HCMC UNIVERSITY OF TECHNOLOGY AND EDUCATION
FACULTY OF MECHANICAL ENGINEERING

HCMUTE

ARTIFICIAL INTELLIGENCE

TRAJECTORY PREDICTION ON HIGHWAYS

Lecturer: Dr. Tran Vu Hoang

Student: Vo Thanh Nguyen - 22134008
Nguyen Trinh Tra Giang - 22134005

HCMC, December 2024

Contents

Chapter 1

Introduction

1.1 Motivation

The exponential growth in highway traffic has resulted in heightened congestion, increased accident rates, and substantial economic losses attributed to travel delays. According to the Federal Highway Administration (FHWA), traffic congestion alone is estimated to cost the United States approximately $87 billion annually in wasted fuel and lost time [1]. The National Highway Traffic Safety Administration (NHTSA) further underscores the severity of this issue by reporting over 6 million police-recorded motor vehicle crashes in 2020, revealing the urgent need for innovative traffic management solutions [2]. The figure 1.1 show the severity of an accident on Highway.

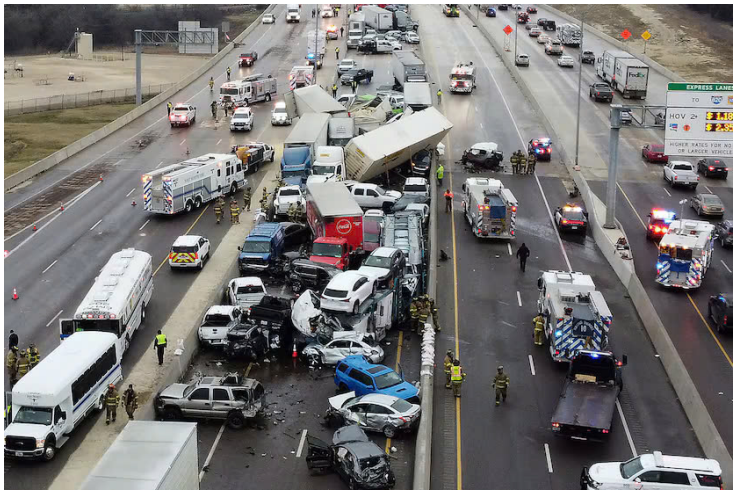

Figure 1.1: At least 6 dead in 133-car pileup in Fort Worth after freezing rain coats road

Accurate prediction of vehicle trajectories is an essential component of intelligent transportation systems (ITS). Such systems aim to optimize traffic flow, enhance road safety, and reduce congestion by proactively managing potential conflicts and bottlenecks. Traditional trajectory prediction methods often struggle to capture the complex interactions and varied driver behaviors common in highway settings. They are also limited by real-time constraints, computational inefficiencies, and the challenge of scaling to large traffic volumes. The Pishguve [3] model provides an advanced alternative by leveraging modern machine learning techniques to improve prediction accuracy and speed, making it well-suited for high-volume datasets such as NGSIM [4].

1.2 Objectives

Motivated by the increasing need to mitigate highway congestion, reduce accident rates, and enhance traffic efficiency, the primary objective of this report is to investigate the trajectory prediction problem, specifically for highway environments. Our goal is to develop a trajectory prediction model that ensures high and stable accuracy while maintaining a lightweight architecture and consistent computational speed.

1.3 Scope

This study focuses on trajectory prediction for vehicles on highways. The research relies on pre-recorded position coordinates, excluding tasks such as direct trajectory prediction from video data or other sensor modalities. However, the model is designed with potential deployment on Cyber-Physical Edge Systems in mind. The scope encompasses:

- **Data Source**: The study relies solely on position coordinates obtained from GPS, which provides detailed vehicle trajectory information on specific highway segments
- **Technical Constraints**: The model is optimized for real-time operation using position data. While the research primarily focuses on offline analysis, the model architecture is adaptable for deployment on Cyber-Physical Edge Systems.
- **Excluded area**: The study does not address trajectory prediction directly from video inputs or extend to broader traffic management functions such as signal control or route optimization.

By concentrating on positional data and designing for edge deployment, the study aims to create a lightweight and efficient solution for real-time trajectory prediction in intelligent transportation systems.

1.4 Report Structure

The report is organized into several chapters, reflecting our methodological progression and key findings. Following this introductory chapter, Chapter 2 (*Related Works*) reviews theoretical underpinnings and the broader landscape of trajectory prediction methodologies, with special attention to existing challenges. In Chapter 3 (*Existing Methods*), we present maneuver-based and interaction-aware models, highlighting their advantages and limitations. Chapter 4 (*Proposed method*) describes our chosen Pishguve model, detailing its system architecture and how it integrates graph neural networks (GNNs) and attention-based convolutional neural networks (CNNs). Chapter 5 (*Experimental Evaluation*) presents the dataset, metrics, and experimental results, followed by Chapter 6 (*Conclusion and Future Work*), where we summarize our findings and propose directions for further research.

1.5 Team Effort and Tasks

This section outlines the responsibilities undertaken by each team member. Table 1.1 shows the broad task distribution.

Table 1.1: Task Distribution Among Team Members

Task Description	**Vo Thanh Nguyen**	**Nguyen Trinh Tra Giang**
Research and review literature on trajectory prediction	X	X
Identify and acquire relevant datasets		X
Data preprocessing and structuring		X
Selection of appropriate algorithms	X	
Model building and evaluation		X
Report generation and visualization	X	
Presentation and recommendations	X	X

Chapter 2

Related Works

2.1 Theoretical Foundations

Trajectory prediction essentially attempts to forecast future positions of vehicles based on their current and historical movements, while also considering the influence of neighboring agents and environmental conditions. Models in this domain must capture spatial-temporal dependencies, driver behavior heterogeneity, and inter-vehicle interactions, all while meeting real-time execution needs. Many traditional methods, including Kalman-based filter [5] and physics-driven approaches, excel in simpler or low-density traffic but are less robust when driver behaviors deviate significantly from idealized assumptions.

2.2 Challenges in Trajectory Prediction

Despite notable progress, several hurdles continue to impede efficient and accurate trajectory prediction. Balancing accuracy and computational speed is a critical dilemma: high model complexity may improve predictive power but slows inference, complicating real-time applications. Longer prediction horizons add more uncertainty, especially on highways where speeds and traffic fluctuations vary widely. Modeling multiple interacting trajectories raises the complexity further, as each vehicle's path can influence its neighbors through lane changes or decelerations to avoid collisions. Highways also manifest dynamic traffic patterns that shift with time of day, weather conditions, and regional factors. Moreover, large-scale datasets like NGSIM [4]contain substantial noise and may have missing values, requiring robust data-cleaning strategies. Finally, scalability and data availability remain central concerns. Models need to process immense volumes of trajectory data without performance degradation, while real-world conditions often supply only partial or imperfect information about driver intentions and surrounding road infrastructures.

2.3 Existing Methods

2.3.1 Maneuver-based Motion Models

Maneuver-based models predict trajectories by identifying actions like lane changes, merges, or turns. These approaches often assume each vehicle makes decisions independently, simplifying interaction dynamics. Common techniques include clustering maneuver types, applying HMMs or RNNs [6] to capture sequential patterns, and exploring reinforcement learning in simplified settings. Although these models can detect critical maneuvers early and provide improved long-horizon predictions, they struggle in dense traffic, where ignoring interactions leads to reduced accuracy.

2.3.2 Interaction-aware Motion Models

Interaction-aware models view traffic as an interconnected system, with each vehicle's motion influenced by its neighbors. This category covers prototype trajectories, dynamic Bayesian networks, and coupled HMMs, but deep learning (especially LSTMs [7] with attention machanism or pooling [8], [9]) has notably improved multi-vehicle trajectory prediction. Despite the improved performance, these models need large datasets and can be computationally expensive, especially if every vehicle-to-vehicle interaction is modeled explicitly.

Representative Interaction-aware Models in the Literature

Early methods, such as Woo et al. [10], use LSTMs for ego-vehicle and a few neighbors, while Kim et al. [11] leverage occupancy grids to predict positions and outperform simpler filters. Lin et al. [12] employ attention-based LSTMs, [9], [8] propose social pooling to emphasize critical interactions. Messaoud et al. [13] use attention to rank the most influential vehicles. These approaches enrich trajectory forecasts under moderate congestion but face scalability challenges in extremely dense traffic.

Newer Graph-based Methods

Graph-based methods model each vehicle as a node and interactions as edges to capture complex spatio-temporal dependencies. GSTCN [14] utilizes spatio-temporal graphs with CNN layers (replacing LSTM) for faster inference and high accuracy. GRIP [15] adopts an LSTM encoder-decoder for multi-object prediction, which offers strong accuracy but higher computational costs. Pishgu [16] combines Graph Isomorphism Networks (GIN) and attention for real-time performance, achieving state-of-the-art results on NGSIM. PishguVe [3] further embeds attention within the GNN and applies Linear Attention Dropout (LAD) to selectively emphasize essential interactions, maintaining a lightweight design suitable for real-time embedded systems.

Table 2.1: Comparison of Existing Trajectory Prediction Methods Based on Challenges

Method	Main Technique	Interaction Modeling	Key Advantages	Challenges
Maneuver-based	Clustering, HMM, RL	Assumes independence	Simple and fast; Early maneuver detection	Struggles with multi-agent scenarios; Poor in dense environments
Woo et al. [10]	LSTM (relative features)	Limited (ego+4 vehicles)	Captures long-term dependencies; Lightweight and interpretable	Limited scalability; Difficult to model dense traffic
Kim et al. [11]	LSTM + occupancy grid	Partial (nearest neighbors)	Probabilistic modeling; Robust in sparse traffic	Sensitive to grid resolution; High computational cost
Lin et al. [12]; Messaoud et al. [13]	LSTM + attention on grid	Explicit attention weighting	Improves interaction modeling; Higher accuracy in complex traffic	Requires large datasets; Computationally expensive; Difficult to model dense traffic
Social LSTM [8], [9]	CNN + LSTM (grid-based)	Partial (grid interaction)	Structured environments; Captures sequential behavior	Ineffective in highly unstructured traffic; Limited interaction modeling; Difficult to model dense traffic
GRIP [15]	Graph convolution + LSTM	Partial (local neighborhood)	Models local interactions; Simultaneous multi-agent prediction	Complex edge design; Limited global context
GSTCN [14]	Graph-based spatio-temporal conv + CNN	Partial (spatial-temporal graph)	Efficient inference; Lightweight model (49.8k params)	Limited spatial reach; Graph complexity in large-scale scenes
Pishgu [16]	GNN + Attention-based CNN	Full (all vehicles)	Real-time inference; Effective in dense traffic	Computationally demanding; Prone to overfitting in rare events
PishguVe [3]	GNN + Local Attention + CNN	Full (all vehicles with global and local attention)	High accuracy in dense scenarios; Robust feature extraction	Complex architecture; Fixed timestep input-output with CNN decoder; Requires optimization for real-time performance

Chapter 3

Proposed method

3.1 Design Requirements

Our design requirements focus on ensuring **high predictive accuracy** under diverse traffic conditions while maintaining **real-time feasibility** and **scalability** for large numbers of interacting vehicles. Because highway traffic can involve sudden maneuvers, frequent lane changes, or variable driver behaviors, the system must handle **uncertainties** in agent dynamics (e.g., peak traffic). Consequently, the architecture is designed to exploit *spatial-temporal interactions* in a way that does not incur excessive computational costs.

To meet the Design Requirments, we adopt the **PishguVe** [3] model, which leverages a of *Graph Isomorphism Networks (GIN)* [17] and *Linear Attention Dropout (LAD)* to effectively capture spatial interactions between vehicles. The key reasons for selecting PishguVe include:

- **Expressive Graph Representation:** By modeling traffic snapshots as fully connected graphs, Selective model can capture intricate relationships between vehicles, enhancing the system's ability to predict nuanced agent interactions over time.
- **LAD for Enhanced Aggregation:** The LAD block integrates linear transformations with attention mechanisms and dropout, refining neighbor features and preventing overfitting. This makes PishguVe well-suited for handling dense traffic scenarios without sacrificing generalization.
- **Temporal Modeling via CNNs [18]:** This module utilizes an attention-based CNN decoder to process sequential embeddings, efficiently learning temporal dependencies across multiple frames.
- **Scalability and Efficiency:** The combination of lightweight GIN [17]layers and efficient LAD modules allows PishguVe to scale with increasing numbers of vehicles, ensuring fast inference while maintaining high accuracy.

By designing the architecture to exploit *spatial-temporal interactions* and incorporating robust attention mechanisms, PishguVe [3] balances **model complexity** with **computational efficiency**. This enables the system to deliver accurate, real-time trajectory predictions, even in highly dynamic and congested traffic conditions.

3.2 Model Architecture

Figure 3.1 illustrates the overall flow of **PishguVe [3]**. It consists of four main steps:

1. **Data Preprocessing:** We extract each vehicle's position from the NGSIM [4] dataset and normalize them to ensure uniform scaling. Additional attributes (e.g., speed, lane ID) may also be included.
2. **GIN Encoder (with LAD):** A fully connected graph is constructed at each time frame, where vehicles are nodes and edges represent potential interactions. *Graph Isomorphism Network* (GIN) [17] for node aggregation, enhanced by our novel **Linear Attention Dropout (LAD)** module for neighbor feature refinement.

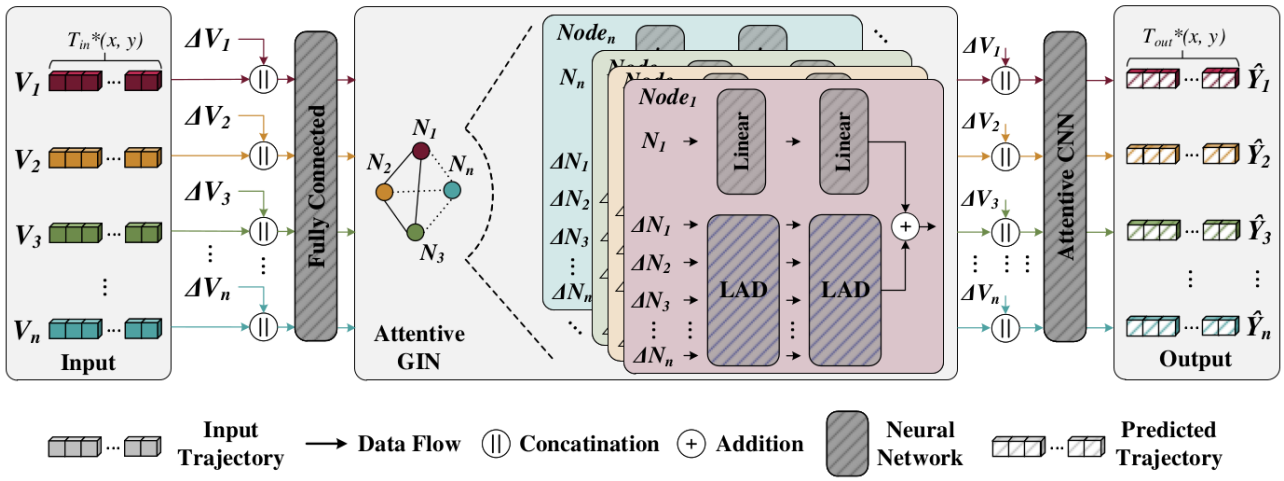

Figure 3.1: Overview of the **PishguVe** Model Architecture. The input pipeline includes data preprocessing, GNN-based encoding of spatial interactions (with LAD), an attention-based CNN decoder for temporal modeling, and final trajectory prediction.

3. **Attention-based CNN Decoder:** The node embeddings from the GIN [19] encoder are stacked over multiple time steps to form a 3D feature map, then passed through a CNN [18]pipeline interspersed with channel and spatial attention blocks [20, 21].

4. **Trajectory Output:** A final series of dense layer(s) produces predicted positions (and optionally velocities) for each vehicle over the chosen prediction horizon T_{out}.

In the following, we provide a deeper look at the key Model components, referencing foundational CNN approaches [18,22] as well as more recent GNN [17]and attention-based enhancements [17,20,21].

3.3 PishguVe Model Implementation

The **PishguVe** model consists of two core components:

- A **Graph Neural Network (GNN) encoder** with **Linear Attention Dropout (LAD)** blocks for capturing spatial relationships among multiple vehicles at a given snapshot in time.
- An **attention-based Convolutional Neural Network (CNN) decoder** for learning temporal dynamics across sequential frames.

3.3.1 Relative Position Encoding (Optional)

In many traffic scenarios, *relative positions* can be crucial for understanding local interactions (e.g., distances or directions to neighboring vehicles). In **PishguVe**, we typically define the relative position of vehicle i at time t with respect to its position at the earliest observation time t_0 (e.g., $t_0 = t - T_{\text{in}}$). Mathematically:

$$\Delta N_i^{(t)} = N_i^{(t)} - N_i^{(t_0)},$$

where $N_i^{(t)}$ is the absolute position of vehicle i at time t. These relative coordinates can be integrated in two primary ways:

1. **Concatenation into Node Features:** Append $\Delta N_i^{(t)}$ directly to each node's feature vector $\mathbf{x}_i$, along with the absolute positions and any additional attributes (e.g., speed, lane ID). In this approach, the GNN treats relative displacement as part of the node's initial input.

2. **Injection during Neighbor Aggregation:** One may apply a small MLP (or linear transform) $\mathbf{P}$ to the pairwise difference $\mathbf{x}_j - \mathbf{x}_i$, thereby obtaining a feature vector

$$\Delta\mathbf{x}_{ij} \;=\; \mathbf{P}\big(\mathbf{x}_j - \mathbf{x}_i\big).$$

This $\Delta\mathbf{x}_{ij}$ is then concatenated into the neighbor sum:

$$\sum_{j\in\mathcal{N}(i)} \big[\mathbf{h}_j^{(l)};\, \Delta\mathbf{x}_{ij}\big],$$

allowing the GNN to learn how relative displacement impacts message passing between nodes.

In practice, the choice between embedding $\Delta N_i^{(t)}$ at the node-feature level or injecting it during neighbor aggregation depends on factors like model complexity, memory constraints, and the specific traffic scenario. Either way, leveraging both absolute and relative information helps **PishguVe** [3] better capture local displacements and global context for robust trajectory forecasting.

3.3.2 GIN-based Encoder with LAD

Graph Construction

We consider each traffic snapshot at time t to be a graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ where:

- $\mathcal{V}_t = \{v_1, v_2, \dots, v_n\}$ is the set of n vehicles present in the scene at time t.
- $\mathcal{E}_t$ represents edges between every pair of vehicles. For a fully connected graph, each vehicle has edges to *all* other vehicles in the same snapshot.

Each node v_i is associated with a feature vector $\mathbf{x}_i$ (containing both absolute and relative positions, plus optional attributes).

Graph Isomorphism Network (GIN)

To effectively capture interactions in our *fully connected* graph, we adopt a **Graph Isomorphism Network (GIN)** [17] rather than a standard GNN [19]. In a fully connected setting, each node has *all other nodes* as neighbors, which can lead to an abundance of information but also requires a robust mechanism to learn from *diverse* neighbor features. GIN [17]is a strong candidate because it has been shown to be as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test in terms of distinguishability and can be adapted for large, dense graphs.

Why GIN for a Fully Connected Graph?

- **Representation Power:** GIN's simple yet expressive aggregation function (scaling self-features by $(1 + \theta)$ and summing neighbor features) provides a strong foundation for learning complex interactions without overcomplicating the message-passing scheme.
- **Flexible Adaptation:** Because our graph is fully connected, we needed an approach that can handle a wide range of neighbor embeddings without saturating or losing key details. GIN's design allows for straightforward modifications to the aggregator, which we exploit to increase *diversity* in the features extracted from different neighbors.

Our Adapted GIN Formulation. A single GIN layer in **PishguVe** updates node i's representation by equation (3.1):

$$\mathbf{h}_i^{(l+1)} \;=\; \mathrm{MLP}_0\Big((1+\theta)\,\mathbf{h}_i^{(l)}\Big) \;+\; \mathrm{MLP}_1\Big(\sum_{j\in\mathcal{V}(i)} \mathbf{h}_j^{(l)}\Big), \tag{3.1}$$

where:

- $\mathbf{h}_i^{(l)}$ is the hidden representation of node i at layer l.
- MLP_0 and MLP_1 are small multi-layer perceptrons (each with at least one hidden layer and a nonlinear activation, e.g., ReLU).
- θ is a learnable scalar that scales the self-feature contribution.

Linear Attention Dropout (LAD) for Neighbor Aggregation

PishguVe [3] introduces a novel *Linear Attention Dropout (LAD)* block designed explicitly for neighbor feature aggregation. Figure 3.2 shows the details of the LAD block. LAD consists of a linear layer followed by attention layers for highlighting the most informative neighbor features to construct a richer representation. We adopt channel and spatial attention from [20, 21] to integrate attention efficiently into the GIN. Also, to avoid overfitting, we leverage two dropout layers. Two back-to-back blocks of LAD are used for neighbor aggregation to extract higher-level features.

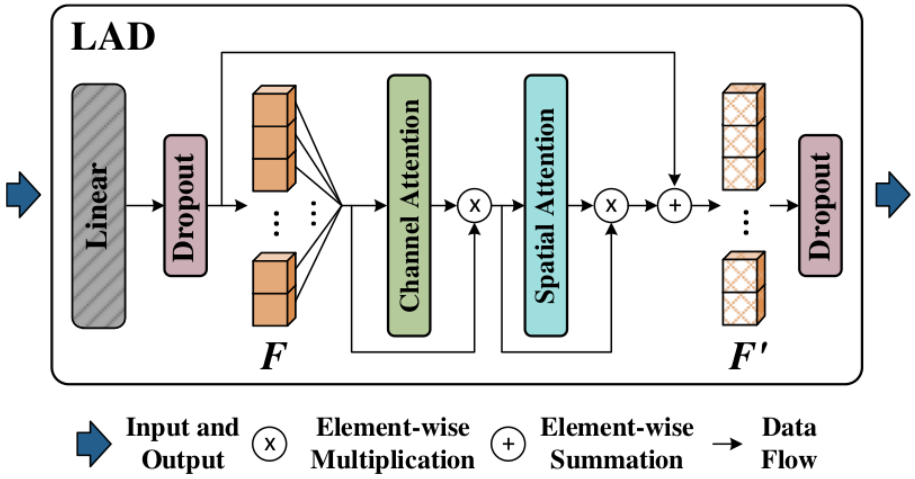

Figure 3.2: **Linear Attention Dropout (LAD) module.** Each neighbor's feature difference ΔN_j is processed by a linear layer (W_4, B_4), then goes through pooling and attention before merging into the GIN.

LAD Operation. Let W_4 and B_4 be the parameters of the linear layer, and MLP_1 be a shared Multi-layer Perceptron with one hidden layer. For a neighbor j of node i, we first compute the difference $\Delta N_j = \mathbf{x}_j - \mathbf{x}_i$ and transform it via

$$W_4 \cdot \Delta N_j \; + \; B_4.$$

We then apply global average pooling ($\mathrm{Pool}_{\mathrm{avg}}$) and global max pooling ($\mathrm{Pool}_{\mathrm{max}}$) separately, passing each through MLP_1. Thus, the LAD output F_j' can be summarized as follows (matching Equation (3.2) in the paper):

$$F_j' \;=\; \underbrace{\mathrm{MLP}_1\Big(\mathrm{Pool}_{\mathrm{avg}}\big(W_4\,\Delta N_j + B_4\big)\Big)}_{\text{Avg branch}} \;+\; \underbrace{\mathrm{MLP}_1\Big(\mathrm{Pool}_{\mathrm{max}}\big(W_4\,\Delta N_j + B_4\big)\Big)}_{\text{Max branch}}. \tag{3.2}$$

In other words, each pooling branch goes through the same MLP, and the two outputs are summed. F_j' is then the final result of one LAD block for neighbor j.

Integration into GIN. The final output of the GIN layer is constructed by summing the results of node aggregation and neighbor aggregation, as described in Equation (3.3) of the paper:

$$\mathrm{GIN}(N_i) = G_i^0 + \sum_{j \in \mathcal{V}(i)} \mathrm{LAD}(F_j'), \tag{3.3}$$

where $\mathcal{V}(i)$ represents the set of neighbors for the i^{th} node, and G_i^0 denotes the node-level (self) aggregation output.

Advantages. By splitting the average-pooled and max-pooled paths and passing each through the same MLP, this formulation more flexibly highlights informative features from multiple neighbors, while two dropout layers mitigate overfitting. Consequently, **LAD** enhances the GIN's ability to capture complex interactions among vehicles for robust trajectory prediction.

3.3.3 Attention-based CNN Decoder

Temporal Modeling Framework

Since traffic data is time-dependent, let T_{in} be the input window of historical frames and T_{out} the prediction horizon. For each vehicle v_i, we gather GIN-encoded embeddings across T_{in} steps:

$$\left[\mathbf{z}_i^{(t_1)}, \mathbf{z}_i^{(t_2)}, \ldots, \mathbf{z}_i^{(t_{T_{\text{in}}})}\right].$$

We then stack these embeddings into a 3D "feature map" and feed it into a CNN for temporal modeling.

CNN with Attention Mechanisms

Convolutional Neural Networks (CNNs) have a proven track record in vision tasks [18,22], and have also been adapted for sequential data. Our decoder uses a lightweight 2D CNN architecture interspersed with attention blocks, specifically:

- **Squeeze-and-Excitation (SE)** [20] for *channel attention*, weighting each channel by importance.
- **Convolutional Block Attention Module (CBAM)** [21] for *spatial attention*, emphasizing salient regions in the feature map.

Channel Attention. Following the Squeeze-and-Excitation (SE) approach [20], for a feature map $\mathbf{F} \in \mathbb{R}^{C \times H \times W}$, we compute global average pooling (GAP) and global max pooling (GMP) across the spatial dimensions to form channel descriptors. As shown in Figure 3.3, these descriptors pass through a small MLP (with a sigmoid activation at the end) to produce an attention vector $\mathbf{M}_c(\mathbf{F}) \in \mathbb{R}^C$. We then multiply $\mathbf{M}_c(\mathbf{F})$ channel-wise with $\mathbf{F}$, thus emphasizing the most important feature channels.

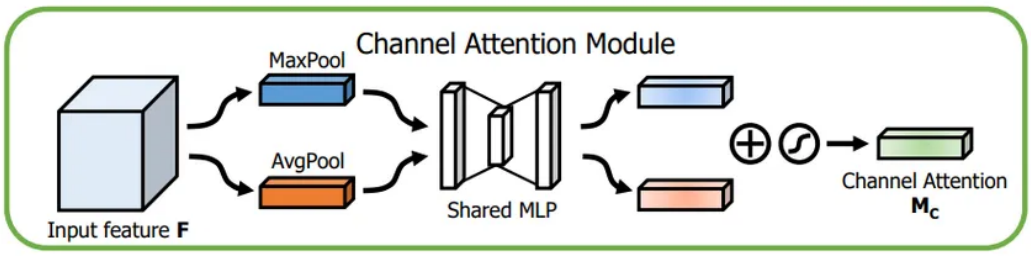

Figure 3.3: **Channel Attention Module** [20,21]. GAP and GMP generate two channel descriptors, which are merged and passed through an MLP + sigmoid to yield the channel-wise attention map $\mathbf{M}_c(\mathbf{F})$.

Spatial Attention. Next, we apply a spatial attention module [21] by stacking the average-pooled and max-pooled feature maps along the channel dimension, and convolving with a $k \times k$ kernel (often $k = 7$) over the spatial dimensions. As illustrated in Figure 3.4, the resulting attention mask $\mathbf{M}_s(\mathbf{F})$ focuses on the most relevant spatial locations in $\mathbf{F}$. We multiply $\mathbf{F}$ element-wise by $\mathbf{M}_s(\mathbf{F})$, thereby guiding the model to attend to salient areas in the feature map.

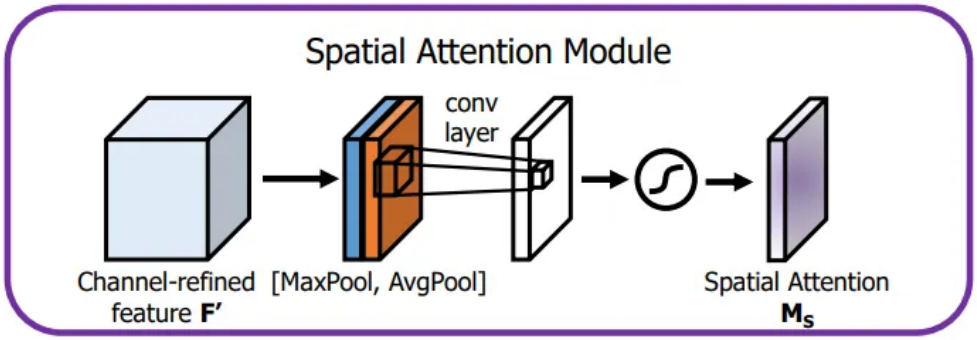

Figure 3.4: **Spatial Attention Module** [21]. We concatenate the average-pooled and max-pooled feature maps (across the channel dimension), then convolve with a $k \times k$ filter to obtain a spatial attention map $\mathbf{M}_s(\mathbf{F})$.

Predicting Future Trajectories

Let $\mathbf{F}_{\text{out}} \in \mathbb{R}^{C_{\text{out}} \times H \times W}$ be the final CNN feature map. We flatten $\mathbf{F}_{\text{out}}$ and apply one or more fully connected layers (FC$(\cdot)$) to output the predicted positions $\hat{\mathbf{p}}_i$ or offsets for each vehicle over T_{out} steps:

$$\left[\hat{\mathbf{p}}_i^{(t+1)}, \ldots, \hat{\mathbf{p}}_i^{(t+T_{\text{out}})}\right] = \text{FC}\Big(\text{Flatten}\big(\mathbf{F}_{\text{out}}\big)\Big). \tag{3.4}$$

Depending on the application, $\hat{\mathbf{p}}_i^{(t)} \in \mathbb{R}^2$ could represent (x, y) positions, or we may also predict velocity or acceleration.

Chapter 4

Experimental Evaluation

4.1 Dataset and Experimental Setup

We use the NGSIM (Next Generation Simulation) dataset [4], which contains fine-grained highway traffic data captured from surveillance cameras. The dataset provides:

$$\{(p_{i,x}^{(t)}, p_{i,y}^{(t)}), (v_{i,x}^{(t)}, v_{i,y}^{(t)}), \dots\}_{i=1\dots n,\, t=1\dots T},$$

covering multiple time intervals with different traffic densities. We select an input window of length T_{in} (e.g., 15 timesteps of trajectory history) and predict T_{out} (e.g., 25 timesteps into the future). Figure 4.1 shows one of the study area schematic and camera coverage.

The dataset is not yet in the optimal format for training. To address this, we preprocess the data by concatenating all vehicles with the same sequence length in the current frame. Any sequences that do not meet the required length of 40 frames (eg. 15 timesteps of trajectory history + 25 timesteps into the future) are discarded. Next, the remaining sequences are processed to form a fully connected graph, ensuring they are disconnected from other sequences.

Once the preprocessing is complete, The data is split as 70% training, 15% validation, and 15% testing. Additionally, the dataset is augmented using random shifting and noise injection to enhance diversity and robustness.

This experiment is carried out on a NVIDIA RTX4050 Laptop GPU with Intel Core I5. The deep learning framework of choice is Python/Pytorch with the latest version

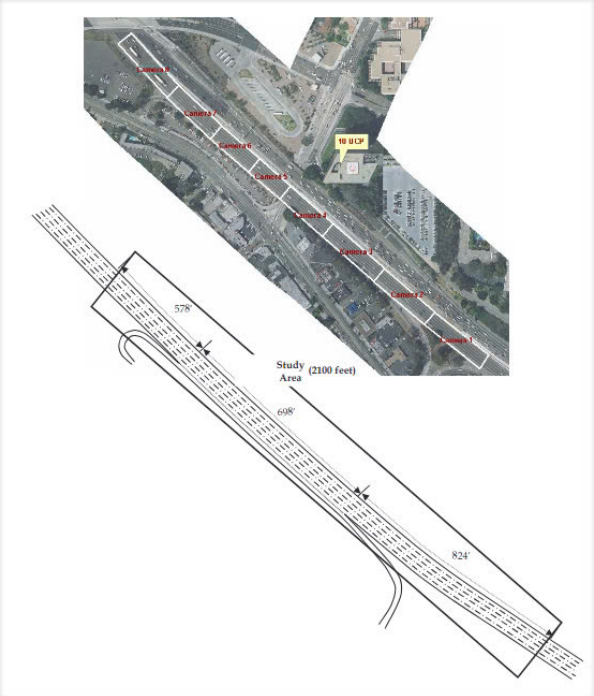

Figure 4.1: Study Area Schematic and Camera Coverage

4.2 Evaluation Metrics

To evaluate how accurately our model predicts the future positions of vehicles, we measure:

1. **Average Displacement Error (ADE):**

$$\text{ADE} = \frac{1}{n\,T_{\text{out}}} \sum_{i=1}^{n} \sum_{k=1}^{T_{\text{out}}} \sqrt{\left(\hat{p}_{i,x}^{(t_k)} - p_{i,x}^{(t_k)}\right)^2 + \left(\hat{p}_{i,y}^{(t_k)} - p_{i,y}^{(t_k)}\right)^2}\,.$$

Purpose: ADE measures the average Euclidean distance between the predicted and actual positions of each vehicle over all predicted timesteps.

What It Demonstrates:

- **Overall Tracking Accuracy:** By averaging the displacement errors across all timesteps and all vehicles, ADE provides a comprehensive view of how well the model tracks the vehicles throughout the entire prediction horizon.
- **Consistency Over Time:** It ensures that the model maintains accuracy not just at specific points but consistently across all future predictions.

Advantages:

- **Simplicity:** ADE is straightforward to compute and interpret.
- **Holistic Assessment:** It captures the general performance of the model across the entire trajectory.

Limitations:

- **Equal Weighting:** All timesteps contribute equally to ADE, which might not account for the varying importance of different prediction horizons in specific applications.

2. **Final Displacement Error (FDE):**

$$\text{FDE} = \frac{1}{n} \sum_{i=1}^{n} \sqrt{\left(\hat{p}_{i,x}^{(t_{T_{\text{out}}})} - p_{i,x}^{(t_{T_{\text{out}}})}\right)^2 + \left(\hat{p}_{i,y}^{(t_{T_{\text{out}}})} - p_{i,y}^{(t_{T_{\text{out}}})}\right)^2}.$$

Purpose: FDE calculates the Euclidean distance between the predicted and actual positions of each vehicle specifically at the final timestep of the prediction horizon.

What It Demonstrates:

- **Endpoint Accuracy:** FDE focuses on the model's ability to predict the exact final position, which is crucial for applications where the destination point is of primary interest, such as in navigation systems or collision prediction.
- **Trajectory Completion:** It assesses whether the model can accurately conclude the trajectory, ensuring that the predicted path leads to the correct endpoint.

Advantages:

- **Focus on Critical Point:** By concentrating on the final position, FDE highlights the model's performance in scenarios where the end location is more important than intermediate positions.
- **Sensitivity to Endpoint Errors:** Small deviations at the final step can significantly impact FDE, making it a stringent metric for endpoint prediction accuracy.

Limitations:

- **Limited Scope:** FDE does not account for errors at intermediate timesteps, potentially overlooking issues in the trajectory leading up to the final position.

3. **Root Mean Square Error (RMSE):**

$$\text{RMSE} = \sqrt{\frac{1}{n\,T_{\text{out}}} \sum_{i=1}^{n} \sum_{k=1}^{T_{\text{out}}} \left(\left(\hat{p}_{i,x}^{(t_k)} - p_{i,x}^{(t_k)} \right)^2 + \left(\hat{p}_{i,y}^{(t_k)} - p_{i,y}^{(t_k)} \right)^2 \right)}.$$

Purpose: RMSE computes the square root of the average of the squared differences between predicted and actual positions across all vehicles and all predicted timesteps.

What It Demonstrates:

- **Error Magnitude Sensitivity:** By squaring the errors before averaging, RMSE disproportionately penalizes larger errors, making it effective in highlighting significant deviations in predictions.
- **Overall Prediction Reliability:** It provides insight into the model's reliability, especially in avoiding substantial mistakes that could be critical in safety-sensitive applications like autonomous driving.

Advantages:

- **Emphasizes Large Errors:** RMSE is particularly useful for identifying and mitigating large prediction errors that could have severe implications.
- **Smooth Differentiable Objective:** Its mathematical properties make it suitable for optimization and gradient-based learning algorithms.

Limitations:

- **Influence of Outliers:** RMSE can be overly influenced by outliers, potentially skewing the evaluation if a few predictions have exceptionally large errors.
- **Less Intuitive Interpretation:** Compared to ADE and FDE, RMSE may be less straightforward to interpret in the context of specific application requirements.

Summary of Metrics' Contributions:

- **ADE** provides a broad assessment of the model's ability to track vehicle positions consistently over time, ensuring that predictions remain accurate throughout the entire trajectory.
- **FDE** zeroes in on the model's proficiency in predicting the final destination, which is crucial for applications requiring precise endpoint estimations.
- **RMSE** offers a nuanced evaluation by emphasizing larger errors, thereby highlighting the model's robustness and reliability in avoiding significant prediction mistakes.

By leveraging these three metrics collectively, we obtain a comprehensive evaluation of the model's predictive performance, capturing both its general accuracy and its ability to handle critical prediction aspects effectively.

4.3 Implementation Details and Training

During training, we minimize a **mean squared error** (MSE) loss between predicted and true positions across all vehicles and time steps:

$$\mathcal{L}_{\text{MSE}}(\Theta) = \frac{1}{n\,T_{\text{out}}} \sum_{i=1}^{n} \sum_{k=1}^{T_{\text{out}}} \left\| \hat{\mathbf{p}}_i^{(t_k)} - \mathbf{p}_i^{(t_k)} \right\|^2 ,$$

where Θ encapsulates all trainable parameters in the GNN encoder and CNN decoder.

Optimizer and Learning Rate Schedule. We use the **Adam** optimizer with:

- Learning rate (lr) = **0.01**
- Weight decay = **0.0005**

Additionally, we employ a **MultiStepLR** scheduler with milestones at $\{10, 30, 45\}$ and a decay factor (γ) of **0.1** to reduce the learning rate if no improvement is observed within those epochs.

Dropout and Batch Normalization.

- LAD linear dropout: **0.02**.
- LAD LinearCBAM dropout: **0.25**. This refers to the dropout rate in the combined linear + CBAM branch inside the LAD module.
- CBAM batch normalization: eps = **1e-5**, momentum = **0.01**.

These dropout layers (in both the LAD blocks and standard CNN/MLP layers) help mitigate overfitting, especially under high-density traffic scenarios. Batch normalization is applied after certain convolutional layers to stabilize training and accelerate convergence.

Layer Parameters and Activation.

- **Fully Connected (FC):** Input 30×2, Output $30 \times 2 \times 2$.
- **GINConv:** Input $30 \times 2 \times 2$, Output $30 \times 2 \times 2$.
- **Conv1:** Input channels 2, Output channels 64.
- **Conv2:** Input channels 64, Output channels $64 \times 2 = 128$.
- **Conv3:** Input channels 128, Output channels $128 \times 2 = 256$.
- **Conv4:** Input channels 256, Output channels 50.
- **CBAM Layers:** Same structure as the corresponding convolutional layers.
- **Activation:** LeakyReLU is applied to all convolutional, fully connected, and GINConv layers.

Model Size. The total number of **trainable parameters** in our model (GNN encoder + CNN decoder) is **133283**. We found that this size provides a good balance between parameter efficiency and expressiveness ensures that the model can operate effectively under high-traffic scenarios while remaining computationally feasible for deployment.

Other Hyperparameters. In addition to the core architecture, the following hyperparameters were explored to fine-tune performance:

- **Batch sizes**: $\{2, 64, 128\}$.
- **Number of GNN layers**: $\{1\}$.
- **CNN filter sizes**: $\{64, 128, 256\}$.
- **Weight initialization methods:** Xavier.

We select the combination that yields the best validation ADE while avoiding overfitting.

Once the model converges, we evaluate on the test split to measure final performance. Our design aims to balance *model complexity* (so we can capture complex interactions) with *inference speed* (to remain feasible in near-real-time traffic applications). By carefully managing the graph size (either fully connected or distance-based adjacency) and limiting the depth of the CNN stacks with attention, we ensure that the **PishguVe** [3] model can run efficiently on a modern GPU for typical highway traffic densities.

4.3.1 Model Summary (PyTorch Implementation)

The table below (Table 4.1) provides a detailed breakdown of the model architecture, outlining each layer, its output shape, and the number of parameters.

4.4 Results and Discussion

4.4.1 Comparative Evaluation

To demonstrate the effectiveness of **PishguVe** [3], we retrained the original state-of-the-art (SOTA) **Pishgu** model [16] to ensure a fair and transparent comparison. Figures 4.2–4.8 illustrate how various error metrics (RMSE at 1s, 2s, 3s, 4s, 5s, ADE, FDE) evolve over 100 training epochs, further emphasizing the improvements achieved by the enhanced model.

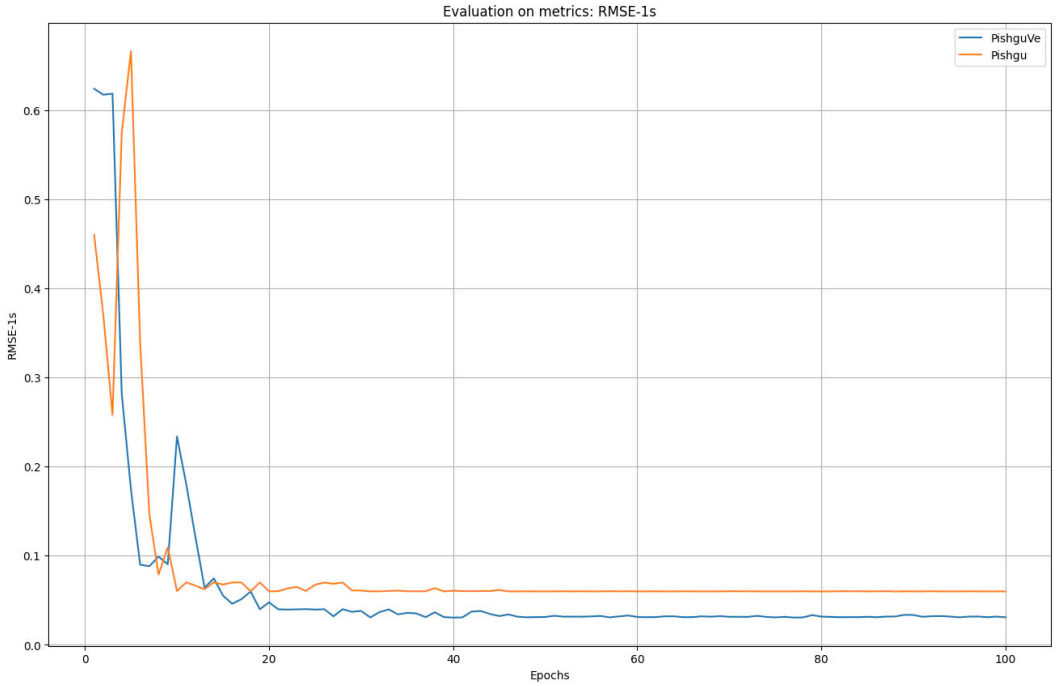

Figure 4.2: **RMSE at 1s** over epochs. PishguVe (blue) converges slightly faster and ends with a lower error floor than Pishgu (orange).

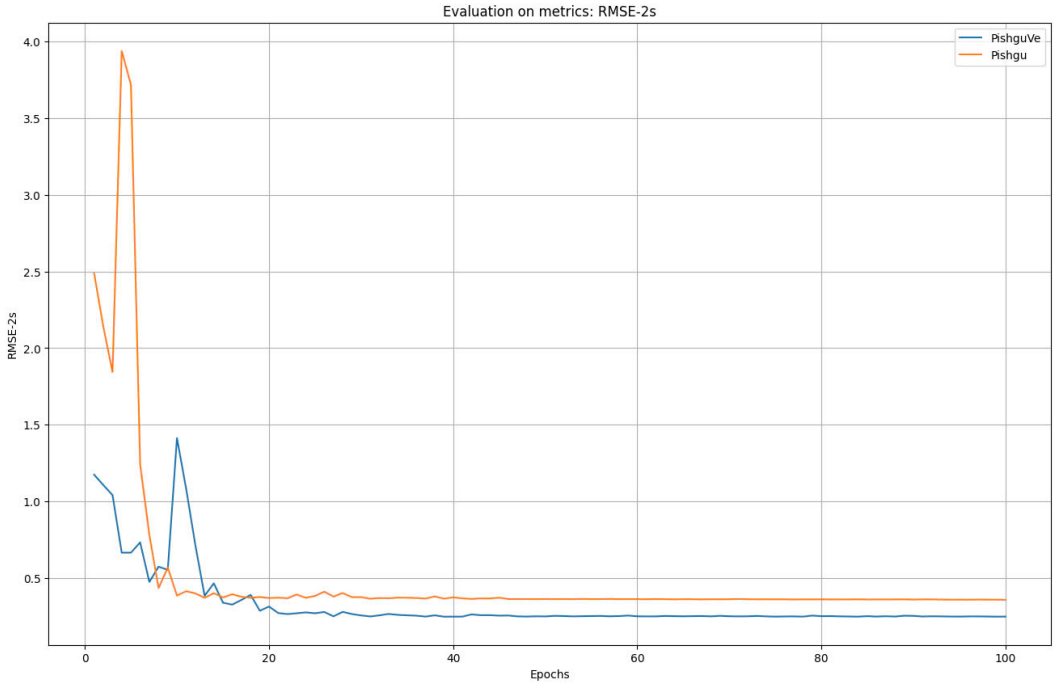

Figure 4.3: **RMSE at 2s** over epochs. Notice PishguVe's quicker drop between epochs 0–10 and its consistently lower RMSE beyond epoch 20.

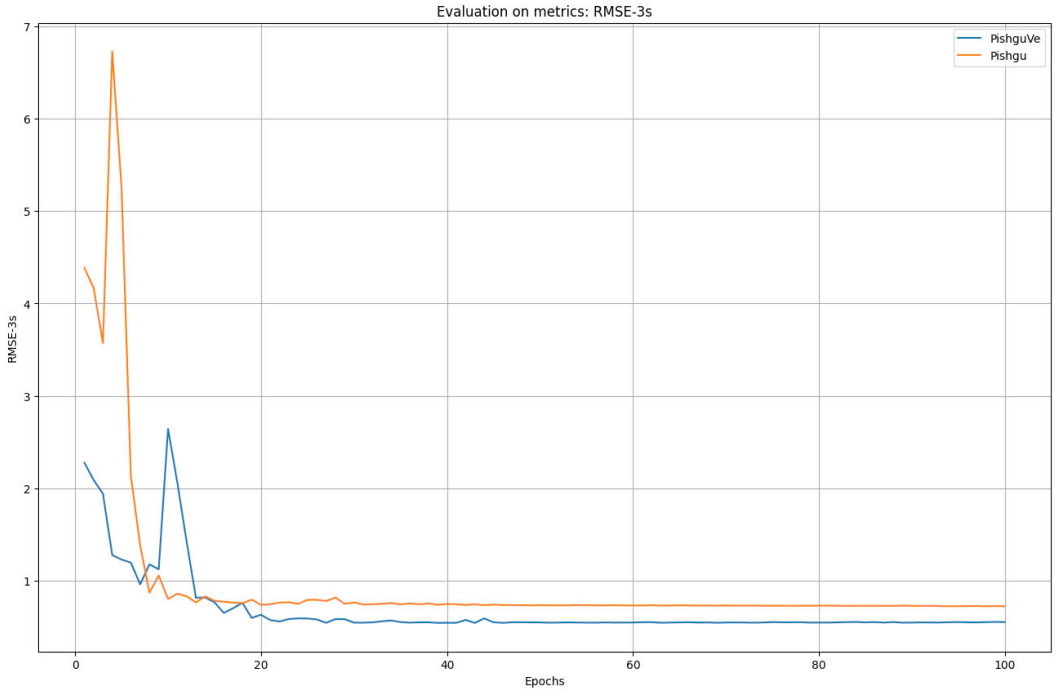

Figure 4.4: **RMSE at 3s** over epochs. Both methods eventually stabilize, but PishguVe yields a smaller final RMSE.

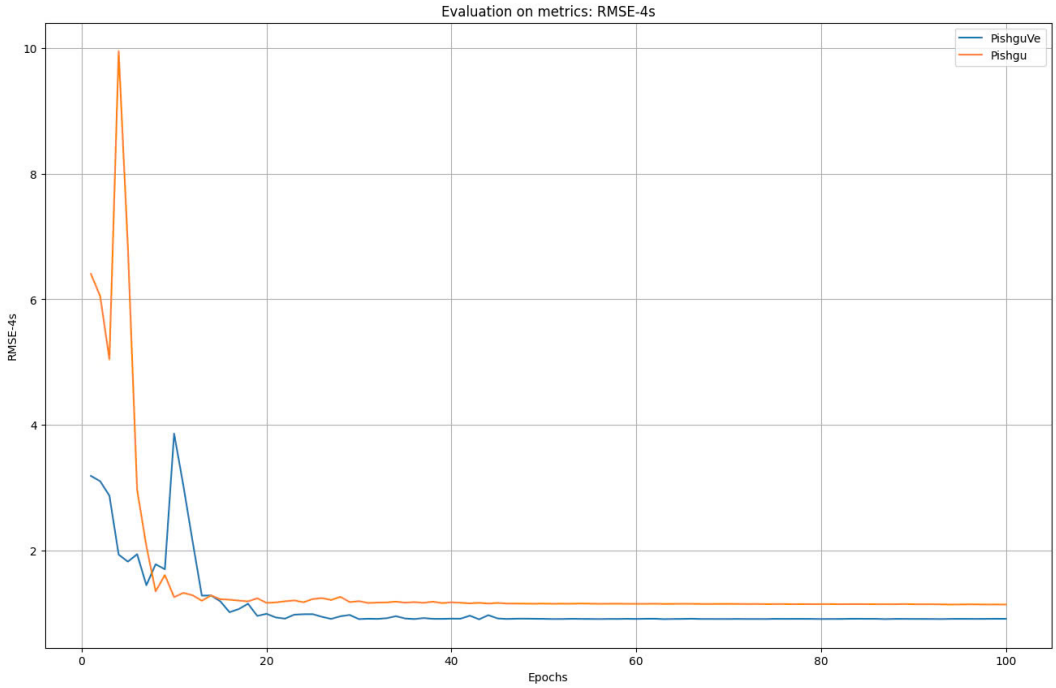

Figure 4.5: **RMSE at 4s** over epochs. PishguVe exhibits sharper initial reductions and maintains a lower error floor than Pishgu.

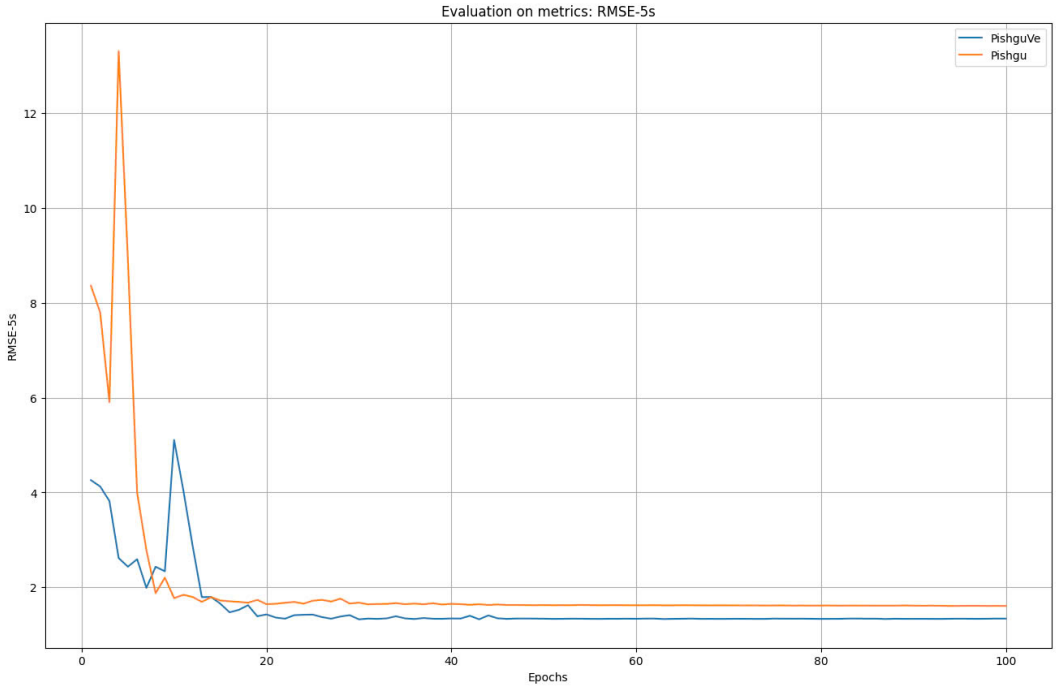

Figure 4.6: **RMSE at 5s** over epochs. PishguVe also outperforms Pishgu at longer horizons, indicating stronger long-range forecasts.

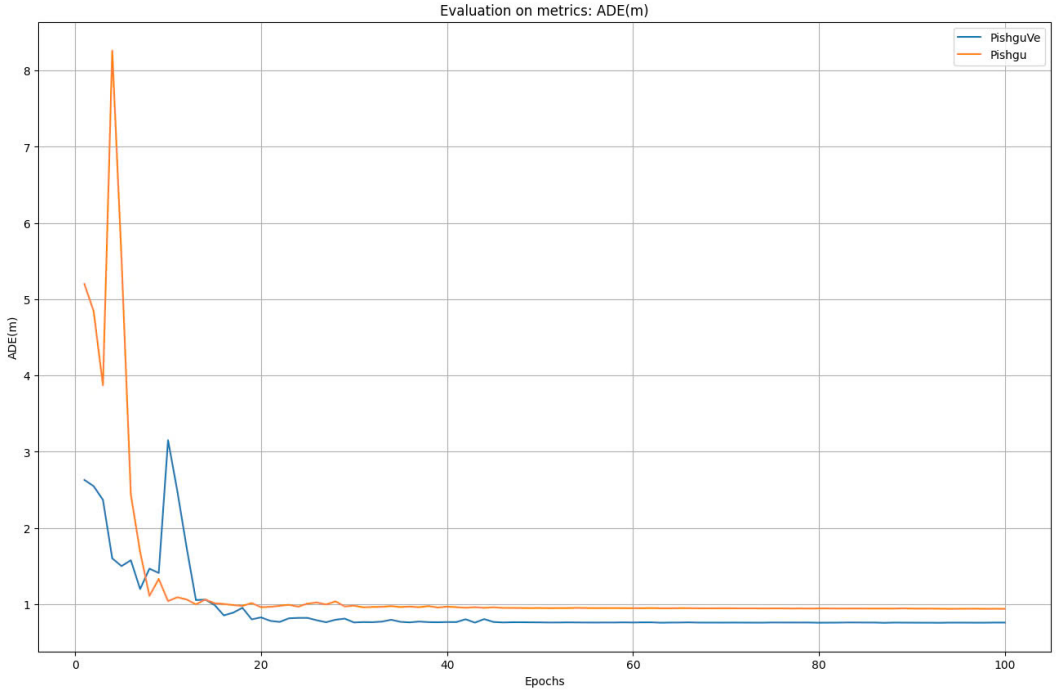

Figure 4.7: **Average Displacement Error (ADE)** over epochs. PishguVe (blue) converges more quickly and remains below Pishgu (orange) throughout training.

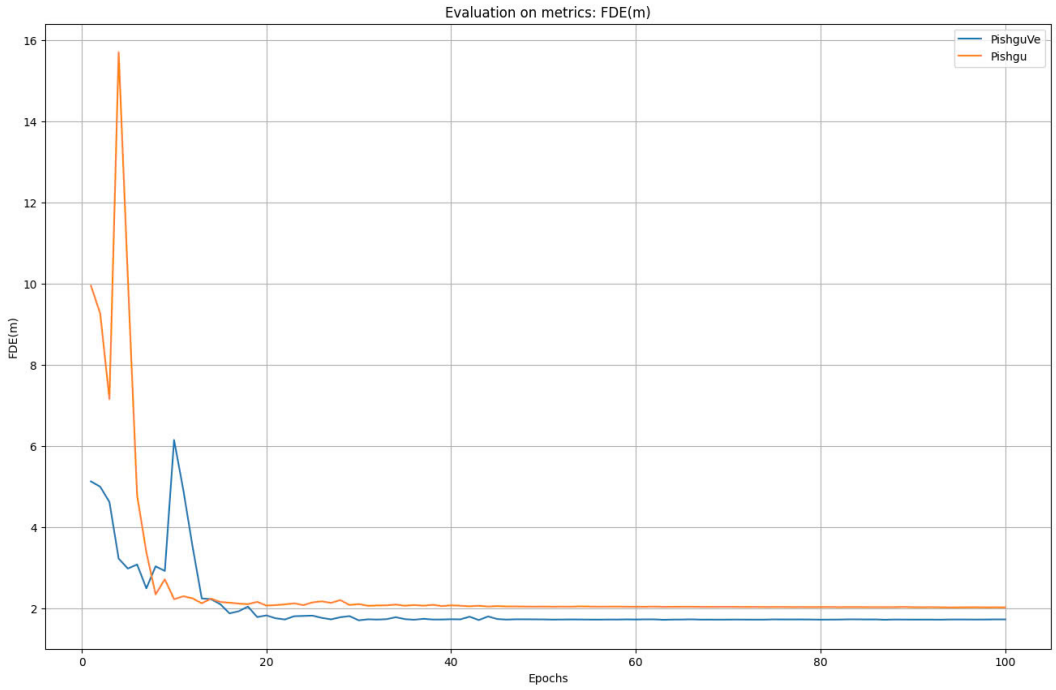

Figure 4.8: **Final Displacement Error (FDE)** over epochs. The gap between PishguVe and Pishgu becomes evident within the first 10 epochs, demonstrating PishguVe's advantage in final position accuracy.

Table 4.1: Detailed Model Summary - Layer Breakdown and Parameters

Layer (type:depth-idx)	Output Shape	Param #
NetGINConv	[6, 50, 1, 1]	–
+– Linear: 1-1	[6, 120]	7,320
+– GINConv: 1-2	[6, 30]	1
| +– SelfAggregate: 2-1	[6, 30]	–
| | +– Linear: 3-1	[6, 60]	7,260
| | +– Linear: 3-2	[6, 30]	1,830
| +– SumAggregation: 2-2	[6, 120]	–
| +– NeighborAggregate: 2-3	[6, 30]	–
| | +– Sequential: 3-3	[6, 60]	7,688
| | +– Sequential: 3-4	[6, 30]	1,926
+– Conv2d: 1-3	[6, 64, 7, 1]	576
+– CBAM: 1-4	[6, 64, 7, 1]	–
| +– ChannelGate: 2-4	[6, 64, 7, 1]	–
| | +– Sequential: 3-5	[6, 64]	580
| | +– Sequential: 3-6	[6, 64]	(recursive)
| +– SpatialGate: 2-5	[6, 64, 7, 1]	–
| | +– ChannelPool: 3-7	[6, 2, 7, 1]	–
| | +– BasicConv: 3-8	[6, 1, 7, 1]	100
+– Conv2d: 1-5	[6, 128, 3, 1]	16,512
+– CBAM: 1-6	[6, 128, 3, 1]	–
| +– ChannelGate: 2-6	[6, 128, 3, 1]	–
| | +– Sequential: 3-9	[6, 128]	2,184
| | +– Sequential: 3-10	[6, 128]	(recursive)
| +– SpatialGate: 2-7	[6, 128, 3, 1]	–
| | +– ChannelPool: 3-11	[6, 2, 3, 1]	–
| | +– BasicConv: 3-12	[6, 1, 3, 1]	100
+– Conv2d: 1-7	[6, 256, 1, 1]	65,792
+– CBAM: 1-8	[6, 256, 1, 1]	–
| +– ChannelGate: 2-8	[6, 256, 1, 1]	–
| | +– Sequential: 3-13	[6, 256]	8,464
| | +– Sequential: 3-14	[6, 256]	(recursive)
| +– SpatialGate: 2-9	[6, 256, 1, 1]	–
| | +– ChannelPool: 3-15	[6, 2, 1, 1]	–
| | +– BasicConv: 3-16	[6, 1, 1, 1]	100
+– Conv2d: 1-9	[6, 50, 1, 1]	12,850
Total params:		**133,283**
Trainable params:		**133,283**
Non-trainable params:		**0**
Total mult-adds (Units.MEGABYTES):		1.09
Input size (MB):		0.00
Forward/backward pass size (MB):		0.12
Params size (MB):		0.53
Estimated Total Size (MB):		0.66

Key Observations.

- **Rapid Convergence:** For shorter horizons (1s–2s), PishguVe's error drops noticeably faster in the early epochs.

- **Consistent Improvement:** At 3s–5s horizons, PishguVe maintains a lower RMSE than Pishgu, indicating its robustness to longer-range predictions.
- **Overall ADE and FDE Gains:** Across training, PishguVe converges to better ADE and FDE, especially beyond epoch 15, underscoring the benefits of *Linear Attention Dropout (LAD)* and the attention-based CNN decoder.

These results confirm that the proposed enhancements in **PishguVe** (LAD and attention-based CNN) contribute to faster training convergence and more accurate trajectory predictions across multiple time horizons.

4.4.2 Visualize Results

To further illustrate the predictive quality of our model, Figure 4.9 presents a snapshot of **trajectory evolution** in a multi-vehicle scenario. The plot compares the *ground-truth* trajectories (green lines) with the *predicted* trajectories (red lines) over several time steps. Blue boxes represent the vehicles' bounding boxes at a specific time index, and blue dots indicate the location at time = 6521 (in simulation timesteps, for example).

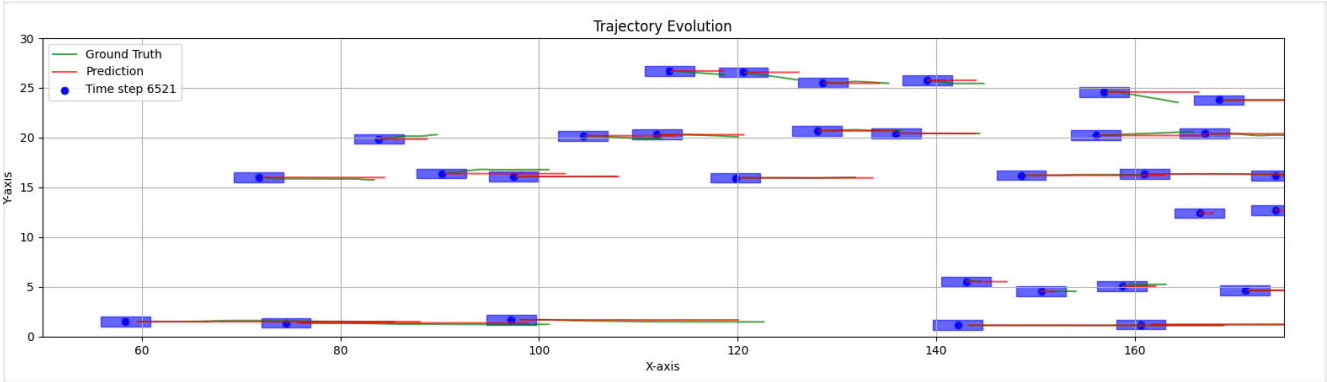

Figure 4.9: Visualize a Comparison of Predicted Trajectories for Selected Highway Segments

Observations.

- **Close Alignment**: Across lanes and various positions on the x-axis, the predicted trajectories (red) largely overlap with the ground truth (green), suggesting that the model captures key motion patterns.
- **Stable Predictions**: Even in scenarios where multiple vehicles move at different speeds and directions, the model avoids large deviations from the true paths.
- **Potential Errors**: Some minor offsets can be observed in areas with sudden lane changes or tighter spacing, indicating that the model could further benefit from additional training data in these edge cases. This is further reinforced by the fact that FDE still has large error (close to 2 meters).

These qualitative results complement the quantitative metrics (RMSE, ADE, FDE), demonstrating that our proposed **PishguVe** [3] approach yields coherent multi-vehicle trajectory forecasts in diverse traffic conditions.

4.4.3 Limitations

While the PishguVe [3] model achieves state-of-the-art performance in trajectory prediction, certain limitations remain. In high-density traffic scenarios, the model occasionally exhibits signs of overfitting, despite the use of Linear Attention Dropout (LAD) and rigorous regularization techniques.

Additionally, the model's accuracy can diminish in rare, edge-case situations such as abrupt lane changes, sudden braking, or near-collision events, primarily due to the limited availability of such examples in the training dataset. Expanding the dataset to include more diverse driving behaviors may mitigate this limitation.

In complex multi-lane environments with frequent vehicle merges and diverges, PishguVe sometimes struggles to capture subtle interactions, leading to minor deviations in long-term predictions. Incorporating more sophisticated spatial-temporal attention mechanisms or hierarchical aggregation strategies could further refine performance.

Another limitation lies in the use of CNN for the decoder. While this architecture is efficient in terms of inference time, it inherently relies on a fixed input-output timestep structure. This restricts the model's flexibility in handling dynamic or variable-length trajectory data, potentially limiting its applicability to scenarios requiring predictions over varying horizons.

Despite these challenges, the model consistently outperforms existing methods in standard highway settings, highlighting its effectiveness in capturing general driving patterns. Addressing the outlined limitations will be key to extending the model's applicability to more intricate urban and congested environments.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This report adopts the Pishguve [3] model for highway trajectory prediction, employing a GIN [17] encoder to capture inter-vehicle interactions and an attention-enhanced CNN [18] decoder to model temporal dependencies. Experimental evaluations on the NGSIM dataset indicate that the proposed approach outperforms baseline methods in terms of ADE, FDE, and RMSE. Results further suggest that combining graph-based spatial relationships with attentive neural architectures leads to improved precision across varying traffic densities, though challenges persist in exceedingly dense or rare-event scenarios.

5.2 Future Work

Moving forward, there is clear incentive to expand the scope of our study in multiple ways. First, exploring richer data augmentation strategies could help the model learn from atypical traffic events, such as abrupt stops or aggressive lane changes. Second, combining GNNs [19] with Transformer-based [23] architectures may further refine long-range temporal dependencies while scaling more naturally to dense traffic. Third, deploying and testing the model in real-time traffic systems would offer a deeper understanding of its practical performance and resilience under live conditions. Lastly, broadening the domain to urban or mixed traffic could confirm the model's capacity to handle a wider array of driving behaviors and patterns, thereby enhancing its generalizability and utility in intelligent transportation systems.

Bibliography

[1] F. H. Administration, "Traffic congestion costs," 2020. [Online]. Available: https://www.fhwa.dot.gov/publications/research/tp/hif13024/index.cfm

[2] N. H. T. S. Administration, "Traffic safety facts: 2020 data," 2021. [Online]. Available: https://www.nhtsa.gov/research-data/traffic-safety-facts

[3] V. Katariya, G. A. Noghre, A. D. Pazho, and H. Tabkhi, "A pov-based highway vehicle trajectory dataset and prediction architecture," *arXiv preprint arXiv:2303.06202*, 2023.

[4] J. Brown, T. Golob, and A. Soltesz, "The next generation simulation (ngsim) program," *Transportation Research Record*, vol. XXX, no. XXX, pp. 1–12, 2009.

[5] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[6] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1468–1476.

[9] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 961–971.

[10] H. Woo, M. Sugimoto, J. Wu, Y. Tamura, A. Yamashita, and H. Asama, "Trajectory prediction of surrounding vehicles using lstm network," in *Proceedings of the 2013 IEEE Intelligent Vehicles Symposium (IV)*, Gold Coast, QLD, Australia, 2013, pp. xx–yy.

[11] B. Kim, C. Kang, J. Kim, S. Lee, C. Chung, and J. Choi, "Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network," in *Proceedings of the 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, Yokohama, Japan, 2017, pp. 399–404.

[12] L. Lin, W. Li, H. Bi, and L. Qin, "Vehicle trajectory prediction using lstms with spatial-temporal attention mechanisms," *IEEE Transactions on Intelligent Transportation Systems*, vol. xx, no. yy, pp. zz–zz, 2021.

[13] K. Messaoud, I. Yahiaoui, A. Verroust-Blondet, and F. Nashashibi, "Attention based vehicle trajectory prediction," in *Proceedings of XYZ Conference*, Somewhere, 2020, pp. xx–yy.

[14] Z. Sheng, Y. Xu, S. Xue, and D. Li, "Graph-based spatial-temporal convolutional network for vehicle trajectory prediction in autonomous driving," 2021.

[15] F. Li and Others, "Grip: Graph-based interaction-aware trajectory prediction," arXiv preprint arXiv:210x.xxxxx, 2021.

[16] A. D. P. C. N. H. T. Ghazal Alinezhad Noghre, Vinit Katariya, "Pishgu: Universal path prediction network architecture for real-time cyber-physical edge systems," in *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)*, 2023, pp. 88–97.

[17] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations (ICLR)*, 2019.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[19] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[20] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 7132–7141.

[21] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 3–19.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.