Real-time Graph-based Interaction-aware Trajectory Prediction for Autonomous Vehicles

Bachelor Thesis in Computer Science

by

Aung Kyaw Tun

CONSTRUCTOR UNIVERSITY

School of Computer Science and Electrical Engineering

Thesis Supervisor: Prof. Dr. Jakob Suchan

August 31, 2024

Statutory Declaration

Family Name, Given/First Name	*Aung Kyaw Tun*
Matriculationnumber	*30005733*
What kind of thesis are you submitting: Bachelor-, Master- or PhD-Thesis	*Bachelor Thesis*

English: Declaration of Authorship

I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however.

The Thesis has been written independently and has not been submitted at any other university for the conferral of a PhD degree; neither has the thesis been previously published in full.

German: Erklärung der Autorenschaft (Urheberschaft)

Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung.

Diese Arbeit wurde in der vorliegenden Form weder einer anderen Prüfungsbehörde vorgelegt noch wurde das Gesamtdokument bisher veröffentlicht.

10.05.2024, AungKyawTun
....

Date, Signature

ABSTRACT

As autonomous vehicles are entering the market more rapidly with advancements in Machine Learning and Computer Vision, it becomes a necessity to develop a robust algorithm that allows the vehicles to predict the trajectory motion of their nearby objects. One such established algorithm, comparable to the state-of-the-art method, is (Enhanced) Graph-based Interaction-aware Trajectory Prediction(GRIP++). The algorithm uses Gated Recurrent Units (GRU) and Long Short-term Memory (LSTM) type of Recurrent Neural Network(RNN) to train the model with the ApolloScape 3D Trajectory dataset. This paper studies the algorithm in detail, implements object detection and semantic segmentation, and evaluates the results using the Joint Attention in Autonomous Driving (JAAD) dataset, which provides data on pedestrian behavior in traffic scenarios.

Keywords:

Motion Prediction, Trajectory Prediction, Computer Vision, Machine Learning, Deep Learning, Long short-term memory, Recurrent neural networks, Graph-based motion prediction

Contents

1 Introduction **1**

2 Related Work **3**

2.1 GRIP Scheme Description 6

2.1.1 Analysis 6

2.2 Objective 13

3 Predicting Pedestrian Trajectories in Driving Environment **14**

3.1 JAAD Dataset 14

3.1.1 Annotation Structure of JAAD 14

3.2 Data Preparation 16

3.2.1 Data Conversion 17

3.2.2 Data Cleaning 18

3.2.3 Data Splitting 19

3.3 Training and Testing the Model 19

3.3.1 Model Training SetUp 19

3.3.2 Hyperparameters and Configurations 20

3.3.3 Preprocessing and Data Preparation 20

3.3.4 Training Process 21

3.3.5 Challenges 22

3.4 Results and Analysis 22

3.4.1 Model Performance Evaluation 22

3.4.2 YOLO and DeepLabV3+ 23

3.4.3 Qualitative Analysis 24

3.4.4 Challenges and Limitations 24

4 Future Work and Conclusion **26**

4.1 Future Work 26

4.2 Conclusion 27

Appendices **29**

A Appendix A: Code Listings and Scripts **29**

A.1 CUDA Adjustments for Colab Environment 29

A.2 Running the GRIP++ Code 29

A.2.1 Training and Testing 30

A.2.2 Visualization 30

A.3 YOLO and DeepLabV3+ for Visualization 30

Bibliography **32**

1. Introduction

Development in the area of Artificial Intelligence, Machine Learning, and Computer vision has been surging in the recent couple of years [5]. With it, autonomous vehicles are becoming more available commercially [10]. Despite advancement in the field, mass production of self-driving cars and their deployment in everyday life would be only possible if their safety is verified [8]. This poses a challenge, especially due to the reported traffic incidents in 2018 from leading companies in the area such as Tesla and Uber [10], and has caused concern for the safety of autonomous vehicles (AV).

One of the aspects that need to be improved in order to ensure the safety of AV is predicting the future positions of the surrounding close-by objects [8] just like in the case of human drivers. It is argued that if AV are to predict the future locations of the nearby traffic participants precisely, traffic accidents can be avoided [10]. Therefore, to ensure their safety, it is imperative to improve the performance of the algorithms used in AV for trajectory prediction.

Various algorithms have been developed to that effect, with numerous car manufacturers working towards improving vehicle safety via better designs of Advanced Driver-Assistance Systems (ADAS) [5]. However, much needs to be done in the area. Developing technologies for self-driving cars is also not a recent phenomenon. We can trace its development back to as early as the 1980s [5], especially with Dean A. Pomerleau's work on ALVINN in 1989 that utilized neural networks for the purpose of autonomous navigation [14]. These early advancements helped us reach where we currently stand today, which offers different algorithms and models tailored for trajectory prediction that deploy different techniques such as the Kalman Filter, Hidden Markov Model, Bayesian Networks, Gaussian Processes, or using Machine Learning models with Convolutional Neural Network (CNN) and Recurrent Neural Network(RNN); more specifically Long Short-Term Memory(LSTM), Gated Recurrent Unit (GRU), Conditional Variational Auto-Encoder (CVAE), etc. [5].

The focus of this paper is on one particular scheme, developed by Li et al. [11], called (Enhanced) Graph-based Interaction-aware Trajectory Prediction (GRIP++) that uses graph convolutions to extract features and applies LSTM to make predictions. In this research, the GRIP++ algorithm was analyzed and applied to the JAAD dataset [15] focused on pedestrian behavior in traffic scenarios. To enhance the analysis, object detection was performed using YOLO [17], identifying traffic agents like pedestrians and vehicles. Additionally, semantic segmentation was applied to the video data using DeepLabV3+ [3], which allowed for the identification of drivable areas, crosswalks, and other relevant scene elements. The predicted trajectories, obtained after training and testing the GRIP++ model, were then overlaid onto the processed video, providing visualization of how well the model predicts pedestrian movement in a given traffic scene.

2. Related Work

There exists a number of challenges in predicting trajectories in the context of self-driving cars. One of those crucial factors is behavior of pedestrians. Due to their ability to move around freely and cross the streets in the same environment as AV, it is important that their safety is ensured by the self-driving cars, irrespective of pedestrians' actions [6]. Because of their "large degree of freedom" [6], it becomes challenging to model their behavior in a dataset. Therefore other largely available datasets don't provide information for autonomous cars to understand the intention of other traffic agents [15]. Studies, such as those by Rasouli et al.[15], emphasize that pedestrians often engage in non-verbal communication, such as eye contact or gestures, which can significantly influence traffic flow and the decisions made by AV systems. To this end, in their paper "Are they going to Cross?" [15], Rasouli et al. introduced a new dataset, Joint Attention in Autonomous Driving (JAAD), which provides not only bounding box information for object detection, but also is annotated with behaviour and contextual details of those present in the traffic scenes. Figure 2.1 gives an example of the annotations provided by JAAD dataset.

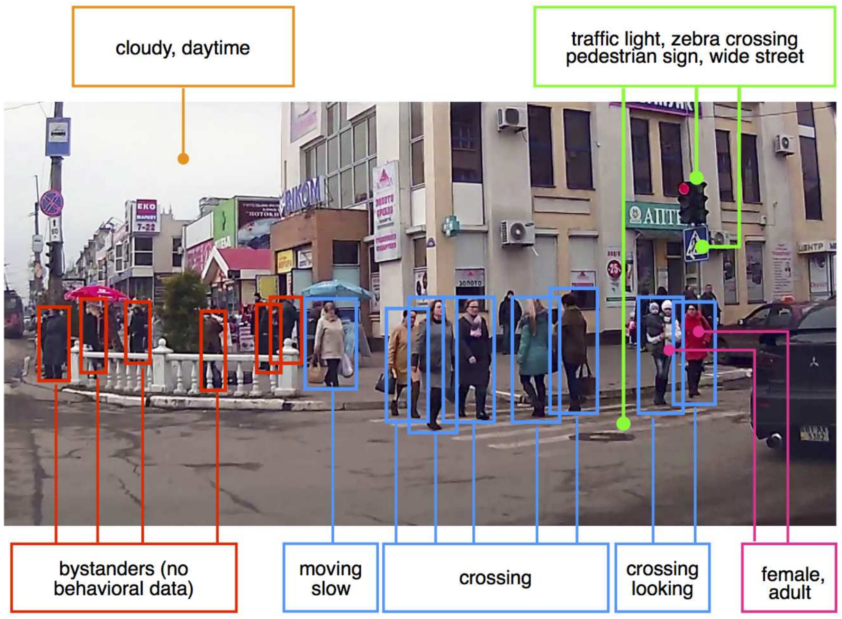
cloudy, daytime
traffic light, zebra crossing pedestrian sign, wide street
bystanders (no behavioral data)
moving slow
crossing
crossing looking
female, adult

Figure 2.1: JAAD annotations with bounding boxes for all pedestrians, behavioral labels and contextual tags

JAAD dataset particularly focuses on numerous crossing scenarios for pedestrians as the authors of the paper consider most of the interaction with the driving cars to happen at the crossing points [15]. Their dataset includes three types of ground truth annotations: bounding boxes of pedestrians for object detection, behavioral tags for denoting pedestrians' states and scene annotations for environmental contextual elements. The behavioral information includes the type and duration of pedestrians' actions, as well as the actions of the driver. Pedestrians' actions are divided into three categories: Precondition - pedestrians' state before crossing (i.e., standing, moving slowly, or fast), Attention - pedestrians' awareness towards approaching vehicle , looking for more than 1 second or glancing for less than 1 second, and Response - actions of the pedestrians after being aware of the approaching vehicle. This includes responses such as stopping, clearing the path, slowing down, speeding up, hand gestures, or nodding in reaction to the vehicle. Additionally, complementary tags come with demographics of the pedestrians like gender, age (adult, child, elderly), and group size. Figure 2.2 describes an example of behavioral labelling.

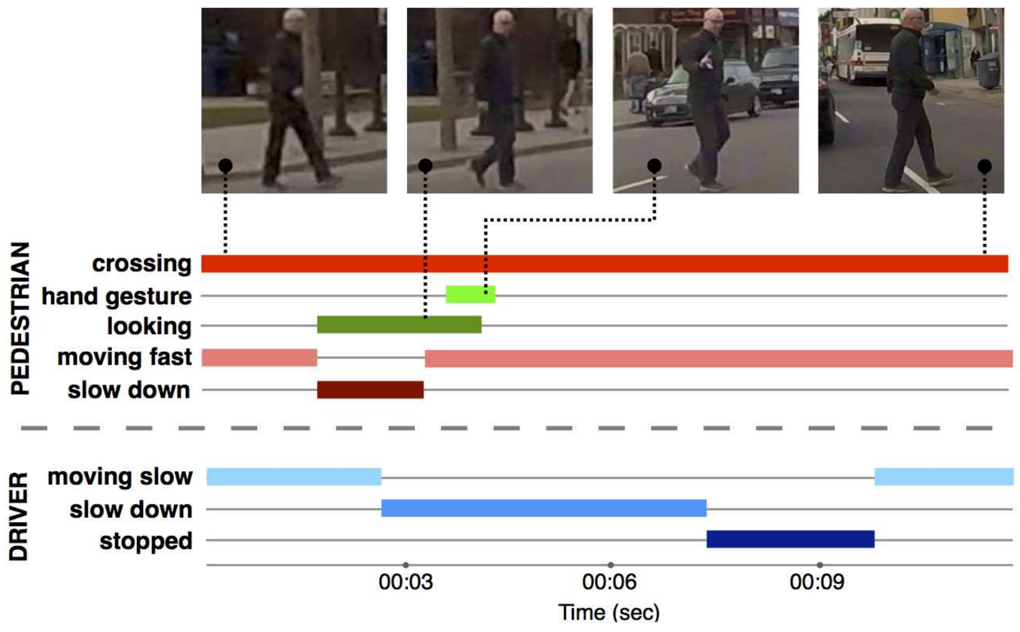

Figure 2.2: JAAD behavioral annotations with timestamps at a crossing point

Another crucial factor in AV system and subsequently for trajectory prediction is to get enhanced information about the driving environment, such as crosswalks, sidewalks, drivable areas and other contextually essential features, that is needed to make informed decisions from the side of AV. The segmentation process assigns semantic labels to each pixel in an image, distinguishing between different object types and scene elements [22]. Semantics segmentation helps to improve the safety of AV's operation by outlining drivable area and lane markings [2]. Deep neural networks employ encoder-decoder structures or spatial pyramid pooling modules to perform semantic segmentation tasks [3]. Encoder-decoder capture sharper object boundaries and gradually recovers the spatial information whereas spatial pyramid pooling encodes multi-scale contextual information by probing the incoming features with filters or pooling operations at multiple rates and multiple effective fields-of-view [3]. DeepLabV3+ [3] combines both these methods that allows them to achieve refined segmentation results along object boundaries. Figure 2.3 shows the application of DeepLabV3+ on Cityscapes dataset.

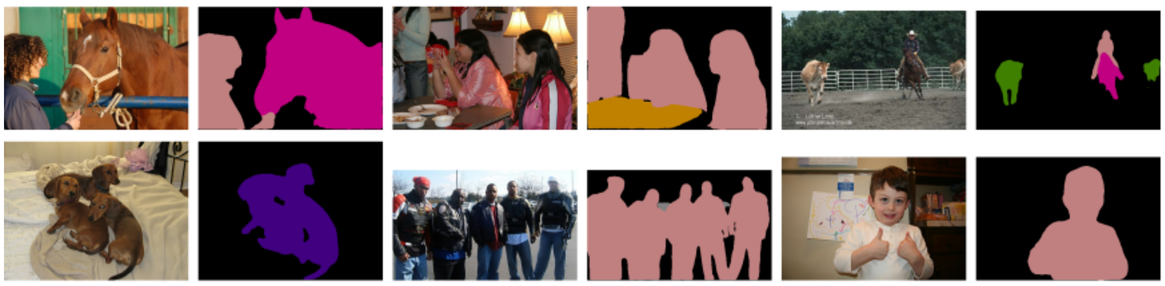

Figure 2.3: Semantics Segmentation of DeepLabV3+ on Cityscapes dataset

Yet another component required in an AV system is object detection. This process involves locating and categorizing objects [20] within an image or a video frame. Development in the field has been made commonly with the use of Deep learning algorithms such as You Only Look Once (YOLO) detectors that is based on Convolutional Neural Network(CNN) [20]. YOLO is able to perform object detection in real-time, with a fast speed and great accuracy [17]. For this reason, YOLO was later applied on the dataset for safe navigation of autonomous vehicles in a traffic scene.

Although a lot is being done for trajectory prediction on objects, only a number of them consider the importance of nearby objects [10]. This issue has been taken into consideration and new researches have been made that takes into account the ”social interaction” [9] between target vehicle, that which we want to predict the trajectory, and its environment that contains other traffic agents such as bicycles, cars and pedestrians.

Cui et al. [5] proposed, focusing on multi-modality of a traffic scenario, a method to predict multiple possible trajectories of a given actor as well as estimating their probabilities. Their proposed network encodes 300×300 bird-eye-view raster image of a nearby traffic actor with $0.2m$ resolution and the actor’s current state, such as velocity, acceleration, heading change rate, as input and outputs M modes of future $x-$ and $y-$ positions. The positions are outputted with $2H$ per mode, where H denotes the number of future consecutive time steps for which the states are predicted along with their probabilities. This overall results in $(2H + 1)M$ outputs for each actor. To ensure that the probability outputs sum up to 1, they pass the results through a softmax layer. Their method allows any CNN architecture to be used as the base network and the authors applied MobileNet-v2.

Luo et al. [12] introduced in their paper a fast detection, tracking and motion predicting of objects. Bird's eye view LiDAR data is given as input and 3D is processed spatially and temporally. Afterwards they add two branches of convolutional layers where one calculates the probability of a vehicle being at a given location and the other forecasts bounding boxes of the subsequent frames. These available approaches require high computational power if they are to perform trajectory predictions of all nearby objects in a traffic scene [10]. Therefore Li et al. developed a robust and more efficient object trajectory prediction scheme to forecast the accurate future positions of those objects surrounding the self-driving cars [10]. Moreover they further improved their scheme [11] and this paper utilizes the enhanced version.

2.1 GRIP Scheme Description

The algorithm that will be the focus of this paper is Enhanced Graph-based Interaction-aware Trajectory Prediction (GRIP++) [11]. It is designed to predict trajectory of traffic participants around an AV more efficiently using graph to represent interaction amongst close objects, applies multiple graph convolutional blocks to extract features and ultimately uses LSTM encoder-decoder to make predictions. This section summarizes and provides a detailed description of the scheme.

2.1.1 Analysis

Most of the algorithms used in the traffic scene are limited in several ways such as one where it doesn't work in crowded scenarios, or with the second example it merely predicts the trajectory of one specific car but since the traffic will be composed of more than one vehicle at a given time, this method becomes highly inefficient. Nonetheless these methods require intensive computation power if they want to predict the trajectory of all surrounding objects. And if they are maneuver based, wrong classification of the maneuver type will negatively impact the trajectory prediction.

Li et al., formulates the problem as follows: trajectory prediction is a problem that estimates the future positions of all objects in a scene based on their trajectory histories. Therefore the inputs X of the model are trajectory histories(over t_h time steps) of all observed objects in a traffic scene:

$$X = [p^{(1)}, p^{(2)}, ..., p^{(t_h)}] \tag{2.1}$$

where,

$$p^{(t)} = [x_0^{(t)}, y_0^{(t)}, x_1^{(t)}, y_1^{(t)}, ..., x_n^{(t)}, y_n^{(t)}] \tag{2.2}$$

are the coordinates of all observed objects at time t, and n is the number of observed objects in that scenario. Their format is the same as another study done by Deo et al. and global coordinates are used. To improve the prediction accuracy, the “ego-vehicle-based” coordinate system could be used as suggested by the author but is not applied for future work. Because their model takes in histories of all observed objects in a traffic scenario, it was argued to be most fitting to predict the future position of all of them simultaneously rather than focusing on one individual object, which is a preferred measure for other schemes that are also predicting future position for autonomous cars. The outputs Y of the proposed model are the predicted positions of all observed objects in a scenario from time step $t_h + 1$ to $t_h + t_f$:

$$Y = [p^{(t_h+1)}, p^{(t_h+2)}, ..., p^{(t_h+t_f)}] \tag{2.3}$$

where $p^{(t)}$ is the same as the second equation described above and t_f is the predicted horizon.

In there proposed scheme , the authors of the paper divide their approach towards their novel model into following components:

1. Input Preprocessing Model,
2. Graph Convolution Model
3. Trajectory Prediction Model
4. Implementation Details

Input Preprocessing Model

A. Input Representation: Prior to feeding the raw data into the model, it is required to convert it into a specific format for subsequent efficient computation. Given n objects in a traffic scene in the past t_h

time steps, it is represented in a 3D array F_{input} with a size of $(n \times t_h \times c)$. The authors set $c = 2$ in the paper to indicate x and y coordinates of an object. Since it is easier to predict the velocity of an object rather than predicting its location(Li et al., 2020), velocities $(p^{t+1} - p^t)$ is calculated before feeding the data into the model.

B. Graph Construction: As in an autonomous driving application scenario, the movement of a target's surrounding objects deeply impacts the motion of the target, the authors represent the inter-object interaction using an undirected graph $G = \{V, E\}$. Each node, V corresponds to an object present in a traffic scene. Given that each object would have different states at different time steps, the node set $V = \{v_{it}|i = 1, ..., n, t = 1, ...t_h\}$ where n is the number of observed objects and t_h is the observed time steps. The feature vector v_it on a node is the coordinate of i_{th} object at time t.

In constructing the graph, objects that have interaction at each time step t are connected via edges. Authors define such interaction in an autonomous driving scenario when two objects are close to each other. Therefore the edge set E consists two parts: (a) the interaction information between two objects in spatial space time t, called "spatial edge" (Li et al., 2020) and denoted as $E_S = \{v_{it}v_{jt}|(i, j \in D)\}$, with D being a set where objects are close to each other. This is demonstrated in the Figure 2.4, using two blue circles with a D_{close} radius on the "Raw Data" section. Objects that are inside the blue circle are regarded as close to the object located in the middle, indicated by a blue dot. Therefore in the diagram, the top object has three close neighbors whereas the one below has one neighbor. (b) The second part is the "inter-frame edges" (Li et al., 2020), that essentially represents the historical information frame by frame in temporal space. The temporal edge connects each observed object in a given time step with itself in another time step, and is denoted as $E_F = \{v_{it}v_{i(t+1)}\}$. In such a way$E_F$ contains all edges of a particular object which represent its trajectory over time steps.

The authors of the algorithm represent this graph using an adjacency matrix $A = \{A_0, A_1\}$, where A_0, an identity matrix I, represents self-connections and A_1, an adjacency matrix, represents spatial connection, for more efficient computation. At a given time t,

$$A_0[i][j](or A_1[i][j]) = \begin{cases} 1, & \text{if edge} 0.5mm\langle v_{it}, v_{jt}\rangle \in E \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

A_0 and A_1 both are of size $(n \times n)$, where n is the number of observed objects in a traffic scene. The authors called it “Fixed Graph” as it is constructed on a manual design rule and is fixed once the input data is given and will not modify during the training phase. It is shown in Figure 2.4 as a blue dots connected graph symbol.

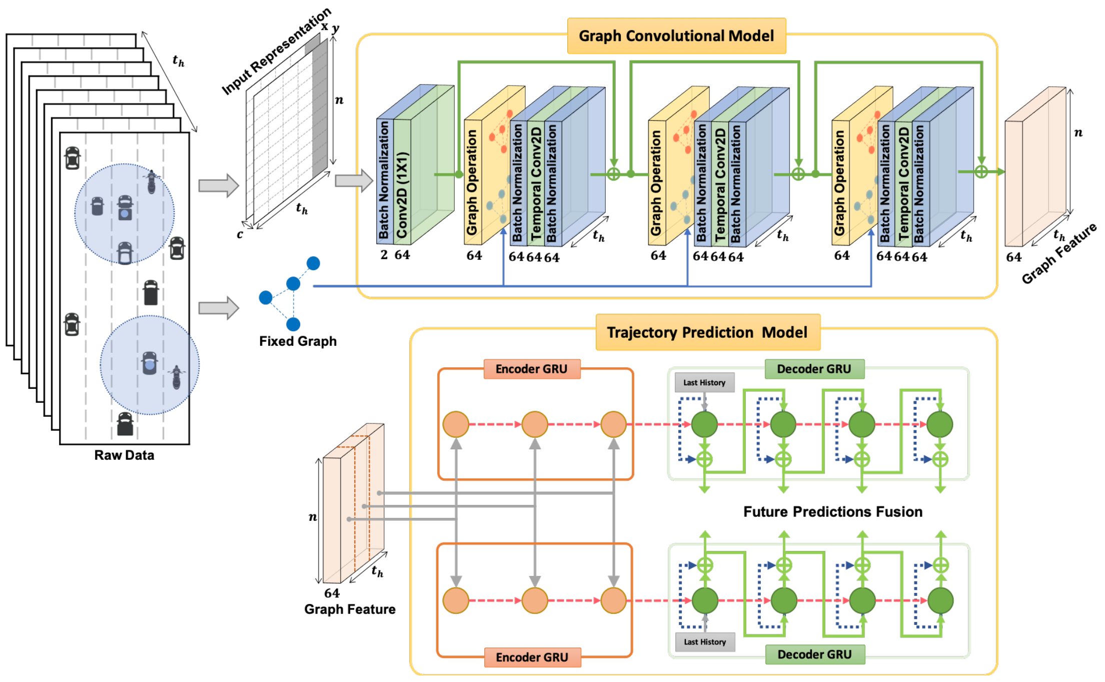

Figure 2.4: GRIP++ proposed Scheme

Graph Convolutional Model

For a given preprocessed input data $F_{input} := \mathbb{R}^{n \times t_h \times c}$, the Graph Convolutional Model passes it through a 2D convolutional layer with (1×1) kernel size, shown in Figure 2.4 as ”Conv2D (1×1)”, so that the number of channel is increased. It is designed to map the 2 dimensional input data (x and y coordinates) into a higher dimensional space to help the model learn a good representation for the trajectory prediction task. The output, therefore, is of the shape $(n \times t_h \times C)$, where C is the new number of channels. In the figure 2.4, C is 64. The input data is then fed into both several graph operations and temporal convolutions where graph operations handle the inter-object interaction in spatial space and temporal convolutions capture useful temporal features such as motion pattern of an object. Therefore, Figure 2.4, with 3 Graph Operation layers and 3 Temporal Convolution layers shown, one Temporal Convolution layer is added at the end of each Graph Operation layer to process the input data both spatially and temporally in every alternate.

Batch Normalization layers are added as well to improve the stability of the model in training.

Skip connections, denoted as green polylines, are used to ascertain that the model can propagate larger gradients to initial layers, which could learn as fast as the final layers.

1) Graph Operation Layer: A graph operation layer takes into account the interactions of surrounding objects. There are two graphs in each Graph Operation layer: (i) a Fixed Graph, which is the adjacency matrix A shown above and the blue graph symbol from Figure 2.4, constructed based on the current input, and (ii) a trainable graph, G_{train}, shown in Figure 2.4 as orange graph symbol in the Graph Operation Model block.

The authors normalize the Fixed Graph A using the below equation to make sure that the value range of feature maps are not changed after the graph operations:

$$G^j_{fixed} = \Lambda_j^{-1/2} A_j \Lambda_j^{-1/2} \tag{2.5}$$

where $\Lambda_j^{-1/2}$ is computed as:

$$\Lambda_j^{ii} = \sum_k (A_j^{ik}) + \alpha \tag{2.6}$$

α is set to 0.001 to avoid empty rows in A_j.

Since the Fixed Graph, G_{fixed} is constructed based on a manually designed rule, it might not be able to represent the interactions of objects properly. Therefore to solve this problem, the authors of the paper sum the Fixed Graph with the trainable graph so that the trainable graph can be trained to overcome this limitation. As a result, after a Graph Operation layer takes an input f_{conv} from the previous layer, the output feature map f_{graph} is calculated as:

$$f_{graph} = \sum_{j=0}^{1} (G^j_{fixed} + G^j_{train}) f_{conv} \tag{2.7}$$

As the Graph Operation layers do not change the size of the features, f_{graph} is of size $(n \times t_h \times C)$.

2) Temporal Convolutional Layer: The generated feature $f_{graph} := \mathbb{R}^{n \times t_h \times C}$ is then fed into a Temporal Convolutional layer. The kernel size of a Temporal layer is set to (1×3) to force the processing of the data along the temporal dimension. Paddings and strides are added appropriately to make sure each layer has an output feature map of expected size.

Trajectory Prediction Model

This model consists of several networks which share the same Seq2Seq structure but is trained for different weights. Two Seq2Seq networks are shown in Figure 2.4. The Graph Feature, generated by the Graph Convolutional Model, is taken by each network as its input. At each temporal dimension, feature vectors of the Graph Feature are fed into the corresponding input cell of the Encoder GRU (Gated Recurrent Unit). Thereafter the hidden feature of the Encoder GRU together with the coordinates of the objects are fed into the Decoder GRU to predict the position coordinates at the current time step. The input of the first decoding step is the coordinates of objects at the ”Last History” step, and the output of the current step is then fed into the next GRU cell. This is illustrated in Figure 2.4 as the gray ”Last History” boxes corresponding to the gray column of the Input Representation. Above described decoding process is to be repeated multiple times until the positions for all expected time steps $(t_f))$ in the future is predicted by the model. Due to velocity of the traffic agents in a given scene not being constant, the authors have incorporated the model to predict the velocity change by adding a residual connection, shown as blue dashed lines in the Figure 2.4, in between the input and the output of each cell of the Decoder GRU.

After getting the predicted results of the Seq2Seq networks, the predicted velocities results are averaged at each time step. After averaging the predicted velocities $(\Delta x, \Delta y)$, they are added back to the last historical location $(p^{(th)})$ to convert the predicted results back to (x, y) coordinates. The main difference between implementation of the algorithms GRIP++ and GRIP are listed down as follow:

- GRIP++ takes velocity $(\Delta x, \Delta y)$ as input whereas GRIP takes (x, y) coordinates as input
- GRIP++ considers both fixed as well as trainable graphs meanwhile GRIP only considers fixed graphs in the graph convolution sub-module
- In the graph convolution model GRIP++ uses 3 blocks, adds batch normalization and uses skip connections while GRIP uses 10 blocks without the batch normalization layers.
- GRIP++ uses GRU networks, three encoder-decoder blocks for trajectory prediction, and average the results whereas GRIP uses LSTM networks and only one encoder-decoder block for trajectory prediction.

Implementation Details

The authors implement the scheme using Python Programming Language and PyTorch Library. This section further details the implementation of the scheme and the settings of necessary parameters.

Input Preprocessing Model: The authors propose, in their paper, a traffic scene within 180 feet (±90 feet), whereby all the objects within the designated region will be observed and predicted in the future. Furthermore, two objects in a scene are defined as *close* if their distance is less than 25 feet $(D_{close} = 25)$ and are connected via a spatial edge, $e_s \in E_S$.

Graph Convolutional Model: As described above with Figure 2.4, the authors utilize a (1×1) convolutional layer to increase the channel of data input to 64, their Graph Convolutional Model contains 3 Graph Operation layers and each layer is followed by a Temporal Convolution layer that has a convolutional kernel of size 1×3. $stride$ is set to 1 and paddings are added appropriately to maintain the feature maps shape. Therefore, the Graph Convolutional Model outputs a size of $(n \times t_h \times 64)$. Additionally, the scheme dropouts features (0.5 probability) after each graph operation to avoid overfitting.

Trajectory Prediction Model: The encoder and decoder of this model are both a two-layer GRU networks. The number of hidden units of the two GRUs are set to r times of the output dimension, $r \times 2 \times n$, where r is to improve the representation ability, n is the number of objects and 2 is the x, y coordinates. For this paper, the authors choose $r = 30$ for best performance. The 64 channels input of the encoder are the same as the Graph Convolutional Model output.

Optimization: The model is trained as a regression task at each time and the overall lose is computes as:

$$Loss = \frac{1}{t_f} \sum_{t=1}^{t_f} loss^t \tag{2.8}$$

$$= \frac{1}{t_f} \sum_{t=1}^{t_f} \| Y^t_{pred} - Y^t_{GT} \| \tag{2.9}$$

where t_f is the time step in the future, which is set to 4 in Figure 2.4, $loss^t$ is the loss at time t, Y_{pred}

and Y_{GT} are predicted positions and ground truth respectively. And the model is trained such that the $Loss$ is minimized.

Training Process: The authors train the model using Adam Optimizer with default settings unchanged in PyTorch Library, with $batch\_size$ set to 64 during training.

2.2 Objective

For this research, primary objective is to answer the following research question: How can the integration of the GRIP++ algorithm with real-time object detection (YOLO) and semantic segmentation (DeepLabv3+) enhance the accuracy of pedestrian trajectory prediction in a self-driving cars context in urban environments?

3. Predicting Pedestrian Trajectories in Driving Environment

The previous chapter presented an in-depth description and an analysis of the GRIP++ algorithm [11], explaining its architecture, theoretical foundations, and advantages in predicting trajectories within dynamic traffic environments. To further assess the performance and applicability of GRIP++, the algorithm was implemented on a different dataset from ApolloScape Trajectory dataset [13], the one originally used by the developers of the algorithm. Instead Joint Attention in Autonomous Driving, (JAAD) [15] dataset was opted for this operation. JAAD provides extensive examination of joint attention in the autonomous driving context [15]. The dataset focuses on the behaviour of pedestrians and drivers in a traffic scenario, specifically at the crossing points as well as the factors influencing them. Due to its comprehensive set of annotated videos, it makes it well-suited for the task of predicting pedestrian trajectories and investigating the essential question ”Are they going to corss?” [15].

As such this chapter focuses on conversion of JAAD dataset to suitable format, training and testing the dataset to receive predicted trajectory results, which was then incorporated with `YOLO` [17] for object detection and DeepLabV3+ [3] for semantics segmentation to enhance overall accuracy and context-awareness of the predictions. Figure 3.1 demonstrates the pipeline of this research.

3.1 JAAD Dataset

As briefly mentioned above JAAD provides annotation of a collection of dataset, specially 346 short video clips of driving footage, is available to the public for research in autonomous vehicle area. These clips, recorded from various regions of North America and Eastern Europe, capture a variety of common scenes typical of everyday urban driving in a range of weather conditions [15].

3.1.1 Annotation Structure of JAAD

The annotations of JAAD videos are based on the video clip titles. Three kinds of labels are available from JAAD: ”pedestrians” (individuals with behavior annotations), ”peds” (bystanders who

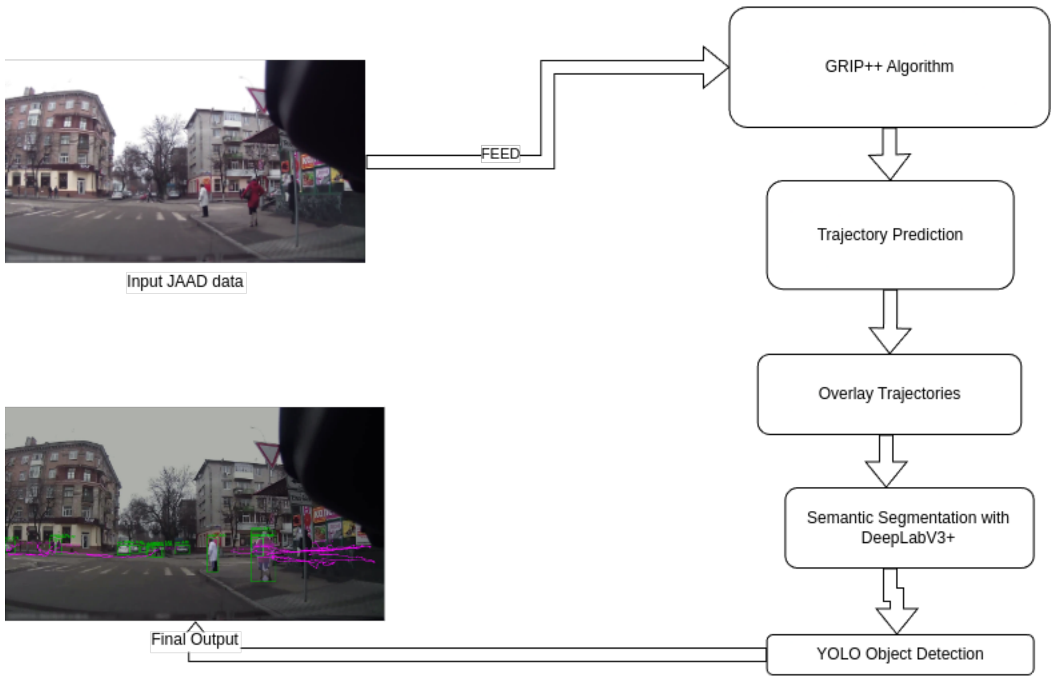

Figure 3.1: Pipeline for predicting trajectory in a dynamic driving context

are distant and do not interact with the driver), and ”people” (groups of pedestrians) [15]. The format of each pedestrian’s unique id is `0_<video_id>_<pedestrian_number>`. Pedestrians with behavior annotation, have ”b” attached at the end of their ids, e.g., `0_1_3b`. Similarly, annotations for people follow the same strucutre but their ids end in the letter ”p,” such as `0_5_2p`, the annotations for people likewise follow the same structure. For the purpose of using this dataset annotation, however, strings and chars like `"_"`, `"p"`, and `"b"` were removed from the annotations during the **data transformation** stage since the data structure of the GRIP++ algorithm does not allow for strings to be included in the dataset while preprocessing [13].

In the annotations, bounding boxes are provided together with occlusion tags for all pedestrians which make the dataset suitable for pedestrian detection. Bounding boxes in each annotation have two coordinate points: the top-left and bottom-right corners, specified as $[x_1, y_1, x_2, y_2]$. Additionally, these bounding boxes are labeled with occlusion tags, where 0 indicates no occlusion, 1 represents partial occlusion (more than 25%), and 2 denotes full occlusion (more than 75%).

The annotations within the dataset are categorized into five distinct groups, each for a specific purpose:

- **Annotations**: This group encompasses general video metadata such as attributes related to the time of day, weather conditions, and location. Additionally, it includes detailed information on pedestrian activities (e.g., walking, looking) and bounding box coordinates, along with corresponding occlusion statuses. Notably, activity information is only available for a selected subset of pedestrians. These annotations are recorded for each frame and apply to every labeled instance.

- **Attributes**: Focused on pedestrians with behavior annotations, this category provides demographic details and information regarding crossing points and crossing behavior characteristics. These annotations are assigned to individual pedestrians rather than frames and are intended to offer insights into specific pedestrian behaviors in various traffic scenarios.
- **Appearance**: For videos characterized by high visibility, this group offers details on pedestrian appearance, including pose, clothing, and any objects being carried. This information is annotated per frame for each pedestrian, allowing for a thorough analysis of pedestrian visual characteristics.
- **Traffic**: This group contains traffic-related annotations, offering information on the presence of traffic signs, traffic lights, and other traffic control devices within the scene. These annotations are captured on a per-frame basis, ensuring a comprehensive understanding of the traffic environment during the recorded events.
- **Vehicle**: Vehicle-related actions, such as acceleration, deceleration, or rapid movement, are included in this category and these actions are recorded per frame

In the application of the GRIP++ algorithm, not all annotation groups are required. The primary focus is on the dataset annotations group "**Annotations**," which provide the necessary information for the task. The other groups are considered irrelevant to this particular application and are therefore disregarded.

3.2 Data Preparation

The JAAD dataset, originally structured for analyzing pedestrian behavior in traffic scenarios, required transformation to be suitable with the data format required by the GRIP++ algorithm. The JAAD dataset provides annotations in XML format, detailing pedestrian positions, actions, and interactions with traffic elements. However, the GRIP++ algorithm expects data in a specific structure, similar to the Apolloscape dataset, which was used by the GRIP++ authors for their experiments [11]. To adapt the JAAD dataset to be compatible with GRIP++ requirement, data conversion was carried out.

3.2.1 Data Conversion

The JAAD dataset contains XML annotations for behavior and interactions of pedestrians in urban environments. These XML annotations were first converted into a structured text format that matches the format used by the Apolloscape dataset. Their data structure of each line in the text file is as follow:

frame_id	object_id	object_type	position_x	position_y	position_z	object_length	object_width	object_height	heading

Table 3.1: Data Structure of ApolloScape Dataset [13]

However, since the JAAD dataset does not include information for position_z, and object_height, only eight columns were extracted from the JAAD annotations. These columns include frame_id, object_id, object_type, position_x, position_y, and heading. The missing columns (i.e. position_z and object_height) were omitted because JAAD lacks the necessary 3D positional information and detailed bounding box dimensions available in Apolloscape.

Heading Calculation

Although data for ”heading” was not readily available, it was computed based on the already accessible information. The heading is an angular measure that represents the orientation or direction of the object with respect to the x-axis [19]. For instance, in a traffic scenario, the heading can indicate whether a vehicle is moving forward, turning, or reversing. The heading was calculated based on the position of the object’s bounding box. Specifically, the angle of the vector formed by the top-left and bottom-right corners of the bounding box with respect to the x-axis was computed using the $atan2$ function. This function calculates the arctangent of the two given coordinates, which provides the angular direction of the bounding box in radians. The formula [1] used is:

$$heading = atan2(y_{br} - y_{tl}, x_{br} - x_{tl}) \tag{3.1}$$

where (x_tl, y_tl) is the top-left corner of the bounding box and (x_br, y_br) is the bottom-right corner. This heading value was then rounded to three decimal places for consistency. The heading provided directional information that the GRIP++ model could use to better predict future trajectories of moving objects.

Object Type Mapping

Each object in the JAAD dataset was assigned a type based on the classification scheme used in the Apolloscape dataset. The object types in Apolloscape include small vehicles, big vehicles, pedestrians, motorcyclists/bicyclists, and others. These types were mapped to corresponding IDs as follows:

Object Type	**Small Vehicles**	**Big Vehicles**	**Pedestrian**	**Motorcyclist and Bicyclist**	**Others**
ID	1	2	3	4	5

Table 3.2: ApolloScape's object types and their corresponding IDs [13]

since the JAAD dataset primarily focuses on pedestrian behavior and interactions, it does not provide detailed information about other types of objects, such as vehicles or bicyclists. As a result, after converting the JAAD dataset into the required format, only two object types are expected to appear in the processed data:

- **Pedestrians** (ID: 3),
- **Others** (ID: 5)

The ”Others” category contains any objects that are not classified as pedestrians, though in practice, the JAAD dataset contains very few examples outside of the pedestrian category. The object type mapping was still retained to ensure compatibility with the GRIP++ algorithm, which was designed with the Apolloscape dataset's object type distribution in mind.

3.2.2 Data Cleaning

After converting the JAAD dataset into the required format for use with the GRIP++ algorithm, an additional step of data cleaning was performed. This cleaning process was necessary for ensuring that the data was consistent with the expectations of the GRIP++ model and did not introduce unnecessary noise during training and testing. The key steps involved in this process were as follows:

1. **Handling Missing Values**: During the conversion process, some object annotations may have been incomplete or missing entirely. These instances were identified, and any lines or entries with missing values were either corrected or removed to maintain the integrity of the dataset.

2. **Filtering Non-Relevant Objects**: As previously noted, the JAAD dataset primarily focuses on pedestrians, with limited information available for other object types such as vehicles. Object types that did not fit into the defined categories of the GRIP++ model were categorized as ”Others” (ID: 5), but were still retained in the dataset for completeness.

3. **Removing Empty Files**: During the preprocessing stage, some files were found to be empty, meaning they contained no objects or annotations. These empty files were automatically removed from the dataset since such files provided no valuable information for training or testing.

As a result of this data cleaning process, the dataset was reduced from the originally available **346** files to **319** files. The removed files either contained no annotations or did not meet the necessary criteria for further processing.

3.2.3 Data Splitting

Once the data was transformed and cleaned, it needed to be divided into training and testing sets to train the model and evaluate its performance. For this purpose, the `train_test_split` function from the `scikit-learn` library was used.

Given the original setup by the GRIP++ developers, where a specific ratio of training to testing data was employed, the same principle was adopted in this implementation. The data was split with a ratio of approximately 55 to 1, following the precedent set by the authors of the GRIP++ algorithm. This ratio was chosen to maintain consistency with their approach and ensure that the training data was sufficiently large while still allowing for adequate evaluation on the test data.

3.3 Training and Testing the Model

3.3.1 Model Training SetUp

The training of the GRIP++ model was conducted on `Google Colab`, which provides access to NVIDIA GPUs, that is required to run the original `Python` file developed by the authors [11], ensuring the required computational power and CUDA support necessary for training deep learning models. `Colab` was chosen due to its free availability and built-in shared GPU support, making it suitable for models that require intensive computations like GRIP++.

`PyTorch` was used as the primary framework for implementing and training the GRIP++ model. `PyTorch` provides flexibility and ease of use, particularly for graph-based models, which are integral to GRIP++. The CUDA library was leveraged to ensure GPU acceleration, which speeds up matrix operations, tensor computations, and the backpropagation process during training [4].

3.3.2 Hyperparameters and Configurations

Several hyperparameters were set for training the GRIP++ model. These include:

- **Learning Rate**: The learning rate was set to an initial value of 0.01. This value was chosen based on experimentation and previous works, balancing between fast convergence and stability of training.
- **Batch Size**: The batch size was set to 64 for training and 32 for validation. This choice balances memory usage and the stability of gradient updates.
- **Number of Epochs**: The training was conducted for 50 epochs. The original paper experimented with various epochs, and similar values were used here to ensure the model converges appropriately.
- **Optimizer**: The Adam optimizer was selected for its adaptive learning rate capabilities, which are particularly beneficial for models like GRIP++ that have complex architectures and large parameter spaces.

These configurations were determined based on the original implementation of GRIP++ and fine-tuning through experimentation to optimize performance on the JAAD dataset.

3.3.3 Preprocessing and Data Preparation

Before training, the transformed dataset underwent preprocessing to ensure it was in the correct format for input into the GRIP++ model. This preprocessing involved:

- **Normalizing Position Data**: The positional data (`position_x` and `position_y`) were normalized based on the maximum values in the dataset to prevent issues related to scale during training.
- **Graph Construction**: Each frame in the sequence was represented as a graph, where objects (e.g., pedestrians, vehicles) were nodes, and the edges represented interactions between them. The features associated with each node included the object's type and its spatial information.
- **Sequence Padding**: The sequences were padded to ensure consistent input dimensions for the GRIP++ model, as deep learning models require fixed input sizes. Shorter sequences were padded, while longer sequences were truncated.

Once preprocessing was complete, the dataset was ready for the training and testing phases.

3.3.4 Training Process

The training process involved the following steps:

- ***Data Loading:*** The preprocessed dataset was loaded using custom data loaders designed to efficiently batch the data for training and testing. This involved dynamically creating batches of data, including the graphs for each frame sequence and the associated labels.
- ***Forward Pass:*** For each batch, the data was passed through the GRIP++ model, where the graph convolutional layers extracted interaction-aware features, and the LSTM layers predicted future trajectories based on these features.
- ***Backpropagation***: Gradients were computed using backpropagation, and the Adam optimizer was used to update the model's parameters.
- ***Validation***: After each epoch, the model was evaluated on the validation set to monitor its performance and adjust the learning rate if necessary.

This training cycle was repeated for the specified number of epochs, with model checkpoints saved at intervals to prevent data loss and to enable further fine-tuning if required.

3.3.5 Challenges

During the training process, one of the main challenges encountered was the limitations imposed by the available GPU resources on `Google Colab`. While `Colab` provides access to powerful GPUs, there are restrictions on memory usage and the maximum time for continuous usage. These limitations required careful management of the available resources. Training had to be paused or restarted at times, and model checkpoints were regularly saved to avoid data loss.

3.4 Results and Analysis

3.4.1 Model Performance Evaluation

After training the GRIP++ model on the JAAD dataset, the model's performance was evaluated on the test set. The evaluation focused on predicting the future trajectories of pedestrians, as JAAD primarily contains pedestrian data. The predictions were compared against the ground truth to assess the accuracy of the model.

Key performance metrics included:

- Average Displacement Error (ADE): Measures the average Euclidean distance between the predicted trajectories and the ground truth over all time steps [13]. This metric is particularly important in trajectory prediction tasks, as it indicates how close the model's predictions are to the actual movement of the objects.
- Final Displacement Error (FDE): Evaluates the distance between the predicted final position and the ground truth final position at the last time step [13]. This metric helps assess how well the model predicts the endpoint of a trajectory, which is critical in safety-critical applications like autonomous driving.

In the first test subset on the JAAD dataset, the GRIP++ model achieved an ADE of 1.66 and an FDE of 0.15. These results indicate that while the model performs reasonably well in predicting pedestrian trajectories overall, it performed exceptionally in final destination prediction, as evidenced by the relatively low FDE. The low FDE suggests that the model's final predictions were close to the actual final positions of pedestrians in the scene.

Method	**Epoch**	**ADE**	**FDE**
TrafficPredict	-	7.1811	11.121
GRIP++	Epoch18	0.7142	1.3732
GRIP++ on JAAD	Epoch21	1.66	0.15
GRIP++ on JAAD	Epoch41	12.26	0.57

Table 3.3: Comparison of methods based on ADE and FDE metrics.

The performance on the second test subset was less accurate, with an ADE of 12.26 and an FDE of 0.57, suggesting that in this scenario, the model struggled more with overall trajectory prediction but still performed relatively better in terms of predicting the final position, though less accurately than in the first test.

3.4.2 YOLO and DeepLabV3+

To get an accurate interpretation of the scene, DeepLabv3+ was employed for semantic segmentation. This deep learning model was used to highlight key features in the driving environment, such as drivable areas and crosswalks. By segmenting the scene, it becomes easier to understand the spatial context within which the pedestrians and vehicles are moving. The segmented output identifies safe zones, walkways, and areas that are off-limits, providing a comprehensive view of how pedestrians interact with their environment. This information is particularly valuable when analyzing whether predicted trajectories falls within safe walking paths or if there is a potential risk of crossing into dangerous areas. The integration of semantic segmentation ensures that trajectory predictions are not only accurate but also contextually aware of the environment.

`YOLO` object detection was used to draw bounding boxes around the objects present in the video frames to keep track of the pedestrians' movements and their interaction in real-time. This helped to distinguish pedestrians from other traffic agents and to associate their predicted trajectory with correct individuals. After the semantics segmentation and YOLO application, predicted trajectories were overlaid on the video frames to depict precisely the movements, ids and the anticipated locations of the pedestrians.

The visualizations showed how the model predicted the next 3 seconds of pedestrian movement. For instance, the model correctly predicted when a pedestrian was likely to stop, move forward, or change direction. This visual validation was essential to verify the model's effectiveness beyond numerical metrics.

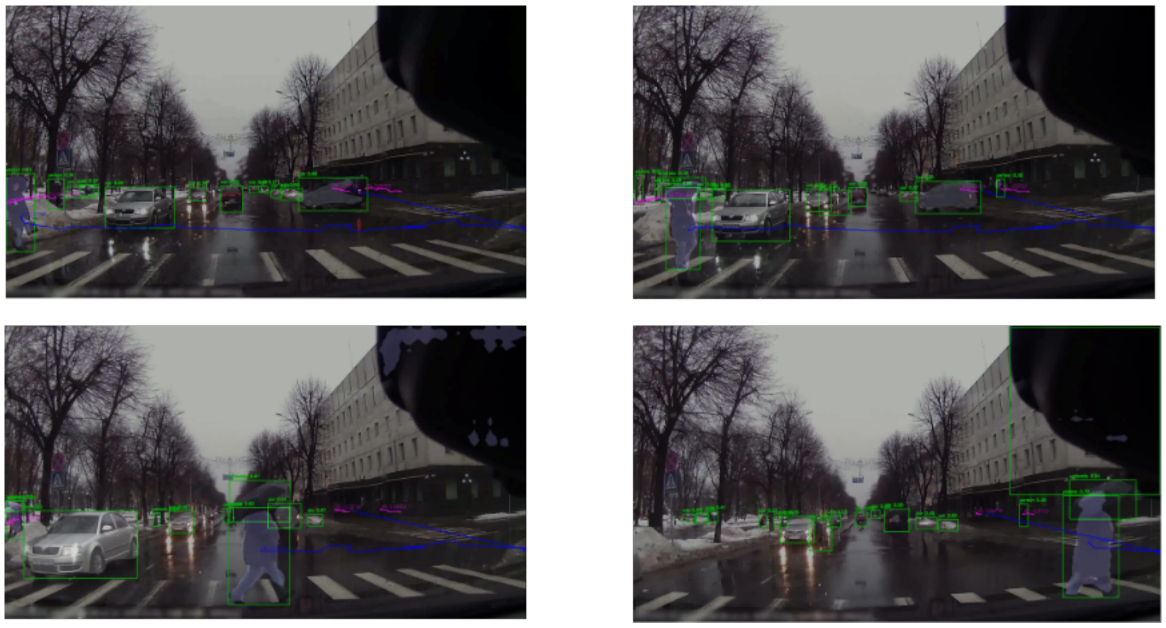

Figure 3.2: Images from the second test subset

3.4.3 Qualitative Analysis

While quantitative metrics like ADE and FDE provide valuable insights, they do not fully capture the complexities of real-world scenarios. One crucial aspect is the inherent uncertainty in pedestrian behavior. Even if the model predicts with 99% accuracy that a pedestrian will not cross a street, there remains a 1% chance that they might unexpectedly decide to cross.

This slim margin of uncertainty can have significant consequences. For example, if an autonomous vehicle concludes that a pedestrian is unlikely to cross and continues driving, but the pedestrian suddenly steps into the street, the outcome could be disastrous. Therefore, even models with high predictive accuracy should be used cautiously in critical real-world applications like autonomous driving. Human behavior is unpredictable, and safety systems must account for these rare but potentially catastrophic deviations.

The challenge lies in striking a balance between the model's confidence and the implementation of safety measures that prevent accidents even in rare cases of prediction failure.

3.4.4 Challenges and Limitations

One of the primary challenges encountered during this project was the need for substantial computational resources. The GRIP++ model relies heavily on graph-based computations, which can be resource-intensive. High-performance GPUs and CUDA-enabled environments were essential for training the model efficiently. However, this requirement poses challenges for deploying such models in real-world systems, especially in environments with limited computational capacity, such as cars for which these algorithms are developed in the first place. Another limitation is the gap between academic algorithms and practical implementations. Many algorithms developed in academic research are tailored for desktop-grade GPUs, which require high computational power. However, real-world autonomous driving systems and Advanced Driver Assistance Systems need to be compatible with embedded systems that operate within the constraints of small devices installed in vehicles.

One other limitation comes from the JAAD dataset, which provides annotations only for pedestrians and does not include other relevant and necessary information for other traffic agents like bikers, cyclists, etc. As the dataset is still relatively smaller [15], it may be limited to only a certain type of information.

4. Future Work and Conclusion

4.1 Future Work

Building on the findings and limitations identified in this research, several areas for future work emerge. These potential directions aim to enhance the practicality, performance, and robustness of graph-based trajectory prediction models like GRIP++, particularly in the context of real-world autonomous driving applications.

Incorporation of Additional Object Types and Environmental Factors: The current implementation of GRIP++ using the JAAD dataset focused primarily on pedestrian trajectory prediction. Expanding the model to accurately predict the trajectories of other objects, such as vehicles, cyclists, and motorcyclists, could improve its applicability in more complex traffic scenarios. Additionally, incorporating environmental factors like road conditions, weather, and traffic signals could enhance the model's decision-making capabilities.

Cross-dataset Generalization: The current implementation was tested on the JAAD dataset, which is focused on pedestrian behavior in specific urban scenarios. Future research could evaluate the generalizability of GRIP++ across different datasets, such as the KITTI [7] or Waymo datasets [21], which contain a broader range of traffic agents and driving conditions. Improving the model's ability to generalize across datasets would be a crucial step toward its deployment in real-world systems.

Collaboration with Industry for Real-world Testing: Finally, collaborating with industry partners in the automotive sector could provide opportunities to test the model in real-world driving conditions. This collaboration could offer insights into the practical challenges of deploying such models in autonomous vehicles and help refine the model to meet the stringent safety and performance standards required in the industry.

Pedestrian Analysis: Although not done during this research, one area of improvement could be to look into if a pedestrian is going to enter a segmented area which is classified as driving lane and pose a threat to themselves. This is a question that needs to be analyzed carefully for safety of autonomous vehicles.

4.2 Conclusion

The research undertaken in this project aimed to replicate and evaluate the performance of the GRIP++ algorithm using a different dataset, JAAD, to test the model's effectiveness in predicting pedestrian trajectories. The study successfully demonstrated the feasibility of adapting the GRIP++ algorithm to a new dataset by transforming the JAAD annotations into the required format. Despite the challenges of dataset transformation, preprocessing, and handling variations in object types, the model achieved reasonable performance in predicting pedestrian movements.

One of the key achievements of this project was the ability to visualize the predicted trajectories alongside actual movements in real-world video footage. This provided an intuitive understanding of the model's strengths and limitations. The results indicate that the graph-based approach of GRIP++ effectively captures the interaction dynamics between different traffic agents, leading to accurate predictions in most cases.

However, several limitations were identified during the course of this research. The reliance on high-performance GPUs for training and testing the model is a notable constraint. In practical applications, such as in autonomous vehicles, the computational power available is often limited to embedded systems, which may not be able to run complex models like GRIP++ efficiently. Moreover, the analysis highlighted the inherent uncertainties in trajectory prediction models. Even with a high prediction accuracy, there remains a small chance of incorrect predictions, which could have significant consequences in safety-critical applications like autonomous driving. These findings underscore the importance of ongoing research to improve the reliability and robustness of trajectory prediction models.

While the GRIP++ algorithm shows promise in enhancing the accuracy of pedestrian trajectory predictions, further work is needed to address the challenges of deploying these models in real-world systems.

Appendices

A. Appendix A: Code Listings and Scripts

A.1 CUDA Adjustments for Colab Environment

The original GRIP++ code was designed for a local environment with CUDA support. When running the code on Google Colab, adjustments were made to ensure compatibility with Colab's environment.

```
# Original code to specify device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Modified code for Colab compatibility
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)

# In some parts of the code, move tensors to device as needed
tensor = tensor.to(device)
```

The main change involves ensuring that all tensors and models are explicitly moved to the GPU (CUDA) when available, which is crucial in a cloud-based environment like `Google Colab`.

A.2 Running the GRIP++ Code

Adjust the `main.py` code to handle any necessary changes for training and testing on your dataset. Example modifications include:

- Adjusting the dataset paths.
- Modifying the train and test split to match the structure of the JAAD dataset.
- Configuring the epochs and other hyperparameters.

```
# Load the dataset (adjusted for JAAD data)
train_dataset = load_dataset('/path_to_train_data')
test_dataset = load_dataset('/path_to_test_data')

# Set epochs and learning rates
num_epochs = 50  # Adjust as needed
learning_rate = 0.001

# Train and test the model
train(model, train_dataset, num_epochs, learning_rate)
test(model, test_dataset)
```

A.2.1 Training and Testing

Train the model by running the modified main.py. During this phase, you can adjust the number of epochs, batch size, and other hyperparameters as necessary.

```
!python main.py --train --epochs 50
```

Once training is complete, select the model that performs the best (based on validation metrics) and run it on the test data.

A.2.2 Visualization

After obtaining the predictions, visualize them by overlaying the predicted trajectories on the test video using the visualization.py script. You can also add additional segmentation layers (e.g., for driveable areas) using models like YOLO and DeepLabV3+.

```
!python visualize.py --video_path /path_to_test_video.mp4 --predictions /path_t
```

A.3 YOLO and DeepLabV3+ for Visualization

To add object detection (using YOLO) and semantic segmentation (using DeepLabV3+) to the prediction visualizations:

```python
# Load pre-trained YOLO and DeepLabV3+ models
yolo_model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
deeplab_model = models.segmentation.deeplabv3_resnet101(pretrained=True).eval()

# Apply object detection and segmentation to each frame
for frame in video_frames:
    results = yolo_model(frame)
    masks = deeplab_model(frame)['out']

    # Overlay predictions on the video frame
    frame_with_predictions = overlay_predictions(frame, results, masks)
    output_video.write(frame_with_predictions)
```

Bibliography

[1] GameDev Academy. *Atan2 C++ Tutorial - Complete Guide*. Accessed: 2024-08-31. 2024. URL: https://gamedevacademy.org/atan2-c-tutorial-complete-guide/.

[2] Quang-Huy Che et al. *TwinLiteNetPlus: A Stronger Model for Real-time Drivable Area and Lane Segmentation*. 2024. arXiv: 2403.16958 [cs.CV]. URL: https://arxiv.org/abs/2403.16958.

[3] Liang-Chieh Chen et al. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: *CoRR* abs/1802.02611 (2018). arXiv: 1802.02611. URL: http://arxiv.org/abs/1802.02611.

[4] NVIDIA Corporation. *CUDA Toolkit Documentation*. Accessed: 2024-08-31. 2024. URL: https://docs.nvidia.com/cuda/.

[5] Henggang Cui et al. "Multimodal Trajectory Predictions for Autonomous Driving using Deep Convolutional Networks". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 2090–2096. DOI: 10.1109/ICRA.2019.8793868.

[6] Kleio Fragkedaki et al. *Pedestrian Motion Prediction Using Transformer-based Behavior Clustering and Data-Driven Reachability Analysis*. 2024. arXiv: 2408.15250 [cs.CV]. URL: https://arxiv.org/abs/2408.15250.

[7] Andreas Geiger et al. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* (2013).

[8] Yanjun Huang et al. "A Survey on Trajectory-Prediction Methods for Autonomous Driving". In: *IEEE Transactions on Intelligent Vehicles* 7.3 (2022), pp. 652–674. DOI: 10.1109/TIV.2022.3167103.

[9] Hao Jiang et al. "Motion Query-based Multimodal Vehicle Trajectory Prediction for Autonomous Driving". In: *2023 7th CAA International Conference on Vehicular Control and Intelligence (CVCI)*. 2023, pp. 1–6. DOI: 10.1109/CVCI59596.2023.10397282.

[10] Xin Li, Xiaowen Ying, and Mooi Choo Chuah. "GRIP: Graph-based Interaction-aware Trajectory Prediction". In: *2019 IEEE INTELLIGENT TRANSPORTATION SYSTEMS CONFERENCE (ITSC)*. IEEE. 2019.

[11] Xin Li, Xiaowen Ying, and Mooi Choo Chuah. “GRIP++: Enhanced Graph-based Interaction-aware Trajectory Prediction for Autonomous Driving”. In: *arXiv preprint arXiv:1907.07792* (2020).

[12] Wenjie Luo, Bin Yang, and Raquel Urtasun. *Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting with a Single Convolutional Net*. 2020. arXiv: 2012.12395 [cs.CV]. URL: https://arxiv.org/abs/2012.12395.

[13] Yuexin Ma et al. “Trafficpredict: Trajectory prediction for heterogeneous traffic-agents”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 6120–6127.

[14] Dean Pomerleau. “ALVINN: An Autonomous Land Vehicle In a Neural Network”. In: *Proceedings of (NeurIPS) Neural Information Processing Systems*. Ed. by D.S. Touretzky. Morgan Kaufmann, Dec. 1989, pp. 305–313.

[15] Amir Rasouli, Iuliia Kotseruba, and John K Tsotsos. “Are they going to cross? A benchmark dataset and baseline for pedestrian crosswalk behavior”. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 206–213.

[16] Amir Rasouli, Iuliia Kotseruba, and John K Tsotsos. “It’s Not All About Size: On the Role of Data Properties in Pedestrian Detection”. In: *ECCVW*. 2018.

[17] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *arXiv* (2018).

[18] Luca Rossi et al. “Vehicle trajectory prediction and generation using LSTM models and GANs”. In: *PLOS ONE* 16.7 (July 2021), pp. 1–28. DOI: 10.1371/journal.pone.0253868.

[19] Raymond A. Serway and John W. Jewett. *Physics for Scientists and Engineers with Modern Physics*. Cengage Learning, 2013.

[20] Diego A. Silva et al. *A Recurrent YOLOv8-based framework for Event-Based Object Detection*. 2024. arXiv: 2408.05321 [cs.CV]. URL: https://arxiv.org/abs/2408.05321.

[21] Pei Sun et al. “Scalability in perception for autonomous driving: Waymo open dataset”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 2446–2454.

[22] Zhigang Sun et al. *SemanticFormer: Holistic and Semantic Traffic Scene Representation for Trajectory Prediction using Knowledge Graphs*. 2024. arXiv: 2404.19379 [cs.CV]. URL: https://arxiv.org/abs/2404.19379.

[23] Arsal Syed and Brendan Tran Morris. “Semantic scene upgrades for trajectory prediction”. In: *Machine Vision and Applications* (2023). URL: https://doi.org/10.1007/s00138-022-01357-z.