electronics

Article

Vehicle Trajectory Prediction Based on Adaptive Edge Generation

He Ren *,† and Yanyan Zhang †

School of Electronics and Information Engineering, Nanjing University of Information Science and Technology, Nanjing 210044, China; 002243@nuist.edu.cn
* Correspondence: 202212180013@nuist.edu.cn
† These authors contributed equally to this work.

Abstract: With the rapid evolution of intelligent driving technology, vehicle trajectory prediction has become a pivotal technique for enhancing road safety and traffic efficiency. In this domain, high-definition vector maps and graph neural networks (GNNs) play a vital role, supporting precise vehicle positioning and optimizing path planning, thereby improving the performance of intelligent driving systems. However, high-definition vector maps and traditional GNNs still encounter several challenges in trajectory prediction, such as high computational resource demands, long training times, and limited modeling capabilities for dynamic traffic environments and complex interactions. To address these challenges, this paper proposes an adaptive edge generator method, this method dynamically constructs and optimizes the connections between nodes in the GNN architecture, effectively enhancing the accuracy and efficiency of trajectory prediction. Specifically, we classify nodes into dynamic and static nodes based on their attributes, and devise differentiated edge construction strategies accordingly. For dynamic nodes, we introduce a relative angle factor, enabling the attention model to comprehensively consider the distance and intersection status between nodes, resulting in more accurate computation of edge weights. For static nodes, we utilize a length threshold to assess the feasibility of establishing connections between vehicles and lane lines, determining whether a connection should be established. Through this approach, we successfully reduce the algorithmic complexity, increase computational speed, and maintain high trajectory prediction accuracy. Tests on the Argoverse motion prediction dataset demonstrate that trajectory prediction utilizing the adaptive edge generator achieves an average displacement error (ADE) of 0.6681, a final displacement error (FDE) of 0.9864, and a miss rate (MR) of 0.0952. Furthermore, the model parameters are significantly reduced, validating the effectiveness of the proposed vehicle trajectory prediction method based on the adaptive edge generator.

Keywords: vehicle trajectory prediction; graph neural network; edge construction and generation

Citation: Ren, H.; Zhang, Y. Vehicle Trajectory Prediction Based on Adaptive Edge Generation. *Electronics* **2024**, *13*, 3787. https://doi.org/10.3390/electronics13183787

Academic Editor: Andrea Bonci

Received: 15 August 2024
Revised: 18 September 2024
Accepted: 21 September 2024
Published: 23 September 2024

1. Introduction

Vehicle trajectory prediction is crucial for autonomous driving technologies [1], especially in complex traffic environments such as urban roads, highways, and intersections. Autonomous vehicles must continuously make decisions to address diverse and dynamic situations, predicting the behaviors of other vehicles and pedestrians to respond promptly and prevent traffic accidents. Accurate vehicle trajectory prediction enhances traffic flow and efficiency by optimizing route planning and speed control, reducing congestion and improving road usage efficiency. However, predicting trajectories in scenarios involving complex interactions [2], such as multiple vehicles in proximity, frequent lane changes, and intersection crossings, is particularly challenging due to increased dynamics and uncertainties. Traditional methods [3], including rule-based, statistical, and machine learning approaches, often fall short in such environments because of their limited capacity to model nonlinear relationships and dynamics, failing to meet the high demands for accuracy, robustness, and real-time performance required by autonomous driving systems [4].

In recent years, the advent of deep learning technologies has progressively become the focal point of vehicle trajectory prediction research. Deep learning-based approaches utilize high-definition (HD) scene maps [5], which offer detailed structural and geographical information about the vehicle's surroundings [6]. This enhanced environmental perception [7] aids significantly in accurate trajectory prediction. HD maps not only serve as a reference for predicting potential future paths but also facilitate the planning and control processes based on these predictions [8]. Furthermore, HD maps are instrumental for vehicle localization. By aligning the vehicle's current sensor data with map information, the system can pinpoint its exact location with greater precision, a critical factor for achieving reliable trajectory predictions [9,10].

Researchers typically transform scene maps into bird's-eye view images and analyze them using convolutional neural networks (CNNs) [11], a method that leverages general image models but is computationally demanding and offers limited perceptual range [12]. Recent studies have shifted focus towards vectorized representations of scene maps [13] for a more compact depiction of scenes. These vectorized maps extract key information from trajectories and map elements, processing these scenes through models such as graph neural networks [14] and Transformers [15]. This approach facilitates learning the relationships between vectorized entities, such as trajectory waypoints and lane segments, offering more detailed and accurate map information, thus enhancing the perceptual capabilities of autonomous driving systems.

However, in complex traffic scenarios like intersections, where lane lines, obstacles, and other elements frequently change and increase in complexity, it remains a critical challenge to determine whether autonomous systems can efficiently identify relevant objects within the map to simplify algorithm complexity and enhance computational speed. Furthermore, there is a pressing need to explore whether reducing computational load during graph construction and incorporating appropriate semantic information into the map can improve overall computational efficiency. These considerations are crucial for advancing the capabilities of autonomous driving technologies in handling dynamic and intricate traffic situations.

In graph convolutional neural networks (GCNNs), edges represent the relationships or interactions between nodes, facilitating the transmission of information and influences across the network. The method of edge construction is pivotal as it directly influences the network's representational capabilities and overall performance [16]. Traditionally, edges in graph neural networks are constructed using a fully connected graph approach. While comprehensive, this method results in complex network structures that significantly increase computational and storage demands. Moreover, fully connected graphs often struggle to capture the inherent sparsity and localized features present in traffic data, which can have a detrimental effect on model performance.

To overcome the aforementioned challenges, this paper introduces an adaptive edge generation scheme tailored for constructing graph convolutional neural networks specifically for vehicle trajectory prediction. This adaptive edge generator utilizes time series and map information as inputs to refine the traditional fully connected graph approach in graph neural networks, particularly focusing on the connections between one's own vehicle and pertinent objects. Nodes are differentiated into static and dynamic categories based on their attributes, and distinct connection strategies are employed for each type. For dynamic nodes, the attention mechanism is enhanced by incorporating a relative angle factor, enabling the model to more comprehensively assess the positional relationships between vehicles. This improvement is particularly beneficial for understanding complex traffic scenarios, such as intersections. For static nodes, edges are established based on a maximum distance criterion. The trajectory prediction method proposed in this paper not only ensures high accuracy but also reduces algorithmic complexity and enhances the computational speed of the model.

The remainder of this paper is structured as follows: Section 2 discusses related work and identifies prevailing challenges in vehicle behavior prediction. Section 3 pro-

vides a concise explanation of the adaptive edge generator's principles and outlines the prediction methodology. Section 4 details the experiments conducted and analyzes the results obtained.

2. Related Work

2.1. Map Information

In recent years, motion prediction has emerged as a central focus of autonomous driving research, particularly due to the critical role of high-definition maps in urban environments. Consequently, significant research efforts have been directed towards the effective encoding of these high-definition maps for motion prediction. The encoding approaches for motion prediction maps, which rely heavily on spatial features, are primarily categorized into rasterized and vectorized forms. The objective is to furnish the computer with a robust foundation for data analysis and statistics, and to digitize complex map features to enhance their readability and recognition by computational systems [17].

Rasterization: Rasterization techniques convert the environment surrounding an agent, typically visualized as a bird's-eye view (BEV), into an image format. This transformation redefines the task of semantically understanding the target's surroundings into a computer vision challenge, allowing the use of numerous image-based methodologies such as convolutional neural networks. Rasterized representations are particularly noted for their ability to encapsulate the spatial context of a target, such as a map, effectively. A notable study in this domain is ChauffeurNet [18], which employs recurrent neural networks (RNNs) [19] to generate predicted trajectories. ChauffeurNet displays maps, navigation data, and other objects in a BEV, focusing on a rectangular region ahead of the target. The road map is depicted as an RGB image, illustrating features like lane centerlines and curbs.

Similarly, MultiPath [20] forecasts movements based on anchor classification and offset regression of these contextual elements. Numerous studies adopt this rasterized map representation, merging it with vectorized depictions to enhance predictions of vehicle motion and interactions. Techniques such as a multi-hypothesis fully connected (FC) prediction head leverage convolutional contextual features and vectorized agent state features. CoverNet [21] innovatively generates anchor trajectories and classifies them using similar feature pairings. Techniques like multiple futures prediction [22] and multi-agent tensor fusion [17] employ RNNs to encode and decode predicted target motions, integrating contextual features at various stages.

While rasterized maps offer the advantage of partitioning map information into a grid format, simplifying computational processing and understanding, they also come with challenges. For extensive areas, rasterization can effectively provide voluminous information. However, this method suffers from the drawbacks of large data volumes, high spatial complexity, and limited adaptability in dynamic environments.

Vectorization: Vectorization techniques prioritize the topology of the map by conceptualizing it as a graph. These graphical representations are significantly more compact than raster images, which often leads to improved efficiency. Graphical representations excel at articulating complex traffic semantics due to their ability to define connectivity clearly. For instance, they can distinctly describe scenarios where two lanes are physically proximate yet separated by a median strip, preventing interaction.

Prominent predictive networks employing vectorization include VectorNet [13], which conceptualizes both the map and vehicle motion using a two-level hierarchy of fully connected graphs. Similarly, LaneGCN [23] represents the map as a heterogeneous directed graph and utilizes a parameterized graph convolutional network (GCN) to train on this structure. Following this, LaneRCNN [24] builds upon the foundation set by LaneGCN by constructing a predicted vehicle graph that models interactions within the map, enhancing the overall prediction accuracy and contextual relevance of the traffic scenarios modeled.

To summarize, vectorized maps offer several advantages over rasterized maps for trajectory prediction:

(1) **Enhanced detail and accuracy**: Vectorized maps provide more comprehensive map information, including road curvature and traffic facilities, which are vital for accurate trajectory prediction. By delivering more precise road shapes and features, vectorized maps facilitate a better understanding of road conditions, aiding vehicles in making more accurate trajectory planning and predictions.
(2) **Superior real-time performance and dynamism**: Vectorized maps are particularly advantageous in trajectory prediction scenarios that demand real-time responsiveness and dynamic adaptation. Road conditions may change due to various factors such as new traffic signs, road construction, or traffic congestion. The ease of updating vectorized maps allows for quick responses to such changes, ensuring that the maps reflect the most current information and thereby enhancing the real-time accuracy of predictions.
(3) **Improved navigation and route planning**: Vectorized maps excel in navigation and route planning, providing more precise navigation guidance and route recommendations. With a deeper understanding of the structure and characteristics of road networks, vectorized maps can offer more accurate route planning for navigation systems.

These attributes highlight the significant application potential and value of vectorized maps in intelligent transportation systems. The method of the adaptive edge generator proposed in this paper, which is based on vectorized map encoding, aligns with these advantages.

2.2. Trajectory Prediction Based on Graph Neural Networks

Within the field of trajectory prediction, traditional neural network architectures such as convolutional neural networks (CNNs) [25] and recurrent neural networks (RNNs) have demonstrated efficacy in various application scenarios. However, they exhibit notable limitations when applied to the complex task of trajectory prediction. This is primarily because CNNs and RNNs are inherently biased toward processing data with regular geometric structures. In contrast, trajectory prediction often involves handling data characterized by significant irregularities and complex spatial dynamics, such as road networks, dynamic vehicle positions, and obstacles. These challenges underscore the need for adapting or developing new neural network architectures better suited to the intricacies of trajectory data.

Graph neural networks (GNNs) offer a compelling solution for trajectory prediction by incorporating graph theoretical principles to adeptly handle data characterized by non-regular structures and dynamic interactions. The foundational strength of GNNs [14] lies in their ability to conceptualize data as a graphical structure, with nodes representing individual entities such as vehicles or pedestrians, and edges indicating the relationships or interactions between these entities. By facilitating information transfer between nodes, GNNs effectively capture complex patterns of inter-entity interactions, enabling the network to uncover deep, underlying dependencies within the data. This capability makes GNNs particularly suited to the multifaceted nature of trajectory prediction tasks.

To address the challenges posed by irregular graph data, the academic community has introduced various innovative approaches and frameworks to enhance the performance and adaptability of GNNs. Notably, Patchy-san [26] proposed a graph normalization technique that enables neighborhoods of varying sizes to be mapped into a fixed-size convolutional layer, facilitating effective information aggregation. Additionally, the GraphSAGE [27] framework leverages neighborhood sampling, employing pooling or LSTM mechanisms to aggregate these samples, thus allowing for efficient integration of vertex features. Graph convolutional networks (GCNs) excel in local information aggregation by employing convolution operations within local neighborhoods, leveraging truncated Chebyshev polynomial approximations to achieve this. Meanwhile, message-passing neural networks (MPNNs) [28] provide a comprehensive framework for graph-based architectures, highlighting the importance of message-passing mechanisms in summarizing and integrating features across various graph network models. These advancements significantly improve GNNs' ability to manage and interpret complex and irregularly structured data.

While scholars have developed a range of solutions for GCNs to handle irregular graph data, significant challenges remain in applying these techniques to dynamic graphs, such as those used in trajectory prediction. Dynamic graphs are characterized by temporal changes in their structure or attributes, including the addition or removal of nodes, the formation or dissolution of edges, and the updating of node or edge attributes. This dynamic nature complicates the direct application of traditional GCN methods, which generally assume a static graph structure throughout training. When changes occur in the graph's structure or attributes, it necessitates the recalculation of the convolution operation for the entire graph, substantially increasing the computational cost. Moreover, efficiently integrating newly emerged nodes and edges into the dynamic graph poses additional challenges, as traditional GCN methods may struggle to effectively manage such updates. Consequently, effectively adapting to dynamic changes in graph structures and capturing complex interactions within the environment represent significant hurdles in processing large-scale dynamic graph data with GNNs.

2.3. Summary of Trajectory Prediction Problems Based on Graph Neural Networks

In the field of trajectory prediction, although significant strides have been made with the introduction of RNNs, GNNs, and attention mechanisms to address vehicle trajectory prediction, certain issues remain either overlooked or unresolved.

(1) **Optimization of graph connectivity**: Traditional methods often employ fully connected graphs for vehicle trajectory prediction, ensuring that all nodes are engaged in the graph convolution process. However, this approach does not reflect the reality where many nodes might not significantly influence the target trajectory. For instance, within the Argoverse dataset [29], numerous nodes do not contribute directly to trajectory generation, such as lane line nodes that are distant from the predicted vehicle, which have minimal impact on the prediction outcomes. As illustrated in Figure 1, out of 11 nodes in the graph, only 7 are related to trajectories. Utilizing a fully connected graph that includes all 11 nodes undeniably adds to the complexity and computational time of the algorithm. Even when only considering the seven trajectory-related nodes, those that are distant from the predicted vehicle still have limited influence on the prediction results. Thus, a more precise screening of nodes that significantly impact prediction outcomes is crucial to optimizing both the performance and efficiency of the algorithm.

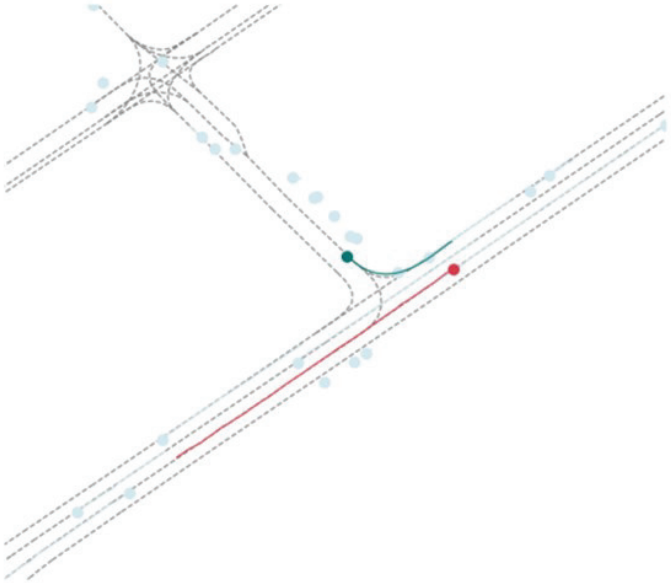

Figure 1. Map of selected map scenes and vehicle locations in the Argoverse dataset.

(2) **Enhanced focus on edges**: Traditionally, research has predominantly focused on nodes, with edges primarily viewed as mere connectors and tools for representing neighborhood relationships during graph convolution. This perspective overlooks the significant role that edges play within the graph. Edges do more than connect nodes [30]; they also serve as conduits for storing and transferring information, playing a crucial role in facilitating communication between nodes. Therefore, fully leveraging the function of edges in graph convolution processes is critical for enhancing the performance and accuracy of algorithms and represents a novel research direction in this paper.

(3) **Refined edge construction**: In the realm of edge construction, the traditional graph attention network (GAT) approach typically calculates edge weights based on the distance between nodes. This method, however, fails to accurately represent the weight relationships between vehicles in specific scenarios. For instance, at intersections, the relevance of one vehicle to another depends not only on their distance but also on their relative angles and other factors. Relying solely on distance for determining edge weights can lead to reduced accuracy and robustness of the prediction model. This paper advocates for a more nuanced approach that considers multiple factors in edge construction to better capture the complex dynamics of vehicle interactions, particularly in challenging environments like intersections.

To address the aforementioned challenges, this paper introduces a dynamic vehicle trajectory prediction model based on adaptive edge generation. This approach mitigates the issues associated with the large computational demand and high complexity of fully connected graphs by implementing varied connection schemes for nodes with different attributes. This method allows for a more efficient and targeted processing of graph data, significantly enhancing the performance and accuracy of trajectory prediction while reducing computational overhead.

3. Trajectory Prediction Based on Adaptive Edge Generation

The dynamic vehicle trajectory prediction method based on adaptive edge generation constructs a new graph structure by generating first-order subgraphs of predicted vehicles to other nodes by adopting two different edge connection strategies for static nodes and dynamic nodes to form adaptively connected edges, which can more fully consider the positional relationships between vehicles, improve the model's comprehension of scenarios such as intersections, and reduce the computational complexity of the heterogeneous graph structure between predicted vehicles and lane lines.

3.1. Overall Workflow

The prediction of future trajectories of vehicles based on graph neural networks is achieved by first vectorizing the traffic map and the time-series trajectory coordinates of vehicles in a given high-resolution map scenario, where the historical trajectories of the vehicles and the high-resolution maps are known.

In the vectorized scenario, any time point is defined as the map

$$G = \{\gamma, \varepsilon\} \tag{1}$$

where γ denotes a point and ε denotes an undirected edge. We address the heterogeneous nature of graphs in vehicle trajectory prediction, where the semantic information contained in the edges between vehicles, and between vehicles and lane lines, differs significantly. The paper introduces a method to first differentiate the static parts of the graph into dynamic nodes and static nodes, utilizing different edge connection schemes based on their attributes. The overarching architecture of our approach is depicted in Figure 2 and encompasses three primary modules: the vectorization module, the encoder module, and the decoder module, described as follows:

(1) **Vectorization module**: This module is responsible for the vectorization of vehicle trajectory and map information, transforming these elements into a format suitable for further processing and analysis.
(2) **Encoder module**: Here, the adaptive edge generator strategy is introduced. For dynamic nodes, this involves calculating the relative positions between the vehicle (node) to be predicted and other vehicles (nodes), assigning weights based on these relative positions through an attention mechanism, and connecting dynamic nodes based on these weight values. For static nodes, a connection strategy that limits the length between vehicles and static elements is implemented for edge connections.
(3) **Decoder module**: A multilayer perceptron (MLP) decoder is employed to achieve accurate trajectory prediction. This module translates the encoded graph data back into predicted vehicle trajectories, ensuring precision and reliability in the output.

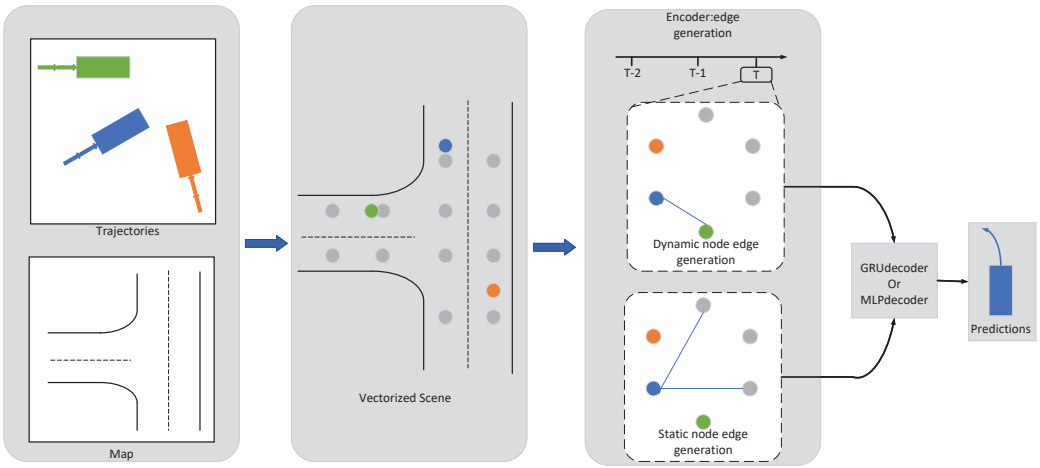

Figure 2. General block diagram of adaptive edge generator; dynamic nodes include dynamic information such as vehicles, and static nodes include static information such as lane lines; these generate edges and weights through different rules.

The innovation of this paper lies in the adaptive edge generator (shown in Figure 3) within the encoder module, which categorizes nodes into dynamic and static types, adopting tailored edge connection strategies for each. The input includes the vehicle's movement direction, calculated from the previous and current time steps, along with the vehicle's coordinates. Additionally, the hidden layer processes the relative angles between vehicles and the hidden information from the previous layer. The hidden layer distinguishes between dynamic and static nodes, determines how to connect the nodes, and uses the decoder to predict the final output. The advantages of this approach are as follows:

(1) **Enhanced model flexibility and adaptability**: Dynamic nodes represent the current positions of vehicles, while static nodes correspond to fixed road sections or landmarks. By distinguishing between these node types, the model can more efficiently handle rapidly changing road conditions, enhancing both the flexibility and adaptability of the prediction model.
(2) **Reduced computational complexity**: Traditional reliance on fully connected graphs can obscure meaningful inter-node relationships and increase computational demands. By processing dynamic and static nodes separately, this approach reduces the graph's connection density, thereby decreasing computational complexity and enhancing both the efficiency and scalability of the algorithm.
(3) **Improved accuracy and stability**: The relationships between vehicles differ from those between vehicles and road markers, such as lane lines, introducing a level of heterogeneity that complicates graph convolution computations. By implementing different connection strategies and dynamically adjusting based on node type and

semantic information, the model can more accurately capture the associations between vehicle trajectories, thus improving the accuracy and stability of predictions.

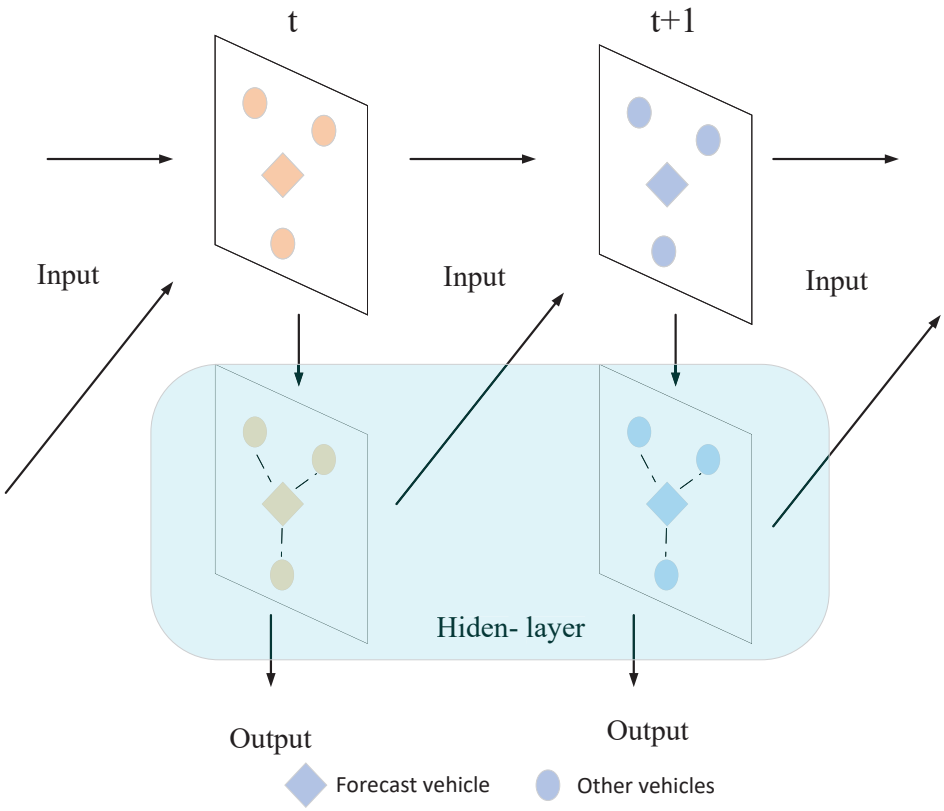

Figure 3. The input includes the vehicle's movement direction, calculated from the previous and current time steps, along with the coordinates of the vehicle and other objects. Additionally, the hidden layer processes the relative angles between vehicles and the hidden information from the previous layer. The hidden layer decides how to connect the nodes and uses the decoder to predict the final output.

3.2. Scene Construction

Vectorized traffic scene construction usually focuses on the absolute position of vehicles or lane lines in the scene, and does not consider the relative positions of predicted vehicles and other vehicles, which can better measure the relative motion relationship between vehicles and is crucial for real-time scene sensing and dynamic traffic management. To address this issue, this paper introduces the relative angle factor between vehicles in order to better model the spatial attributes of predicted vehicles in a dynamic traffic scene, as shown in Figure 4. The details are as follows:

In the trajectory prediction of a vehicle, the positional feature information of vehicle i at time t is firstly expressed as $\left\{p_i^t - p_i^{t-1}\right\}^T$, where p_i^t is the feature information of vehicle i and T is the history time of the vehicle. For the lane line ξ, its geometric attributes are given by $p_\xi^1 - p_\xi^0$, where p_ξ^1 and p_ξ^0 are the end and start coordinates of ξ, respectively. The relative position information of vehicle i and vehicle j at time t is defined as $p_i^t - p_j^t$, and the relative position information includes the relative angle factor, distance, speed relationship, and other information between the two vehicles.

In order to represent the relative spatial relationship between vehicle i and vehicle j, as shown in Figure 4, the relative angle factor between the two vehicles is defined as θ_{ij}^t. In this paper, the east direction is defined as the positive direction of the x-axis, and θ_i^t, θ_j^t is the direction information of the vehicle, at time t, then the relative angle between the two

vehicles at time t is defined as $\theta_{ij}^t = \left|\theta_i^t - \theta_j^t\right| (\theta_{ij}^t \in (-\frac{\pi}{2}, \frac{\pi}{2}))$. According to Equation (2), the relative angle value can be calculated.

$$\theta_{ij}^t = \arctan(\frac{x^t - x^{t-2}}{y^t - y^{t-2}}) \tag{2}$$

As can be seen from Equation (2), θ_{ij}^t describes the spatial relationship and the relative position of two vehicles, which can be used to model the spatial attributes between the predicted vehicles and other vehicles in a dynamic traffic scenario. In the subsequent dynamic node edge generation module, the relative angle factor is used to calculate the attentional weights between the vehicles and realize the edge connections of the dynamic nodes based on the weight values.

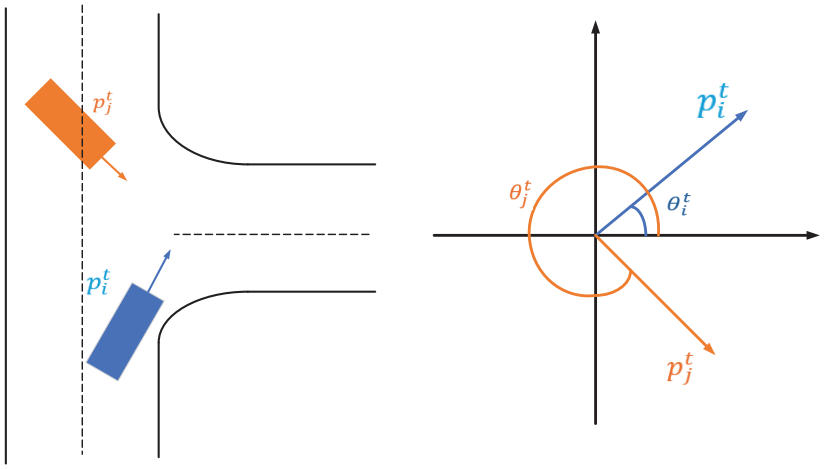

Figure 4. Vehicle scenario construction diagram for some cases.

3.3. Adaptive Edge Generator

3.3.1. Dynamic Node Edge Generation Based on Relative Angles

Dynamic nodes usually refer to mobile objects such as vehicles moving on the road. For vehicle trajectory prediction, the positions and speeds of dynamic nodes change over time, and their dynamics and changing trends need to be considered in trajectory prediction, so the edge connection scheme for dynamic nodes needs to consider their node characteristics.

GAT is a commonly used graph neural network model that is widely used for trajectory prediction tasks. GAT can learn the dynamic relationships between nodes (e.g., vehicles or pedestrians) in a traffic network and predict the motion trajectories [31]. In the GAT model, nodes denote traffic vehicles and edges denote interactions between nodes. Through a multilayer attention mechanism, GAT can dynamically learn the importance between each node and its neighboring nodes in the graph and perform information aggregation and trajectory prediction accordingly [32]. The learning formula is as follows:

$$e_{ij} = \phi_{rel}(Wp_i^t, Wp_j^t) \tag{3}$$

$$a_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})} \tag{4}$$

In Equations (3) and (4), W is the learnable matrix, ϕ_{rel} is the activation function, which is generally a two-layer multilayer perceptron in a graph neural network, after calculating the weights of the edges, the operation of normalization is carried out and its required attention is calculated according to Equation (5), and the overall formula is as follows:

$$a_{ij} = softmax(\frac{F_{t-1}^Q (F_{t-1}^K)^T}{\sqrt{d_k}}) F_{t-1}^V \tag{5}$$

F_{t-1}^{Q}, F_{t-1}^{K}, F_{t-1}^{V} is a linear transformation of the p_i^{t-1} feature at the previous moment. It can be seen that Equation (5) focuses on the influence of distance d_k on the attention coefficient, while in the actual traffic scene, distance is not the only key factor. And under the GAT formula, every point is involved in the calculation of the graph, which leads to problems such as distraction of attention and increase in calculation cost.

In this paper, we address this problem by introducing the relative position factor on the basis of the above GAT equation. The relative position information between vehicles is integrated into the calculation process of the attention weight according to Equations (6) and (7), and the angular relationship between the dynamic vehicles is added into the model as a characteristic factor. The specific formulas are as follows:

$$Z_i^t = \phi_{rel}[(p_i^t - p_i^{t-1}), a_i] \tag{6}$$

$$Z_{ij}^t = \phi_{rel}[(p_i^t - p_i^{t-1}), (p_j^t - p_j^{t-1}), a_i] \tag{7}$$

where ϕ_{rel} is the activation function, which is generally a two-layer multilayer perceptron in a graph neural network, then the three values of Q, K, and V in attention are

$$F_i^Q = W^Q z_i^t \qquad F_{ij}^K = W^K z_{ij}^t \qquad F_{ij}^V = W^V z_{ij}^t$$

where W^Q, W^K, and W^V are learnable matrices, according to Equation (8). The weights of the edges are

$$a_{ij}^t = softmax[\frac{F_i^Q (F_{ij}^K)^T}{\sqrt{d_k}} + A(F_i^Q, F_{ij}^K)]F_{ij}^V \tag{8}$$

where $A(F_i^Q, F_{ij}^K)$ is the angular characterization as defined in Equation (9):

$$A(F_i^Q, F_{ij}^K) = F_i^Q \cdot (F_{ij}^K)^T \cdot |\sin\theta_{ij}^t| \cdot W^\theta \tag{9}$$

where W^θ is the learnable matrix, $\sin\theta_{ij}^t$ is the role of the relative angle factor, according to the understanding of the remote interaction between vehicles; when the relative angle between the two vehicles is smaller (such as 0—the vehicles are parallel at this time), the attention required at this time is smaller; while when the relative angle factor between the two vehicles is larger (such as $\frac{\pi}{2}$), the attention required is larger.

The original Transformer model is optimized and improved by introducing the angle characteristics between vehicles. In the original model, the attention mechanism only relies on the variable of distance, which makes it difficult to pay full attention to the intersection state of vehicles. The introduction of the relative angle factor enables the model to more fully consider the positional relationship between vehicles, and the weight of the edges is thus improved, which in turn improves the model's understanding of scenarios such as intersections as a way to increase the accuracy and robustness of trajectory prediction.

As shown in Figure 5, three vehicles can be observed from the figure, green vehicle A, blue vehicle B, and orange vehicle C. Vehicle A is closer to vehicle B. According to the traditional attention mechanism, when calculating the attentional weights of the edges of vehicle B that needs to be predicted, the weights of vehicle A and vehicle B will be heavier than those of vehicle B and vehicle C. Vehicle A is closer to vehicle B than vehicle B is to vehicle C. Vehicle A is closer to vehicle B than vehicle B and vehicle B. However, in real traffic scenarios, vehicle B needs to pay more attention to the trajectory of vehicle C to avoid possible collisions and adjust its position to obtain a reasonable turning path. The figure shows that the angle between vehicle A and vehicle B is approximately $\theta_1 = \frac{\pi}{4}$, and $|\sin\theta_1| = \frac{\sqrt{2}}{2}$, the angle between vehicle B and vehicle C is approximately $\theta_2 = \frac{\pi}{2}$, $|\sin\theta_2| = 1$. By calculating the angle of vehicle B with vehicle C and vehicle A and adding it to the attention mechanism, their attention weights can be adjusted and the attention between vehicle b and vehicle C is increased by a moderate amount.

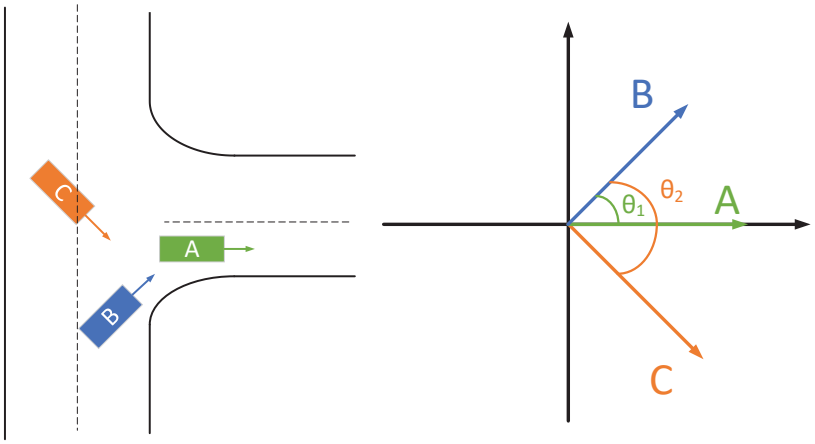

Figure 5. Schematic of dynamic edge generation for specific scenarios.

After calculating the weights of the edges a_{ij}^t, a first-order subgraph centered on the vehicles to be predicted is generated according to Equations (10)–(12) as follows:

$$m_i^t = \sum_{j \in N_i} a_{ij}^t \cdot F_i^t \tag{10}$$

$$g_i^t = sigmoid(W^{gate}[Z_i^t, m_i^t]) \tag{11}$$

$$\widetilde{Z}_i^t = g_i^t \odot W^{self} Z_i^t + (1 - g_i^t) \odot m_i^t \tag{12}$$

where N_i is the set of neighbors of the predicted vehicles, W^{gate}, W^{self} is a learnable matrix that fuses the weight features m_i^t between the vehicles and their own features Z_i^t through a *sigmoid* activation function, and finally, connects the edges through a selection gate function [33], and the output is $\widetilde{Z}_i^t$. The graph generated after filtering through the gate function is a first-order adjacency graph centered on the predicted target. With the gate function, the number of edges can be controlled and the rationality of the generated edges can be increased.

Graph neural networks suffer from state-space explosion when dealing with large-scale data. This is because graph convolutional computation in graph neural network graphs is achieved by nodes aggregating information from the domain as a means of updating their own nodes, and a linear increase in the number of nodes and edges in the graph may lead to an exponential increase in computation. The dynamic edge generation module limits the number of edges generated by updating the physical state of the vehicle (e.g., speed, relative angle, etc.) and filters the number of edges that need to be generated by applying a gate function to compute the weight of the edges in order to compute the dynamic part of an object such as a vehicle. If the number of connected edges is reduced, the number of updates is greatly reduced and the amount of floating point operations and training time required is correspondingly reduced, thus reducing the state-space explosion problem. In the subsequent experimental section, the floating point operations and testing times before and after applying the method are compared. Also, a comparison graph comparing the number of updated nodes and total points is given.

3.3.2. Static Node Edge Generation Based on Length Thresholding

In traffic scenarios, static nodes are objects or locations with a fixed position in space, which are objects or locations that do not move, such as road junctions, buildings, and traffic signs. Unlike dynamic nodes, the positions of static nodes do not change over time, so their motion characteristics usually do not need to be considered in trajectory prediction or other related tasks. In static nodes, local map information, such as the position and direction of lane lines, has a greater impact on the future trajectory of the vehicle; i.e., local map information can represent the future intention of the vehicle. According to the above

characteristics, in the edge generation of static nodes, the distance between the static node and the predicted object is an important factor for the vehicle trajectory prediction, and the length threshold is adopted as the strategy for the edge generation of the static node; i.e., the maximum length of the connected edges between the vehicle and the lane lines is measured to decide whether to connect the edges between the vehicle and the lane lines or not. The specific steps are as follows:

First, the relative position information between the vehicle and the map lane lines is calculated, as shown in Equation (13):

$$z_{i\xi} = \phi_{lane}[(p_{\xi}^{1} - p_{\xi}^{0}), (p_{\xi}^{0} - p_{i}^{T}), a_{\xi}] \tag{13}$$

ϕ_{lane} is the MLP encoder of the lane segment, $(p_{\xi}^{1} - p_{\xi}^{0}), (p_{\xi}^{0} - p_{i}^{T})$, and a_{ξ} are the start position, end position, and feature vector of the lane segment ξ, respectively. The spatio-temporal features of the predicted vehicles are used as query inputs [34] and the MLP-encoded lane segment features are used as key/value inputs, and the weights between the vehicles and lane lines are calculated.

Next, a hyperparameter threshold L is used to measure the maximum length of the connecting edges between the vehicle and the lane line; l indicates the length between the vehicle and the lane line. When $l < L$, the weight b_{ij} between the vehicle and the lane line is computed by softmax, and the edges of the nodes of the vehicle and the lane line are generated based on the weight b_{ij} between the vehicle and the lane line, and the first-order subgraph centered on the predicted vehicle is built.

Compared with the traditional GAT, this static edge generation strategy determines the edge connection relationship between vehicles and lane lines by setting a certain length threshold. This static edge connection strategy based on a length threshold has obvious advantages in computational efficiency. In addition, the use of hyperparameters can flexibly control the distance of the connecting edges between vehicles and lane lines, thus controlling the number and distance of the generated static edges, which further optimizes the performance of the prediction model.

3.4. MLP Decoder and Loss Function

In this paper, a two-layer MLP is used as a tool for decoding. The MLP receives as input the representation of the edge generator and outputs the position h_i^t of the vehicle in the local coordinate frame that needs to be predicted for each future time step and its associated uncertainty c_i^t.

In the subsequent experimental sections, the experimental results of the commonly used GRU decoder [35], MLP decoder [36], and LSTM decoder [37] are compared to verify the performance of the MLP decoder.

Diversity loss was used to promote diversity in trajectory predictions, and the training process was optimized only on the best prediction in each prediction. Prior to optimization, the error between the model-predicted mixture component and the ground truth position was first calculated for each predicted vehicle at each time step. Subsequently, the errors for all time steps were summed to form a matrix of dimension [F; N]. Based on this matrix, the trajectory with the smallest error in each predicted vehicle was selected, i.e., the minimum value of each column in the error matrix was found. The final loss function consists of a regression loss L_{reg} and an equally weighted classification loss L_{cls}:

$$L = L_{reg} + L_{cls} \tag{14}$$

In practice, the optimal balance between these terms may vary depending on the specific task or dataset. Different scaling of the two terms could affect model performance.

Overweighting the regression loss might improve the trajectory prediction but reduce the model's ability to promote diverse trajectory predictions. Overweighting the classification loss could encourage more diverse trajectories but might compromise the accuracy of the predicted positions. The equal weighting is empirically optimal [38].

A negative log-likelihood [39] was used as the regression loss:

$$L_{reg} = -\frac{1}{NH}\sum_{i=1}^{N}\sum_{t=T+1}^{T+H}\log P((p_i^t - p_i^T)|\hat{u}_i^t, \hat{b}_i^t) \tag{15}$$

$P(\cdot|\cdot)$ in Equation (15) is the probability density function of the Laplace distribution, u_i^t and b_i^t are the location and uncertainty of the best predicted trajectory of vehicle i. We optimize the mixing coefficients using the cross-entropy loss as the classification loss.

4. Experimental Results and Analysis

In order to verify the accuracy and lightness of this design in vehicle trajectory prediction, experiments are conducted on the Argoverse dataset, and the ablation experiments analyze the impact on model performance of different modules, and the experimental results prove the effectiveness of the method in this paper. In the comparison experiments, our method outperforms some current mainstream algorithms in terms of comprehensive performance. Finally, the corresponding visualization analysis is carried out to show the experimental results more intuitively.

4.1. Datasets and Evaluation Indicators

The Argoverse motion prediction dataset provides detailed trajectory data and high-definition map information for vehicles. The dataset comprises 323,557 real-world driving scenarios, which are divided into training (205,942 samples), validation (39,472 samples), and test (78,143 samples) sets.

The trajectory data in the Argoverse dataset mainly include the x- and y-coordinates of vehicles, which are sampled at 0.1 s intervals. The dataset also contains high-precision map information that provides details about roads, lane boundaries, stop signs, traffic signals, and other related features. All training and validation scenarios consist of 5 s sequences sampled at 10 Hz, which corresponds to 50 time steps per scenario. For the test set, only the first 2 s (i.e., 20 time steps) of the trajectory data are publicly available, and the task requires predicting the vehicle's motion for the subsequent 3 s (i.e., 30 time steps). This setup mimics real-world applications, where only partial observations are available and the goal is to predict future trajectories.

To ensure the quality and accuracy of the data used in our experiments, the dataset applied rigorous filtering methods to remove missing or noisy data points. For instances with minor missing values, linear interpolation was used to maintain the continuity of the trajectory. This cleaning process ensured that only the most relevant and challenging driving scenarios were included, providing a robust testing ground for motion prediction algorithms.

Performance metrics for motion prediction typically include minimum average displacement error (MinADE), minimum final displacement error (MinFDE), and missing rate (MR), which allow the model to make six predictions for each trajectory. MinADE is the average distance (in meters) between the best predicted trajectory and the ground truth trajectory for all future time steps; MinFDE is the future error for the final time step, the best predicted trajectory is defined as the trajectory with the smallest endpoint error; and MR is the ratio of scenarios in which the distance between the ground truth endpoint and the best predicted endpoint is greater than 2 m.

Let there be a predicted trajectory P and a true trajectory F, where a trajectory is a series of positional points in a time series. For each time step t, the predicted location is $P_t = (p_{tx}, p_{ty})$; p_{tx} represents the predicted x -coordinate of the vehicle's position at time step t, and p_{ty} represents the predicted y-coordinate of the vehicle's position at time step t. The true location is $F_t = (f_{tx}, f_{ty})$; f_{tx} represents the true x-coordinate of the vehicle's position at time step t, and f_{ty} represents the true y-coordinate of the vehicle's position at time step t. N represents the total number of time steps used in the trajectory prediction; the average displacement error (ADE) is defined as

$$ADE = \frac{1}{N}\sum_{t=1}^{N}\sqrt{(p_{tx}-f_{tx})^2+(p_{ty}-f_{ty})^2} \tag{16}$$

the final displacement error (FDE) is defined as

$$FDE = \sqrt{(p_{tx}-f_{tx})^2+(p_{ty}-f_{ty})^2} \tag{17}$$

MR is the missing rate, which is a measure of the number of true targets that the model fails to correctly predict or detect as a proportion of the total number of true targets, and is defined as follows:

$$MR = \frac{\text{Number of real targets not detected}}{\text{Total number of real targets}} \tag{18}$$

4.2. Experimental Environment and Hyperparameter Settings

In this paper, we use the AdamW optimizer to train on RTX 4070 super GPUs and the VectorNet encoder for data preprocessing. The experimental environment is shown in Table 1.

Table 1. Experimental environment.

Configuration	Parameter
Operating System	Linux
CPU	Intel i5-13600KF
Memory	32 G
GPU	NVIDIA RTX4070 super
Software Platform	Python 3.10, PyTorch 2.2.0, CUDA 12.1

The learning rate determines the step size for parameter updates. A higher learning rate can lead to unstable training, where parameters oscillate around the optimal value and fail to converge. In most deep learning tasks, especially for trajectory prediction involving complex multi-modal data, a small learning rate (3×10^{-4}) tends to work well. Both weight decay and dropout rate serve as regularization techniques to prevent overfitting. A dropout rate of 0.2 and weight decay of 1×10^{-4} is a reasonable balance, constraining the model enough to avoid overfitting while still allowing it to adapt to complex patterns in the data. The use of 64 epochs and a batch size of 32 in deep learning models is not arbitrary but has been validated in a wide range of experiments across various domains, including trajectory prediction.

The choice of a two-layer edge generator and one-layer decoder suggests that the model designers aimed to keep the model architecture simple while ensuring it effectively captures both local and global interactions in trajectory prediction. Fewer layers reduce computational complexity and increase model efficiency, while having enough layers ensures the model can still capture complex interaction patterns. This setup is a compromise between computational cost and model complexity. The radius of the region at the number of static node connections $L = 20$ m. The parameters are set as shown in Table 2.

Table 2. Experimental parameterization.

Parameter Setting	Parameter
Epoch	64
Batch size	32
Initial learning rate	3×10^{-4}
Weight decay	1×10^{-4}
Dropout rate	0.2
L	20 m

4.3. Ablation Experiment

In this study, ablation experiments are conducted on the Argoverse motion prediction dataset to assess the contribution of each module to the overall prediction performance. By systematically removing one component at a time—specifically the dynamic edge generation module, the static edge generation module, and the map information—we evaluate the effectiveness of each part. Each ablation configuration is trained and tested three times to ensure stability and reliability in the results. The mean and standard deviation of each performance metric (ADE, FDE, and MR) are presented in Table 3 to reflect the consistency of the outcomes.

Table 3. Ablation experiment.

Dynamic Edge Generation Module	Static Edge Generation Module	Map Information	ADE (mean ± std)	FDE (mean ± std)	MR (mean ± std)
	√	√	0.902 ± 0.015	1.574 ± 0.022	0.138 ± 0.005
√		√	0.681 ± 0.012	1.039 ± 0.018	0.102 ± 0.004
√	√		0.73 ± 0.010	1.16 ± 0.020	0.122 ± 0.003
√	√	√	0.663 ± 0.009	0.973 ± 0.013	0.094 ± 0.002

As can be seen from Table 3, each module improves performance to some extent. After removing the dynamic edge generation module, the structure of the model is basically similar to that of VectorNet, and therefore, has the lowest accuracy. After removing the static edge generation module, the model performance remains basically unchanged because the principle of static edge generation is similar to that of GAT, except that a range threshold is added as a way to reduce the complexity of the update ratio In addition, the map information is crucial in motion prediction, and the accuracy decreases significantly after removing the map information.

In this paper, an experiment is also performed to compute the number of floating point numbers as well as the computation time to verify the effect of the adaptive edge generation module of this paper on the model's floating point computation volume and test time, as shown in Table 4.

Table 4. Floating point computation volume and test time.

Dynamic Edge Generation Module	Floating Point Capacity	Time Required for Individual Scenario Testing	GPU Memory
√	0.7 M	0.04 s	5.2 G
	2.6 M	0.08 s	8.4 G

As can be seen from Table 4, the experimental data show that there is a significant reduction in the floating point computation of the model after the introduction of the dynamic edge generation mechanism. Further, in the comparison experiments for the testing time of individual scenarios, the method proposed in this paper also shows some advantages compared to the traditional fully connected graph method.

In addition, this paper provides an in-depth analysis of the impact of the edge generation mechanism on the node update strategy in graph neural networks (GNNs), due to the core mechanism of GNNs aggregating neighborhood information to update node states through graph convolution operations, and the inherent characteristics of scene independence and randomness in dynamic scenes, e.g., generation also exhibits significant randomness and dynamics characteristics. Therefore, by counting the ratio of the number of updated nodes to the total number of nodes in a single scene, it is possible to intuitively quantify the number of edges generated under each scene, as shown in Figure 1, from which it can be seen that, compared to the inherent pattern that all nodes participate in updating (i.e., the number of updated nodes equals the number of summarized points) under the traditional fully connected graph architecture, this paper's experiments have

been exhaustively analyzed for 2242 independent dynamic scenes. The results show that the average number of total nodes in these scenarios is 55.21, while the average number of update nodes is only 28.07, which is much lower than the update ratio in the fully connected graph. This illustrates the effectiveness of the dynamic edge generation mechanism in this paper in reducing unnecessary node updates and optimizing the allocation of computational resources, and shows the advantages and efficiency enhancement of this paper's method in coping with complex dynamic graph processing tasks.

In order to deeply analyze the node update characteristics of each independent scenario, we adopted a sampling analysis method, randomly selected 40 sample scenarios to explore, and compared the number of summary points with the number of updated nodes, as shown in Figure 6.

In the bar chart presented in Figure 6, the blue bar represents the number of summary points in a single scene, while the red bar corresponds to the number of nodes actually updated in that scene. The comparative analysis shows that the number of updated nodes is generally significantly lower than the total number of summary points in different dynamic scenarios.

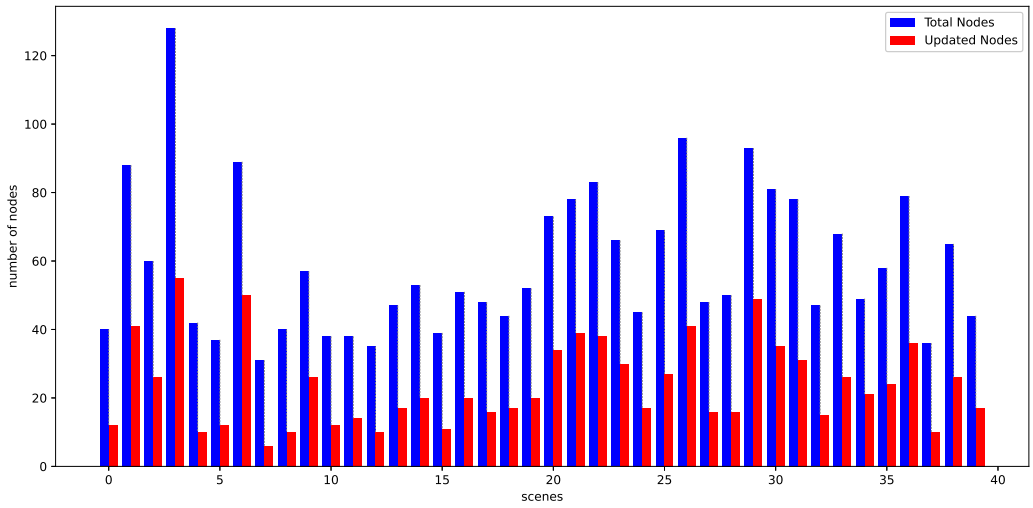

Figure 6. The number of total nodes compared with the number of updated nodes.

4.4. Decoder Module Comparison

Since vehicle trajectory prediction is characterized by complex tasks, a large amount of data, and high real-time requirements, this paper compares and analyzes the performance of the MLP decoder and the GRU decoder in terms of the number of parameters and accuracy, and the results are shown in Table 5.

Table 5. Comparison of MLP, GRU, and LSRTM decoders' performance.

	ADE	**FDE**	**Param.**
MLP Decoder	0.668	0.986	2360 K
GRU Decoder	0.663	0.973	3156 K
LSTM Decoder	0.665	0.965	4820 K

From Table 5, it can be seen that the GRU and LSTM decoders need more resources to calculate the gate states in terms of the number of parameters. In this paper, the test is carried out on the 4070 super platform; using the ordinary MLP decoder, the video memory used is about 5.2 G, and the epoch time is about 40 min; if the GRU decoder is used, the video memory used is about 8 G, and the epoch time is about 50 min; meanwhile, if the LSTM decoder is used, the video memory used is about 10 G, and the epoch time is about 60 min. The latter two encoders do not show stronger performance compared to

the ordinary MLP decoder while increasing the parameters and running cost. Therefore, this paper finally adopts the MLP encoder with relatively faster decoding speed and lower number of parameters.

4.5. Baseline Models

In this section, we provide detailed descriptions of the baseline models used for comparison on the Argoverse dataset, as shown in Table 6. These models represent state-of-the-art approaches to motion prediction tasks, each with different architectures and parameter complexities.

Table 6. Comparison with models on other Argoverse datasets.

Model	ADE	FDE	MR	Param.
VectorNet (baseline)	0.9260	1.8623	0.2736	12,651 K
Scene Transformer	0.8026	1.2321	0.1255	15,296 K
LaneGCN	0.8679	1.3640	0.1634	3710 K
DenseTNT	0.8817	1.2815	0.1258	1130 K
Ours	**0.6681**	**0.9864**	**0.0952**	**2360 K**

4.5.1. VectorNet [13] (Baseline)

VectorNet is a graph-based motion prediction model designed for autonomous driving tasks. It represents each object's trajectory as a series of vectorized points and uses a hierarchical graph neural network (GNN) to capture both local interactions between objects and global scene information. The model uses the form of a fully connected graph and aggregates information from the entire scene to predict future trajectories. In this study, VectorNet is used as a baseline, demonstrating the effectiveness of graph neural networks in trajectory prediction tasks.

4.5.2. Scene Transformer [40]

Scene Transformer leverages a Transformer-based architecture to model interactions between objects in a scene. By using attention mechanisms, it captures the importance of different entities in the scene for motion prediction. Transformers have been widely adopted in sequence modeling tasks due to their ability to capture long-range dependencies.

4.5.3. LaneGCN [23]

LaneGCN is specifically designed to capture lane-level interactions between vehicles in driving environments. It uses a graph convolutional network (GCN) to represent the lane structure and vehicle interactions. LaneGCN performs particularly well in scenarios with dense traffic, as it can effectively model the lane topology and vehicle dynamics.

4.5.4. DenseTNT [41]

DenseTNT is an advanced motion prediction model that extends the TNT (Target-Driven Trajectory Prediction) framework by incorporating dense goal candidates. This model predicts future trajectories by first identifying potential goal points, and then, generating feasible trajectories that lead to these goals. DenseTNT leverages both trajectory and goal prediction tasks to enhance accuracy.

4.6. Comparison with Advanced Methods

As demonstrated in Table 6, our proposed method outperforms the baseline models, including VectorNet, Scene Transformer, LaneGCN, and DenseTNT, across all key metrics such as ADE, FDE, and MR.

Specifically, our method shows significant improvements across key metrics while maintaining a much smaller model size compared to other approaches.

VectorNet: VectorNet leverages vector representations for trajectory prediction but struggles with accuracy over extended sequences. It achieves an average displacement error (ADE) of 0.9260, which is considerably higher than our model's 0.6681, indicating lower prediction accuracy over time. In terms of miss rate (MR), VectorNet has a rate of 0.2736, much higher than our method's 0.0952, demonstrating our model's robustness in reducing missed targets. VectorNet's parameter count is 926 K, making it significantly larger than our model's 2360 K, yet our method outperforms it in both accuracy and efficiency.

Scene Transformer: Scene Transformer relies on attention mechanisms to capture long-range dependencies, excelling in modeling global interactions. However, it comes with a large parameter count of 15,290 K, which increases computational overhead. Scene Transformer's ADE is 0.8026 and final displacement error (FDE) is 1.2815, both higher than our model's (ADE 0.6681, FDE 0.9864), indicating that our method offers better trajectory accuracy and final position prediction. Additionally, Scene Transformer's MR is 0.1255, while our model's MR is only 0.0952, showing our approach's superior performance in reducing missed targets while using significantly fewer parameters (2360 K).

LaneGCN: LaneGCN uses graph structures to model lane interactions effectively but suffers from high complexity and a large parameter count of 3710 K. LaneGCN achieves an ADE of 0.8679 and an FDE of 1.2815, both of which are higher than our method's performance. Moreover, LaneGCN's MR is 0.1255, further indicating its limitations in handling complex interactions as efficiently as our model. Our approach combines dynamic edge generation with static edge information, resulting in higher predictive accuracy while being more parameter-efficient.

DenseTNT: DenseTNT focuses on goal-directed trajectory prediction but sacrifices accuracy in multi-modal scenarios due to its dense candidate sampling strategy. It records an FDE of 1.2815, significantly higher than our model's 0.9864. While DenseTNT performs well in goal-oriented predictions, its inability to manage diverse interactions in dynamic environments limits its overall performance. In contrast, our method handles multi-modal interactions effectively, maintaining superior accuracy in final position prediction. DenseTNT's parameter count is 1281.5 K, which is still higher than the 2360 K used by our method.

In summary, our model demonstrates superior performance across ADE (0.6681), FDE (0.9864), and MR (0.0952) while keeping the parameter size minimal at 2360 K. This efficiency makes our model well suited for dynamic, multi-modal trajectory prediction in resource-constrained environments, outperforming other models that require significantly more parameters.

4.7. Visualization of Results

The results of vehicle trajectory prediction on the Argoverse dataset are shown in Figure 7. The red trajectory represents the known trajectory of the vehicle in the first two seconds given by the dataset, which is the historical trajectory; the green trajectory represents the real trajectory of the vehicle in the last three seconds; the blue trajectory represents the predicted trajectory using the method of this paper; and the black trajectory represents the predicted trajectory using the VectorNet method. The prediction of the adaptive edge generator method can accurately predict the turning performance of the vehicle as well as the acceleration performance of the vehicle, which are difficult to realize in the VectorNet method. Overall, the method in this paper shows a great improvement in the accuracy of prediction compared to VectorNet.

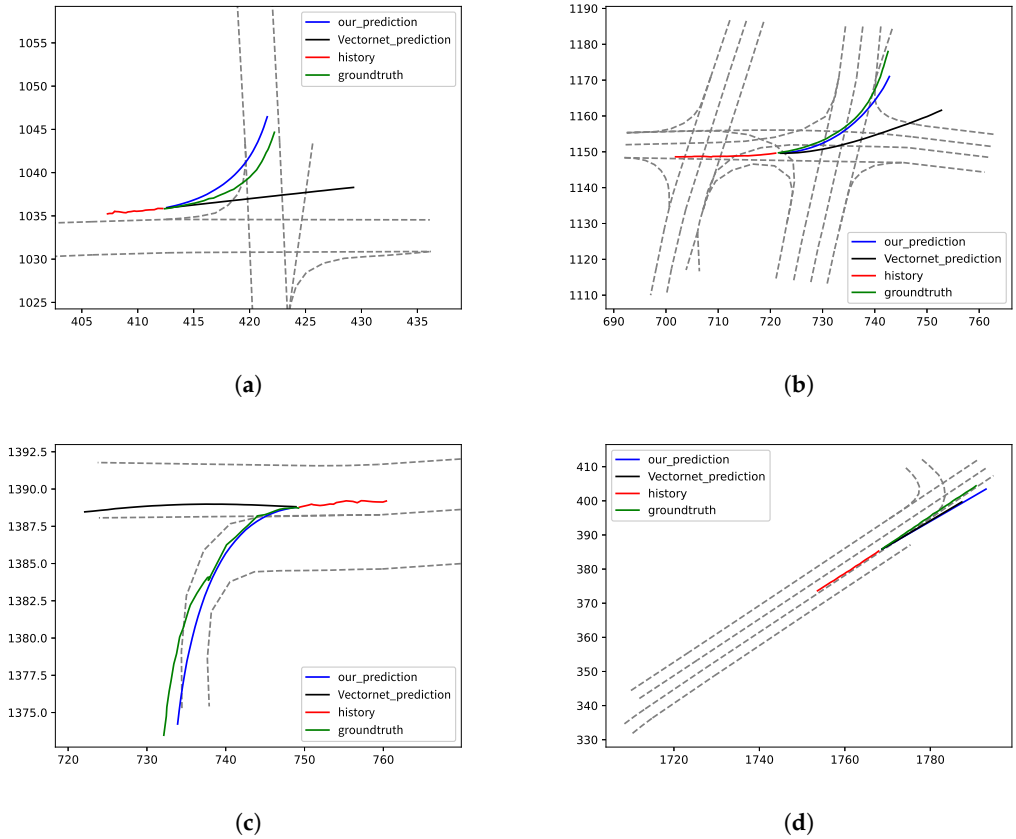

Figure 7. Qualitative results of the adaptive edge generator, with past trajectories shown in red, real trajectories in green, the method in this paper in blue, and the VectorNet method in black. Compared with the VectorNet method, the method proposed in this paper can successfully predict the turning and acceleration performance of complex intersections. (**a**) Predicted intersection turn; (**b**) prediction of complex intersections; (**c**) predicted intersection turn; (**d**) prediction of acceleration.

5. Conclusions and Future Work

Vehicle trajectory prediction in dynamic traffic scenarios is challenging due to inadequate modeling of dynamic traffic elements and complex interactions and requiring high computational complexity and a long training time. Traditional graph neural networks (GNNs) are inefficient in dealing with these complexities due to their reliance on fully connected graphs, which cannot effectively represent the sparsity and localization of traffic data.

In this paper, we introduce an adaptive edge generator that dynamically constructs and optimizes connected edges between nodes in high-definition vectorized maps by fusing map features and applying differentiated edge construction strategies to different types of nodes. For dynamic nodes, this paper generates first-order subgraphs for nodes that need to be predicted for edge connection. In the calculation of the weights of the connected edges, different from the traditional Transformer module, the angle factor is added to make the weights of the generated dynamic edges more reasonable. And through the gating function, the number of generated edges is controlled. For static nodes, this paper adopts the method of limiting the length of edges for edge connection. The method is realized within the framework of a traditional graph neural network, which reduces the complexity of the algorithm and improves the computational speed.

Tested on the Argoverse motion prediction dataset, this method shows superior performance compared to existing techniques, achieving a lower average displacement error (ADE), final displacement error (FDE), and missing rate (MR), indicating high prediction accuracy and stability. The adaptive edge generator effectively solves the core problem

of vehicle trajectory prediction, demonstrating its potential for practical application in intelligent driving systems.

Future research could focus on further optimizing the interaction between dynamic and static nodes. While the current approach utilizes relative angles to generate edges between dynamic nodes, exploring more complex features such as rate of speed changes or environmental factors could enhance prediction accuracy. Additionally, for static nodes, moving beyond distance-based thresholds to incorporate more geographical features could better capture the relationships between trajectories and the environment. Moreover, integrating multi-modal data, including camera images, LiDAR, and radar data, can significantly improve robustness and accuracy. This would be particularly beneficial for autonomous driving and intelligent traffic systems, where a more comprehensive understanding of the environment is critical for accurate vehicle trajectory predictions.

Author Contributions: All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by H.R. and Y.Z. The first draft of the manuscript was written by H.R. and Y.Z. and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding: The work was supported by National Key Research and Development Program of China (Grant number 2022YFB4401301) and National Natural Science Foundation of China, General Program (Grant number 62272234).

Data Availability Statement: Some or all data, models, or code generated or used during the study are available from the corresponding author by request.

Acknowledgments: The authors would like to express their sincere gratitude to Hanfei Cui, a Ph.D. student at University College London (UCL), for his invaluable support and insightful discussions that greatly contributed to the development of this manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Shadrin, S.S.; Ivanova, A.A. Analytical review of standard Sae J3016 «taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles» with latest updates. *Avtomobil'. Doroga. Infrastruktura.* **2019**, *3*, 10.
2. Dagli, I.; Breuel, G.; Schittenhelm, H.; Schanz, A. Cutting-in vehicle recognition for ACC systems-towards feasible situation analysis methodologies. In Proceedings of the IEEE Intelligent Vehicles Symposium, Parma, Italy, 14–17 June 2004; IEEE: New York, NY, USA, 2004; pp. 925–930.
3. Huang, Y.; Du, J.; Yang, Z.; Zhou, Z.; Zhang, L.; Chen, H. A survey on trajectory-prediction methods for autonomous driving. *IEEE Trans. Intell. Veh.* **2022**, *7*, 652–674. [CrossRef]
4. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
5. Cui, H.; Radosavljevic, V.; Chou, F.C.; Lin, T.H.; Nguyen, T.; Huang, T.K.; Schneider, J.; Djuric, N. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; IEEE: New York, NY, USA, 2019; pp. 2090–2096.
6. Bao, Z.; Hossain, S.; Lang, H.; Lin, X. High-definition map generation technologies for autonomous driving. *arXiv* **2022**, arXiv:2206.05400.
7. Casas, S.; Gulino, C.; Liao, R.; Urtasun, R. Spatially-aware graph neural networks for relational behavior forecasting from sensor data. *arXiv* **2019**, arXiv:1910.08233.
8. Ou, C.; Karray, F. Deep learning-based driving maneuver prediction system. *IEEE Trans. Veh. Technol.* **2019**, *69*, 1328–1340. [CrossRef]
9. Fernández-Llorca, D.; Biparva, M.; Izquierdo-Gonzalo, R.; Tsotsos, J.K. Two-stream networks for lane-change prediction of surrounding vehicles. In Proceedings of the 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), Rhodes, Greece, 20–23 September 2020; IEEE: New York, NY, USA, 2020; pp. 1–6.
10. Schmidt, J.; Jordan, J.; Gritschneder, F.; Dietmayer, K. Crat-pred: Vehicle trajectory prediction with crystal graph convolutional neural networks and multi-head self-attention. In Proceedings of the 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 23–27 May 2022; IEEE: New York, NY, USA, 2022; pp. 7799–7805.
11. Hong, J.; Sapp, B.; Philbin, J. Rules of the road: Predicting driving behavior with a convolutional model of semantic interactions. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 8454–8462.

12. Chai, Y.; Sapp, B.; Bansal, M.; Anguelov, D. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. *arXiv* **2019**, arXiv:1910.05449.
13. Gao, J.; Sun, C.; Zhao, H.; Shen, Y.; Anguelov, D.; Li, C.; Schmid, C. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 11525–11533.
14. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Philip, S.Y. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 4–24. [CrossRef]
15. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30* .
16. Liu, Q.; Xu, S.; Lu, C.; Yao, H.; Chen, H. Early recognition of driving intention for lane change based on recurrent hidden semi-Markov model. *IEEE Trans. Veh. Technol.* **2020**, *69*, 10545–10557. [CrossRef]
17. Zhao, T.; Xu, Y.; Monfort, M.; Choi, W.; Baker, C.; Zhao, Y.; Wang, Y.; Wu, Y.N. Multi-agent tensor fusion for contextual trajectory prediction. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 12126–12134.
18. Bansal, M.; Krizhevsky, A.; Ogale, A. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv* **2018**, arXiv:1812.03079.
19. Lipton, Z.C.; Berkowitz, J.; Elkan, C. A critical review of recurrent neural networks for sequence learning. *arXiv* **2015**, arXiv:1506.00019.
20. Varadarajan, B.; Hefny, A.; Srivastava, A.; Refaat, K.S.; Nayakanti, N.; Cornman, A.; Chen, K.; Douillard, B.; Lam, C.P.; Anguelov, D.; et al. Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction. In Proceedings of the 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 23–27 May 2022; IEEE: New York, NY, USA, 2022; pp. 7814–7821.
21. Phan-Minh, T.; Grigore, E.C.; Boulton, F.A.; Beijbom, O.; Wolff, E.M. Covernet: Multimodal behavior prediction using trajectory sets. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 14074–14083.
22. Tang, C.; Salakhutdinov, R.R. Multiple futures prediction. *Adv. Neural Inf. Process. Syst.* **2019**, *32*.
23. Liang, M.; Yang, B.; Hu, R.; Chen, Y.; Liao, R.; Feng, S.; Urtasun, R. Learning lane graph representations for motion forecasting. In Proceedings of the Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, 23–28 August 2020; Proceedings, Part II 16; Springer: Berlin/Heidelberg, Germany, 2020; pp. 541–556.
24. Zeng, W.; Liang, M.; Liao, R.; Urtasun, R. Lanercnn: Distributed representations for graph-centric motion forecasting. In Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 27 September–1 October 2021; IEEE: New York, NY, USA, 2021; pp. 532–539.
25. Chua, L.O. *CNN: A Paradigm for Complexity*; World Scientific: Singapore, 1998; Volume 31.
26. Niepert, M.; Ahmed, M.; Kutzkov, K. Learning convolutional neural networks for graphs. In Proceedings of the International Conference on Machine Learning, PMLR, New York, NY, USA, 19–24 June 2016; pp. 2014–2023.
27. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. *Adv. Neural Inf. Process. Syst.* **2017**, *30*.
28. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural message passing for quantum chemistry. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 1263–1272.
29. Chang, M.F.; Lambert, J.; Sangkloy, P.; Singh, J.; Bak, S.; Hartnett, A.; Wang, D.; Carr, P.; Lucey, S.; Ramanan, D.; et al. Argoverse: 3d tracking and forecasting with rich maps. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 8748–8757.
30. Rong, Y.; Huang, W.; Xu, T.; Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv* **2019**, arXiv:1907.10903.
31. Mo, X.; Huang, Z.; Xing, Y.; Lv, C. Multi-agent trajectory prediction with heterogeneous edge-enhanced graph attention network. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 9554–9567. [CrossRef]
32. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. *arXiv* **2017**, arXiv:1710.10903.
33. Lee, D.D.; Pham, P.; Largman, Y.; Ng, A. Advances in neural information processing systems 22. *Technol. Rep.* **2009** .
34. Zhou, Z.; Ye, L.; Wang, J.; Wu, K.; Lu, K. Hivt: Hierarchical vector transformer for multi-agent motion prediction. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 8823–8833.
35. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv* **2014**, arXiv:1412.3555.
36. Tolstikhin, I.O.; Houlsby, N.; Kolesnikov, A.; Beyer, L.; Zhai, X.; Unterthiner, T.; Yung, J.; Steiner, A.; Keysers, D.; Uszkoreit, J.; et al. Mlp-mixer: An all-mlp architecture for vision. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 24261–24272.
37. Bahdanau, D. Neural machine translation by jointly learning to align and translate. *arXiv* **2014**, arXiv:1409.0473.
38. Zhao, H.; Gao, J.; Lan, T.; Sun, C.; Sapp, B.; Varadarajan, B.; Shen, Y.; Shen, Y.; Chai, Y.; Schmid, C.; et al. Tnt: Target-driven trajectory prediction. In Proceedings of the Conference on Robot Learning, PMLR, London, UK, 8–11 November 2021; pp. 895–904.
39. Yao, H.; Zhu, D.l.; Jiang, B.; Yu, P. Negative log likelihood ratio loss for deep neural network classification. In Proceedings of the Future Technologies Conference (FTC) 2019, San Francisco, CA, USA, 25–26 October 2019; Springer: Berlin/Heidelberg, Germany, 2020; Volume 1, pp. 276–282.

40. Ngiam, J.; Caine, B.; Vasudevan, V.; Zhang, Z.; Chiang, H.T.L.; Ling, J.; Roelofs, R.; Bewley, A.; Liu, C.; Venugopal, A.; et al. Scene transformer: A unified architecture for predicting multiple agent trajectories. *arXiv* **2021**, arXiv:2106.08417.
41. Gu, J.; Sun, C.; Zhao, H. Densetnt: End-to-end trajectory prediction from dense goal sets. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Montreal, QC, Canada, 11–17 October 2021; pp. 15303–15312.