Dissertation on

"Emulating Emergency Vehicle Navigation in Mixed Traffic Environment using Graph Prediction and Simulation"

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

UE20CS461A – Capstone Project Phase - 2

Submitted by:

Shubham S	**PES1UG20CS420**
Shuchith B U	**PES1UG20CS421**
Siddarth M P	**PES1UG20CS423**
Amogh N Rao	**PES1UG20CS625**

Under the guidance of

Prof. Bhaskarjyoti Das

Professor

PES University

June - Nov 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

"Emulating Emergency Vehicle Navigation in Mixed Traffic Environment using Graph Prediction and Simulation"

is a bonafide work carried out by

Shubham S	**PES1UG20CS420**
Shuchith B U	**PES1UG20CS421**
Siddarth M P	**PES1UG20CS423**
Amogh N Rao	**PES1UG20CS625**

in partial fulfillment for the completion of seventh semester Capstone Project Phase - 2 (UE20CS461A) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period June. 2023 – Nov. 2023. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 8th semester academic requirements in respect of project work.

Signature	Signature	Signature
Prof. Bhaskarjyoti Das	Dr. Shylaja S S	Dr. B K Keshavan
Professor	Chairperson	Dean of Faculty

External Viva

Name of the Examiners	**Signature with Date**
1. ______________________	______________________
2. ______________________	______________________

DECLARATION

We hereby declare that the Capstone Project Phase - 2 entitled **“Emulating Emergency Vehicle Navigation in Mixed Traffic Environment using Graph Prediction and Simulation”** has been carried out by us under the guidance of Prof. Bhaskarjyoti Das,Professor and submitted in partial fulfillment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester June – November 2023. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

Shubham S	**PES1UG20CS420**	Shubham
Shuchith B U	**PES1UG20CS421**	shuchith
Siddarth M P	**PES1UG20CS423**	Siddarth
Amogh N Rao	**PES1UG20CS625**	Amogh

ACKNOWLEDGEMENT

We would like to express our gratitude to Prof. Bhaskarjyoti Das, Department of Computer Science and Engineering, PES University, for his continuous guidance, assistance, and encouragement throughout the development of this UE20CS461A - Capstone Project Phase - 2

We are grateful to the project coordinator, Dr. Priyanka H., all the panel members & the supporting staff for organizing, managing, and helping the entire process.

We take this opportunity to thank Dr. Shylaja S S, Chairperson, Department ofComputer Science and Engineering, PES University, for all the knowledge and support we have received from her.

We are grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, Dr. B.K. Keshavan, Dean of Faculty, PES University for providing us various opportunities and enlightenment during every step of the way.

Finally, this project could not have been completed without the continual support and encouragement we have received from our family and friends.

ABSTRACT

Emergency vehicles (EVs) must move through areas with mixed traffic as fast as they can in order to deliver aid in a timely manner. The dynamic and unpredictable nature of traffic makes this a difficult task. In some bottleneck situations, such as highway ramping and intersections, reinforcement learning (RL) has been used to simulate traffic and improve EV navigation. However, the complexity of real-world traffic environments is not fully captured by RL-based simulations, which are frequently restricted to a small number of entities. In this paper, we propose a novel method that makes use of GraphSAGE and a custom action policy function to simulate EV navigation in mixed-traffic environments. A graph neural network (GNN) algorithm called GraphSAGE can be used to learn node representations, or the representations of the relationships between nodes in a graph, through training. The custom action policy function uses the link states that we predict using GraphSAGE to determine the EVs' next course of action.

The custom action policy function makes decisions about lane changes, speed adjustments, and acceleration based on the predicted link states, vehicle types, and relative positions of vehicles. We demonstrate that, when compared to baseline approaches, our approach can significantly improve the navigation efficiency of EVs. Our findings demonstrate how well graphs model and simulate intricate traffic situations. Our simulation can be used to investigate how various traffic situations and vehicle kinds affect EV navigation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The integration of autonomous vehicles (AVs) into our traffic networks is happening quite quickly. Although completely autonomous driving is still a ways off, in the near future, traffic will be composed of both human-driven and autonomous vehicles. The unpredictable behavior of other drivers, the restricted sensor range, and the intricate traffic networks present a variety of difficulties for emergency vehicles (EVs) operating in this mixed traffic environment. In situations where there is mixed traffic, EV navigation must be both safe and effective in order to quickly assist individuals in need. However, because traffic is dynamic and complex, this task is difficult.

Learning representations of nodes in a graph is possible with the use of machine learning models called graph neural networks (GNNs). The efficacy of GNNs has been demonstrated in numerous traffic network-related tasks, including congestion prediction and traffic forecasting. As well as learning representations of EVs that capture their objectives and current state, GNNs can be used to model the traffic network's overall state. Equipped with this data, an action policy function that is comprehensive in nature can be developed to assist EVs in safely and efficiently navigating mixed traffic environments.

In this paper, we propose an innovative method that uses GNNs for efficient EV navigation in mixed traffic environments. Our method tackles the problem of sensor failures in real life situations by utilizing GraphSAGE for link prediction. Additionally, we create a unique action policy function that considers the EV's current condition as well as the types of vehicles in the network and the traffic network's overall state. We assess our suggested method on the generated environment and demonstrate that, in terms of both navigation efficiency and safety, it performs noticeably better than baseline methods.

CHAPTER 2

PROBLEM STATEMENT

Since autonomous vehicles (AVs) have been seamlessly incorporated into contemporary traffic environments, emergency vehicles (EVs) are required to navigate alongside their human-driven and autonomous counterparts. This produces a challenging dynamic. This integration is hampered by a number of factors, including the unpredictability of human drivers, the limitations of sensors, and the complexity of traffic networks. In order to ensure an efficient emergency response, it becomes more challenging for emergency vehicles to travel swiftly and securely in scenarios with mixed traffic because of all these problems.

Current Limitations and Research Gap:

Although some bottleneck problems have been addressed by simulations based on reinforcement learning (RL), these approaches often fail to capture the intricate dynamics of real-world traffic conditions. These simulations are typically constrained by a narrow focus on a limited set of entities, which makes it challenging for them to accurately represent the range of interactions and situations that emergency vehicles face. Furthermore, the potential for sensor failures in real-world situations adds to the complexity of EV navigation. This means that there is a big gap in the methods that can provide EVs running in the unexpected and dynamic mixed traffic settings with dependable and adaptable navigation strategies.

Proposed Research Objectives :

Here, we propose a unique approach to capture and model the complex relationships inside traffic networks using graph neural networks (GNNs), specifically GraphSAGE, in order to address these problems. A more comprehensive understanding of the mixed traffic environment is made possible by the GNN architecture, which makes it possible to learn representations for both emergency vehicles and the larger traffic network. As an addition, a customized action policy function is presented, which uses GraphSAGE's projected link states to guide decision-making over lane changes, speed modifications, and acceleration. The main objective is to create a technique that greatly improves EV navigation efficiency and safety, especially in situations where there is mixed traffic, sensor failures, and other complications. This will further the progress of autonomous and emergency vehicle integration in modern traffic environments.

CHAPTER 3

LITERATURE SURVEY

In this chapter, we present the current knowledge of the area and review substantial findings that help shape, inform and reform our study.

3.1 Simulation Environment :

A diverse range of simulation environments has been employed to assess and validate autonomous driving systems in mixed traffic environments. The most prevalent choice is the SUMO platform [1][2][3][4][9], which provides a realistic traffic simulation framework for modeling various road networks and the interaction between autonomous and human-driven vehicles. Additionally, the CARLA simulation [15] has been used to train and evaluate the DiGNet system, offering high-fidelity scenarios across complex maps, including urban, rural, and highway environments.

The robustness and generalizability of the suggested autonomous driving systems are greatly enhanced by the versatility and depth of these simulation platforms. However, rather than concentrating on particular use cases, the majority of research studies currently in existence assess the overall performance of autonomous driving systems. The lack of research in this area restricts the development and assessment of customized solutions for particular traffic situations, like emergency vehicle navigation.

3.2 Graph Representation and GNN :

Graph representation and the application of Graph Neural Networks (GNN) emerge as pivotal elements in enhancing decision-making for autonomous vehicles. Several papers leverage graph-based models to represent vehicle interactions and properties. In [1], every vehicle is depicted as a node, and interactions as edges in an undirected network, forming the basis for the proposed modular framework's state space. In [3], the Graphic Convolution Q network (GCQ) integrates Deep Q Network (DQN) and Graph Convolutional Neural Network (GCN) to facilitate cooperative lane change decisions, using graph-based techniques to aggregate collaborative sensing data. The adoption of GNN is further exemplified in [5][6][13][15], where graph structures capture spatial and temporal relationships among vehicles, enabling efficient decision-making in various cooperative and mixed-traffic scenarios. These approaches underscore the efficacy of graph-based representations and GNN architectures in modeling complex interactions and improving the decision-making capabilities of autonomous vehicles.

3.3 Action Policy and outcome :

The proposed methodologies in the surveyed papers exhibit a variety of action policies designed to optimize autonomous vehicle decision-making. Reinforcement Learning (RL) algorithms, such as Deep Q-Learning [3], Curriculum through Self-Play [2], and Policy-based training methods [9][13], are recurrent themes. The utilization of advanced RL techniques, like Duelling Double DQN [1], Curriculum through Self-Play [2], and LSTM-Q [3], highlights the importance of effective learning strategies in achieving superior outcomes in interactive traffic scenarios. The incorporation of novel approaches, such as the Multi-Agent Reinforcement Learning (MARL) framework with Connected Automated Vehicle Graph [4] and the DiGNet system [15], showcases the diversity of methodologies aimed at improving efficiency, safety, and cooperation in autonomous driving.

However, lane changes are rarely taken into consideration, and the majority of current action spaces are restricted to simple movements like acceleration and deceleration. In addition, the number of vehicles that could be simulated was constrained by the complexity of RL algorithms. This limitation is addressed by our proposed custom action policy function, which allows complex maneuvers such as lane changes based on the current graph state. This enables us to model more realistic traffic scenarios and simulate a larger network.

3.4 Graph Convolution-Based Deep Reinforcement Learning for Multi-Agent Decision-Making in Mixed Traffic Environments [1]

Liu, Qi, et al

The paper[1] proposes a modular framework for multi-agent decision-making in interactive traffic scenarios that combines Deep Reinforcement Learning (DRL) and Graph Neural Network (GNN). The framework in intelligent transportation systems tries to improve the precision of coordinated choices. Several GRL strategies are used and evaluated in a driving simulation. According to the findings, combining GNN and DRL improves the creation of lane-change behaviors, and using GNN successfully displays road interactions. The framework's modular nature allows it to be compared to other GRL algorithms.

Python and a variety of third-party libraries served as the foundation for its development. The three-lane highway with two ramp exits is part of the SUMO platform, which is used to construct the simulation environment. In the current circumstance, autonomous vehicles (AVs) and human-driven vehicles (HVs) must collaborate to complete driving tasks successfully. The Intelligent Driver Model (IDM) is used for longitudinal control of automobiles, while the recommended GRL algorithm is used for lateral control of autonomous vehicles. The simulation allows the GRL network to be trained constantly by updating the status of each vehicle based on the control commands generated.

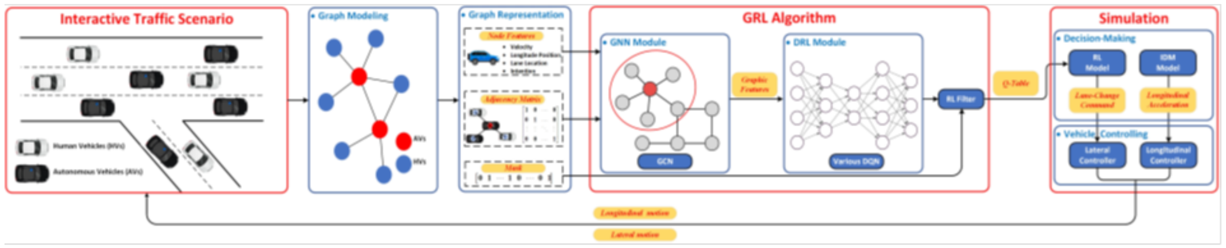


Fig. 1 Proposed framework.

In the scenario, every vehicle is represented as a node, and the interactions between them are represented as edges in an undirected network. Graph representation is used to model vehicle interactions and properties. The state space of a graph is defined by the matrices of adjacency (At), RL-filter (Ft), and node characteristics (Nt). The adjacency matrix portrays vehicle interaction by assuming that every vehicle can communicate with every other vehicle, every AV can communicate with every HV within its sensing range, and every vehicle can communicate with every other AV.

The scenario's action space has three unique lane-changing instructions: lane-changing to the left, lane-changing to the right, and continue ahead. Each agent's decision-making behaviour is generated via the GRL algorithm. The average testing reward is used to compare how well various GRL procedures operate. The results show that, when compared to rule-based and LSTM-based techniques, the proposed framework—specifically, the Duelling Double DQN method—outperforms them. In an interactive traffic environment, the combination of GNN and DRL effectively delivers cooperative lane-change decisions for autonomous vehicles

3.5 Multi-Agent Deep Reinforcement Learning to Manage Connected Autonomous Vehicles at Tomorrow's Intersections [2]

Antonio and C. Maria-Dolores

The paper introduces advanced Reinforced Autonomous Intersection Management (adv.RAIM), a pioneering approach to Autonomous Intersection Management (AIM) employing Multi-Agent Deep Reinforcement Learning (MADRL) and trained through Curriculum via Self-Play. This innovative technique not only constructs intelligent AIMs capable of proactively managing Connected and Autonomous Vehicles (CAVs) in complex scenarios but also enables collaborative control among CAVs. The simulation environment consists of a junction with four approaches and three lanes per approach, utilizing SUMO for microscopic traffic simulation. Simulated scenarios include various flow types (vertical and horizontal origin) and fixed/variable flow situations. Results demonstrate adv.RAIM's superior efficiency in regulating CAVs at junctions, surpassing alternative traffic management algorithms across scenarios. Graphs depict performance metrics such as average travel time, waiting time, and queue length, highlighting adv.RAIM's consistent outperformance.

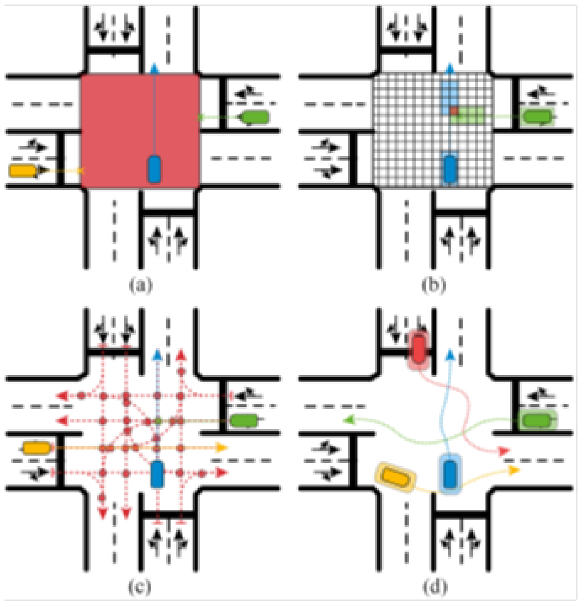


Fig. 2. Strategies for managing conflicts in AIM

The model's learning curve illustrates improvement over time, driven by a deep reinforcement learning algorithm utilizing Self-Play and curriculum-based learning. The study concludes that adv.RAIM excels in mastering complex real-world traffic dynamics, establishing its superiority over existing traffic control algorithms [2].

3.6 A drl-based multiagent cooperative control framework for cav networks: a graphic convolution q network [3]

Dong, Jiqian, Sikai Chen, Paul Young Joun Ha, Yujie Li, and Samuel Labi

The paper introduces the Graphic Convolution Q network (GCQ), a Deep Reinforcement Learning (DRL) technique applied to a network of connected autonomous vehicles (CAVs) to facilitate cooperative lane change decisions. The GCQ model merges the strengths of Deep Q Network (DQN) and Graphic Convolution Neural Network (GCN), enabling multiple CAVs to collaboratively and safely navigate multi-lane routes. Utilizing Q Learning with Target Network and Experience Replay, the model architecture incorporates a Q network, an output layer, a GCN layer, and an FCN encoder. A "warming up" phase precedes the main training loop.

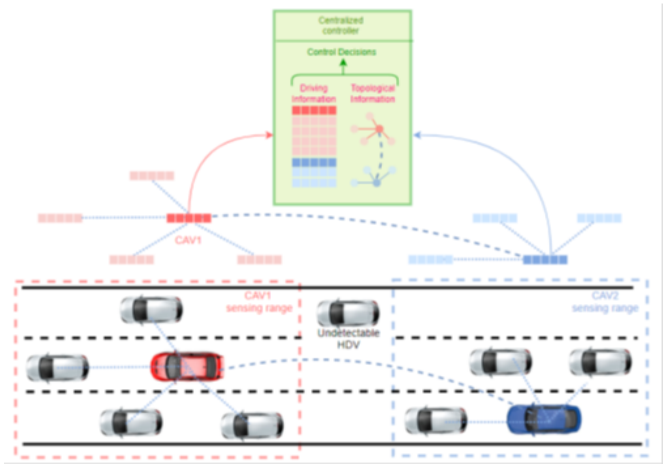


Figure3. Visual representation of CAV Network

Results demonstrate that both LSTM-Q and GCQ models can outperform a rule-based model, converging within a predefined number of steps. Through testing in diverse traffic density scenarios, metrics such as mean, median, and standard deviation of episode payouts are calculated. In summary, the paper proposes a DRL-based cooperative lane change mechanism for CAV networks, highlighting the effectiveness of the GCQ model in integrating information and making informed decisions. Trained with Q Learning, the GCQ model showcases superior performance compared to a rule-based model, emphasizing its potential for enhancing safety and mobility in connected autonomous vehicle networks [3].

3.7 Efficient Connected and Automated Driving System with Multi-agent Graph Reinforcement Learning [4]

T. Shi, J. Wang, Y. Wu, L. Miranda-Moreno, and L. Sun

The paper introduces a novel solution to enhance cooperation in mixed autonomous vehicle networks, focusing on connected and autonomous vehicles (CAVs). Employing a multi-agent reinforcement learning (MARL) framework, the authors integrate the Connected Automated Vehicle Graph (CAVG) to simulate effective interactions among CAVs. The proposed approach surpasses current benchmarks in CAV control, as demonstrated through comprehensive tests.

The CAV control framework, depicted in Figure 4, combines SUMO and OpenAI Gym within the CAVG framework to model CAV traffic flow and evaluate the MARL's effectiveness. The simulation scenario involves a four-way intersection with both autonomous vehicles (CAVs) and human-driven vehicles (HDVs), considering varying traffic volumes and CAV percentages.

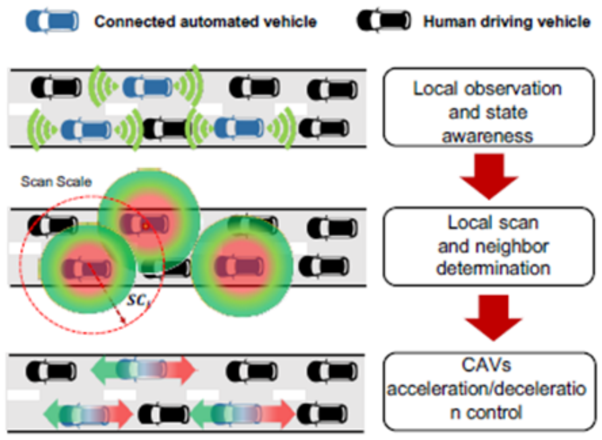

Connected automated vehicle
Human driving vehicle
Local observation and state awareness
Scan Scale
SC_i
Local scan and neighbor determination
CAVs acceleration/deceleration control

Fig. 4. CAV control framework.

The authors introduce a graph-based method, the CAVG, to simulate reciprocal interactions between CAVs. They utilize attention-based graph convolutional networks (GCNs) for training CAV representations and interactions, introducing a novel graph attention mechanism that considers the significance of different CAVs in decision-making processes.

The MARL-CAVG architecture follows the centralised training and decentralised execution (CTDE) paradigm, where actor and critic networks are centrally trained using an experience replay buffer. During execution, each CAV makes decisions based on local observations and the learned action policy. The proposed strategy is evaluated for efficacy and safety, demonstrating superior performance in average speed, travel time, and safety compared to advanced baselines [4].

3.8 Automatic Intersection Management in Mixed Traffic Using Reinforcement Learning and Graph Neural Networks [5]

Klimke, Marvin, Benjamin Völz, and Michael Buchholz

The paper introduces a novel approach for cooperative multi-agent planning in mixed traffic scenarios, leveraging graph neural networks (GNN) and reinforcement learning. The proposed architecture addresses uncertainty around non-connected vehicles, employing relation-dependent edges and an attention mechanism. Tested in the Highway-env simulation framework for mixed-autonomy traffic, the scenario involves a four-way intersection with human-driven vehicles (HVs) and automated vehicles (AVs). Equipped with sensors and communication technology, AVs interact with both the intersection management system and other vehicles. The graph-based input representation captures the complex interactions, and the proposed GNN architecture enhances cooperative planning. Utilizing reinforcement learning, the system calculates optimal actions, demonstrating superior traffic efficiency and safety over a broad range of automation rates compared to baseline techniques, particularly in high traffic demand situations [5].

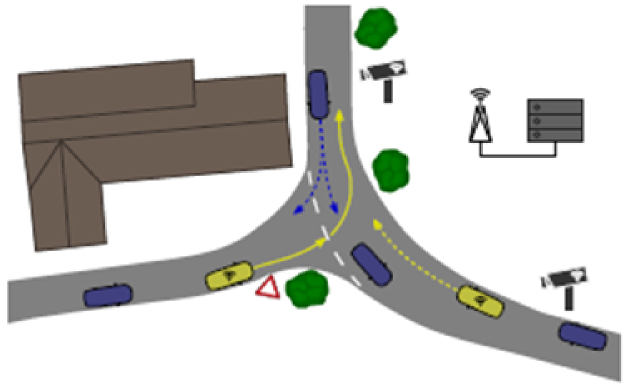


Fig. 5. Collaborative planning in mixed traffic.

3.9 Cooperative Behavior Planning for Automated Driving using Graph Neural Networks [6]

Klimke, Marvin, Benjamin Völz, and Michael Buchholz

The study [6] introduces a reinforcement learning (RL) technique for cooperative multi-agent planning in urban autonomous driving, aiming to manage intersections based on real traffic data. Utilizing machine learning, specifically graph neural networks (GNN), the proposed approach learns the action policy and encodes input features. The simulation environment, based on the Highway Environment, offers a realistic setting for multi-agent planning with dynamic vehicle interactions. The graph-based input representation (Figure 6) models each vehicle as a node, with edges capturing relationships, and employs GNNs to analyze the structure and extract representations. The agent's behaviors are guided by reward signals, fostering successful and safe driving. Evaluation with artificial models and real traffic data demonstrates the learned planner's ability to significantly improve traffic flow by increasing vehicle throughput and reducing induced pauses. Overall, the research presents a promising approach for cooperative multi-agent planning in urban autonomous driving, leveraging GNNs and reinforcement learning to enhance traffic efficiency and address complex planning concerns [6].

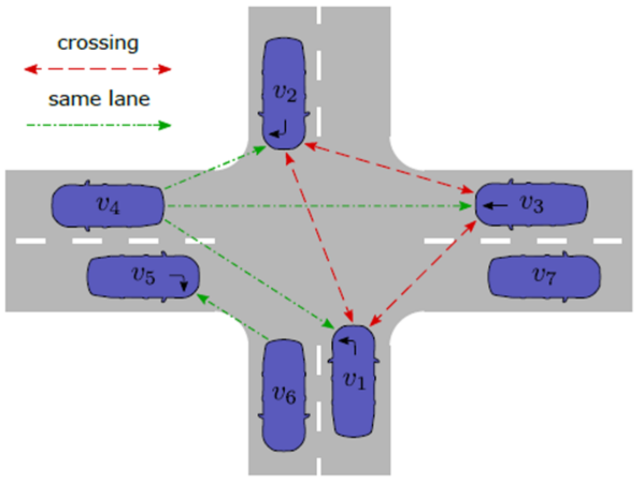


Fig. 6. The graph-based input representation

3.10 An Enhanced Graph Representation for Machine Learning Based Automatic Intersection Management [7]

Klimke, Marvin, Benjamin Völz, and Michael Buchholz

The paper [7] focuses on optimizing traffic signal management through reinforcement learning to alleviate congestion and improve traffic flow in urban areas. The proposed system utilizes deep Q-learning to learn optimal signal timings from real-time traffic data. The simulation environment, mimicking an actual road network with intersections and traffic patterns, assesses the algorithm's effectiveness under diverse traffic scenarios, including high peak hour traffic and varying traffic volumes. Graph-based representations, such as Figure 7, illustrate the simulation results, showcasing performance indicators like wait periods and transit times. Bar charts highlight the proposed algorithm's significant reductions in queue length and travel time compared to baseline approaches. The reinforcement learning-derived action policy for traffic light control, based on past data and real-time statistics, yields notable improvements, reducing queues by up to 40% and travel times by up to 30%. The study suggests that deep Q-learning offers a promising avenue for enhancing urban mobility and mitigating traffic congestion through efficient traffic signal management [7].

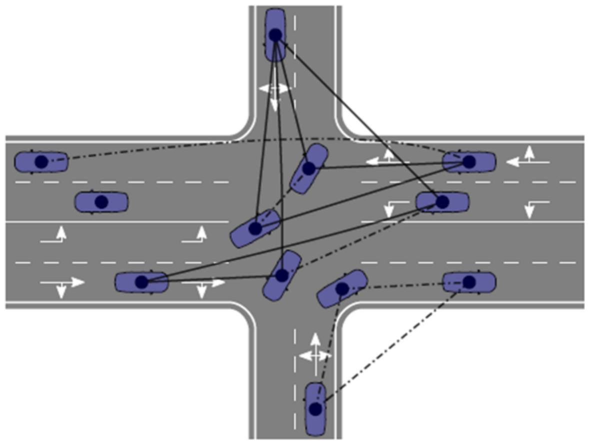


Fig.7. Graph based representation of autonomous vehicles

3.11 Spatio-weighted information fusion and DRL-based control for connected autonomous vehicles [8]

J. Dong et al

The paper [8] introduces a deep set Q-learning model for autonomous vehicle lane-changing decisions within a simulated corridor environment featuring multiple lanes. The model aims to develop an action plan for autonomous vehicles (CAVs) to strategically change lanes in response to environmental signals. Comparative evaluation against four baseline models highlights the superior performance of the recommended linear weighted deep set Q-learning model. This model consistently outperforms others, enhancing CAV operational efficiency by proactively making decisions to avoid traffic congestion and slowdowns. The study presents median and mean performance metrics with a 95% confidence interval across diverse scenarios with varying vehicle numbers. The action strategy relies on the deep set Q-learning architecture, incorporating the output layer, Q network, and encoding network. Trained through an experience replay buffer and optimized with the Adam optimizer, the model's performance is evaluated based on rewards obtained in the simulated environment. The research underscores the effectiveness of the proposed model for enhancing autonomous vehicle lane-changing decisions [8]..

3.12 Multi-agent decision-making modes in uncertain interactive traffic scenarios via graph convolution-based deep reinforcement learning [9]

Gao, Xin, et al

The paper [9] introduces a multi-agent decision-making framework for complex interactive traffic scenarios, leveraging graph convolution-based deep reinforcement learning. The simulation environment, set up using the SUMO platform, features a three-lane highway scenario with both human-driven vehicles (HVs) and autonomous vehicles (CAVs). The framework aims to provide CAVs with decision-making modes that prioritize various objectives, including driving task, traffic efficiency, ride comfort, and safety. The graph representation of vehicles as nodes, with an adjacency matrix capturing interactions, is evaluated using a graph neural network module comprising fully connected and graph convolutional layers. The proposed reward function matrix allows the adjustment of decision-making modes to emphasize different objectives. Evaluation metrics include efficiency, safety, comfort, and task indexes. The deep reinforcement learning-based action policy, utilizing graph convolution, demonstrates potential improvements in the decision-making abilities of networked autonomous vehicles in uncertain traffic environments [9].

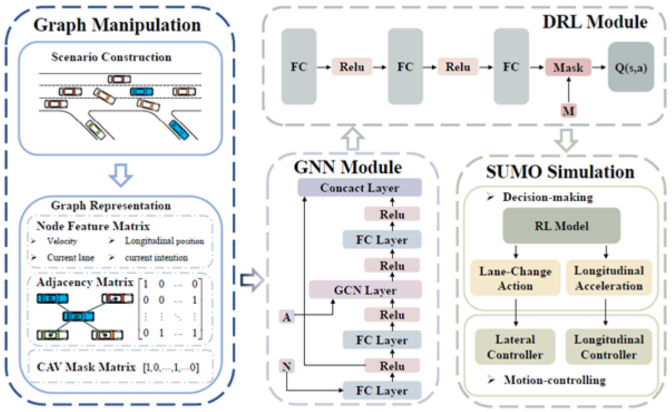


Fig.8. Representation of Designed Framework

3.13 Graph Reinforcement Learning-Based Decision-Making Technology for Connected and Autonomous Vehicles: Framework, Review, and Future Trends [10]

Liu, Qi, Xueyuan Li, Yujie Tang, Xin Gao, Fan Yang, and Zirui Li

The article [10] conducts a thorough evaluation of deep reinforcement learning applications in autonomous vehicle decision-making, specifically utilizing graph neural networks (GNNs). The literature retrieval process, involving databases like IEEE Xplore and Google Scholar, targeted relevant publications using keywords such as deep reinforcement learning and graph reinforcement learning.

The assessment focuses on simulation scenarios, employing open-source frameworks like "highway-env" and "Flow" to mimic highway driving with partial autonomy.
Various graph types, including graph attention networks (GATs), spatial-temporal graph convolutional networks (STGCNs), and graph convolutional networks (GCNs), are explored for their contributions to autonomous vehicle decision-making. These graph-based models capture spatial and temporal relationships in the data, aiding decision-makers.

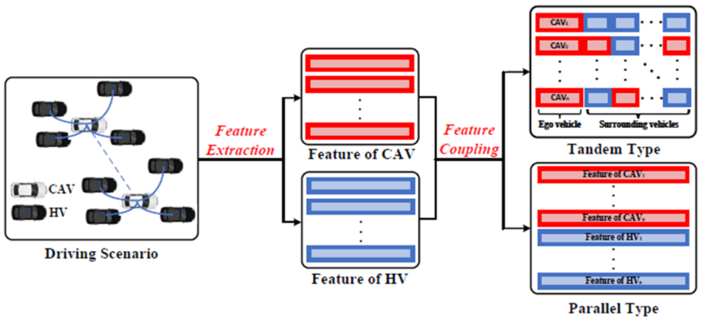


Figure 9. formulations of the node feature matrix.

The study delves into the use of deep reinforcement learning algorithms to derive optimal action and outcome rules for autonomous vehicles, considering choices like lane changes, speed modifications, and collision avoidance. In conclusion, the paper provides a comprehensive exploration of challenges and opportunities associated with deep reinforcement learning in autonomous driving decision-making using graph neural networks [10].

3.14 Vehicle trajectory prediction in connected environments via heterogeneous context-aware graph convolutional networks [11]

Lu, Yuhuan, Wei Wang, Xiping Hu, Pengpeng Xu, Shengwei Zhou, and Ming Cai

The approach described in the paper[11] proposes a model called Heterogeneous Context-Aware Graph Convolutional Networks for vehicle trajectory prediction in a networked context. The model is composed of three main components: scene context extraction, individual context extraction, and interaction model.

1. Individual Context Extraction: This section focuses on determining the distinctive dynamics of linked automobiles. A Spatio-Temporal Graph Convolutional Network is used to simulate the dynamic interactional patterns (STGCN). The distinct context is provided by the prior statuses of the connected automobiles.

2. Scene Context Extraction: By including details about the driving environment, this section aims to improve the accuracy of the trajectory prediction. The scene picture stream is transformed into a compressed scene context. This backdrop provides further information about the roads and infrastructure.

3. Interaction Context Modelling: This section documents how connected vehicles communicate with one another. It represents the network nodes using the collected scene and human contexts. A Spatio-Temporal Dynamic Graph Convolutional Network (STDGCN) is used to simulate the spatial and temporal evolutions of interactional patterns. This contributes to the preservation of the context of the high-fidelity interaction.

The decoder generates future trajectories by merging and feeding in the three contexts (interaction, scene, and individual). After being validated on real-world datasets, the proposed model outperforms cutting-edge approaches in terms of prediction accuracy and stability.

The technology combines graph neural networks, trajectory prediction, and connected vehicles to correctly estimate automotive paths in a connected environment.

3.15 GraphSAGE-Based Dynamic Spatial–Temporal Graph Convolutional Network for Traffic Prediction [12]

Liu, Tao, Aimin Jiang, Jia Zhou, Min Li, and Hon Keung Kwan

The research [12] presents a new deep learning architecture for traffic prediction known as the DST-Graph Convolutional Network, which is based on the GraphSAGE-Based Dynamic Spatial-Temporal Graph. Because the proposed model simultaneously captures dynamic geographical and temporal linkages, it may be able to predict traffic patterns more correctly. The model consists of three modules: a localised spatial-temporal embedding module, a long-range temporal convolution module, and an output module. The model is evaluated on five real-world datasets, and the results show that it has competitive computation efficiency and outperforms baselines.

The proposed model is evaluated using five real-world datasets. These datasets cover a variety of traffic networks (such as urban road networks and highway networks), traffic data types (such as flow and speed), and data sizes. For the experiments, Python 3.7 is utilized, and the server is outfitted with an NVIDIA Tesla P100 GPU and an Intel Xeon E5-2620 v4 CPU. PyTorch is a deep learning framework that was used to build the model.

The paper employs a graph-based framework to simulate traffic networks. The traffic network is represented as a graph, with nodes representing traffic sensors and edges representing spatial interactions between the sensors. The two types of graphs utilized in the essay are static and dynamic graphs. Static graphs are used to describe the geographical relationships between traffic sensors, whereas dynamic graphs are used to depict the time connections between traffic sensors.

Using a multi-head attention method, the proposed model dynamically learns weights between traffic nodes. Weights are utilised to obtain the spatial correlations among traffic sensors and compile information from adjacent nodes. The model may also represent the long-range temporal interactions between traffic sensors through the use of dilated causal convolutions. The results demonstrate that the proposed model outperforms baselines and demonstrates competitive processing efficiency. The proposed approach may be extended to other spatiotemporal prediction challenges and used to traffic prediction tasks such as traffic flow and speed prediction.

3.16 Improving Graph Neural Network Models in Link Prediction Task via A Policy-Based Training Method [13]

Shang, Yigeng, Zhigang Hao, Chao Yao, and Guoliang Li

The paper[13] proposes a policy-based training method (PbTRM) to improve the performance of Graph Neural Network (GNN) models in link prediction tasks. When the sampling period is over, the PbTRM updates the policy network parameters and creates negative samples using a negative sample selection. Experiments on two datasets with three GNN models demonstrate the effectiveness of the proposed technique.

For the testing, the Cora and CiteSeer databases were employed. The benchmarking strategy, which randomly selects unobserved edges to generate negative examples in the training set, and the recommended PbTRM were utilized to train the GNN models. The GNN models were trained over 100 epochs.

The paper lacks a detailed discussion of graph kinds. It does point out that the node set V, the observed edge set E, and the node feature matrix X—which may be represented as a simple undirected graph G = (V, E, X)—are the usual components of a network dataset.

The PbTRM uses a negative sample selector to generate negative samples and adjusts the policy network's parameters at each sampling interval. By using the negative sample selection technique in line with Algorithm 1, the GNN models trained with the suggested PbTRM (referred to as GNN+PbTRM) generate negative samples in the training sets. The benchmarking GNN models generate the negative samples for the training set by randomly selecting unseen edges. The effectiveness of the proposed PbTRM is demonstrated by comparing the GNN+PbTRM models' performance with that of the benchmarking GNN models. The results show that the PbTRM significantly improves the link prediction tasks performance of the GNN models.

3.17 GNN-RL: Dynamic Reward Mechanism for Connected Vehicle Security using Graph Neural Networks and Reinforcement Learning [14]

Rathore, Heena, and Henry Griffith

The paper [14] introduces a framework extending the Graph Neural Network-Reinforcement Learning (GNN-RL) approach to provide consensus-based reputation estimations in Connected Vehicle (CV) networks. The method incorporates reputation estimations and vehicle topology data from roadside units (RSUs) into a separate network structure. Employing a centralized RL agent, the proposed approach encourages vehicles to deliver more accurate reputation evaluations of neighboring vehicles. In a three-lane traffic scenario simulation, the RL agent assigns reputation ratings to vehicles, demonstrating dynamic incentives based on their reputation scores. The adaptation of the graph structure includes RSU reputation estimations and vehicle topological information, utilizing the Laplace matrix to analyze CV behavior and connectivity. The proposed method effectively combines reputation measures with GNN and RL to incentivize accurate predictions and dynamic rewards for vehicles in connected car networks [14].

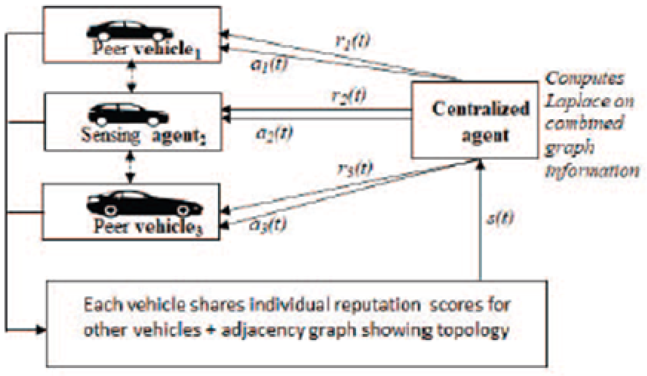

Peer vehicle$_1$
Sensing agent$_2$
Peer vehicle$_3$
$r_1(t)$
$a_1(t)$
$r_2(t)$
$a_2(t)$
$r_3(t)$
$a_3(t)$
Centralized agent
Computes Laplace on combined graph information
$s(t)$
Each vehicle shares individual reputation scores for other vehicles + adjacency graph showing topology

Fig. 10: Proposed framework for CV consensus-based security

3.18 DiGNet: Learning scalable self-driving policies for generic traffic scenarios with graph neural networks [15]

Cai, Peide, Hengli Wang, Yuxiang Sun, and Ming Liu

The paper [15] introduces DiGNet, a deep network based on graphs, designed for scalable self-driving across various traffic scenarios. Rigorously tested in the CARLA simulation (0.9.9).

DiGNet demonstrates its ability to safely operate vehicles in highways, rural areas, and cities. Outperforming other deep
learning-based self-driving algorithms, DiGNet excels in generalization and requires less manual engineering effort.

Utilizing the CARLA simulation with six complex maps, including Town04 and Town10, the authors code ten routes for each map to create diverse driving experiences. DiGNet employs a graph neural network (GNN) to handle input data represented as a graph. Comparative analyses include a baseline with a multi-layer perceptron (MLP), GAT, and DiGNet(CTL), an alternative approach generating device-level control instructions.

DiGNet's policy network processes hybrid inputs like road maps, traffic signals, route plans, and dynamic objects to generate waypoints. Evaluation metrics encompass average speed, success rate, and accident rate, showcasing DiGNet's superior performance over baselines and achieving state-of-the-art results on the CARLA benchmark. The study highlights DiGNet's robust generalization capabilities across unseen scenarios and maps [15].

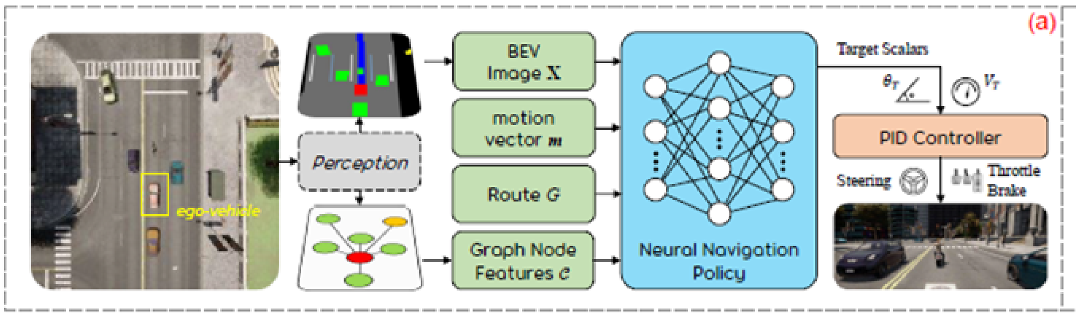


Fig 11.Schematic overview of the proposed method for self-driving.

CHAPTER 4

METHODOLOGY

.

4.1 Graph Representation :

At each timestep of the simulation, the vehicles present are identified and categorized as either HV i.e. Human Vehicle, AV i.e. Autonomous Vehicle, or EV i.e. Emergency Vehicles. The TraCI library is used to extract their attributes, such as position, speed, and lane information. This library makes communication with the SUMO traffic simulation software easier. Next, using the extracted vehicle data, a graph is built, with nodes standing in for the vehicles. Based on the position data of the two vehicles, an edge is established by calculating the distance between them using the formulae mentioned in equation (1). An edge is added to the graph if this distance drops below a set threshold.

$$d = ((x2 — x1)\hat{}2 — (y2 — y1)\hat{}2)\hat{}½$$

Edge formation is limited to interactions between AVs and HVs, EVs and HVs, or AVs and EVs in order to better represent real-world scenarios. Connections between HVs themselves are not included in this. This distinction arises from the fact that HVs are treated as passive actors in the simulation, with decision-making powers exclusively assigned to AVs and EVs.
The graph better captures the dynamics of traffic scenarios involving AVs, EVs, and HVs by restricting edge formation to these interactions

4.2 GraphSAGE and Link Prediction :

Emergency vehicle routing refers to the optimization of paths for emergency vehicles, such as ambulances, fire trucks, or police cars, to reach their destinations quickly and efficiently. Link prediction can enhance this process by predicting the likelihood of connections or interactions between vehicles, which in turn can be used to optimize emergency response strategies. In real-world scenarios, sensors are used to gather positional data of vehicles, which is then employed to establish connections between them based on their proximity . However, sensor failures can lead to data gaps, resulting in the absence of links between vehicles. Link prediction techniques can be employed to bridge these gaps by predicting the missing edges, which can then be utilized to enhance the decision making process for emergency vehicles. To predict the missing links between vehicles, we employ a specialized model called GraphSageNet. This model comprises two main components:

4.2.1. GraphSAGE:

This component is responsible for generating node embeddings for vehicles. Node embeddings are vector representations of nodes in a graph that capture their unique characteristics and their relationships with neighboring nodes. In this case, the GraphSAGE model generates node embeddings that capture the characteristics of vehicles and their interactions with other vehicles in the traffic network.

4.2.2. LinkPredictorNN:

This component is responsible for predicting the existence of links between vehicles. It takes the node embeddings generated by GraphSAGE as input and produces a link prediction score for each pair of nodes. The higher the link prediction score, the more likely it is that a link exists between the two corresponding nodes.

The GraphSAGE model consists of two layers of SAGEConv, which is a variant of GraphSage graph neural network (GNN), that aggregates information from neighboring nodes to generate node representations. The LinkPredictionNN consists of two fully connected layers. The first layer takes the output-dimensional node embeddings from the GraphSAGE model as input and produces hidden-dimensional representations. The second layer takes the hidden-dimensional representations as input and produces a single output value, which represents the probability of a link existing between the two corresponding nodes. The combined GraphSageNet model effectively predicts the missing links between vehicles by leveraging the strengths of both the GraphSAGE model and the LinkPredictionNN. The GraphSAGE model generates informative node embeddings that capture the interactions between vehicles, while the LinkPredictionNN effectively learns patterns in these embeddings to identify potential links. The GraphSageNet model is trained using a dataset obtained from the SUMO (Simulation of Urban MObility) software, utilizing the TraCI (Traffic Control Interface) API. The dataset represents a traffic network extracted from OpenStreetMap (OSM) data, encompassing X vehicles. The model is trained on a snapshot of the traffic network at a specific time instant. Edge construction for the training graph follows the same procedure outlined in the graph representation section. Within the real simulation process, once the traffic graph is constructed, GraphSageNet is employed to predict any missing links that may exist. Subsequently, the Custom Action policy is utilized to make informed decisions for both autonomous and emergency vehicles within the simulated environment.

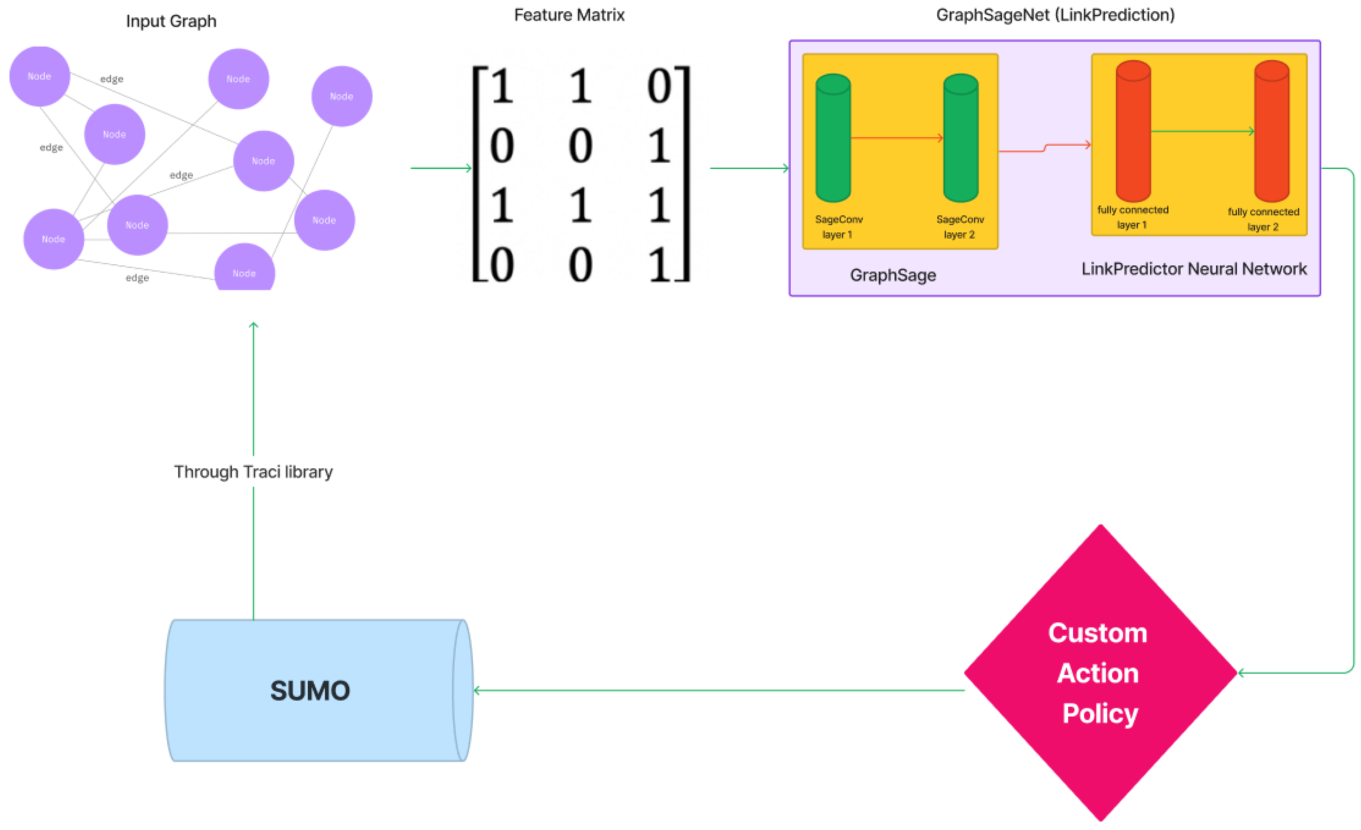
4.3 Custom Action Policy Function :

The main element in charge of choosing actions based on the traffic graph's current state is the custom action policy function. The presence of edges in the graph is necessary for it to function. The action policy function is triggered to determine a score for every vehicle when there are sufficient edges in the graph. Depending on the type of vehicle and the characteristics of its surrounding nodes and edges, this score is the basis for decision-making.

For emergency vehicles (EVs), a high score is assigned, prompting an adjustment of their speed towards their maximum allowable limit. This guarantees that electric vehicles (EVs) can quickly and effectively navigate the traffic network to get to their destinations.

The action policy function for autonomous vehicles (AVs) uses neighborhood data to decide what speed modifications are appropriate. The action policy function tells an AV to switch lanes to make room for an EV if they are in the same lane. When the autonomous vehicle (AV) is in a separate lane, it is advised to reduce speed to prevent possible collisions. The custom action policy function uses the vast amount of information found in the traffic graph to plan an orderly and effective vehicle movement that guarantees emergency vehicles can navigate through mixed traffic situations in a timely and safe manner.

4.4 Architecture :



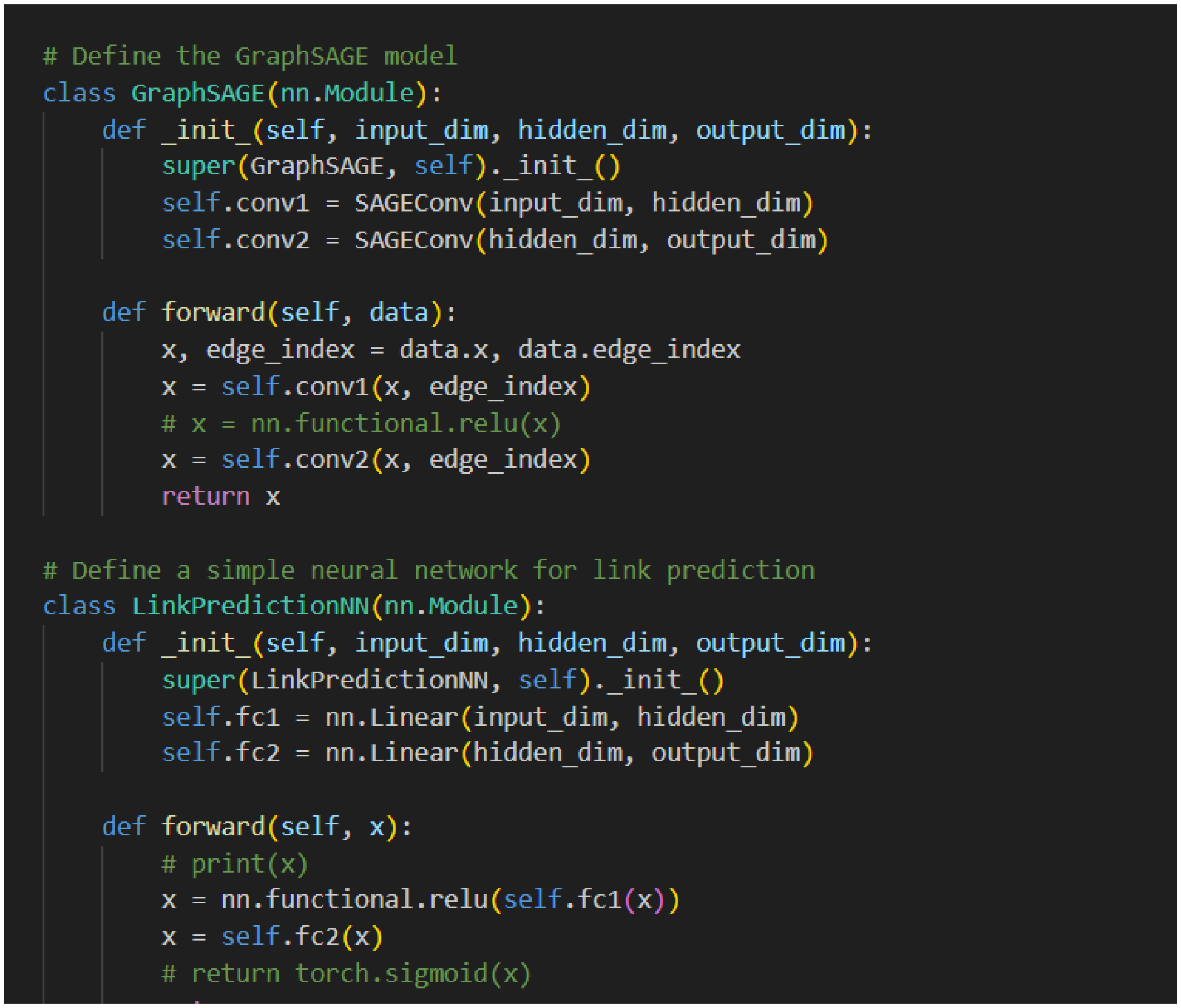
$$\begin{bmatrix}1 & 1 & 0\\0 & 0 & 1\\1 & 1 & 1\\0 & 0 & 1\end{bmatrix}$$

Fig. 12. Proposed Architecture

CHAPTER 5

IMPLEMENTATION

5.1 Graphsage , Link predictor Neural network :



```
# Define the GraphSAGE model
class GraphSAGE(nn.Module):
    def _init_(self, input_dim, hidden_dim, output_dim):
        super(GraphSAGE, self)._init_()
        self.conv1 = SAGEConv(input_dim, hidden_dim)
        self.conv2 = SAGEConv(hidden_dim, output_dim)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        # x = nn.functional.relu(x)
        x = self.conv2(x, edge_index)
        return x

# Define a simple neural network for link prediction
class LinkPredictionNN(nn.Module):
    def _init_(self, input_dim, hidden_dim, output_dim):
        super(LinkPredictionNN, self)._init_()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        # print(x)
        x = nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        # return torch.sigmoid(x)
```

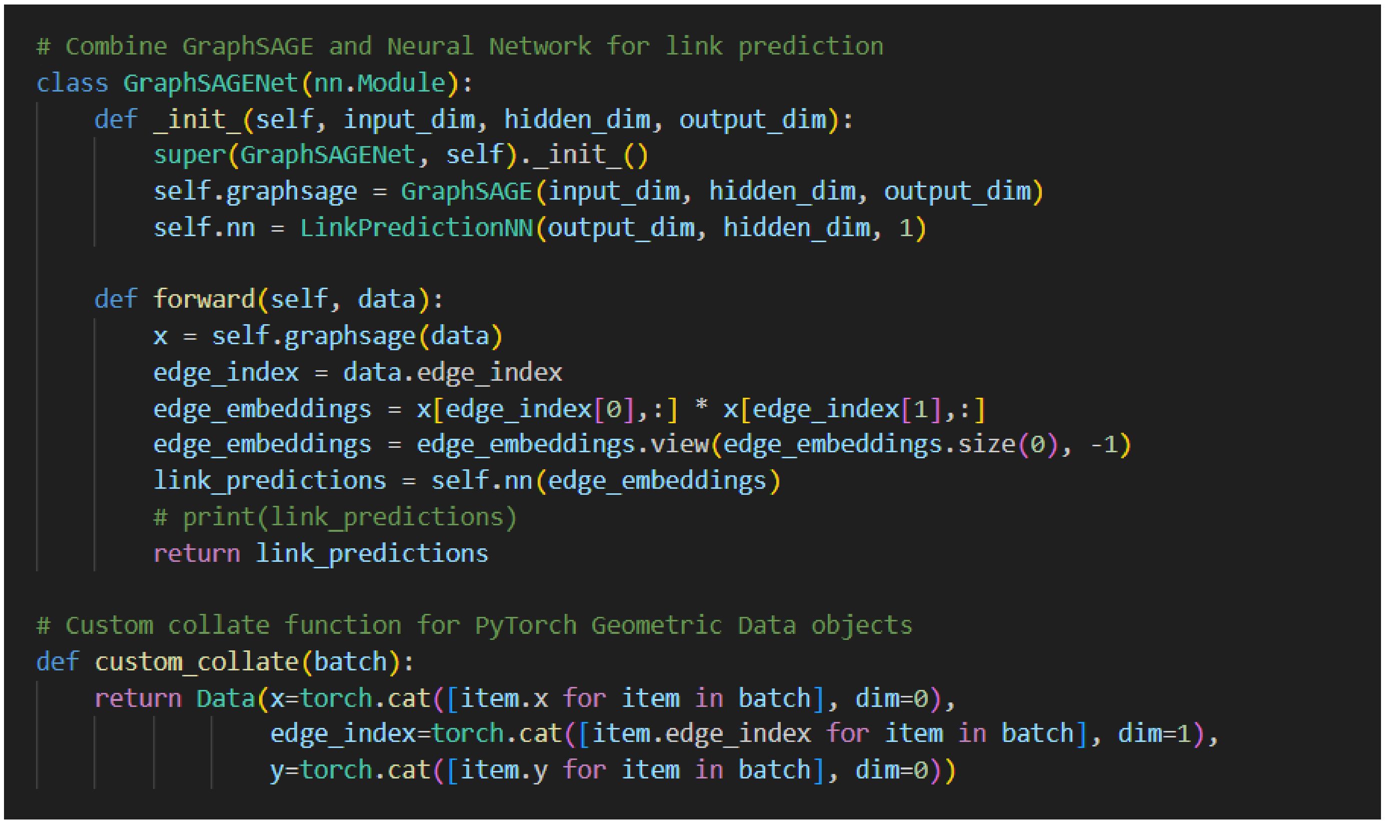
The above code defines two neural network modules: a GraphSAGE model and a Link Prediction Neural Network. The GraphSAGE model is designed for graph-based representation learning, while the Link Prediction Neural Network focuses on predicting links between nodes.

The GraphSAGE model is initialized with input, hidden, and output dimensions. It consists of two SAGEConv layers, which perform graph convolution operations. These layers take node features (x) and edge indices (edge_index) as input, transforming the input features through the graph structure. The model's forward function applies these convolutional layers to generate node embeddings, capturing the graph's structural information.

The Link Prediction Neural Network is initialized with input, hidden, and output dimensions. It consists of two fully connected layers (fc1 and fc2), with rectified linear unit (ReLU) activation applied between them. This network takes node embeddings as input and outputs a prediction score for link existence between nodes. The ReLU activation introduces non-linearity, and the final layer produces an output representing the likelihood of a link.

Both models demonstrate modular and flexible designs, suitable for tasks involving graph-based learning and link prediction, respectively. The GraphSAGE model is intended for extracting informative node embeddings from graph-structured data, while the Link Prediction Neural Network focuses on predicting relationships between nodes based on the generated embeddings.

5.2 GrapheSageNet :



```
# Combine GraphSAGE and Neural Network for link prediction
class GraphSAGENet(nn.Module):
    def _init_(self, input_dim, hidden_dim, output_dim):
        super(GraphSAGENet, self)._init_()
        self.graphsage = GraphSAGE(input_dim, hidden_dim, output_dim)
        self.nn = LinkPredictionNN(output_dim, hidden_dim, 1)

    def forward(self, data):
        x = self.graphsage(data)
        edge_index = data.edge_index
        edge_embeddings = x[edge_index[0],:] * x[edge_index[1],:]
        edge_embeddings = edge_embeddings.view(edge_embeddings.size(0), -1)
        link_predictions = self.nn(edge_embeddings)
        # print(link_predictions)
        return link_predictions

# Custom collate function for PyTorch Geometric Data objects
def custom_collate(batch):
    return Data(x=torch.cat([item.x for item in batch], dim=0),
                edge_index=torch.cat([item.edge_index for item in batch], dim=1),
                y=torch.cat([item.y for item in batch], dim=0))
```

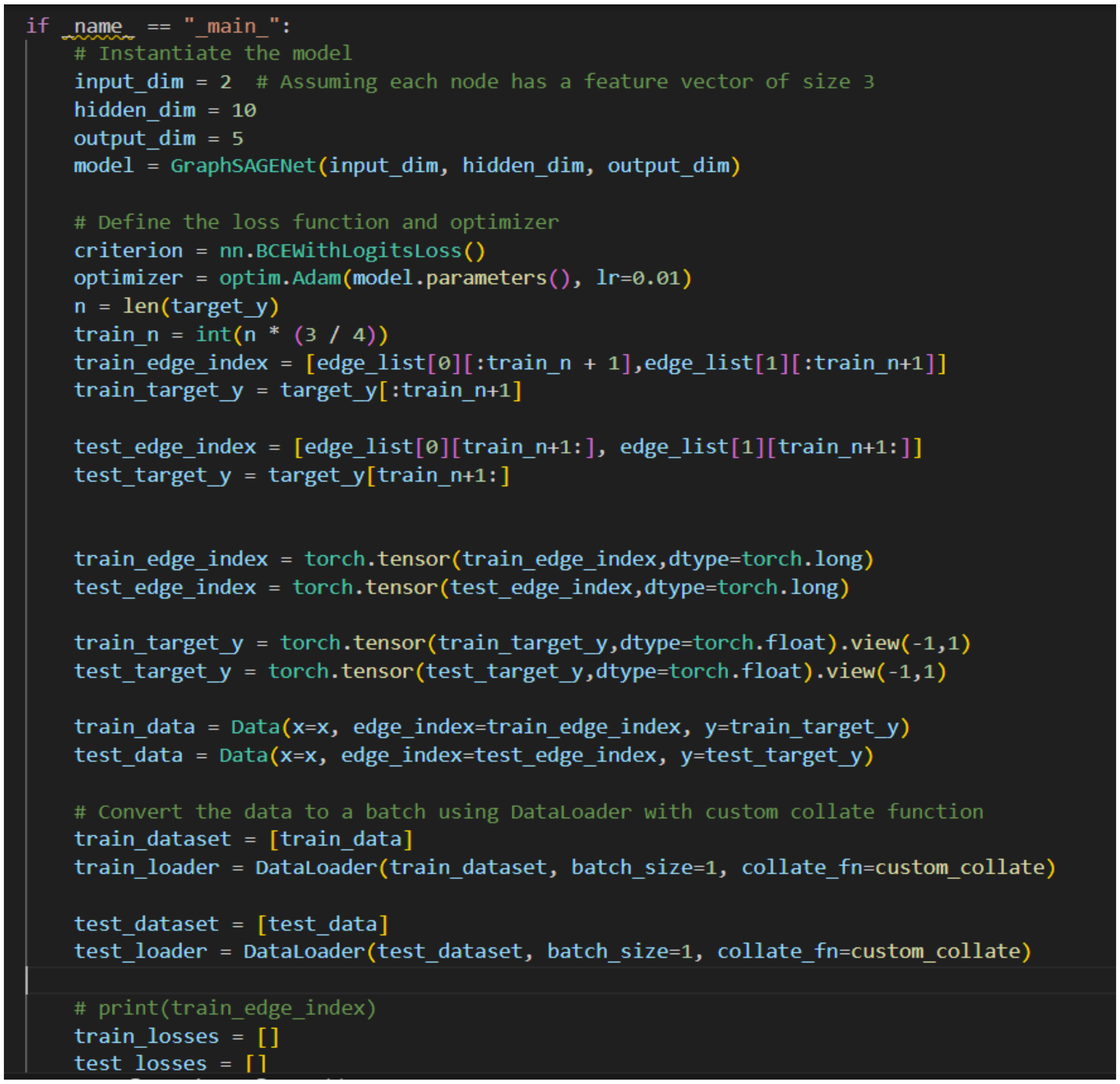
The above code defines a combined model, GraphSAGENet, which integrates the GraphSAGE model for graph-based representation learning with a neural network for link prediction. The model is designed for tasks where understanding relationships in a graph is crucial, such as link prediction in the context of emergency vehicle routing.

The GraphSAGENet class is initialized with input, hidden, and output dimensions. It contains two main components: the graphsage module, which is an instance of the previously defined GraphSAGE model, and the nn module, which is an instance of the LinkPredictionNN neural network. The forward function of GraphSAGENet first passes the input graph data through the graphsage model to obtain node embeddings. It then computes edge embeddings based on these node embeddings and utilizes them as input for the link prediction neural network (nn), producing predictions for link existence.

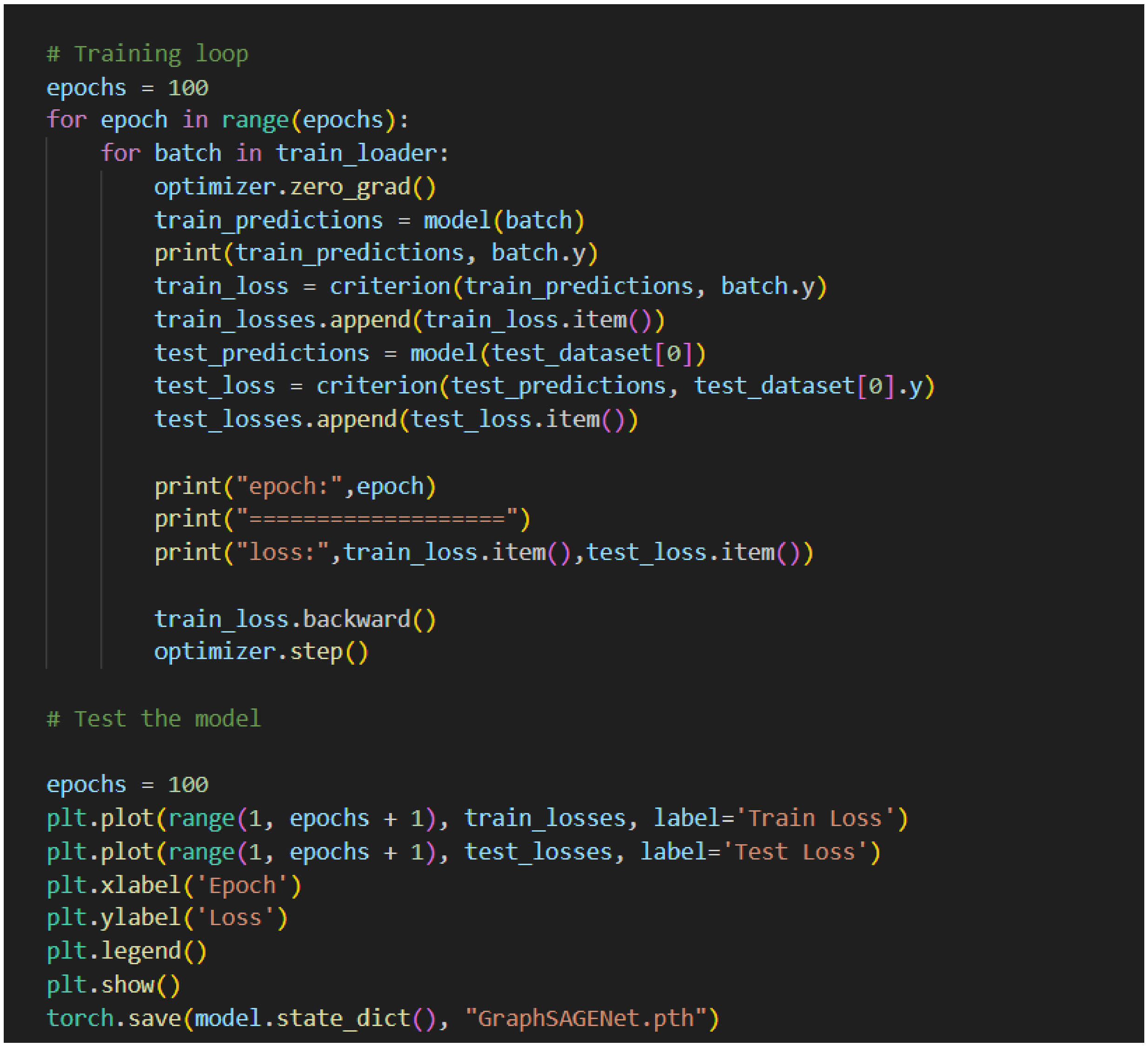
Additionally, the code includes a custom collate function for PyTorch Geometric Data objects. This function is designed for handling batches of graph data during training, concatenating node features, edge indices, and labels across the batch.

Overall, GraphSAGENet seamlessly integrates graph-based learning through GraphSAGE with link prediction capabilities, creating a comprehensive model.

5.3 Training Model :



```python
if _name_ == "_main_":
    # Instantiate the model
    input_dim = 2  # Assuming each node has a feature vector of size 3
    hidden_dim = 10
    output_dim = 5
    model = GraphSAGENet(input_dim, hidden_dim, output_dim)

    # Define the loss function and optimizer
    criterion = nn.BCEWithLogitsLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.01)
    n = len(target_y)
    train_n = int(n * (3 / 4))
    train_edge_index = [edge_list[0][:train_n + 1],edge_list[1][:train_n+1]]
    train_target_y = target_y[:train_n+1]

    test_edge_index = [edge_list[0][train_n+1:], edge_list[1][train_n+1:]]
    test_target_y = target_y[train_n+1:]


    train_edge_index = torch.tensor(train_edge_index,dtype=torch.long)
    test_edge_index = torch.tensor(test_edge_index,dtype=torch.long)

    train_target_y = torch.tensor(train_target_y,dtype=torch.float).view(-1,1)
    test_target_y = torch.tensor(test_target_y,dtype=torch.float).view(-1,1)

    train_data = Data(x=x, edge_index=train_edge_index, y=train_target_y)
    test_data = Data(x=x, edge_index=test_edge_index, y=test_target_y)

    # Convert the data to a batch using DataLoader with custom collate function
    train_dataset = [train_data]
    train_loader = DataLoader(train_dataset, batch_size=1, collate_fn=custom_collate)

    test_dataset = [test_data]
    test_loader = DataLoader(test_dataset, batch_size=1, collate_fn=custom_collate)

    # print(train_edge_index)
    train_losses = []
    test_losses = []
```



```
# Training loop
epochs = 100
for epoch in range(epochs):
    for batch in train_loader:
        optimizer.zero_grad()
        train_predictions = model(batch)
        print(train_predictions, batch.y)
        train_loss = criterion(train_predictions, batch.y)
        train_losses.append(train_loss.item())
        test_predictions = model(test_dataset[0])
        test_loss = criterion(test_predictions, test_dataset[0].y)
        test_losses.append(test_loss.item())

        print("epoch:",epoch)
        print("===================")
        print("loss:",train_loss.item(),test_loss.item())

        train_loss.backward()
        optimizer.step()

# Test the model

epochs = 100
plt.plot(range(1, epochs + 1), train_losses, label='Train Loss')
plt.plot(range(1, epochs + 1), test_losses, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
torch.save(model.state_dict(), "GraphSAGENet.pth")
```

The above code implements a GraphSAGENet model for link prediction in graph-structured data using PyTorch. The key components include model instantiation, data preprocessing, training, evaluation, and visualization.

Model Instantiation:

The GraphSAGENet model is created with specified dimensions for input features (input_dim), hidden layers (hidden_dim), and output dimensions (output_dim).

Loss Function and Optimizer:
The loss function is defined as Binary Cross Entropy with Logits (nn.BCEWithLogitsLoss()), and the optimizer is set as Adam with a learning rate of 0.01.
Data Splitting:

The dataset is split into training and test sets, with 75% of the data used for training. Edge indices and target labels are extracted accordingly.
Data Formatting and DataLoader:

The data is formatted into PyTorch Geometric Data objects, and custom collate functions are defined for handling batches during training and testing. DataLoader instances are created using these collate functions.

Training Loop:
The model is trained through a loop over a specified number of epochs (100 in this case). Within each epoch, the training dataset is iterated through batches using the DataLoader. The optimizer is zeroed, predictions are obtained, and the BCEWithLogitsLoss is calculated. Both training and test losses are logged.

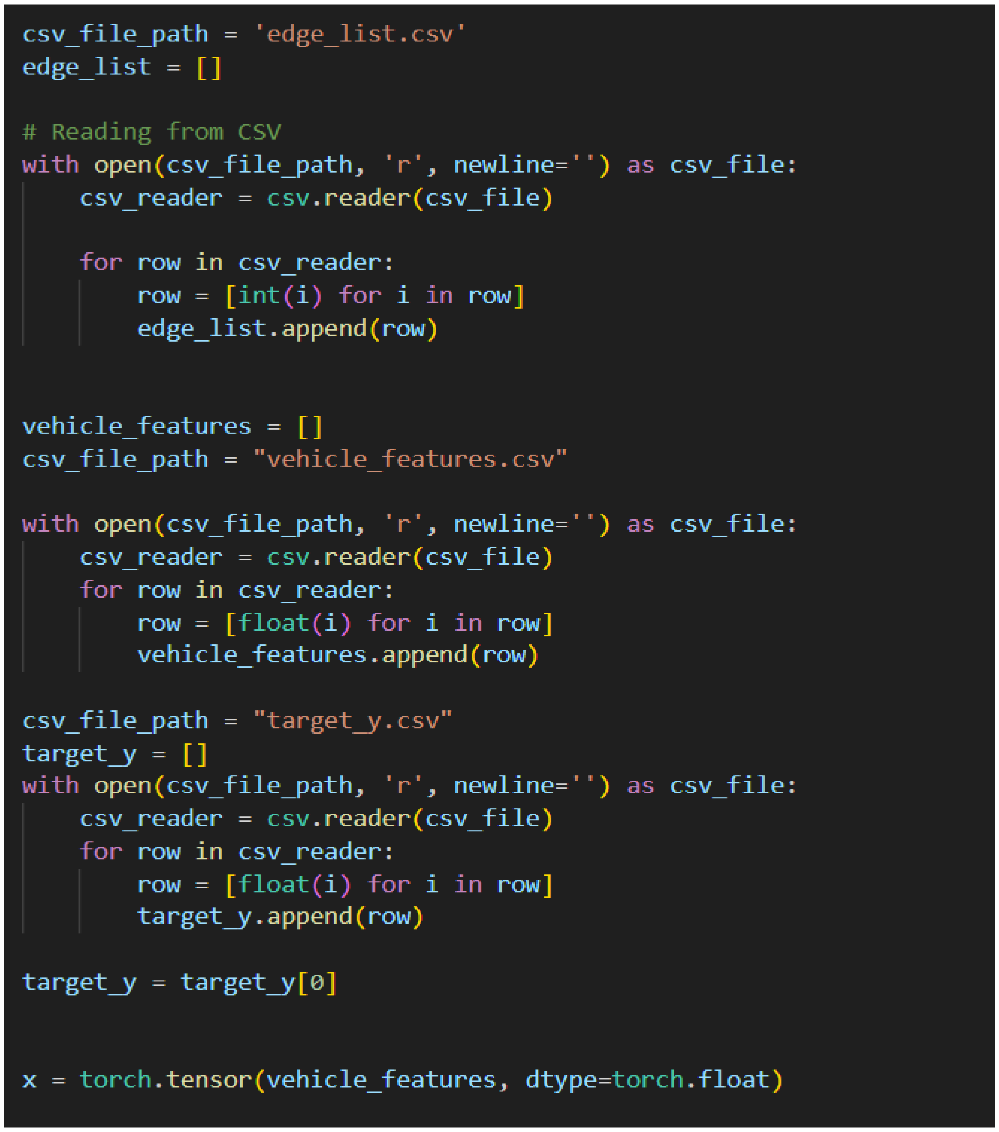
Visualization:
Training and test losses are plotted against the number of epochs using matplotlib. This visualization provides insights into the model's learning progress and potential overfitting.

Model Saving:
The trained model's state dictionary is saved to a file named "GraphSAGENet.pth" using torch.save().

Testing Phase:

The model is evaluated on the test dataset, and the results are printed, including the loss for each epoch. The code concludes with a visualization of the training and test loss curves and the saving of the trained model's parameters. This implementation provides a comprehensive pipeline for training and evaluating a GraphSAGENet model for link prediction in a graph-structured dataset.

5.4 Read / Write Dataset :



```python
csv_file_path = 'edge_list.csv'
edge_list = []

# Reading from CSV
with open(csv_file_path, 'r', newline='') as csv_file:
    csv_reader = csv.reader(csv_file)

    for row in csv_reader:
        row = [int(i) for i in row]
        edge_list.append(row)


vehicle_features = []
csv_file_path = "vehicle_features.csv"

with open(csv_file_path, 'r', newline='') as csv_file:
    csv_reader = csv.reader(csv_file)
    for row in csv_reader:
        row = [float(i) for i in row]
        vehicle_features.append(row)

csv_file_path = "target_y.csv"
target_y = []
with open(csv_file_path, 'r', newline='') as csv_file:
    csv_reader = csv.reader(csv_file)
    for row in csv_reader:
        row = [float(i) for i in row]
        target_y.append(row)

target_y = target_y[0]


x = torch.tensor(vehicle_features, dtype=torch.float)
```

The above code snippet performs data loading and preprocessing for a graph-based machine learning task, presumably involving vehicle interactions. It reads data from three CSV files: 'edge_list.csv', 'vehicle_features.csv', and 'target_y.csv'.

Edge List Loading:
The 'edge_list.csv' file contains information about edges in a graph, likely representing connections or interactions between vehicles. The code reads this file, converting the read values to integers, and constructs the edge_list list.
Vehicle Features Loading:

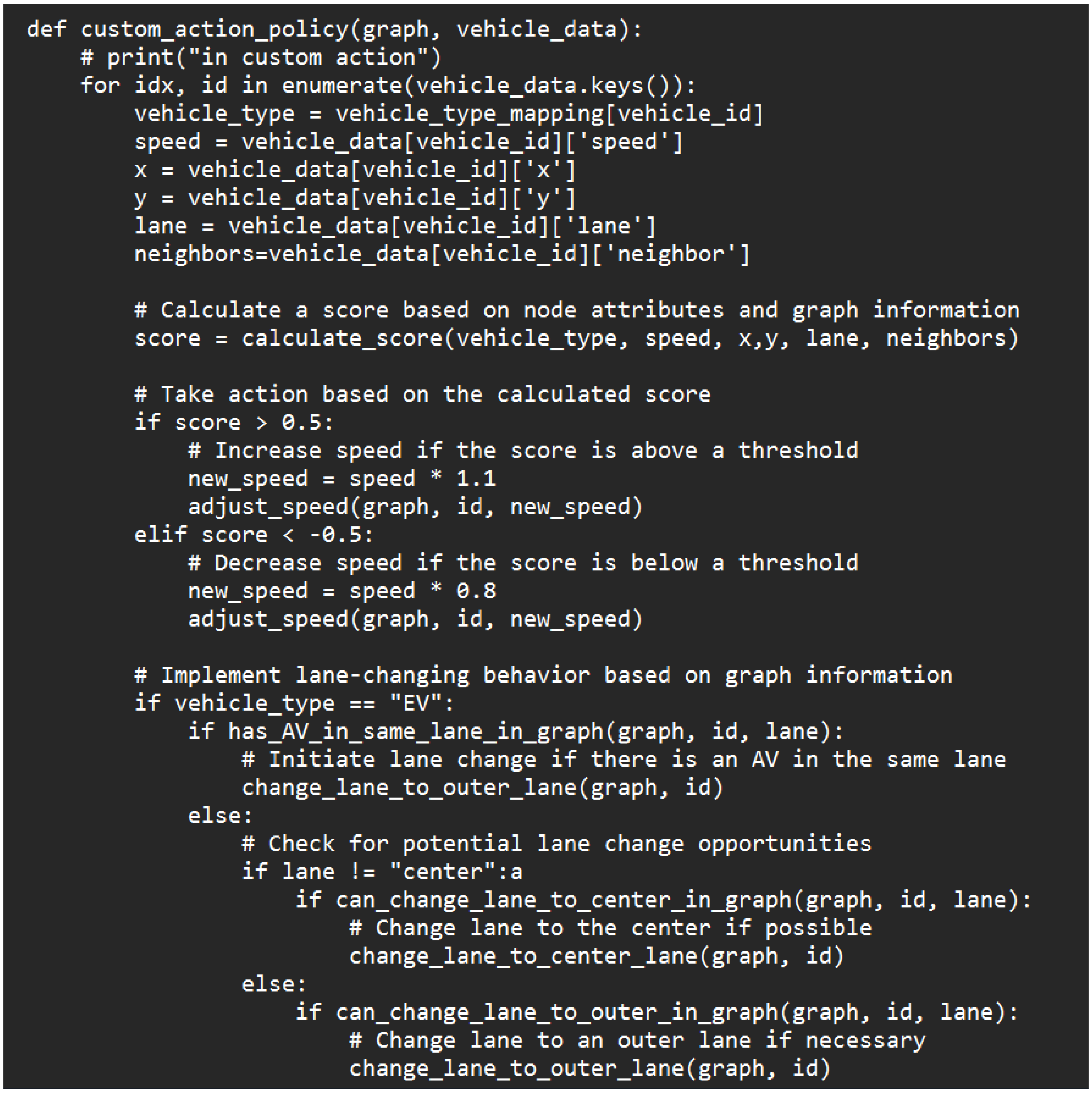
The 'vehicle_features.csv' file contains features associated with each vehicle in the graph. The code reads this file, converting the read values to floats, and constructs the vehicle_features list.

Target Labels Loading:
The 'target_y.csv' file contains target labels, possibly indicating the presence or absence of links between corresponding vehicles. The code reads this file, converting the read values to floats, and constructs the target_y list. The list is then converted to a one-dimensional torch tensor.

Conversion to Torch Tensor:
The vehicle_features list is converted to a torch tensor named x with a data type of float, likely to be used as input features for a machine learning model.
Overall, this code establishes the necessary data structures to represent the graph, with edge_list denoting the graph's edges, vehicle_features containing features for each node (vehicle), and target_y representing the target labels. The torch tensor x holds the feature vectors for each

vehicle, facilitating compatibility with PyTorch-based machine learning models. This data loading and preprocessing script is foundational for subsequent steps in building and training graph-based models for our tasks such as link prediction or vehicle interaction analysis.

5.5 Custom Action Policy :



```
def custom_action_policy(graph, vehicle_data):
    # print("in custom action")
    for idx, id in enumerate(vehicle_data.keys()):
        vehicle_type = vehicle_type_mapping[vehicle_id]
        speed = vehicle_data[vehicle_id]['speed']
        x = vehicle_data[vehicle_id]['x']
        y = vehicle_data[vehicle_id]['y']
        lane = vehicle_data[vehicle_id]['lane']
        neighbors=vehicle_data[vehicle_id]['neighbor']

        # Calculate a score based on node attributes and graph information
        score = calculate_score(vehicle_type, speed, x,y, lane, neighbors)

        # Take action based on the calculated score
        if score > 0.5:
            # Increase speed if the score is above a threshold
            new_speed = speed * 1.1
            adjust_speed(graph, id, new_speed)
        elif score < -0.5:
            # Decrease speed if the score is below a threshold
            new_speed = speed * 0.8
            adjust_speed(graph, id, new_speed)

        # Implement lane-changing behavior based on graph information
        if vehicle_type == "EV":
            if has_AV_in_same_lane_in_graph(graph, id, lane):
                # Initiate lane change if there is an AV in the same lane
                change_lane_to_outer_lane(graph, id)
            else:
                # Check for potential lane change opportunities
                if lane != "center":a
                    if can_change_lane_to_center_in_graph(graph, id, lane):
                        # Change lane to the center if possible
                        change_lane_to_center_lane(graph, id)
                else:
                    if can_change_lane_to_outer_in_graph(graph, id, lane):
                        # Change lane to an outer lane if necessary
                        change_lane_to_outer_lane(graph, id)
```

The custom action policy function is designed to govern the behavior of vehicles in a simulation scenario, using a graph-based representation and individual vehicle data. It iterates through the vehicles in the simulation, extracting relevant information such as vehicle type, speed, position, lane, and neighbors from the provided vehicle data. The function then calculates a score for each vehicle based on its attributes and the graph's information, determining whether the vehicle should adjust its speed or change lanes.

The scoring mechanism considers factors such as vehicle type, speed, position, and the presence of neighbors. If the calculated score exceeds a threshold, the vehicle is prompted to increase its speed, while a score below another threshold triggers a speed reduction. Additionally, for emergency vehicles (EVs), the function evaluates the graph structure to initiate lane changes. EVs may change lanes to the center if no autonomous vehicles (AVs) are present or to an outer lane when needed, enhancing their ability to navigate through traffic efficiently.

The custom action policy dynamically adjusts vehicle behavior based on a combination of individual vehicle attributes and the global traffic network structure, promoting effective and adaptive vehicle navigation in mixed traffic environments.

The presence of edges in the graph is necessary for it to function. The action policy function is triggered to determine a score for every vehicle when there are sufficient edges in the graph. Depending on the type of vehicle and the characteristics of its surrounding nodes and edges, this score is the basis for decision-making. For emergency vehicles (EVs), a high score is assigned, prompting an adjustment of their speed towards their maximum allowable limit. This guarantees that electric vehicles (EVs) can quickly and effectively navigate the traffic network to get to their destinations.

5.5 Evaluation :

Three metrics are taken into consideration in order to fully assess the effectiveness of our suggested approach: overall traffic waiting time, collision rate, and navigation waiting time.
The effectiveness of the navigation strategy is evaluated by measuring the amount of time an emergency vehicle waits while attempting to reach its destination, a measure known as
navigation waiting time. The quantity of collisions determines safety, which assesses how well the action policy function performs in averting collisions and guaranteeing the security of every vehicle. The effect of the navigation strategy on the larger traffic system is measured by overall traffic waiting time, which is determined by the overall flow and congestion of the traffic network. We compare two baseline approaches: Baseline 1 uses the
IDM3 algorithm only and does not use any custom action policy, while Baseline 2 uses a custom action policy function that uses simulation obtained by the traCI library instead of GraphSAGE to predict links.

5.6 Workflow :

1. Import necessary libraries
2. Read edge list from CSV file
3. Read vehicle features from CSV file
4. Read target values from CSV file
5. Convert vehicle features to a PyTorch tensor
6. Create a PyTorch Geometric Data object
7. Define the GraphSAGE model
8. Define a simple neural network for link prediction
9. Combine GraphSAGE and Neural Network for link prediction
10. Define a custom collate function for PyTorch Geometric Data objects
11. Instantiate the model
12. Define the loss function and optimizer
13. Split data into train and test sets
14. Convert train and test data to batches using DataLoader with custom collate function
15. Define training loop and epoch count
16. Iterate through each batch in the training set
17. Zero the gradients of the optimizer
18. Make predictions for the current batch
19. Calculate the loss for the current batch
20. Append the loss to the training loss list
21. Make predictions for the test set
22. Calculate the loss for the test set
23. Append the loss to the test loss list
24. Print the current epoch, training loss, and test loss
25. Backpropagate the loss
26. Update the model parameters
27. Test the model
28. Plot the training and test losses and save model parameters

CHAPTER 6

DATASET

6.1 Simulation Environment :

Three simulation scenarios—highway merge shown in Fig.1, intersection shown in Fig. 2, and citywide scenarios shown in Fig. 3—are used to assess the efficacy of our suggested method for emergency vehicle (EV) navigation in mixed traffic environments. All the scenarios are based on real-world traffic situations with human vehicles, autonomous vehicles, and emergency vehicles. For all scenarios, the complex traffic network is modeled using SUMO, a well-known and reliable simulation platform.

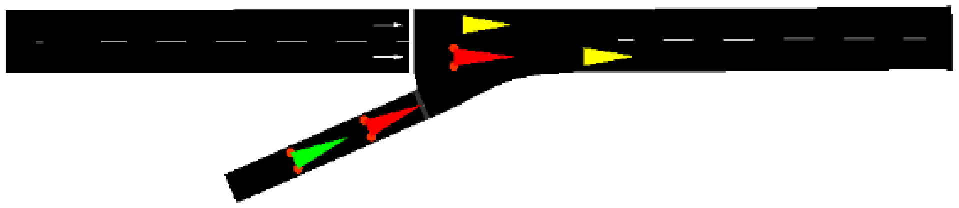


Fig. 1. Merge Network

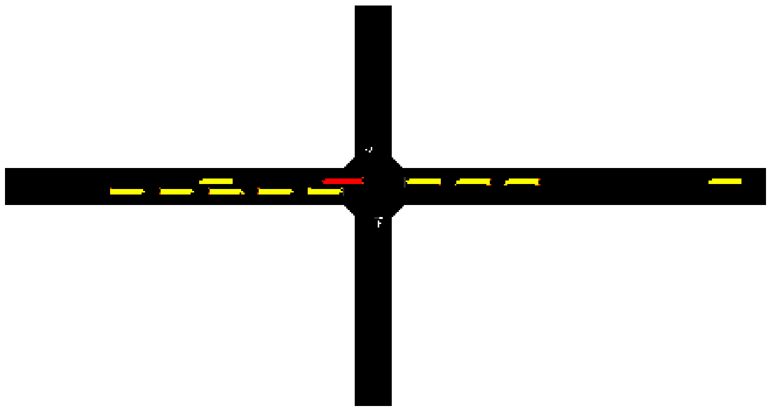


Fig. 2. Intersection Network

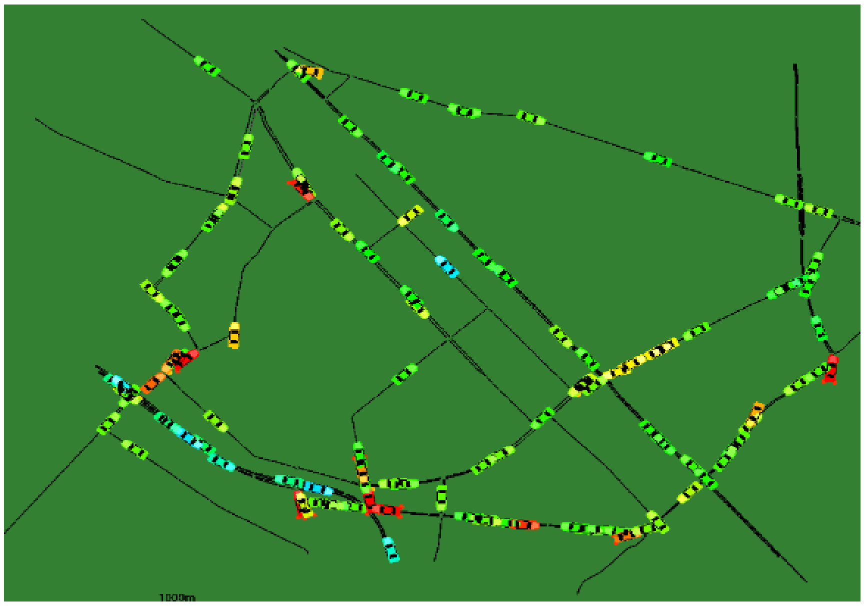


Fig. 3. City Network

The IDM3 algorithm, which accurately replicates human travel behavior, is used to model human-driven vehicles. Our proposed action policy function is used by autonomous vehicles to receive actions and also applies the IDM3 algorithm. This combination ensures safe and effective navigation by enabling autonomous vehicles to make decisions based on the action policy function. Emergency vehicles are able to move through the traffic network faster because they are given priority over other vehicles. By setting priorities, emergency vehicles can get to their destinations faster and possibly save more lives

CHAPTER 7

RESULTS AND DISCUSSION

This chapter is a compilation of all the major results along with associated discussion.

7.1 Overview :

In contrast to earlier research employing reinforcement learning to choose actions, our approach doesn't need the high GPU hardware processing power to model the connect. Large-scale calculations are not required because Our unique function for action policy produces succinct actions. According to the current state of the traffic graph.For every one of the, we looked at three different cases. Three simulated environments to evaluate the effectiveness of our working methods. Without the use of algorithms, the

The first case serves as a baseline and simulates a situation. The second example uses our custom action to construct the graph. Policy function was used, making use of SUMO-sourced simulation attributes. In the last case, our custom action policy function is used based on the generated graph after link states are inferred using GraphSAGE.

The experimental parameters for the three scenarios merge, intersection, and city network scenarios are presented in TABLE I.

Parameters	**Scenarios**		
	Merge	***Intersection***	***City***
Number of HVs	23	26	47
Number of AVs	14	16	26
Number of EVs	3	2	7
Speed limit of EVs and AVs	140 km/h	140 km/h	120 km/h
Speed limit of HVs	120 km/h	120km/h	100 km/h
Number of Lanes	2	2	<3

TABLE I. PARAMETERS SETTING

7.2 Merge Network :

The relatively sparse distribution of cars in the merge scenario suggests that individual vehicle driving behavior has little effect on the overall flow of traffic. This finding seemingly clarifies the performance patterns shown in TABLE II. Even though the improvement in performance over the baseline is not very large, it is still noteworthy. Furthermore, the GraphSAGE-based approach's performance is close to that of the condition where the graph was created with simulation attributes.

Simulation Scenario	**Parameters**		
	Emergency WaitingTime	**Depart Delay**	**Average Waiting Time**
Baseline	7.2s	1.2s	12.8s
Graph made by Simulation attributes	5.5s	0s	10.2s
Graph made by GraphSAGE	6.1s	0.3s	11.1s

TABLE II. PERFORMANCE ON MERGE

7.3 Intersection network :

The optimization effect attained in the intersection scenario was more noticeable than that seen in the merge scenario, as clearly seen in TABLE III. This discrepancy is probably due to the increased mutual influence between cars in the intersection situation, where their movements and interactions have a bigger impact on the direction of traffic flow as a whole. Interestingly, the simulation-based approach's performance and the GraphSAGE-based approach's closely match which highlights how well GraphSAGE captures the complex dynamics of the intersection scenario.

Simulation Scenario	**Parameters**		
	Emergency Waiting Time	**Depart Delay**	**Average Waiting Time**
Baseline	9.2s	2.4s	15.8s
Graph made by Simulation attributes	3.5s	0.2s	3.2s
Graph made by GraphSAGE	3.8s	0.5s	3.1s

TABLE III. PERFORMANCE ON INTERSECTION

7.4 City Network :

As can be seen from TABLE IV, the custom action policy function significantly outperforms the baseline when combined with the graph for the city network. Furthermore, the simulation-based approach and the GraphSAGE-based approach for graph construction closely match, indicating GraphSAGE's strong prediction abilities. Due to computational constraints, large-scale traffic simulations are just not possible, so this is a significant improvement over RL-based action systems.

Simulation Scenario	Parameters		
	Emergency Waiting Time	Depart Delay	Average Waiting Time
Baseline	12.2s	2.6s	20.8s
Graph made by Simulation attributes	4.8s	0.4s	8.1s
Graph made by GraphSAGE	4.9s	0.8s	9.6s

TABLE IV. PERFORMANCE ON CITY NETWORK

The outcomes show that the use of a graph in conjunction with a custom action policy function has greatly increased the simulation's efficiency. Moreover, GraphSAGE is an effective way for predicting links for graphs in such forms of traffic networks and can be used efficiently in these kinds of situations. The train-test curve for GraphSAGE, as depicted in Fig 4. illustrates the loss and highlights its low value

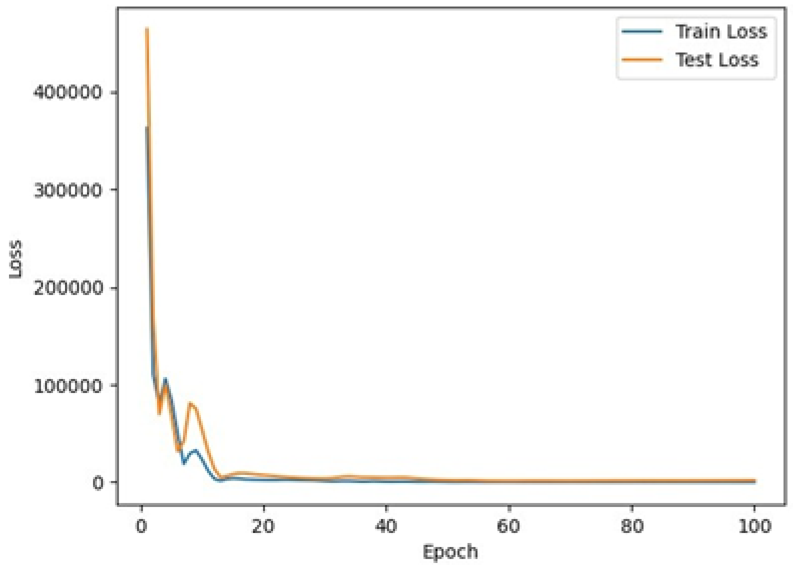


Fig. 4. Train Test curve for GraphSAGE

CHAPTER 8

CONCLUSION AND FUTURE WORK

This paper uses GraphSAGE and a custom action policy function to present a novel approach to EV navigation in mixedtraffic environments. By capturing the dynamic and unpredictable nature of traffic, our method outperforms baseline approaches in enabling EVs to navigate. EV navigation efficiency has been significantly improved by integrating GraphSAGE for graph representation and link prediction with a custom action policy function for EV navigation optimisation. This reveals the potential of graph-based traffic modelling and prediction methods in practical traffic control situations, especially for vital applications such as emergency vehicle navigation. The suggested method has a number of benefits over conventional RL-based techniques. GraphSAGE enables more precise link prediction and EV navigation by efficiently capturing the intricate spatial relationships present in the traffic network. Second, these relationships are taken into account by the custom action policy function, which optimises traffic flow for EVs and results in more effective navigation. Third, the SUMO simulation framework's integration of GraphSAGE and the custom action policy function enables large-scale traffic simulations, getting around the computational constraints of RLbased techniques. All things considered, our research shows the potential of graph-based traffic modelling and prediction methods in practical traffic management applications, especially for crucial situations such as electric vehicle (EV) navigation. Our suggested method provides a viable means of enhancing the effectiveness of EV navigation and guaranteeing prompt assistance delivery during emergencies.Future research will examine how various graph neural network architectures affect our method's effectiveness, create a more complex action policy function that accounts for other variables like pedestrians and road conditions, and test our method on a bigger and more varied dataset of traffic scenarios. These efforts will improve the efficacy of our suggested methodology and expand its suitability to an expanded array of actual traffic situations.

BIBLIOGRAPHY

[1] Liu, Qi, et al. "Graph Convolution-Based Deep Reinforcement Learning for Multi-Agent Decision-Making in Mixed Traffic Environments." arXiv preprint arXiv:2201.12776 (2022).

[2] G. -P. Antonio and C. Maria-Dolores, "Multi-Agent Deep Reinforcement Learning to Manage Connected Autonomous Vehicles at Tomorrow's Intersections," in IEEE Transactions on Vehicular Technology, vol. 71, no. 7, pp. 7033-7043, July 2022, doi: 10.1109/TVT.2022.3169907.

[3] Dong, Jiqian, Sikai Chen, Paul Young Joun Ha, Yujie Li, and Samuel Labi. "A drl-based multiagent cooperative control framework for cav networks: a graphic convolution q network." arXiv preprint arXiv:2010.05437 (2020).

[4] T. Shi, J. Wang, Y. Wu, L. Miranda-Moreno, and L. Sun, "Efficient Connected and Automated Driving System with Multi-agent Graph Reinforcement Learning," IEEE Transactions on Intelligent Transportation Systems, doi: 10.1109/TITS.2021.3067931, 2021.

[5] Klimke, Marvin, Benjamin Völz, and Michael Buchholz. "Automatic Intersection Management in Mixed Traffic Using Reinforcement Learning and Graph Neural Networks." arXiv preprint arXiv:2301.12717 (2023).

[6] Klimke, Marvin, Benjamin Völz, and Michael Buchholz. "Cooperative Behavior Planning for Automated Driving using Graph Neural Networks." In 2022 IEEE Intelligent Vehicles Symposium (IV), pp. 167-174. IEEE, 2022.

[7] Klimke, Marvin, et al. "An Enhanced Graph Representation for Machine Learning Based Automatic Intersection Management." 2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2022.

[8] J. Dong et al., "Spatio-weighted information fusion and DRL-based control for connected autonomous vehicles," 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), Rhodes, Greece, 2020, pp. 1-6, doi: 10.1109/ITSC45102.2020.9294550.

[9] Gao, Xin, et al. "Multi-agent decision-making modes in uncertain interactive traffic scenarios via graph convolution-based deep reinforcement learning." Sensors 22.12 (2022): 4586.

[10] Liu, Qi, Xueyuan Li, Yujie Tang, Xin Gao, Fan Yang, and Zirui Li. "Graph Reinforcement Learning-Based Decision-Making Technology for Connected and Autonomous Vehicles: Framework, Review, and Future Trends." *Sensors* 23, no. 19 (2023): 8229.

[11] Lu, Yuhuan, Wei Wang, Xiping Hu, Pengpeng Xu, Shengwei Zhou, and Ming Cai. "Vehicle trajectory prediction in connected environments via heterogeneous context-aware graph convolutional networks." *IEEE Transactions on Intelligent Transportation Systems* (2022).

[12] Liu, Tao, Aimin Jiang, Jia Zhou, Min Li, and Hon Keung Kwan. "GraphSAGE-Based Dynamic Spatial–Temporal Graph Convolutional Network for Traffic Prediction." *IEEE Transactions on Intelligent Transportation Systems* (2023).

[13] Shang, Yigeng, Zhigang Hao, Chao Yao, and Guoliang Li. "Improving Graph Neural Network Models in Link Prediction Task via A Policy-Based Training Method." *Applied Sciences* 13, no. 1 (2022): 297.

[14] Rathore, Heena, and Henry Griffith. "GNN-RL: Dynamic Reward Mechanism for Connected Vehicle Security using Graph Neural Networks and Reinforcement Learning." In *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 201-203. IEEE, 2023.

[15] Cai, Peide, Hengli Wang, Yuxiang Sun, and Ming Liu. "DiGNet: Learning scalable self-driving policies for generic traffic scenarios with graph neural networks." In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8979-8984. IEEE, 2021.