Boosting (XGBoost)**: XGBoost is a tree-boosting algorithm known for its efficiency and strong performance across various regression tasks. Our adoption of XGBoost on the trajectory prediction task is motivated by its strong performance in time series forecasting . We train two XGBoost regression models to predict the x and y coordinate of the ego agent's next position, respectively, using the position differences of the last 50 time steps as features. Each predicted position is used to update the feature vector to predict the subsequent position in an autoregressive manner. The two XGBoost models are trained to minimize the MSE loss.

5.2 Evaluation

The models are compared by their test set prediction MSEs.

5.3 Implementation Details

We used Kaggle as our development platform. Leveraging the NVIDIA Tesla P100 GPU available in the Kaggle notebook environment, each epoch took approximately 16 seconds. Our model employed the following configuration:

- **Batch size**: 128

- **Optimizer**: Adam with initial learning rate 1×10^{-3}
- **Regularization**: L2 weight decay (1×10^{-3})
- **Scheduler**: ReduceLROnPlateau (factor=0.15, patience=4 epochs)
- **Training**: Up to 100 epochs, batch size 128, early stopping (patience=10)
- **Architecture**: 2-layer LSTM with 256 hidden units and 0.2 dropout
- **Loss**: Sum of MSE for position, velocity, and yaw angle predictions

Hyperparameters were tuned using random search over the following ranges:

Table 1: Hyperparameter Search Space

Parameter	**Search Range**
Learning rate	$\{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 5 \times 10^{-3}\}$
Weight decay	$\{1 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}\}$
Hidden layer size	$\{128, 256, 512\}$
LSTM layers	$\{1, 2, 3\}$
Dropout rate	$\{0.1, 0.2, 0.3, 0.5\}$
Batch size	$\{64, 128, 256\}$
Scheduler factor	$\{0.1, 0.15, 0.2\}$
Scheduler patience	$\{3, 4, 5\}$

The search ranges are chosen based on our past experiences tuning neural network models.

5.4 Results

The prediction MSE for the last 60 steps of each model on each data split is reported in Table 2. Additionally, we visualize the predictions of each model on several validation scenes in Figure 3.

From Figure 3, we observe that while XGBoost produces smoother and less noisy trajectories, it struggles to capture the ego agent's intention to turn or initiate movement in a new direction. This observation helps explain the relatively small MSE performance gap between XGBoost and the Constant Acceleration baseline, which simply extrapolates the agent's trajectory in a straight line based on its most recent orientation. In contrast, the LSTM model produces noisier predictions but more accurately follows the direction of the ego agent's actual motion.

It is important to note that our XGBoost model uses only past position differences as input features, whereas the LSTM model leverages the full history of available data. This design choice stems from the nature of the models: XGBoost, like most traditional regression algorithms, is structured to predict a single target and does not naturally accommodate multi-dimensional outputs. In contrast, LSTM models can be easily adapted to predict multiple features in parallel by simply adjusting the size of the final output layer—a straightforward modification that introduces minimal computational overhead.

Table 2: Performance across different datasets

Model	**Train**	**Validation**	**Test**
CA	-	-	17.59
XGBoost	11.90	14.32	15.72
LSTM	7.34	8.57	8.53

5.5 Ablations

Figure 4 shows the training and validation loss curves of our proposed model. To evaluate the contribution of each component introduced in Section 4, we disable one technique at a time while keeping the rest unchanged.

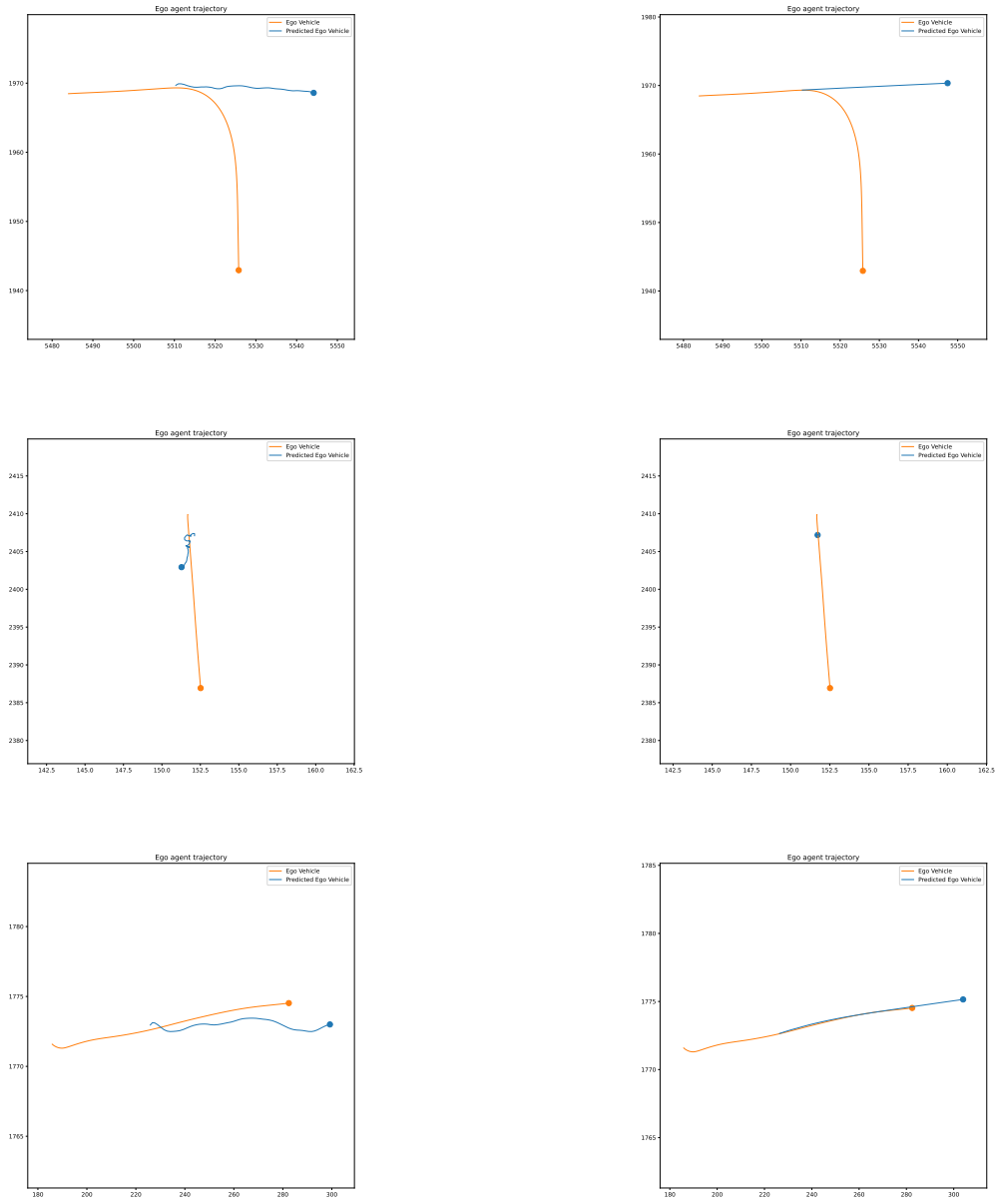

Figure 3: Comparison of trajectories predicted by LSTM and XGBoost.

Heading angle alignment: When initial heading angle alignment is disabled (Figure 5), the model converges more slowly and shows signs of overfitting, with the validation loss plateauing at a higher value than the full model. This suggests that the alignment aids both convergence and generalization.

Gaussian noise regularization: Without Gaussian noise regularization (Figure 6), the validation loss exhibits greater fluctuations during training, though the final performance remains similar. This indicates that the noise primarily stabilizes training rather than improving final accuracy.

Predicting velocity and heading angle features in parallel: Using only past positions as features to predict future positions (Figure 7), both train and validation losses stay above 10, evidencing the significant improvement brought by parallel prediction of velocity and heading angles.

These results demonstrate that all techniques meaningfully contribute to the prediction performance of our proposed LSTM model.

6 Conclusion

6.1 Lessons Learned and Insights

One of the key lessons we learned in this project is the importance of rolling input construction. We adopted a sliding temporal window that updates the input feature vector at each timestep during autoregressive prediction. This taught us that autoregression does not have to reside solely within the model structure (e.g., RNN hidden states), but can also be implemented at the data level through feature window updates.

We also learned that careful input preprocessing and feature design can be as impactful as architectural improvements. For instance, aligning trajectories to a common reference frame significantly stabilized training. For deep learning beginners working on similar prediction tasks, we recommend starting with focusing on only ego agent at the early stage and try to include more interaction between agents gradually. What's more, try to spend more time on the visualization and data augmentation which may help a lot.

6.2 Limitation Observed

The greatest limitation is that the current approach is lack of consideration for the interaction modeling between different agents. Our current model only consider the ego agent while choose to ignore the other objects. As a result, the current approach may fail to capture the coordinated behaviors in dense or multi-agent scenarios, thus the social or realistic physical interactions such as following, yielding, or collision avoidance had been excluded from the learning. The greatest bottleneck is that: due to the limitations metioned above, our model struggles to accurately predict the behavior of ego agents in scenarios that involve stopping or making abrupt direction changes to avoid collisions with other vehicles or pedestrians. Because of the limitation of the model itself, the ego agent often continues to extrapolate along the presumed predicted-path of motion. As a result, significant prediction errors often occur in such cases.

6.3 Future Work

We plan to study QCNet, the winning solution of the Argoverse 2 Challenge. Our goal is to understand how query-based center prediction and multi-mode reasoning contribute to improved trajectory quality. Specifically, we aim to analyze the network's architecture, input encoding strategies, and candidate center generation mechanisms. We might experiment with adapting QCNet to our own motion prediction pipeline and compare its performance against our current LSTM-based baselines to evaluate its effectiveness in capturing multimodal and interaction-aware behaviors.

7 Contributions

Haochen implemented Gaussian noise regularization, ego agent heading angle alignment, multi-target (position, velocity and angle) prediction, trajectory visualization, and the Constant Acceleration model. He also completed section 2, 5 of the report.

Yuankai implemented the XGBoost model and noise injection to the heading angle alignment matrix. He also performed data analysis, created the presentation slides, and completed section 1, 2, 3, 4, 6 of the report.

References

Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *CVPR*, pages 961–971, 2016.

Yuning Chai, Barrett Sapp, Mangal Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses. In *CVPR*, pages 13624–13633, 2019.

Nachiket Deo and Mohan M Trivedi. Convolutional social pooling for vehicle trajectory prediction. In *CVPR Workshops*, 2018.

Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *CVPR*, pages 336–345, 2017.

Mingxu Liu, Hong Li, Hang Zhao, and Masayoshi Tomizuka. Multimodal motion prediction with stacked transformers. In *CoRL*, 2021.

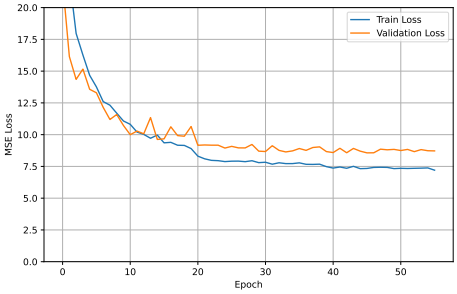

Figure 4: Loss curves of proposed model

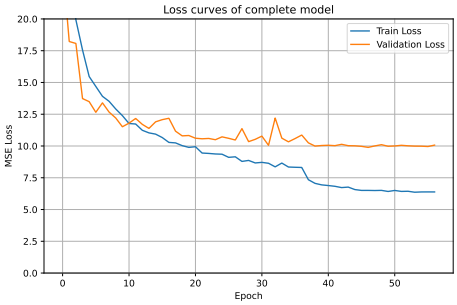

Figure 5: Loss curves of proposed model without ego agent heading angle alignment

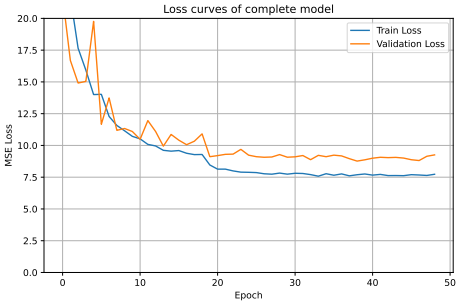

Figure 6: Loss curves of proposed model without Gaussian noise regularization

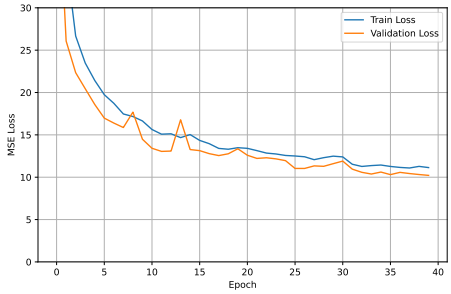

Figure 7: Loss curves of proposed model using only past positions as feature