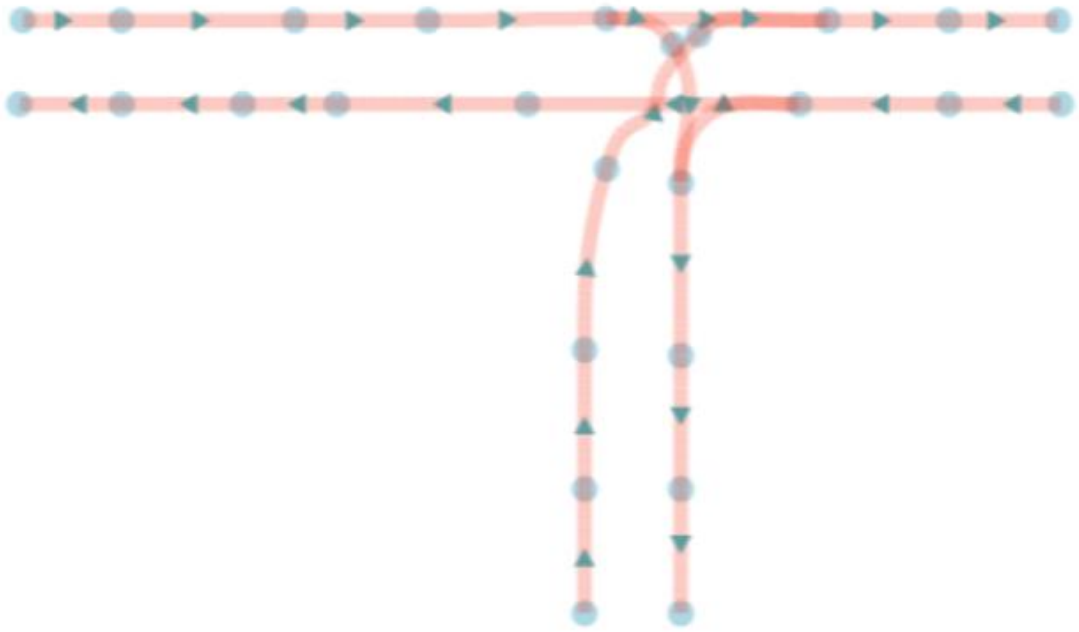

Graph Neural Networks for Mobile Robots

Subtitle: A Systemic GNN Design Solution for Traffic in AGV Systems

Master's thesis in Engineering Mathematics and Computational Science

BOSHEN ZHANG

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Graph Neural Networks for Mobile Robots

A Systemic GNN Design Solution for Traffic in AGV Systems

BOSHEN ZHANG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2025

Graph Neural Networks for Mobile Robots
A Systemic GNN Design Solution for Traffic in AGV Systems
BOSHEN ZHANG

Supervisor: Rasmus Åkerlund, Kollmorgen Automation AB
Examiner: Serik Sagitov, Department of Mathematical Sciences

Master's Thesis 2025
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An example layout of a T-intersection in AGV systems.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Graph Neural Networks for Mobile Robots
A Systemic GNN Design Solution for Traffic in AGV Systems
BOSHEN ZHANG
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Automated Guided Vehicle (AGV) systems improve modern warehouse efficiency but require extensive effort in designing the virtual road networks (also known as layouts). A key evaluation metric in this process is waiting time. Traditional simulation-based methods for waiting time estimation are time-consuming, highlighting the need for faster predictive models. In this thesis, we explore Graph Neural Networks (GNNs) for predicting the waiting time on each road segment. We propose a hierarchical GNN framework that integrates a classifier to detect congested segments and a regressor to estimate the waiting time, effectively addressing the wide-spreading data imbalance issues. Experimental results demonstrate that the framework captures meaningful patterns, providing a potential alternative to traditional simulations in layout design.

Keywords: automated guided vehicle, layout design, deep learning, graph neural networks, hierarchical framework, waiting time prediction.

Acknowledgements

The six months at Kollmorgen passed quickly, and I learned a great deal during this time. First, I would like to express my gratitude to Kollmorgen for providing me with the opportunity to work on my master's thesis. I am especially grateful to my supervisor, Rasmus Åkerlund, whose expertise, insights, and domain knowledge have been invaluable to me. His support greatly contributed to the progress of my thesis, and his guidance significantly improved my thesis writing. I would also like to thank my examiner, Serik Sagitov, for his guidance, valuable suggestions, and the freedom he gave me to explore my research. Finally, I want to express my appreciation to my colleagues at Kollmorgen, who made my time there truly enjoyable.

Boshen Zhang, Gothenburg, March 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AGV	Automated Guided Vehicle
GNN	Graph Neural Network
GCN	Graph Convolutional Network
GAT	Graph Attention Network
GNN4AGV	Graph Neural Network for Automated Guided Vehicle system
CNN	Convolutional Neural Network
MLP	Multi-layer Perceptron

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

In recent years, the demand for increased efficiency in modern warehouses has driven the adoption of more logistics automation systems. The introduction of automated systems often significantly enhances transportation efficiency, enabling faster deliveries. Among these, Automated Guided Vehicle (AGV) systems have emerged as a key solution[1][2].

As a subset of mobile robot systems, AGV systems have been proven through years of application in modern warehouses to be a mature technology that can save labor and reduce product damage[3]. In AGV systems, a fleet of mobile robots automatically transports goods within a predefined virtual road network, referred to as the AGV network or layout. This distinguishes AGV systems from most other mobile robot systems: the movement of vehicles is guided by the virtual network rather than being self-determined. The primary objective is to complete global transportation tasks, with the road network directly tied to the allocation of these tasks.

The layout is custom-designed, as different warehouses require distinct AGV networks to ensure smooth operation of the vehicles. Therefore, layout design is critically important. However, designing a layout is not a straightforward or linear process; it typically requires continuous iterative adjustments and simulation verification. In the design process, every adjustment must go through the simulation to mimic how the AGV system would operate in a real warehouse. The schematic diagram of the layout design process is shown in Figure 1.1. Designers need to ensure that the AGV system can correctly fulfill the required orders in the simulation before considering applying it in a real-world environment.

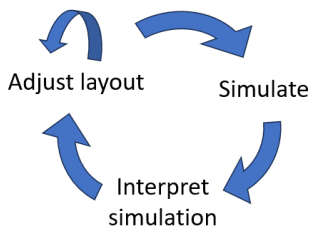

Figure 1.1: Layout design process.

Beyond ensuring correct pick-up and delivery of goods, another critical metric is whether the AGV system can complete orders on time, as timeliness is essential in logistics. The fundamental components of an AGV network are segments, where each segment can accommodate only one vehicle at a time. This constraint can lead to traffic congestion, where a vehicle must wait for another to leave the desired segment. Such congestion events result in delays in AGV task execution, adversely affecting the system's timelines.

In general, we aim to prevent excessive vehicle waiting times. The waiting time for each segment should be in a controlled range. An optimal layout design should ensure that AGVs keep moving continuously for pick-up or delivery tasks rather than frequently waiting at some points in the network. In other words, in the simulation, the cumulative waiting time for each segment of the AGV network should be minimized as much as possible to enhance efficiency.

If we can accurately and quickly predict the accumulative waiting time per segment for each AGV network layout design in the simulation, we can make targeted adjustments to mitigate AGV system congestion. This results in fewer design iterations, i.e., less design time consumed, which translates into faster product deliveries, increased throughput, and increased profits.

The essence of an AGV network is a combination of road segments and segment intersections. Thus, a graph composed of edges and nodes becomes a very natural method to represent an AGV network. Using graph representation learning methods, such as graph neural networks, to predict waiting times is a research direction worth exploring. Moreover, compared to traditional computer simulations, the use of machine learning algorithms can save a lot of time.

1.2 Goals and objective

Graph Neural Networks (GNNs) are a relatively new deep learning model specialized in processing graph data[4][5]. GNN synthesizes information about both the geometric structure of the network and the features of the nodes, enabling edge-level, node-level, and graph-level prediction. Currently, much of the research in this area focuses on domain-specific applications rather than the algorithms themselves[8]. For example, in the field of biochemistry, many structures such as molecules can be represented by graphs, making GNNs a hotspot for molecular design and quantum chemistry[6][7]. However, research on GNNs themselves and their ability to generalize at the task level is less mature than in more traditional deep learning applications such as computer vision.

Kollmorgen has explored graph level and node level GNNs in two previous projects [9]. The former uses GNNs to predict a single metric characterizing the layout design quality of AGV network; the latter predicts the congestion level of each segment in simulation. These studies have demonstrated the feasibility of using GNNs to process AGV network data. And this thesis is also based on node level GNNs and uses

the segment wise waiting time as the target.

However, systemic exploration and knowledge of Graph Neural Network for Automated Guided Vehicle system (GNN4AGV) is still missing. As mentioned earlier, current GNN research mostly focuses on specific tasks, so when focusing on a new domain, it often requires manual adjustments based on domain knowledge. This is exactly the challenge we are facing, as resources for GNN4AGV have not yet appeared in published research, and there are not yet even uniform standards for AGV systems themselves. Therefore, this thesis attempts to establish a systemic framework and explore GNN4AGV in depth based on it.

This thesis aims to address several key questions about GNN4AGVs through theoretical derivation and experimental validation. The goal is to explore the potential of using GNNs to accelerate the AGV network design process and to lay the foundation for their future application in this field.

1.3 Scope

This thesis is divided into two main parts. The first part focuses on the specific knowledge of AGV systems, discussing the core needs of AGV system designers, the unique characteristics of AGV system data, and how this domain knowledge impacts the construction of our GNN model.

Building on this foundation, the second part develops a targeted GNN framework and explores the design space of GNNs using AGV data. Together, these two parts establish a systematic understanding of GNN4AGVs.

2
Traffic in AGV systems

In this section, we introduce key concepts related to traffic in AGV systems, including fundamental elements such as layout, congestion, and simulation. Additionally, we model the layout as a graph and discuss traffic rules. Finally, we provide a detailed description of the dataset used in this thesis.

2.1 AGV Systems

2.1.1 Virtual Road Networks and layouts

Virtual Road Networks are the cornerstone of AGV systems and are also referred to as layouts. These layouts are pre-designed, and all AGVs in the system run within them. Each layout consists of multiple segments connected by points and has directional properties. The connectivity and directionality of these segments determine how AGVs operate, ensuring they follow predefined segments and directions. Each segment also possesses various attributes, such as length and speed constraints. Additionally, during the layout design process, traffic rules may be introduced in specific areas based on operational requirements to achieve certain objectives.

In summary, the information contained within a layout can be classified into three categories: layout topology, segment attributes, and traffic rules.

2.1.2 Automated Guided Vehicles

AGVs are the fundamental units responsible for logistics delivery within an AGV system. Multiple AGVs operate within the layout, navigating between designated locations such as loading/drop-off bays and home areas to fulfill order deliveries. In the following part of the thesis, we will use the term "vehicle" as a simplified reference to AGVs.

Each vehicle is not fully autonomous; its route planning and order allocation are managed by the host system, also known as the fleet manager. Therefore, it is automated rather than autonomous.

2.1.3 Traffic congestion and waiting time

As mentioned in Section 1.1, to ensure the efficient operation of the AGV system, we aim to keep the waiting time at each segment at a minimal level. Delving deeper

into the motivation behind this, certain locations within the AGV system—such as areas where multiple segments converge—can create bottlenecks in vehicle traffic within overlapping regions. These bottlenecks may lead to vehicle congestion. Since each segment can allow only one vehicle running at a time, the likelihood of vehicles coming to a halt while waiting for others to pass increases significantly, ultimately resulting in traffic congestion.

Notably, localized traffic congestion tends to propagate, affecting adjacent areas. This is intuitive like traffic jams on urban roads, which rarely remain confined to the initial problematic segment but instead spread further. Identifying these initial problematic segments is valuable for AGV system designers (also known as application engineers), as modifying their design can not only alleviate congestion in those segments but also potentially ease traffic in the surrounding segments.

In this context, accumulated waiting time becomes highly valuable since it directly reflects congestion levels. The segment with the longest waiting time not only indicates the most severe congestion but is also likely to be one of the primary sources of congestion propagation.

On the other hand, when designing the layout, application engineers may manually introduce traffic rules in certain segments to prevent potential issues. These rules sometimes enforce vehicle waiting under particular conditions, contributing to accumulated waiting time. However, this designed waiting time is not necessarily a drawback. Therefore, in layout design, merely minimizing waiting time is not always desirable, as real-world scenarios are often more complex. Nonetheless, waiting time remains a valuable metric for gaining critical insights.

2.1.4 Simulation in AGV system

Once the layout is designed, it must undergo simulation for validation and assessment before real-world testing. In the simulation, computer software replicates the movement of vehicles within the physical AGV network, with route planning and management still handled by the fleet manager. Although simulation results may differ from real-world testing, its significantly faster execution and the lack of a need for prior local deployment make it a highly time-efficient approach. Furthermore, simulation helps mitigate risks. Conducting real-world tests on an unverified layout may lead to vehicle collisions and additional losses. Therefore, simulation is an essential step in the design process. Only when the simulation results meet the expected objectives will real-world testing be considered for further evaluation.

In this thesis, the waiting time we discussed is from simulation rather than real-world testing.

2.2 Graph representations of layout

A fundamental prerequisite for applying GNNs to AGV systems is that the AGV layout is essentially a graph. GNNs are specifically designed for graph-structured data. In this section, we will explain the correspondence between the AGV layout and graph structure.

2.2.1 Graph theory

A graph is a powerful data structure that can be used to model different systems in many research fields. For example, in the study of social networks and knowledge networks, graphs are a natural choice for modeling these systems. For the definition of a basic graph, a basic graph g is composed of a set of nodes N and a set of edges E. For each edge e, it is defined by a pair of nodes n_1 and n_2, where $n_1, n_2 \in N$, that is, (n_1, n_2) represents the start point and the end point of the edge, respectively.

The most intuitive way to represent a basic graph is through the adjacency matrix $A \in \mathbb{R}^{|N|\times|N|}$, where the row and column indices represent the node indices. For a basic graph, its adjacency matrix is naturally a symmetric matrix, also referred to as an undirected graph[4].

2.2.2 Dual graph representation

The structure of the AGV layout closely aligns with that of a graph, where the fundamental elements are segments, each identified by its start point and end point, i.e. the two nodes it connects. In AGV systems, we focus particularly on segment-level waiting time. If a basic graph structure is used to represent the layout, intuitively, segments correspond to edges in the graph, while points correspond to nodes. However, experience suggests that this direct basic graph representation may introduce challenges for subsequent analysis[9][10]. Most GNN methods primarily focus on nodes or entire graphs as prediction targets rather than edges.

Therefore, we use a dual-graph representation for the layout, where segments in the layout correspond to nodes in the dual graph. An adjacency matrix is then used to define the structure of the dual graph. This approach better accommodates GNN analysis and enhances the modeling of segment-level characteristics.

This dual transformation of the directed graph is illustrated in Figure 2.1.

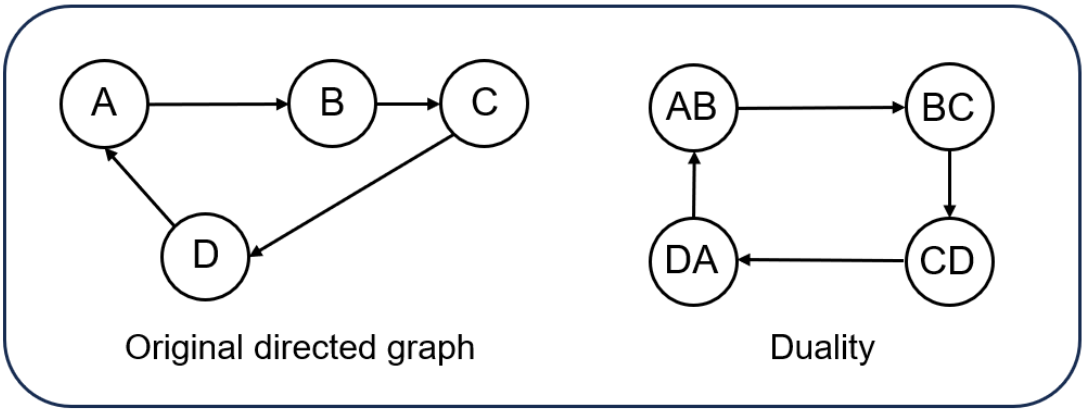

Figure 2.1: The original graph and its dual graph: the direction within the graph is preserved during the dual transformation.

2.2.3 Directed graph

In graph structures, edges can have directional properties, meaning that the two nodes forming an edge can be distinguished as a starting node and an end node. As a result, the adjacency matrix is not necessarily symmetric. In other words, the existence of an edge from node i to node j does not guarantee the existence of an edge from node j to node i. Similarly, in an AGV system, the layout also has directional constraints, requiring vehicles to follow predefined route directions. Each segment has a starting point and an endpoint, and these segments are connected to form the overall movement paths within the AGV system.

In the dual graph structure, this local directionality is preserved and transformed into the directionality between segments. To illustrate this, consider a simple example: In the original graph, a vehicle can only travel sequentially from point A to point B and then to point C. After the dual graph transformation, this directional property changes, meaning the vehicle can only move from node AB to node BC. This transformation is also represented in Figure 2.1.

2.2.4 Representation of traffic rules

We have already discussed traffic rules in AGV systems in Section 2.1.3. Typically, traffic rules are introduced to prevent vehicle collisions. Although the AGV system enforces a "one vehicle per segment" restriction, when segments are closely spaced, vehicles traveling on them may still face collision risks.

The most common traffic rules include point blocking and segment blocking, both of which perform at the segment level. The former means that when a vehicle occupies a certain point (i.e., the starting or ending point of a segment), another segment affected by this rule will be locked, preventing other vehicles from entering. The latter indicates that when a vehicle is already traveling on a segment, another segment will be blocked, making it inaccessible. Illustration of point blocking rule can

be found in Figures 2.2.

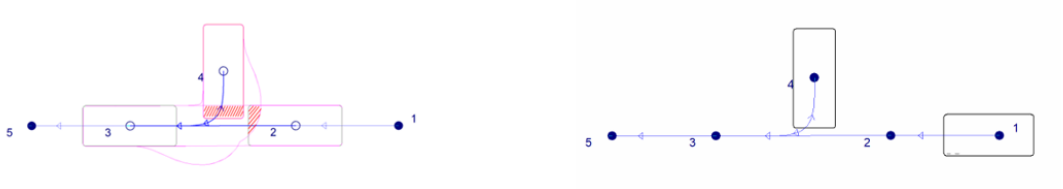

Figure 2.2: Traffic rule illustration: On the left, a potential collision may occur between two vehicles. On the right, after implementing a traffic rule that blocks segment (2,3) when a vehicle is positioned at point 4, the collision is effectively prevented because the other vehicle is prohibited from entering segment (2,3).

Traffic rules are crucial for AGV systems. For a layout, it is essential to represent both its topological graph information and traffic rules. We have already addressed the former by utilizing a directed dual graph. In Chapter 4, we will provide a detailed explanation of how traffic rules can be incorporated into the graph representation.

2.3 Datasets

2.3.1 Layouts and targets

All layouts used in this thesis are derived from 48 variations of a T-intersection shape road segment, with one example shown in Figure 2.3. We conducted a 30-minutes simulation for each of these 48 layouts and recorded the cumulative waiting time of all segments (in seconds) as the core research target. Additionally, we represented these layouts using the directed dual graph introduced in this chapter and employed the corresponding adjacency matrix for subsequent analysis and prediction.

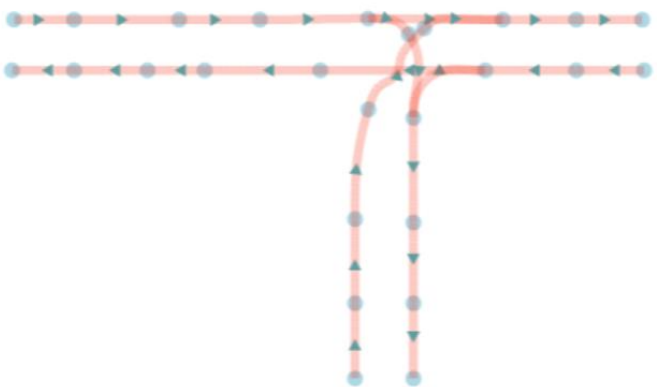

Figure 2.3: An example layout of a T-intersection.

2.3.2 Segment features of AGV systems

Each segment in the layout has three key features, all strongly related to AGV traffic, especially congestion. These features include travel time through the segment, predicted active usage, and predicted blocking probability.

The first feature represents segment attributes—length and driving speed—where travel time is the ratio of the two. The other two features are more advanced: the former approximates how frequently a segment is used during the simulation, while the latter estimates the probability of a vehicle being blocked within the segment. Both are derived from Kollmorgen's internal tools. Although these values are not ground truth, their usability has been validated, and they provide valuable insights.

In the dual graph representation of the layout, the above three segment features correspond to node features in the dual graph.

3

Graph neural networks

This chapter focuses on the theory of Graph Neural Networks. We introduce the core principles and unique characteristics of GNNs. In addition, we also present several specific GNN models that are utilized in this thesis.

3.1 Neural message passing

3.1.1 Motivations behind the theory of Graph Neural Networks

GNNs are a more advanced type of Graph Encoder Model. Compared to traditional Node Embedding models based on neighborhood reconstruction or matrix factorization, GNNs make a significant contribution by being able to integrate both graph structural information and node-level features. Thus, the logic of GNNs becomes straightforward: for each node in the graph, during the GNN operation, feature information is iteratively aggregated from directly or indirectly connected neighboring nodes, in a manner similar to the layer-wise operations of traditional neural networks, but guided by the graph's structure/connectivity.

On the other hand, GNNs utilize convolution operations in this process, enabling them to capture more complex patterns within the neighborhoods of nodes. The convolution operation in GNN shares some homology with Convolutional Neural Network (CNN), as it extends the concept of convolution from Euclidean spaces (e.g., images) to non-Euclidean spaces (e.g., graphs)[11][12].GNN approximate convolution operations through local aggregation, capturing the local structural and feature information of the node's neighbourhood. And this local aggregation technique ensures the permutation equivariance of GNNs—a property analogous to the local translational invariance in CNNs and crucial for graph representation learning, which will be explained in detail in the next section[12].

In addition, the local aggregation operation, similar to the layer-wise operation in neural networks, enables GNNs to be trained using backpropagation techniques. Specifically, during backpropagation, the model computes the gradient of a predefined loss function with respect to the learnable parameters, allowing for iterative weight updates. This process typically involves propagating the gradient information layer by layer, adjusting the weights based on some optimization algorithms

(e.g., stochastic gradient descent) to minimize the loss. This significantly improves computational efficiency and model performance. As a result, this local aggregation mechanism facilitates the development of practical message-passing layers framework, which dynamically update node embeddings by capturing dependencies between nodes and their connected neighborhoods—an inherent characteristic of GNNs that is essential for processing graph-structured data[13].

3.1.2 Permutation equivariance

Unlike pixel-based images, the rows and columns of an adjacency matrix representing node connectivity in a graph lack an inherent order. This order is determined by our labeling scheme and is more dependent on relative relationships than absolute ordering. Consequently, for the same AGV network, different adjacency matrices may be generated. The challenge, therefore, lies in ensuring that different adjacency matrix inputs representing the same graph yield consistent prediction results. To address this, GNNs must satisfy the condition of permutation equivariance, meaning the output of the node embedding matrix should reflect corresponding changes when the row/column order is altered in the input of the adjacency matrix. Conventional neural networks, such as Multi-Layer Perceptron (MLP), do not satisfy permutation equivariance because the output of an MLP depends on the order of the input features.

Permutation equivariance can be mathematically described in matrix form as described in Equation 3.1[4]. Let $\mathbf{P}$ be a permutation matrix and $\mathbf{A}$ an adjacency matrix. If a function f satisfies permutation equivariance, then:

$$f(\mathbf{PAP}^\top) = \mathbf{P}f(\mathbf{A}) \tag{3.1}$$

3.1.3 Message passing in GNNs

The message-passing framework is a fundamental component of GNNs. This framework consists of two main parts: the aggregation function (aggregator) and the update function. Both functions are differentiable, ensuring that GNNs align with the overall neural network architecture, where each message-passing iteration corresponds to a layer in a neural network. During each message-passing process, information undergoes one aggregation and one update, which refers to layers. In layer t of the GNN, the embedding $h_v^{(t)}$ of each node in the graph gets updated based on its own embedding in the previous layer, $h_v^{(t-1)}$, and on an aggregation over its neighbors' previous layer embeddings, $h_u^{(t-1)}$, $u \in \mathcal{N}(v) \setminus \{v\}$, where $N(v)$ is is the set of neighbors of node v, including the node itself.

More formally,

$$m_{\mathcal{N}(v)}^{(t)} = \text{AGGREGATE}\left(\{h_u^{(t)} \mid u \in \mathcal{N}(v)\}\right) \tag{3.2}$$

$$h_v^{(t)} = \text{UPDATE}(h_v^{(t-1)}, m_{\mathcal{N}(v)}^{(k)}) \tag{3.3}$$

Where:

- $\mathbf{h}_v^{(t)}$ is the embedding of node v at layer t,
- x_v is the original node embeddings,
- $\mathcal{N}(v)$ denotes the set of neighboring nodes of node v,
- AGGREGATE$(\cdot)$ is the aggregator applied to the embeddings of the neighboring nodes.

And for the first layer we set

$$h_v^{(0)} = x_v \tag{3.4}$$

Aggregation refers to the process of collecting information from a node's neighboring nodes in each iteration. The properties of the aggregation function ensure that GNNs exhibit the permutation equivariance described in Section 3.1.2, allowing neural networks to effectively process graph-structured data. Specifically, this property stems from the design of the aggregation function itself—regardless of the input order of neighboring nodes, aggregation functions such as mean or sum will always yield the same result. This invariance to input order is crucial for ensuring that GNNs can effectively handle graph-structured data without being affected by arbitrary changes in node ordering.

The update function combines the output of the aggregator with the current node's previous state to produce a new node embedding, which is why it is also called the combine function. This framework iteratively runs, resulting in a neural network that operates on graphs.

It is exactly this layer-based message-passing framework that enables GNNs not only to inherit the usual characteristics of neural networks—stacking more layers to capture more complex patterns—but also to align intuitively with the context of graphs representation. In each iteration, new node embeddings aggregate information from further neighbors, with this aggregation guided by the structural information of the graph, i.e. the connectivity of nodes[4].

The operations of message passing are illustrated in the Figure 3.1. After k iterations, each node will contain information from its k-hop neighborhood. In traffic network graphs, the congestion of a particular road segment can be affected by more distant neighbors. The message passing framework aligns well with this scenario.

The message passing layer serves as the core of graph neural networks. A complete graph neural network model may include layers other than message passing, but the message passing layer is essential as it ensures permutation equivariance.

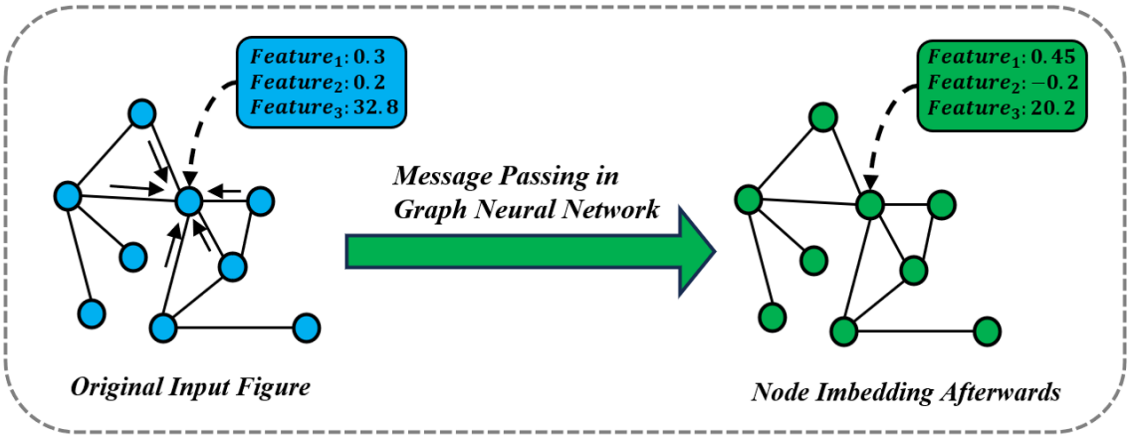

Figure 3.1: Message passing in Graph Neural Network.

3.1.4 Input of GNNs

Compared to neural networks used in image processing, such as CNNs, the input to GNNs is no longer a multi-dimensional matrix based on pixels but rather a adjacency matrix representing a graph consisting of nodes and edges. Similar to traditional node embedding methods (e.g., random walks), the graph's adjacency matrix serves as the primary input to represent structural connections between nodes. Here, we use the adjacency matrices corresponding to the 48 layouts stated in Section 2.3.1. Additionally, GNNs incorporate node feature information as part of their input. These input features serve as the initial node embeddings, denoted as $h_v^{(t=0)}$, as shown in Equation 3.4, and participate in message-passing computations, as described in Equation 3.2. Due to the excellent scalability of GNNs, some of the latest GNN algorithms also incorporate edge-level feature information as input. However, this study considers only node features and the adjacency matrix.

3.1.5 Raw Output of GNNs

The term "raw output" in GNNs refers to the output of the message-passing phase. The message-passing layer is a core component of graph neural networks as it ensures permutation equivariance. The raw output of any GNN is node embeddings, which are generated after multiple iterations of message passing. The dimensionality of these output node embeddings is determined by the specific prediction task and the structure of the model following the message-passing layers.

3.2 Taxonomy of graph neural networks

As discussed in Section 3.1.5, the different dimensions of GNN output are primarily based on node embedding levels. On the other hand, GNN model output can be more diverse and is not necessarily limited to the node level, it also includes the graph level and even edge level in some cases. As a subset of neural networks, the flexible output of GNNs enables them to handle a wide range of prediction tasks.

The prediction target can be at the node, graph, or edge level.

3.2.1 Node level task

For node-level classification or regression tasks, we can directly set the output dimension of the node embeddings to the desired output dimension. In this case, the final node embedding, obtained after the last message passing step, will directly serve as the output for softmax computation or as the final prediction.

A complete GNN model may include additional layers beyond message passing, such as a MLP layer following the message-passing layer, with the MLP's output serving as the final model output. The models used in this thesis include both of these cases. This approach is also applicable to graph-level and edge-level GNNs.

3.2.2 Graph level task

In the graph level case, in addition to the message passing process based on the aggregator and update functions, a readout function should be incorporated to aggregate the node embeddings and generate a graph-level embedding. This is essentially a pooling-like dimensionality reduction operation that aggregates the features of each node (or edge) to generate a single feature representation for the entire graph.

For graph-level GNNs, the expressions for the aggregator, update function, and readout function are given in Equation 3.5 and Equation 3.6.

$$h_v^{(t)} = \text{UPDATE}\left(h_v^{(t-1)}, \text{AGGREGATE}\left(\{m_u^{(t)} \mid u \in \mathcal{N}(v)\}\right)\right) \tag{3.5}$$

$$h_G = \text{READOUT}\left(\{h_v^{(t)} \mid v \in G\}\right) \tag{3.6}$$

3.2.3 Edge level task

Edge level tasks exhibit greater complexity. Following each iteration of node embedding updates, an additional Edge Feature Update is applied on top of the existing updates to facilitate the enhancement of edge embeddings. The expression of edge-level GNNs are given in Equation 3.7 and 3.8.

$$m_{uv}^{(t)} = \text{AGGREGATE}\left(h_u^{(t-1)}, h_v^{(t-1)}, e_{uv}^{(t-1)}\right) \tag{3.7}$$

$$e_{uv}^{(t)} = \text{UPDATE}\left(e_{uv}^{(t-1)}, m_{uv}^{(t)}\right) \tag{3.8}$$

3.3 Choices of GNN models

In this thesis, we utilize four of the most widely used GNN models: Basic GNN, GCN, GraphSAGE, and GAT. The following provides an introduction to each of these models. In fact, the primary differences between various GNNs lie in the aggregator part in Equation 3.2. In GNN algorithm research, the aggregator receives more attention from researchers, as the core of GNN studies revolves around how to aggregate information from neighbors[4].

The default choice of update function is typically implemented as a linear interpolation between the node itself and the aggregation result from its neighborhood, followed by a non-linear transformation using an activation function. Such composition of aggregator and update functions are as shown in Equation 3.9.

$$\mathbf{h}_v^{(k+1)} = \sigma\left(\mathbf{W}_1 \cdot \mathbf{h}_v^{(k)} + \mathbf{W}_2 \cdot \text{AGGREGATE}(\{\mathbf{h}_u^{(k)} : u \in \mathcal{N}(v)\})\right) \tag{3.9}$$

where:

- $\mathbf{W}_1$ and $\mathbf{W}_2$ are learnable weight parameters, and
- $\sigma(\cdot)$ is a non-linear activation function (e.g., ReLU).

With the expanding application of GNNs, the importance of the update function has also become more important when dealing with more complex data structures, such as molecular structures. Studies have also shown that more complex MLPs are more powerful compared to the default one-layer perceptron used in the default update function[22]. However, this thesis focuses solely on the most popular GNNs and adopts the default update function.

In the following introduction to specific GNN models, we will continue to use the basic update function defined in Equation 3.9, with W set as trainable parameters, while focusing on the explanation of the aggregator.

3.3.1 Basic GNN

First, we introduce the most basic GNN. In this GNN structure, aggregation is simply defined as the sum of node embeddings from neighboring nodes. Therefore, we can use this sum to replace AGGREGATE in Equation 3.9, as shown in Equation 3.10:

$$h_v^{(t)} = \sigma\left(W_{\text{self}} h_v^{(t-1)} + \sum_{u \in \mathcal{N}(v)} W_{\text{neigh}} h_u^{(t-1)}\right) \tag{3.10}$$

In practical applications, to simplify computation, we can add self-loops to the input graph, making each node explicitly connected to itself. This corresponds to an addition operation on the adjacency matrix:

$$A = A + I \tag{3.11}$$

By doing so, we treat self-nodes and neighboring nodes equally, effectively sharing parameters between the W_{self} and W_{neigh} matrices. The corresponding expression of the basic GNN is given in Equation 3.12:

$$h_v^{(t)} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{u\}} W h_u^{(t-1)}\right) \tag{3.12}$$

It is worth noting that whether or not self-looping is included not only affects the presentation of the GNN model but also its parameterization. In Equation 3.10, the model employs two sets of trainable parameters, corresponding to self-nodes and neighboring nodes separately. However, in Equation 3.12, only a single set of parameters is used, as self-nodes are treated as part of the neighboring nodes. In this study, the term "basic GNN" refers to models without self-loops, whereas other GNN variants may also incorporate self-looping mechanisms.

3.3.2 Graph Convolutional Networks (GCNs)

Compared to Basic GNN, GCN performs a normalization operation when aggregating information from neighbors, with the normalization based on the node's degree $d(u)$[12].

$$h_u^{(t)} = \sigma\left(W^{(t)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{h_v^{t-1}}{\sqrt{d_u d_v}}\right) \tag{3.13}$$

where:

- $d(u) = |\mathcal{N}(u)|$.

This degree-based normalization eliminates the effect of the degree compared to Basic GNN. In Basic GNN, a node with 100 neighbors is likely to lead to larger aggregation results than a node with 1 neighbor. GCN, on the other hand, shifts the focus towards the feature itself.

3.3.3 GraphSAGE

Graph SAGE is a highly popular and powerful framework for constructing aggregators[15]. It separates the aggregator into two components: sampling and aggregating. The former aims to improve training efficiency and reduce runtime, while the latter introduces using concatenation and applying different aggregation variants. In this thesis, we used the mean variant and max pooling variant of GraphSAGE without applying sampling techniques.

The framework of GraphSAGE is as follows:

$$\mathbf{m}_{\mathcal{N}(v)}^{k} \leftarrow \text{AGGREGATE}^{k}\left(\{\mathbf{h}_{u}^{k-1} : u \in \mathcal{N}(v)\}\right) \tag{3.14}$$

$$\mathbf{h}_{v}^{k} \leftarrow \sigma\left(\mathbf{W}^{k} \cdot \text{CONCAT}\left(\mathbf{h}_{v}^{k-1}, \mathbf{m}_{\mathcal{N}(v)}^{k}\right)\right) \tag{3.15}$$

In this study, we explore two types of aggregators: Mean aggregator and Pooling aggregator. The variant of mean aggregator with update function are as follows:

$$\mathbf{h}_{v}^{k} = \sigma\left(\mathbf{W} \cdot \text{MEAN}\left(\mathbf{h}_{v}^{k-1}, \{\mathbf{h}_{u}^{k-1} : u \in \mathcal{N}(v)\}\right)\right) \tag{3.16}$$

It is worth mentioning that the mean variant does not use the concatenation strategy described in Equation 3.15. Instead, it adopts a strategy similar to the normalization in GCNs as shown in Equation 3.13, where the feature average is taken based on the node and its neighbors. This approach can be seen as an approximate local spectral convolution.

The variant of max pooling aggregator is as follows:

$$\mathrm{m}_{\text{pool}}^{k} = \max\left(\mathbf{W}_{\text{pool}}\mathbf{h}_{u}^{k} + \mathbf{b}, \forall u \in \mathcal{N}(v)\right) \tag{3.17}$$

The max pooling aggregator implements the complete GraphSAGE framework, including explicit concatenation in Equation 3.15. The concatenation in GraphSAGE is similar to a skip connection, where the features of the node itself are retained during each message-passing iteration. This often leads to better model performance.

3.3.4 GAT

GAT introduces an attention mechanism by adding a additional attention metric α to the basic GNN aggregator, as shown in Equation 3.19[14]. This parameter indicates the importance of node i's features to node j. The attention-based aggregator allows GNNs to selectively process neighboring nodes, thereby improving model performance on certain types of graphs. However, computing α requires additional computational resources, increasing the overall computational burden and making it a trade-off in practical applications.

The specific GAT expressions are divided into two parts, Equation 3.18 and Equation 3.19, corresponding to the calculation of α and message passing, respectively.

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^{\top}[\mathbf{W}\mathbf{h}_{i} \underset{\text{concatenation}}{\|} \mathbf{W}\mathbf{h}_{j}]\right)\right)}{\sum_{k \in \mathcal{N}(i)\cup\{i\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^{\top}[\mathbf{W}\mathbf{h}_{i} \underset{\text{concatenation}}{\|} \mathbf{W}\mathbf{h}_{k}]\right)\right)} \tag{3.18}$$

$$\mathbf{h}_v^{(t)} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{i\}} \alpha_{uv}^{(t-1)} \cdot \mathbf{W} \cdot \mathbf{h}_u^{(t-1)}\right) \tag{3.19}$$

where:

- $\mathbf{W}$ and $\mathbf{a}$ are trainable parameters.
- α_{ij} represents the attention weight between node i and node j.

It is worth noting that the attention weight is dynamically computed based on node features and learnable parameters (W and a), which are globally shared.

4

Graph neural networks for AGV system

This chapter provides a more detailed modeling of the AGV dual graph, incorporating the specific blocking rules within the AGV system. It lays the groundwork for the application of GNN4AGV.

4.1 Graph modeling of traffic rules

In Chapter 2, we have already constructed a dual graph representation of the AGV network. However, before using the dual graph as input to the GNN, it is necessary to incorporate the traffic rules as part of the modeling process, as information about the traffic rules is not reflected in the basic dual graph.

4.1.1 Multi relational modeling

As introduced in Section 2.2.4, two types of rules are considered in the database used in this study: segment blocking and point blocking. And both essentially reflect inter-segment relationships that lead to blocking, indicating that a segment is more likely to experience vehicle standing still due to traffic conditions in another segment. Therefore, we can categorize them as "blocking" relationships.

This means that in the modeling of the Dual Graph, in addition to the structural connectivity information represented by the adjacency matrix, there are also distinctions between edge types. So the graph of AGV systems is actually multi-relational graph. Shown as follows:

For nodes u and v, the relation between them is defined as follows:

$$\text{Relation}(u, v) \in \{0, 1, 2\},$$

where:

- 0: u and v are not adjacent/connected,
- 1: u and v are adjacent/connected,
- 2: u and v are blocked by traffic rules.

Note: Relation 2 will overlap with either 0 or 1, as it can apply to both physically disconnected nodes and connected nodes.

In the four GNN models introduced in the Chapter 3, none of them take edge type as input. Then for GNN models, the input is naturally represented in the following form:

$$\text{Input} = (\mathbf{A}, \mathbf{X}),$$

where:

- $\mathbf{A}$: the adjacency matrix representing structural connectivity/node relation between nodes, meaning it contains only two types of node relations, 0 and 1, which indicate whether nodes are adjacent or not,
- $\mathbf{X}$: the node feature matrix.

In this case, a multi-relational modeling approach is proposed. This approach creates different parameter paths based on the number of edge types and the connectivity of each type. Each path models a sub-graph corresponding to a specific edge type. These paths jointly participate in the overall model's forward and backward propagation. Finally, the outputs from the individual parameter paths are aggregated using a sum[17].

The following is the mathematical formulation for multi-relational modeling, built upon the foundation of Basic GNN:

$$\mathbf{h}_i^{(t)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_r(i)} c_{i,r}\mathbf{W}_r^{(t-1)}\mathbf{h}_j^{(t-1)} + \mathbf{W}_0^{(t-1)}\mathbf{h}_i^{(t-1)}\right), \tag{4.1}$$

where:

- $\mathcal{R}$: The set of edge types {1, 2}.
- $c_{i,r}$: A scaling factor for edge type r.
- $\mathbf{W}_r^{(t)}$: The learnable weight matrix for edge type r at layer t.
- $\mathbf{W}_0^{(t)}$: The learnable self-loop weight matrix at layer t.

As a result, the relationships between different nodes such as the traffic rules between adjacent segments can be effectively modeled by the GNN.

4.1.2 Heterogeneous modeling

The multi-relational graph mentioned in Section 4.1.1 is actually a type of heterogeneous graph. A heterogeneous graph refers to a graph where both edges and nodes can have multiple types. Here, the multiple types of nodes do not refer to class levels but usually signify fundamental differences in the nature of the nodes. For instance, in a citation network, nodes may be categorized as researchers, articles, or

affiliations, which are entirely distinct concepts[21].

Modeling based on heterogeneous graphs extends the Equation 4.1 by creating parameter paths not only based on edge types but also node types. In real-world AGV systems, such scenarios often exist. For example, in AGV systems, there are concepts beyond segments, such as clusters, stations, and so on. This modeling approach based on heterogeneous graphs provides the possibility of incorporating more AGV-related elements into the GNN input in the future. We leave this for future work to incorporate more realistic elements of AGV systems.

The Figure 4.1 illustrates a potential heterogeneous modeling scheme, where two node types are present: segments and clusters. For these two types, there exist both multi-type connectivity within each node type and multi-type connectivity between the two. As a result, we ultimately derive more diverse parameter paths.

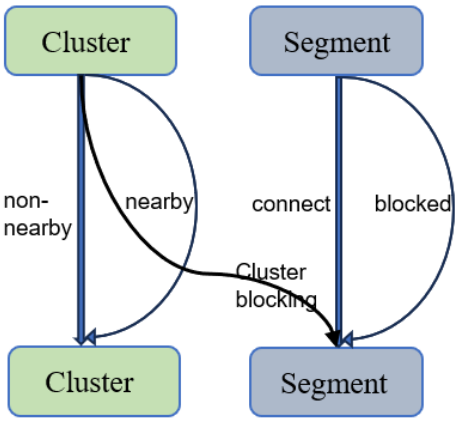

Figure 4.1: An example of potential heterogeneous modeling in GNN4AGV.

4.2 Nature of the GNN4AGV

4.2.1 The application scenarios of waiting time in AGV system design

As briefly mentioned in Section 1.1, the ideal application of GNNs is in predicting traffic congestion, specifically accumulated waiting time, to partially replace the role of simulation in the AGV network design process.

In the AGV system network design process, the application scenarios for using waiting time can be divided into two categories. In the first category, application engineers evaluate the design after each adjustment to assess whether the targeted modifications are effective. Specifically, they examine whether the congestion of those segments has improved, as reflected by a reduction in the waiting times. Un-

der the current circumstances, however, all waiting time data must be obtained through complete computer simulations. This reliance on simulations is one of the primary motivations for utilizing GNNs—to leverage the efficiency of machine learning to reduce the time-consuming nature of simulations and obtain fast and robust cumulative waiting time predictions for each segment.

In the second case, when application engineers complete an early version of the design, they may discover through simulation or prior experience that the design is not optimal and requires adjustments. Since the AGV network's fundamental elements are segments, application engineers need to identify which segments should be prioritized for modification. At this stage, waiting time plays a crucial role, indicating which segments experience the most severe traffic congestion and thus should be addressed prior to others.

When applying GNNs to predict waiting time, the output of the GNN should take into account the two use cases above. In the previous project of Kollmorgen, an attempt was made to transform the regression problem into a classification problem by dividing the waiting time of each segment into four intervals: [0,50), [50,400), [400,800), and [800,) seconds. However, this approach is clearly insufficient. Predicting one of these four categories does not provide application engineers with enough information to prioritize design adjustments, as segments in the same category cannot be distinguished.

Additionally, it fails to reveal the effectiveness of design modifications, since the predicted waiting time for a segment after adjustment might still fall within the same category. Moreover, identifying a data split that holds significant value for designers and aligns with the AGV domain remains a challenging task. Therefore, the prediction target of GNN4AGV should still be waiting time in seconds, making it a regression problem rather than a classification problem.

4.2.2 The output space, difficulty of GNN4AGV and core research question

In summary, the focus of this thesis remains on establishing a nodel-level regression GNN model to accurately predict the exact waiting time for each segment, aligning with the two application scenarios discussed in Section 4.2.1. However, constructing such a regression model is no easy task.

In the AGV system, traffic congestion is not a frequent event. During simulations, most segments do not experience vehicle blockages that lead to the accumulation of waiting time. We refer to these segments as "zero-blocking segments," where the target value (waiting time) is zero. This is because the primary traffic load in an AGV system occurs on the main highways, while the rest of the layout experiences only minimal traffic. As a result, congestion in the system tends to be localized. Most segments do not experience heavy vehicle flow during AGV system operation,

thereby reducing the risk of blocking and subsequent congestion. In the dataset we used, the zero-blocking segments dominate as expected, with 1443 segments belonging to zero-blocking, accounting for the vast majority. The distribution of this situation is shown in Table 4.1 and Figure 4.2.

	Number of segments	Ratio
Zero-blocking segments	1443	83%
Other segments	306	17%

Table 4.1: Segments distribution

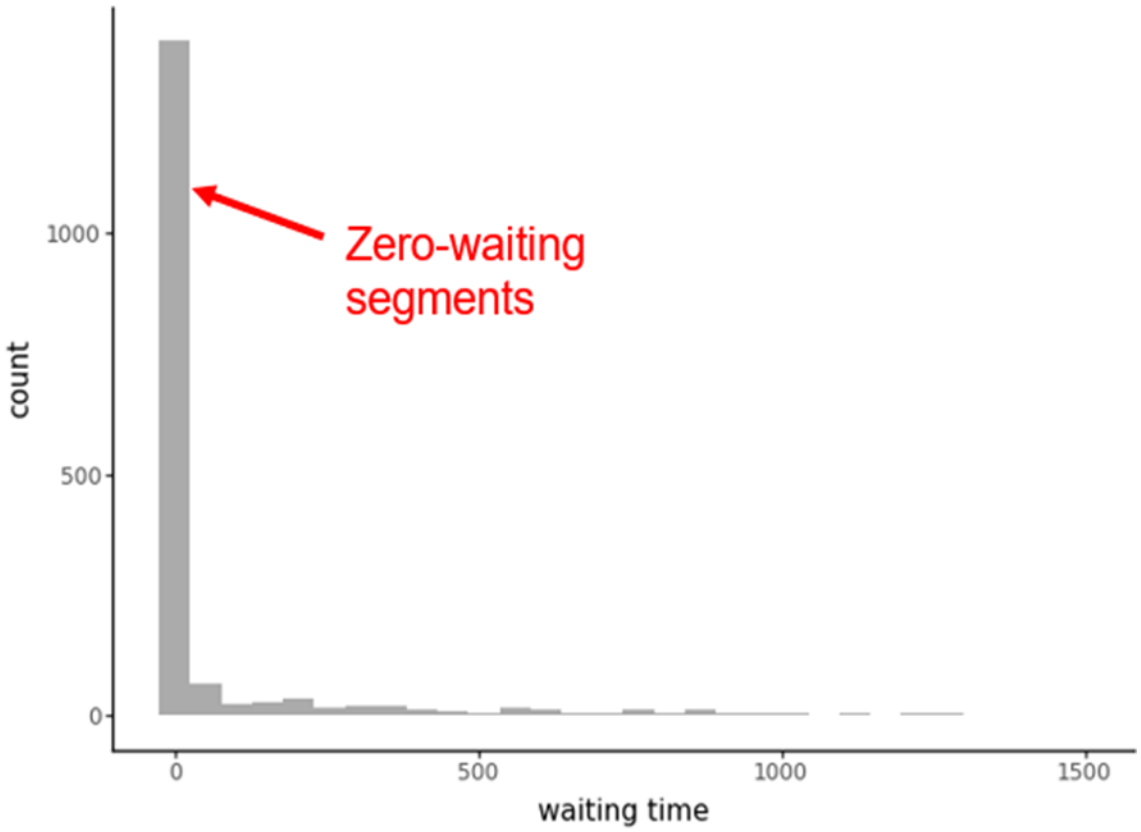

Figure 4.2: Class imbalance: more than 80% of the segments are zero-blocking segments.

Building on this idea, we can further infer that congestion is a traffic phenomenon specific to areas with a certain level of traffic load, whereas regions with low traffic load exhibit different traffic patterns. When the majority of segments in the graph fall into the category of zero-blocking, a significant class imbalance arises. This imbalance leads to the model being heavily biased toward zero-blocking segments, making it difficult to learn patterns from non-zero-blocking segments, which are the primary focus of this study. Resolving the imbalance issue is the key to building an effective regression model.

This leads to the core research question of GNN4AGV: how to develop a node-level regression model for predicting segment waiting time while addressing the issue of imbalance.

Based on our discussion on zero-blocking and imbalance, the reasoning behind waiting time prediction can be divided into two stages: first, distinguishing between high and low traffic load areas, and second, making the final prediction within high-load segments.

4.2.3 The position of GNN4AGV in GNN taxonomy

GNNs at the node level are typically categorized into two types: transductive learning and inductive learning. In transductive learning, the model has access to both the training and test data. A typical approach involves using the target nodes with known labels to predict the labels of unknown nodes within the same graph, effectively completing the graph. This method often exhibits characteristics of semi-supervised learning. In contrast, inductive learning is fully supervised, where the model uses graphs with known node labels to predict labels for entirely unseen graphs[4]. GNN4AGV clearly falls into the latter category, as it aims to learn from the waiting time in past AGV projects to make predictions on layout designs for new AGV projects that are not represented in the training data.

In inductive learning, the structure of the graph is often of greater significance, particularly in AGV systems. The connections between nodes and edges in the graph directly reflect the traffic connections within the AGV network, which are closely linked to congestion. Understanding this relationship is crucial.

5
Method

This chapter introduces a regression hierarchical GNN framework designed to address imbalance. We provide a detailed introduction to this framework for both the training and inference phases. Furthermore, an in-depth exploration of GNN construction methods is included. Finally, we discuss the model evaluation scheme and implementation details.

5.1 Model input and output

Based on the earlier multi-relational graph representation and the definition of message passing in Section 3.1.1 and 4.1.1, the input for a node-level regression GNN model consists of three components: the adjacency matrix representing the layout graph structure, multi-relational data encoding traffic rules, and the three segment features introduced in Section 2.3.2—travel time, predicted active usage, and blocking probability. As discussed in Section 4.2, the model output is the waiting time in seconds.

5.2 Hierarchical graph neural network framework

According to Section 4.2.2, the core issue of GNN4AGV has shifted to: "How to construct a regression model in the presence of imbalance." In this scenario, handling the imbalance problem becomes the key challenge in model development.

In traditional machine learning, widely used techniques such as upsampling and downsampling are often employed. However, these methods cannot be directly applied in our context. In node-level classification tasks, upsampling and downsampling artificially increase or decrease the number of nodes, which disrupts the original graph structure (i.e., the layout topology), making these approaches unsuitable.

This challenge motivates the construction of a hierarchical framework. The framework consists of two distinct GNN models, responsible for classification and regression tasks, respectively. Logically, this hierarchical data flow structure avoids the difficulty of directly building a regressor on a highly imbalanced dataset.

We will elaborate on this hierarchical framework from two perspectives: inference and training.

5.2.1 Inference phase

In the Hierarchical Framework, there are two node-level GNN models. The first is a binary classifier that predicts whether the waiting time of each segment is zero, i.e., whether it is a zero-blocking segment. The second is a regressor that estimates the exact waiting time for each segment. As discussed in Section 4.2.2, zero-waiting segments and non-zero-waiting segments may exhibit fundamentally different AGV system behaviors. Therefore, using two separate models to handle these behaviors is a more reasonable approach.

During framework execution, for a given segment m, if the binary classifier predicts it as a zero-blocking segment, a waiting time of zero seconds will be output as the final regression result. If it is classified as a non-zero-blocking segment, the regressor's prediction will be used as the final output of the framework.

The reason this GNN framework is referred to as "Hierarchical" is that for segments classified as non-zero blocking by the classifier, they appear to flow through a selection process before reaching the regressor, which then computes the exact waiting time prediction as the final output. However, in reality, the inference of both GNN models is performed independently and simultaneously for all segments, including zero-blocking segments. Therefore, in a strict sense, the relationship between the two models is more parallel rather than strictly hierarchical.

The schematic diagram of the Hierarchical framework is as follows:

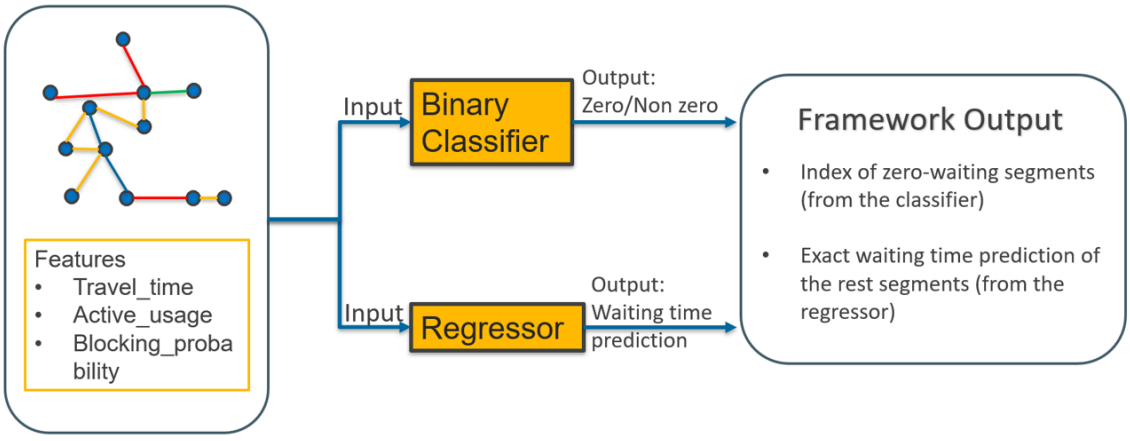

Figure 5.1: Hierarchical framework.

The final output is as follows:

$$\hat{y} = \begin{cases} 0, & \text{if the classifier predicts a zero-blocking segment} \\ \hat{y}_{regressor}, & \text{otherwise} \end{cases}$$

where:

- $\hat{y}_{regressor}$: The prediction from the regression GNN in the framework.

5.2.2 Training phase

The logic in the inference phase is relatively straightforward and concise, whereas the training phase is more critical. The core question is: how can we train the models so that the classifier learns the pattern of zero/non-zero blocking while the regressor focuses on learning the exact waiting time for non-zero blocking segments? For the classifier, standard training methods can be applied directly, as every segment can be categorized as either a zero-waiting or non-zero-waiting segment. However, for the regressor— which is the more crucial part— our solution is to use a "data masking" technique.

As introduced in Section 3.1.1, the message-passing structure of GNNs aligns with the layer-wise operations of traditional neural networks. This allows us to train the model using the Backpropagation algorithm. The training process always begins with the computation of the loss function, followed by backpropagation to iteratively update the model parameters. Each parameter update is determined by propagating the gradient step by step, gradually refining the model parameters throughout training to minimize the gap between the predicted and actual values.

When training the classifier, since the classification of a segment as non-zero/zero-blocking applies to all segments, we use the standard cross-entropy loss function. The model is then trained in the same manner as a conventional neural network.

The cross-entropy loss function is defined as:

$$\mathcal{L}_{CE} = -\sum_{i=1}^{N} y_i \log(p(\hat{y}_i)) + (1 - y_i) \log(1 - p(\hat{y}_i)) \tag{5.1}$$

where:

- N: The total number of segments.
- y_i: The true label (0 for zero-blocking, 1 for non-zero-blocking).
- $p(\hat{y}_i)$: The predicted probability of a segment being non-zero-blocking.

When training the regressor, this is where the data mask technique comes into play. To ensure that the GNN only learns from non-zero waiting segments, we apply a mask during loss computation to filter out all zero-waiting segments, allowing only non-zero waiting segments to contribute to the loss calculation.

As a result, the GNN training process focuses solely on minimizing the prediction error for non-zero waiting segments. The chosen loss function is the Mean Squared Error (MSE) loss, defined as:

$$\mathcal{L}_{MSE} = \frac{1}{N'} \sum_{i \in \mathcal{M}} (y_i - \hat{y}_i)^2 \tag{5.2}$$

where:

- N' is the number of non-zero waiting segments.
- $\mathcal{M}$ represents the set of indices corresponding to non-zero waiting segments.
- y_i is the true waiting time of segment i.
- $\hat{y}_i$ is the predicted waiting time of segment i.

By applying this masked Mean Squared Error loss, the model effectively ignores zero-waiting segments and focuses on optimizing predictions for segments with congestion.

5.3 Systemic exploration of design space of GNN4AGV

Now that we have established a GNN framework suitable for AGV systems, the next step is to populate the framework with two GNN models. This chapter systematically explores the design space of GNNs from a theoretical perspective while incorporating domain-specific factors relevant to AGV systems as much as possible. The experimental results of these explorations will be presented in the next chapter.

5.3.1 GNN design space

The concept of the design space was introduced to address the need to construct GNN in a systemic way and has been widely recognized in GNN related research[16]. The design space divides GNNs designs into 12 distinct dimensions, which are categorized into three levels: intra-layer design, inter-layer design, and learning configuration. GNN researchers should design GNN based on these dimensions. Furthermore, through extensive experiments, several general insights have emerged, such as using the PReLU activation function often yields better results than the basic ReLU, and the application of batch normalization techniques can also enhance performance.

On the other hand, if we return to the core of GNN—the message passing layers, we observe that the update function is not included within the design space. In fact, the majority of GNN research, including that on the design space, has focused on expanding the variance of the aggregator component. The differences among the four GNN models used in this study are primarily rooted in the design of the aggregator. Recently, however, an increasing body of research has shifted focus toward the update function. In some domains, the basic update function, which

is essentially a one-layer perceptron, is inadequate for capturing complex patterns in data. Therefore, we should consider incorporating the update function into the design space. In fact, the connectivity dimension within the original design space (which includes residual connection[18], dense connection[19], etc.) can be viewed as a subset of the update function. Taking the residual connection as an example, it is essentially a form of linear interpolation applied to the basic update function.

The final design space is illustrated in the figure below, where we have added the dimension of the update function.

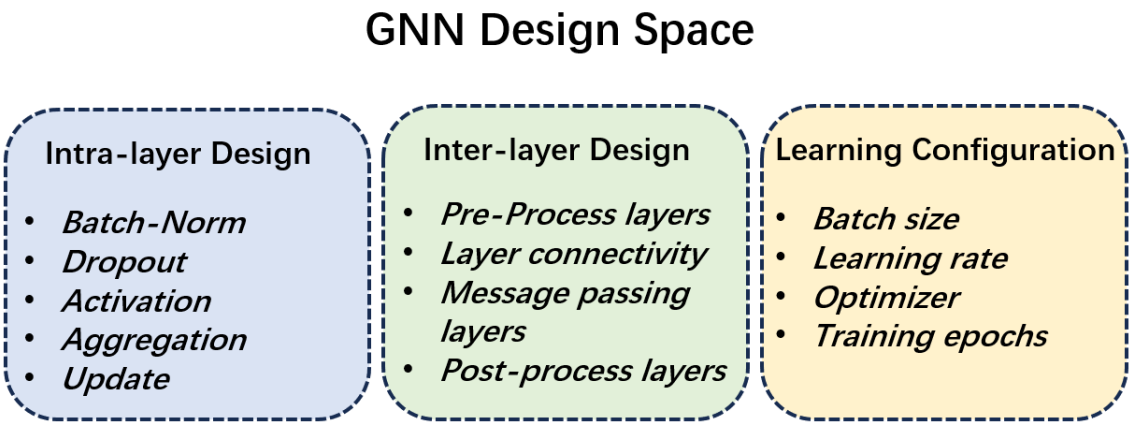

Figure 5.2: Design space of GNN4AGV.

5.3.2 How many Layers?

The number of layers is the most intuitive hyperparameter for any type of neural network.

When exploring GNN4AGV, an important aspect that cannot be overlooked is the strong correspondence between the dual graph to be trained and predicted, and the physical AGV network. For instance, the nodes represent the traffic segments within the AGV network. This differs significantly from knowledge graphs or recommendation systems, where all nodes and edges are derived through abstraction and modeling. Among these connections, one particularly important aspect is the relationship between traffic propagation and GNN depth. In an AGV system, the propagation range of traffic congestion is a crucial factor to consider. For example, when congestion occurs in a specific segment, how far can its impact extend to other segments? Conversely, could the congestion of a segment be attributed to events occurring several segments away? Answering these questions requires deep domain knowledge.

As a result, this range consideration adds complexity to determining the appropriate number of layers in a GNN. In GNNs, each additional layer corresponds to an extra step of forward or backward propagation of traffic congestion. Therefore, the depth

of the network should be determined by the range of congestion propagation within the AGV system. In other words, in GNN4AGV, the choice of GNN depth should be guided by domain knowledge.

On the other hand, beyond the depth needed from AGV domain, it is worth investigating the question: Can GNNs be as deep as AGV systems require? If AGV system engineers believe that traffic congestion typically arises from conditions beyond five segments, then a GNN with a depth of five should be designed accordingly. However, this raises a crucial question: is this feasible? This leads to a fundamental issue in GNNs called over-smoothing[20]. In message-passing based GNN models, the feature information at the original node level (i.e., the segment) becomes increasingly diluted as the number of layers increases. Nodes gradually absorb information from other nodes in the AGV network, leading to a situation where, after several rounds of message passing, the information across different nodes converges, making them increasingly similar. As a result, the model fails to distinguish between nodes, causing GNN training to fail.

Therefore, for data that requires deep structural representation, it is crucial to employ Graph Neural Networks (GNNs) that can prevent the loss of a node's original identity during message passing. An intuitive approach to achieve this is through the use of residual connections, where interpolation of previous node features helps to mitigate feature dilution[18]. This corresponds to the update or connectivity dimension within the design space. In fact, when the data exhibits more complex patterns, more sophisticated update functions can be utilized to construct complex GNNs. These advanced update functions are also capable of addressing the issue of over-smoothing.

5.3.3 Theoretical consideration regarding aggregator choices

The core of the GNN model lies in the aggregator, making the selection of an appropriate aggregator crucial. Exploring the theoretical foundations of specific aggregators is not the focus of this paper. Instead, this section aims to discuss the considerations involved in selecting an appropriate aggregator for the AGV system. There is a wide range of available aggregators, including the most prominent ones such as GCN, GAT, and GraphSage, along with many emerging alternatives. Despite the abundance of options, insights from graph theory and the earlier discussion on GNNs for AGVs provide valuable guidance for making the decision.

In graph theory, the update process of the WL (Weisfeiler-Lehman) algorithm, used for graph isomorphism testing, is similar to the message-passing process in Graph Neural Networks (GNNs). Both methods involve aggregating information from neighboring nodes and representing each node accordingly, which establishes a clear analogy between the two. Consequently, the WL algorithm can be used as a tool to evaluate the ability of GNNs to distinguish graph structures and assess their representational power, making WL a standard benchmark in this context. From

this analogy, we can deduce the following two facts: 1) the WL algorithm defines the theoretical upper bound for GNNs, and 2) to achieve this theoretical upper bound, both the aggregator and update function in GNNs must be capable of modeling injective functions[22][23]. It is worth noting that the key difference between the WL algorithm and GNNs is that the former processes discrete labels, while the latter focuses on node embeddings, which are not necessarily discrete. Therefore, the performance upper bound determined by the WL algorithm is not entirely accurate.

Moreover, Xu et al. suggests that incorporating learnable parameters in the aggregation stage and simply stacking more single-layer perceptrons in the update stage can further approximate the upper bound[22]. Therefore, in practical applications, we should not rigidly adhere to the performance upper bound defined by the WL algorithm as the sole criterion but rather adopt a flexible approach.

For the update part, the most common choice is to use a basic one-layer perceptron which is also the one that we used in this thesis. According to Morris, a basic GNN with such an update function is sufficient to reach the upper bound. If we treat the update mechanism as the baseline, the focus should shift to the aggregator. In fact, the basic sum aggregator satisfies the injectivity condition we discussed earlier. However, introducing neighbourhood-wise information extraction techniques typically leads to the loss of this injective mapping relationship within the aggregator. Among the four models discussed in this paper, three variants—GCN, GAT, and GraphSage—do not satisfy the injectivity condition. Although recent research has modified these mainstream models to meet theoretical upper bounds, this is not the focus of our current study. From a theoretical perspective, we should always begin with the basic GNN, as it can express injectivity, which enables the model to reach the upper bound. Then, depending on the specific needs of the AGV domain, different aggregators should be chosen. Generally speaking, aggregators based on advanced neighbourhood-wise information extraction techniques tend to violate the theoretical upper bound. Thus, the consideration here is similar to the previous section, where domain-specific knowledge in AGV should be the key factor in selecting the appropriate aggregator.

For Graph Convolutional Networks (GCNs), the normalized GNN aggregator model should only be considered in scenarios where the structural information of the domain is not critical and the focus is primarily on the features, such as in transductive learning tasks (e.g., knowledge graphs). This is because normalization is effective in capturing statistical distribution (proportion) information related to graph neighborhoods, but it tends to dilute structural information. In contrast, our case presents the opposite situation: GNN4AGV is an inductive learning task, and the structural connectivity between segments directly contributes to congestion. Therefore, local structural information, such as the connectivity between segments and the degree of connectivity, plays a dominant and crucial role.

As for the GraphSAGE network, this approach is essentially a general GNN framework. Taking the max variant as an example, the use of max pooling similarly

results in the loss of some information. However, the advantage of this approach lies in its emphasis on the maximum value, focusing solely on the most significant node features within the neighboring nodes during the message passing process. This can be considered as capturing the "skeleton" or "main path" of the graph. Additionally, GraphSAGE demonstrates robustness against noise and outliers. While the characteristics of GraphSAGE exhibit some similarities to those of AGV systems, the exact alignment remains unclear.

For Graph Attention Networks (GAT), the attention-based aggregator enables message passing to treat neighboring nodes differently, distinguishing it from GraphSAGE, which focuses more on features. This ability to differentiate between neighboring nodes relies on additional α calculations, making it suitable for handling complex graph structures. However, excessive reliance on such computations may be counterproductive. In the current experimental dataset, all layouts originate from a constrained T-intersection, and the number of features is relatively limited. Therefore, we should be cautious when selecting attention-based GNNs. Nevertheless, as future datasets become more diverse and comprehensive, GAT remains a promising approach worth considering.

In considering potential alternative aggregator options, it is imperative that we adopt a similar analytical approach.

5.4 Training configurations

First, regarding the learning configuration at the general machine learning level, this section covers four dimensions in the design space outlined in Section 4.2.1—learning rate, batch size, optimizer, and training epochs. For the optimizer and learning rate, we adopt the commonly used Adam optimizer with a learning rate of 0.001. This relatively small learning rate is chosen considering the limited dataset size. Likewise, the batch size is set to a small value of 8 for the same reason.

Building upon these three aspects, we will explore the remaining dimension—training epochs—using a basic GNN model with four message-passing layers and one post-message-passing layer. This approach helps establish a rough range for epochs before conducting comprehensive model optimization to determine the optimal configuration, thereby saving time from testing all possible combinations. It is worth noting that, in industrial-scale projects, an extensive hyperparameter search is advisable, and this is a trade-off. The experimental results for this part will be presented in Section 6.1.

5.5 Model evaluating scheme

In this thesis, we adopt the k-fold cross-validation strategy to train and evaluate the GNN model. Under this strategy, we divide 48 layouts into training and validation sets in pre-defined proportions and perform k iterations. In each iteration, the model is trained on the training set and evaluated on the validation set. With this setup, each data point serves as a validation sample once and as a training sample in the remaining $k - 1$ iterations[24].

More specifically, to train the GNN on as much data as possible, we employ the leave-one-out scheme, a special case of cross-validation where $k = 48$. In each iteration, only one layout is held out as the validation set, while the remaining 47 layouts are used for training the model.

This leave-one-out scheme is applied to all aspects of GNN training and evaluation in this thesis[25]. Specifically, in each iteration, we obtain the predicted values for the segments of an unseen layout, and the predictions for this validation set are crucial for evaluating the performance of the GNN. We compute the model evaluation metrics on the 48 validation layouts and take their average as the evaluation metric for the GNN. The choice of evaluation metric will be discussed in detail in the next section.

In addition, during the model optimization and hyperparameter search process, considering the randomness involved in neural network training, we performed three repeated experiments with the above scheme employed and took the average of the evaluation metric scores to mitigate the impact of random factors on the assessment of GNN performance.

5.6 Performance metrics

Due to the presence of both a classifier and a regressor in our framework, the evaluation metrics used for model optimization and best model analysis also differ accordingly.

For the classifier, we chose the Macro-F1 score as the evaluation metric. The reason for this choice is still related to the imbalance issue. As shown in Table 4.1, 83% of the target data belongs to zero blocking segments. In this case, even a dummy classifier that only predicts zero blocking no matter the input would achieve a very high accuracy of 0.83. Since F1-score takes both Precision and Recall into account, with Precision tending to score higher in imbalanced datasets and Recall tending to be lower, F1-score combines both factors, making it a more objective reflection of model performance with imbalanced data scenarios[26]. The expression for F1-score is as follows:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.3}$$

Where:

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

And the macro F1 score is:

$$MacroF1 = \frac{1}{N} \sum_{i=1}^{N} F1_i$$

Where N is the number of classes, 2 in our case, and $F1_i$ is the $F1$ score for class i (i.e. zero-blocking or not).

For the regressor, the most commonly used evaluation metric is Mean Squared Error (MSE). However, since we have already used MSE as the training loss function, we chose the more intuitive metric, Mean Absolute Error (MAE), in this case. The regressor's predicted results will serve as the predicted waiting time for non-zero blocking segments, and thus the absolute error is directly related to time. MAE reflects the average difference between the predicted and true values, measured in seconds, with a smaller MAE indicating better model performance. The Mean Absolute Error (MAE) is defined as:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{5.4}$$

On the other hand, as mentioned in Section 4.2.1, the second use case waiting time in the layout design process is to provide ranking. Compared to accurate prediction, ranking is a more lenient requirement. Therefore, even if there is a discrepancy between the regression prediction and truth, as long as a good ranking can be provided, it can still provide important insight. To assess the ranking performance, we chose Spearman's rank correlation coefficient ρ, which is essentially the Spearman correlation calculated based on the ranks of the predictions and true values[27]. It reflects how well the relationship between two variables can be described using a monotonic function and is calculated according to Equation 5.5, where d_i is the difference between the ranks of each pair of prediction and true target of the segment i, and n is the number of data points. When ρ equals 1, it indicates a perfectly correct ranking, and a score close to 1 is considered good. However, it is worth noting that MAE remains the core metric for evaluating the regressor.

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)} \tag{5.5}$$

5.7 Implementation details

This section details the specific code implementation of this thesis, including the programming languages, as well as the core packages and frameworks.

Python
Python has become one of the preferred programming languages for machine learning and data analysis in recent years. Its concise and intuitive syntax allows us to focus on the data itself. Additionally, Python offers a rich ecosystem of deep learning libraries and frameworks, such as PyTorch and PyTorch Geometric. In this work, Python is used for data preprocessing, model construction, training, and prediction.

Pytorch Geometry
The GNN models used in this study are implemented based on PyTorch Geometric (pyG). PyG is a Python library specifically designed for handling graph-structured data, built on top of PyTorch. It offers a variety of GNN models for use and extends PyTorch's deep learning modules to support graph structures, enabling efficient GNN construction and training.

MLflow
MLflow is an open-source machine learning platform that focuses on the full lifecycle of machine learning projects, ensuring that each phase is manageable, traceable, and reproducible. In this thesis, we utilize MLflow to track and manage all GNN experiment results, making the hyperparameter search process more transparent and organized.

R and ggplot2
For results visualization, the ggplot2 package in R language provides significant convenience with its unique Grammar of Graphics syntax. While Python offers similar visualization tools, such as plotnine, R's native ggplot2 is more comprehensive in terms of visualization options. Therefore, we choose to switch from python to R for visualizing the experimental results. We extract all experiment results from the MLflow platform using the MLflow API for R and present them through ggplot2.

6
Results

This chapter presents various results of the Hierarchical Framework in terms of training and prediction, including an evaluation of its predictive performance and core analyses related to GNN4AGV.

6.1 Training epochs

As stated in Section 5.4, with the other three dimensions in the Learning Configuration of the Design space confirmed, we trained the classifier and regressor for a larger number of epochs and recorded the training loss, validation loss, and performance metrics. The experimental results for the training epochs are shown in Figure 6.1, 6.2 and 6.3.

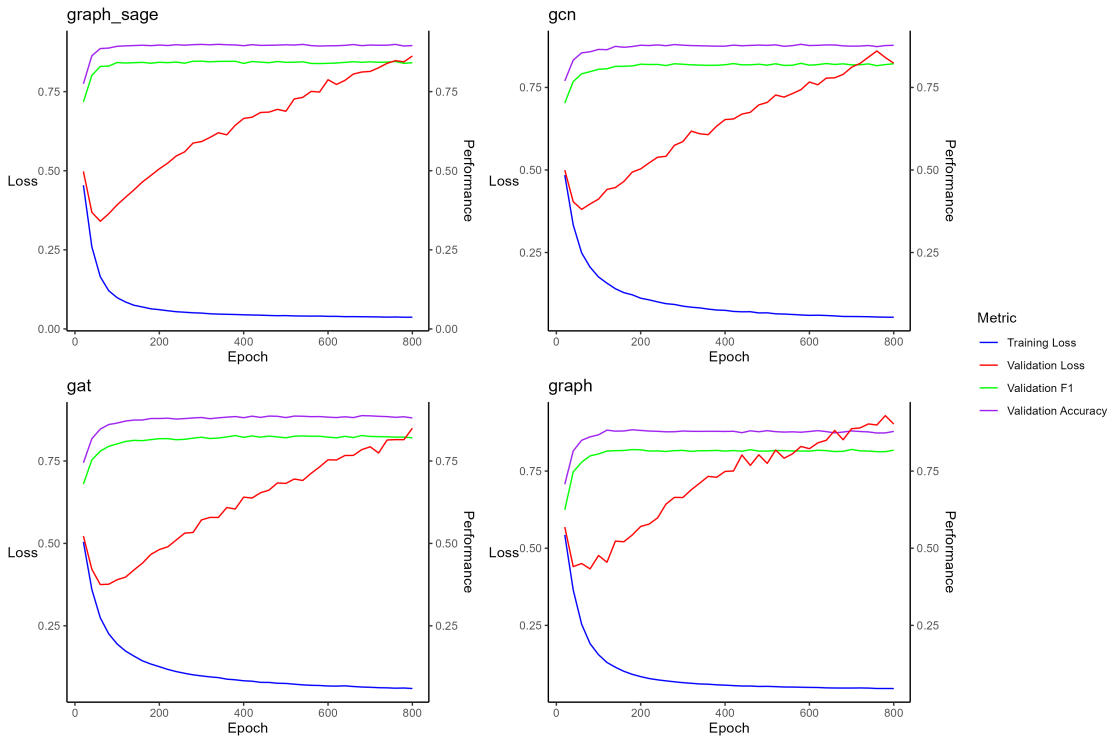

Figure 6.1: Training and validation performance metrics (MSE and MVE) across epochs for GraphSAGE, GAT, GCN and Basic GNN classifier.

As shown in Figure 6.1, the training loss of the classifier continues to decrease throughout the training process, while the validation loss stops decreasing and starts increasing at an earlier stage. In contrast, the F1 score and accuracy on the validation set do not exhibit a clear trend of rising and then falling, which differs from the trend of the validation loss.

This discrepancy is understandable because the loss function (cross-entropy loss) is based on class probabilities rather than class predictions. In classification problems, the final prediction is obtained by applying the softmax function to the class probabilities. Therefore, as training progresses, the model tends to overfit the training data, leading to smaller fluctuations in its parameters. During this process, the model becomes more confident in its predictions on the validation set, regardless of whether they are correct. These predicted class probabilities approach extreme values (close to 0 or 1), and the corresponding class probabilities stabilize, which is why the F1 score and accuracy on the validation set remain stable rather than decreasing as the validation loss decreases.

Based on these observations, we conclude that considering the region near the inflection point of the validation loss and the stability of the validation F1 score, the optimal number of training epochs should lie between 40 and 120. Therefore, we should search for the optimal GNN classifier parameter configuration within this range.

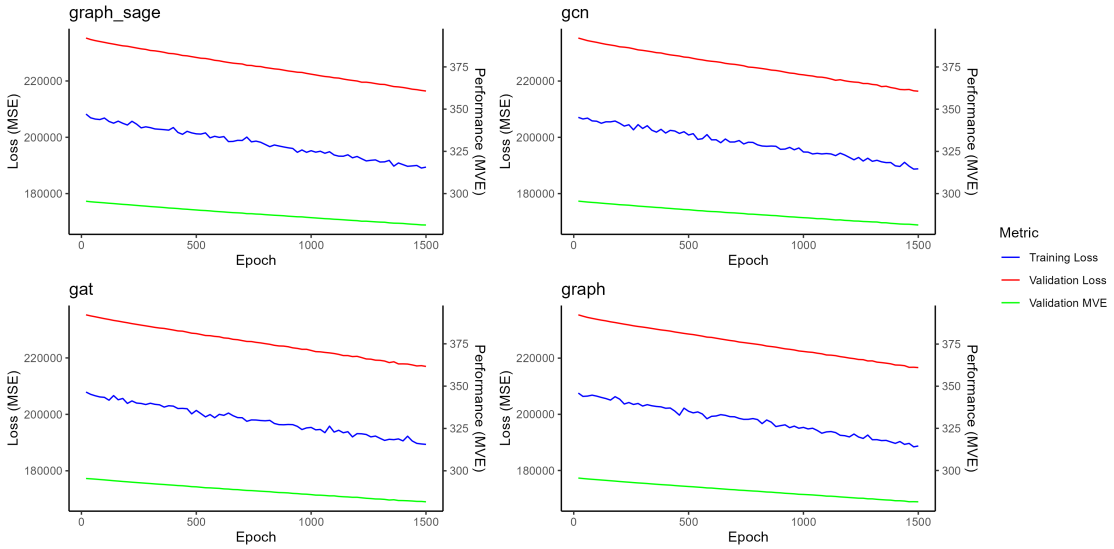

Figure 6.2: Training and validation performance metrics (MSE and MVE) across epochs for GraphSAGE, GAT, GCN and Basic GNN regressor.

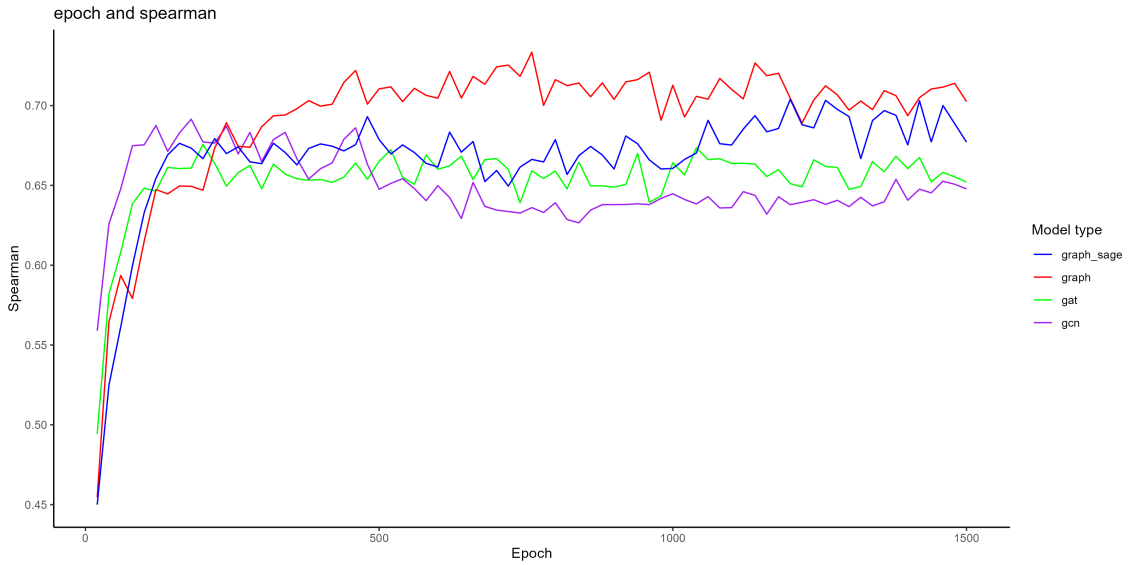

Figure 6.3: Spearman correlation across epochs for GraphSAGE, GAT, GCN, and basic GNN regressor.

For regressors, the situation is different. Even when conducting experiments with a large number of epochs, the training loss and validation loss continued to decrease and did not exhibit the expected trend observed in classifiers. Additionally, compared to mean squared error (MSE), mean absolute error (MAE) serves as a more intuitive evaluation metric, and its trend also showed a continuous decline. On the other hand, based on our monitoring of the Spearman rank correlation on the validation set, its trend remained unclear over a large number of epochs. Given this, to minimize the validation loss as much as possible, a larger number of epochs should still be used. Ultimately, we chose 1000 epochs.

6.2 Batch normalization and advanced activation choice

Based on numerous experimental results from the GNN design space paper, two prominent conclusions are that using Batch PReLU[28] and Normalization[29] rather than ReLU often leads to better performance. Therefore, we also conducted related experiments based on AGV data, and the results are shown in the Figure 6.4 and 6.5.

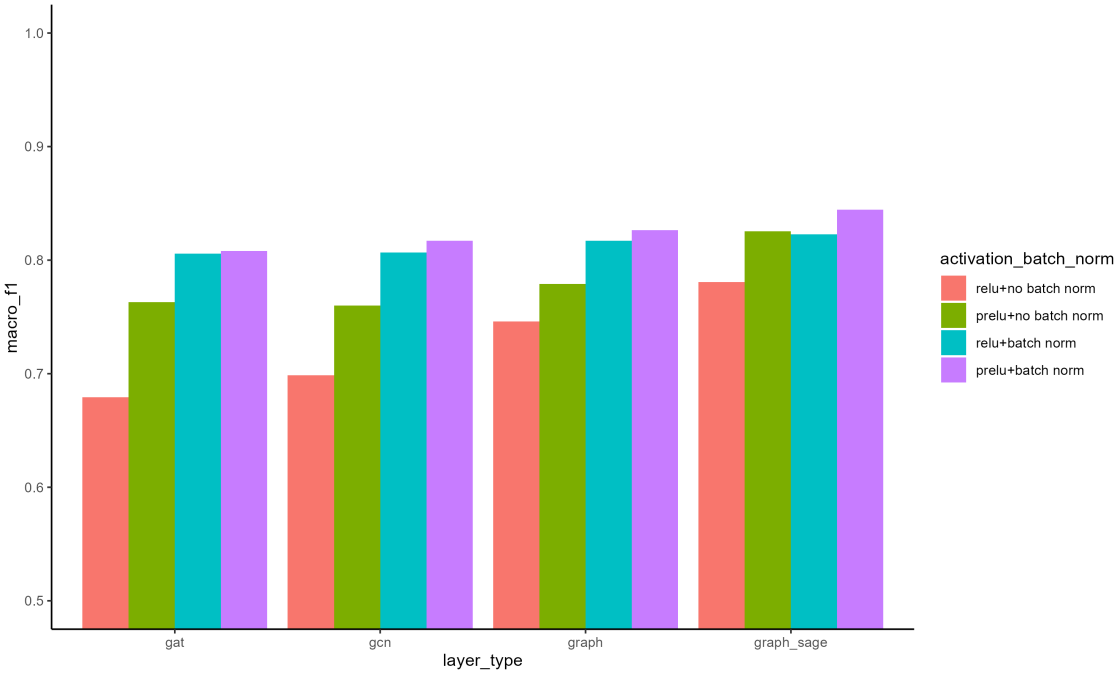

Figure 6.4: Macro-F1 across binary combinations of activation functions and batch normalization for classifier.

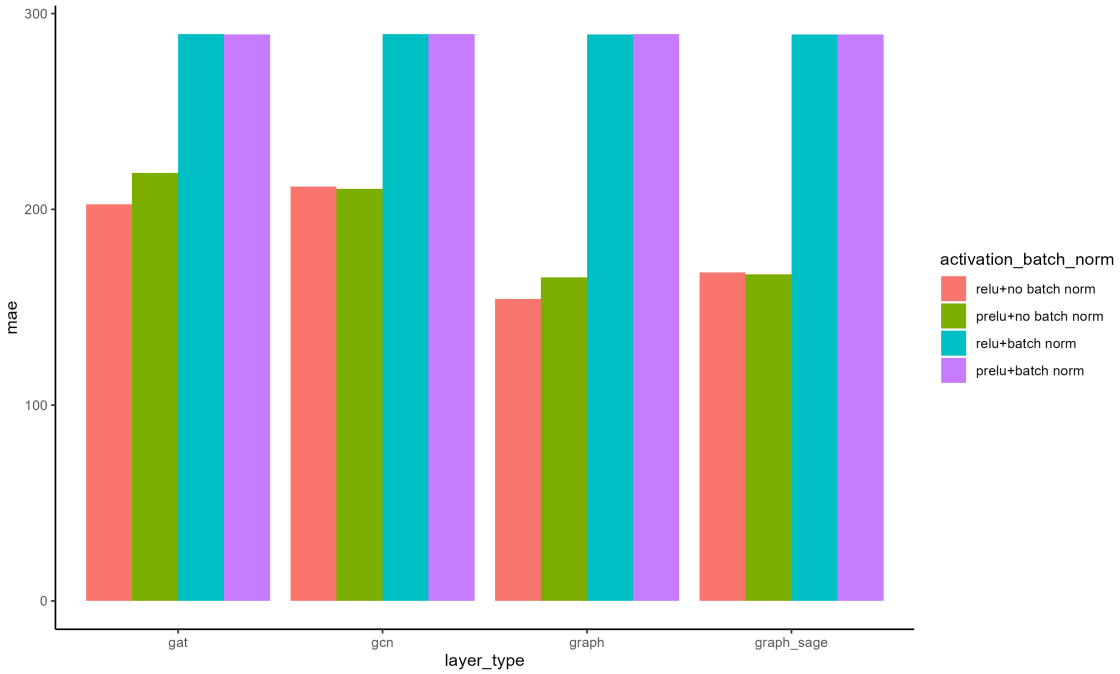

Figure 6.5: Mean absolute error (MAE) across binary combinations of activation functions and batch normalization for regressor.

As shown in the figure, the classifier and regressor once again exhibited significant inconsistencies. The classifier's behavior aligns with previous experiments in the design space, indicating that GNN models should perform better with the addition of

batch normalization and PReLU rather than ReLU as the activation function. However, for regressors, the use of batch normalization appears to harm performance, while the effect of PReLU remains unclear.

In summary, When performing model optimization on the classifier and regressor respectively, we should choose different strategies. For classifier, batch normalization and PReLU should always be used. For regressor, batch normalization should not be applied, while for the activation function, both ReLU and PReLU should be tested.

6.3 Model optimization

In this section we present the detailed model optimization process for the two GNNs in the Hierarchical framework.

6.3.1 Hyperparameter search

In the previous sections, we have explored some dimensions of the GNN design space. In the remaining parts, the Message Passing Layers, Post-process Layers, and Aggregation Layers constitute the core of GNN design. Therefore, we conducted a more comprehensive hyperparameter search.

For the classifier, the final hyperparameter search space is as follows:

- GNN aggregators: Basic GNN, GCN, GAT, GraphSage
- Message passing layers: 2, 3, 4, 5, 6
- Post layers: 0, 1
- Hidden neuron size: 16, 32
- Training epochs: 20 120
- Activation: PReLU
- Batch normalization: Yes

And as for regressor, things are a bit different:

- GNN aggregators: Basic GNN, GCN, GAT, GraphSage
- Message passing layers: 2, 3, 4, 5, 6
- Post layers: 0, 1
- Hidden neuron size: 16, 32
- Training epochs: 1000
- Activation: PReLU, ReLU
- Batch normalization: No

6.3.2 Best models

We conducted a comprehensive training exploration on the GNN hyperparameter range mentioned in Section 6.3.1 and optimized the validation set scores using different performance metrics for classifiers and regressors. The metrics used and their optimal scores are shown in the table 6.1.

Model Type	Evaluation Metric	Best Value
Classifier	Macro F1	0.856
Regressor	Mean Absolute Error (MAE)	119

Table 6.1: Evaluation metrics and optimal scores

The model hyperparameters corresponding to the optimal scores are shown in the Table 6.2 and 6.3.

Mp	Post	Hidden Neuron	Epoch	Aggregator
4	1	32	120	GraphSage (mean variant)

Table 6.2: Best classifier (Mp: message passing layers, Post: post message passing layers)

Mp	Post	Hidden neuron	Epoch	Activation	Aggregator
5	1	32	1000	PReLU	Basic GNN

Table 6.3: Best Regressor

6.4 Best model performance

This section will present more results on the prediction performance evaluation of the best model, focusing on the core issue of the GNN4AGV — can our framework predict segment waiting time?

6.4.1 Classifier results

For the best classifier, given that it is a binary classifier, the most intuitive evaluation method, in addition to MAE as the performance metric, is the confusion matrix, as shown in Figure 6.6. As visualized in the confusion matrix, the classifier successfully identified the majority of both zero-blocking and non-zero-blocking segments, and this is reflected in their high recall rates of 0.91 and 0.87, respectively. In terms of precision, the model performed better in identifying zero-blocking segments (0.95) than for, for non-zero-blocking segments (0.81). These results indicate that the binary classifier effectively learned patterns for both categories.

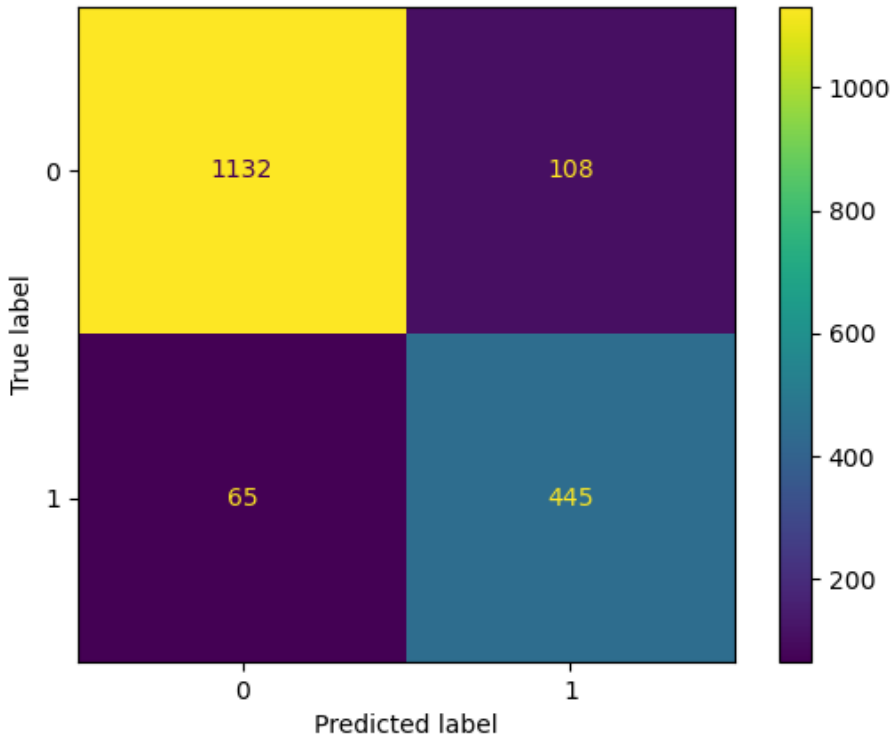

Figure 6.6: Confusion matrix of the best classfier: The best classifier successfully identified the majority of both zero-blocking and non-zero-blocking segments, achieving high recall rates of 0.91 and 0.87, respectively. In terms of precision, the model performed better in identifying zero-blocking segments, with a precision of 0.95, while still maintaining a relatively high precision of 0.81 for non-zero-blocking segments. These results indicate that the binary classifier effectively learned patterns for both categories.

6.4.2 Best regressor results

Since in our hierarchical framework, the output of the non-zero blocking segment is directly derived from the regressor, its evaluation should refer back to the two use scenarios of waiting time described in Section 4.2.1.

The first scenario requires the model to predict the waiting time for each segment with high accuracy, ensuring that even minor changes resulting from segment adjustments can be reflected. The second scenario has relatively lower accuracy requirements, focusing instead on distinguishing significant differences in waiting times between segments.

In addition to MAE in Table 6.1, which directly measures prediction accuracy, we can assess whether the model's predictions are reasonable by examining the similarity between the distribution of predicted waiting times and the distribution of

ground truth. The results are shown in the Figure 6.7, and the corresponding distribution of error is shown in Figure 6.8. As seen in Figure 6.7, the shape of the distributions are quite similar, meaning the trained model is mapping the inputs to a reasonable output range with a diversity that closely resembles the ground truth. To get a better understanding of the models behavior we also examine the distribution of the regressor's error, see Figure 6.8. It shows that the error is mostly between 0 and 100 seconds, with a long tail of larger errors. This aligns with the MAE score of 119 seconds stated above in Table 6.1. In other words, while the regressor struggles with accurately predicting the wait times of individual segments, with typical errors of 1-2 minutes during a 30 minute simulation, its output distribution closely matches the distribution of the simulated wait times.

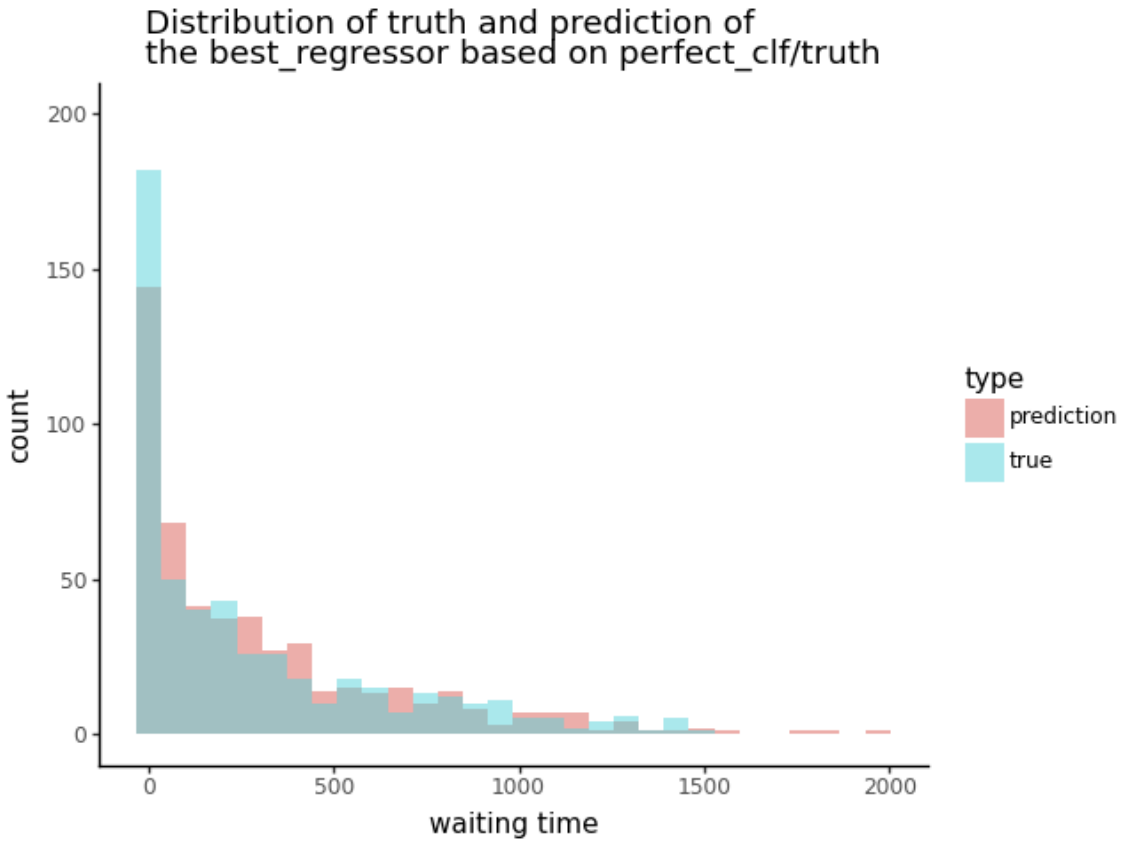

Figure 6.7: Comparison of true and predicted waiting time distribution using best regressor based on truth: The distribution shapes of the predicted and actual values are quite similar, with nearly identical ranges. While there are a few outliers in the tail of the predicted distribution that deviate from reality, their occurrence is minimal.

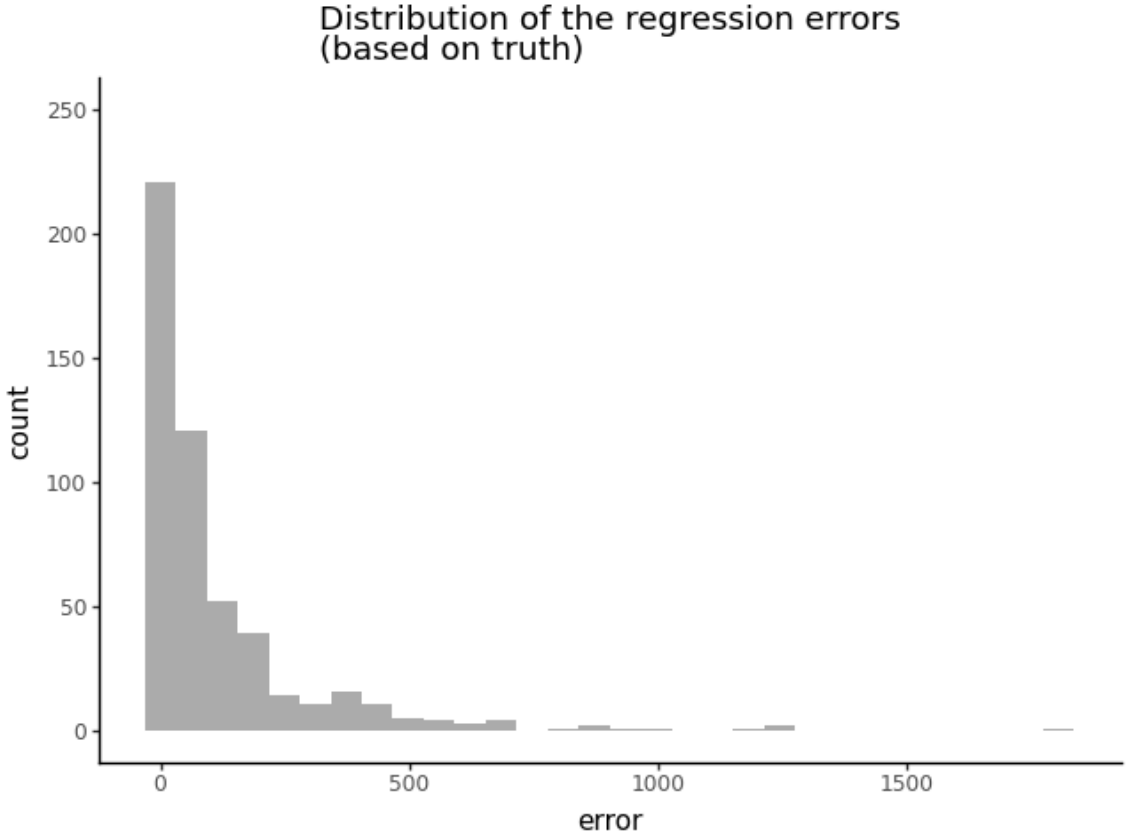

Figure 6.8: Distribution of regression errors based on truth: Most waiting time prediction errors are within 100, which corresponds to approximately 1.5 minutes in the context of a 30-minute window. This indicates that the regressor provides reasonably accurate predictions for most segments. However, a few errors exceed 500, suggesting that the framework completely failed on certain segments.

For the second scenario, we can also additionally check the spearman rank metric of each fold during the cross-validation process. The results shown in Figure 6.9 demonstrate a relatively accurate ordering for most layouts, with Spearman's rank correlation of around 0.8, while for a minority of the layouts the ranking fails to be effective.

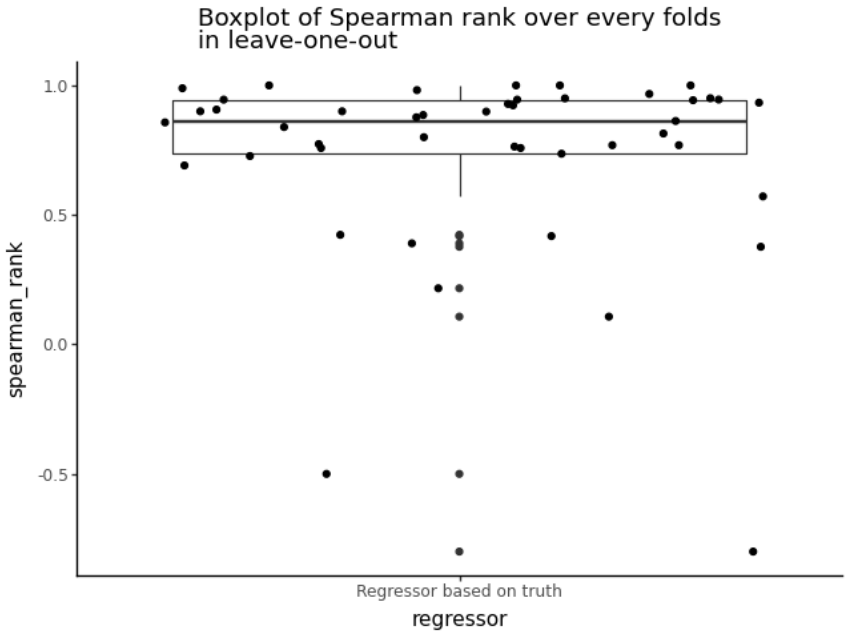

Figure 6.9: Boxplot of spearman rank correlation across folds in leave-one-out cross-validation using best regressor and truth: Regarding ranking performance, the regressor demonstrates relatively accurate ordering for most layouts, with Spearman's rank correlation around 0.8. However, in a few layouts, the ranking fails to be effective. This aligns with the error distribution pattern, where the framework performs significantly worse on certain segments or layouts compared to others.

6.4.3 Hierarchical framework results

As stated in Chapter 3, the fundamental difference between the training stage and the inference stage lies in the fact that, during inference, the determination of zero blocking segments is based on the classifier's predictions rather than the ground truth. This reflects how the trained model operates in real-world scenarios. Therefore, in Section 6.4.2, we present the prediction results of the best regressor under the condition of a perfect classifier, whereas in reality, the predictions are based on the combination of the best regressor and the best classifier. The corresponding results are shown in Figure 6.10, 6.11 and 6.12. The distribution of outputs and the distribution of errors are both strikingly similar to the previous results where a perfect classifier was used (see Figures 6.7 and 6.8). When it comes to ranking, the performance is somewhat degraded when compared to using the perfect classifier (Figure 6.9).

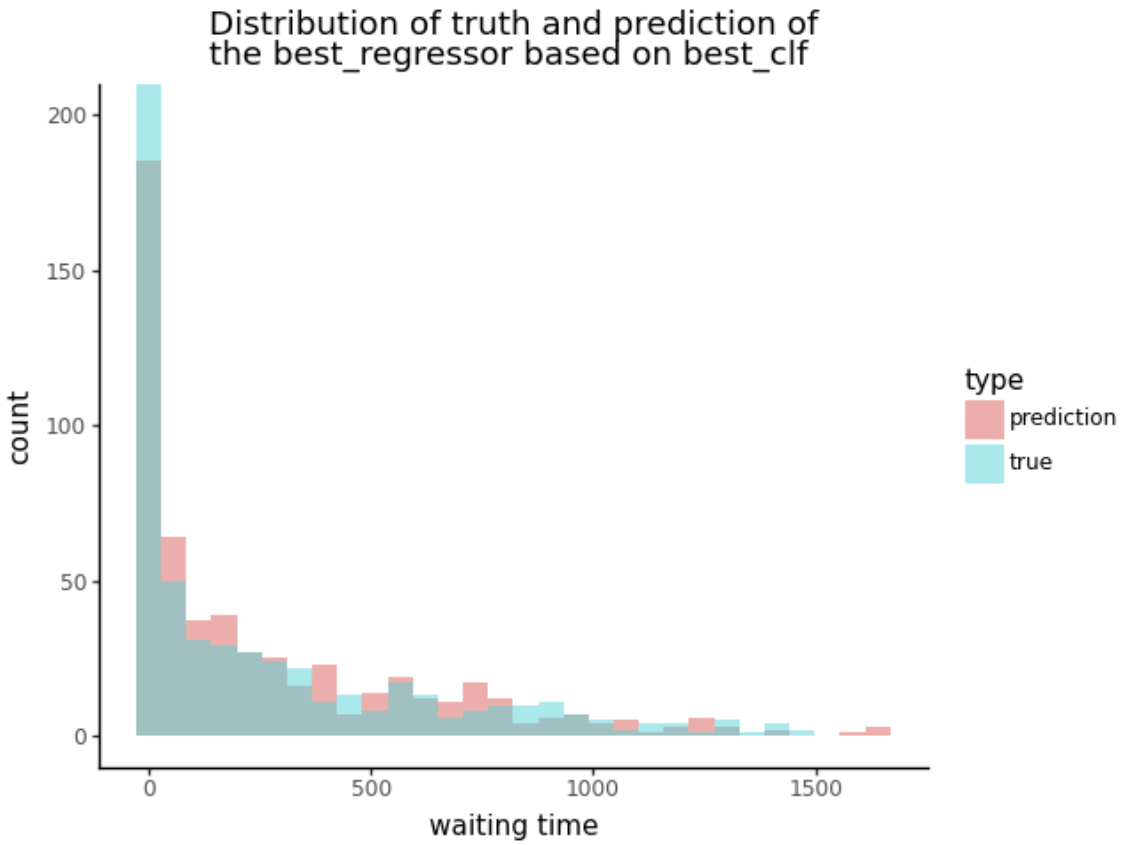

Figure 6.10: Comparison of true and predicted waiting time distribution using best regressor based on best classifier: Compared to the regressor based on binary truth in Figure 6.7, the regressor using the best classifier's results still produces a predicted distribution that closely matches the ground truth. This further demonstrates the strong performance of our classifier.

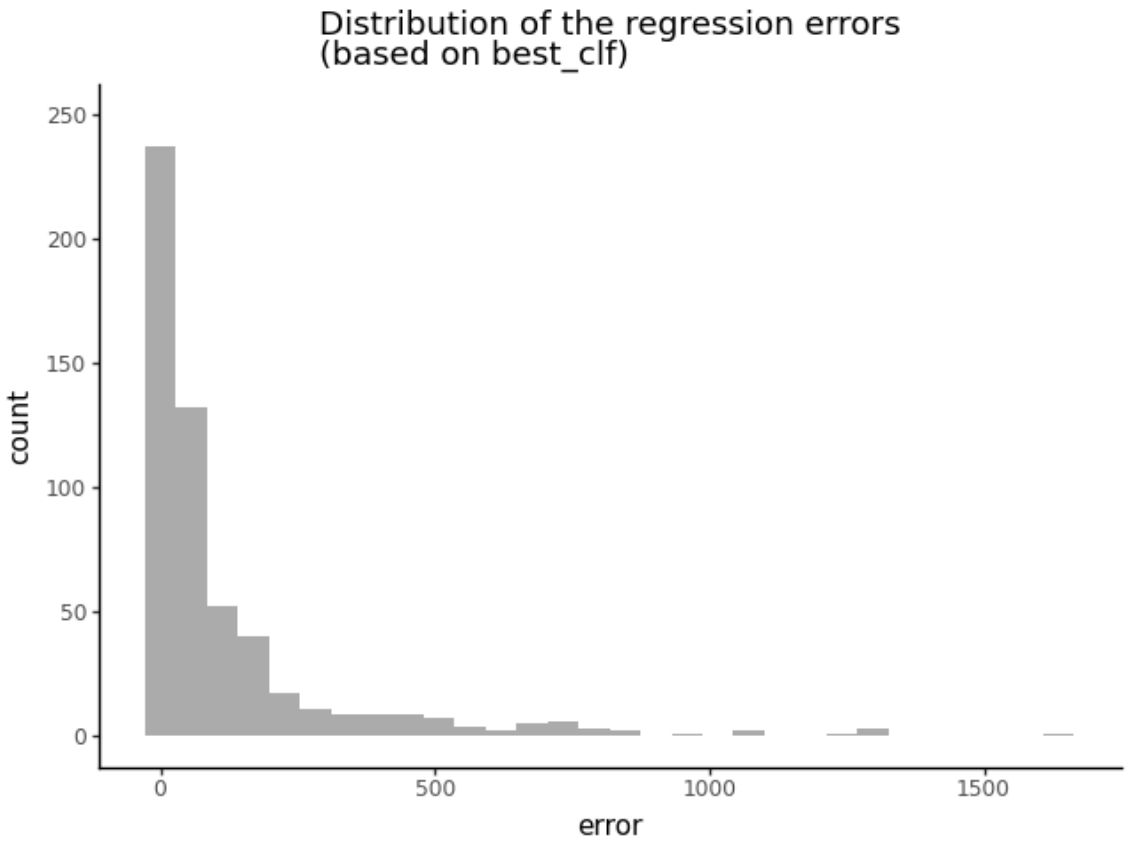

Figure 6.11: Distribution of regression errors based on best classifier: The error distribution exhibits a pattern similar to that of the best regressor based on truth in Figure 6.8.

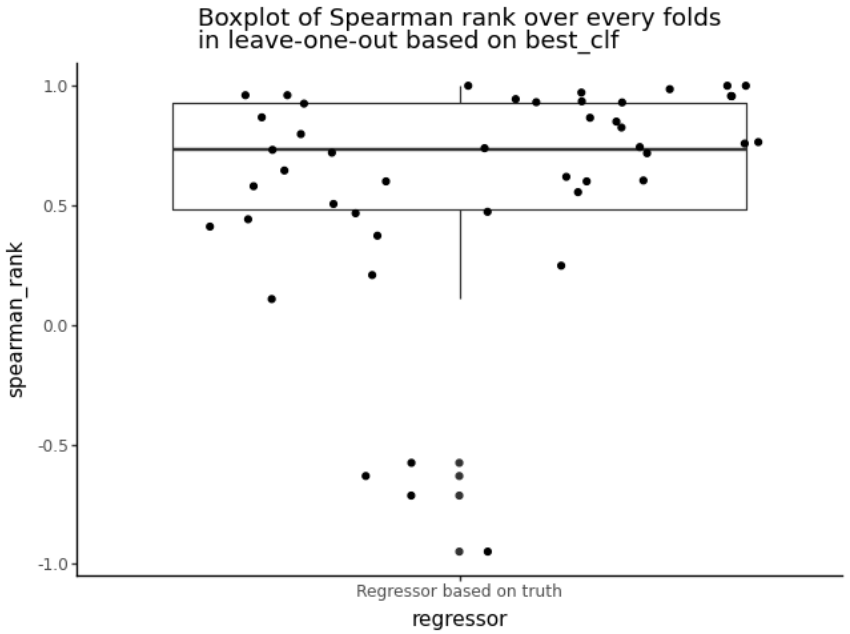

Figure 6.12: Boxplot of spearman rank correlation across folds in leave-one-out cross-validation using best regressor and best classifier: The framework's ranking performance remains similar to previous results in Figure 6.9 but fails to provide satisfactory outcomes for more layouts. This finding also reflects the limitations of ranking in the context of an imbalance issue. When only a very small number of segments in a layout bear vehicle waiting—such as just one or two—ranking and evaluating the results become significantly less meaningful.

Moreover, we can incorporate the classifier's predictions, specifically the segments predicted as zero blocking, to obtain the distribution of the entire framework's predictions. As shown in Figure 6.13, distributions of the predicted and actual value remain highly similar.

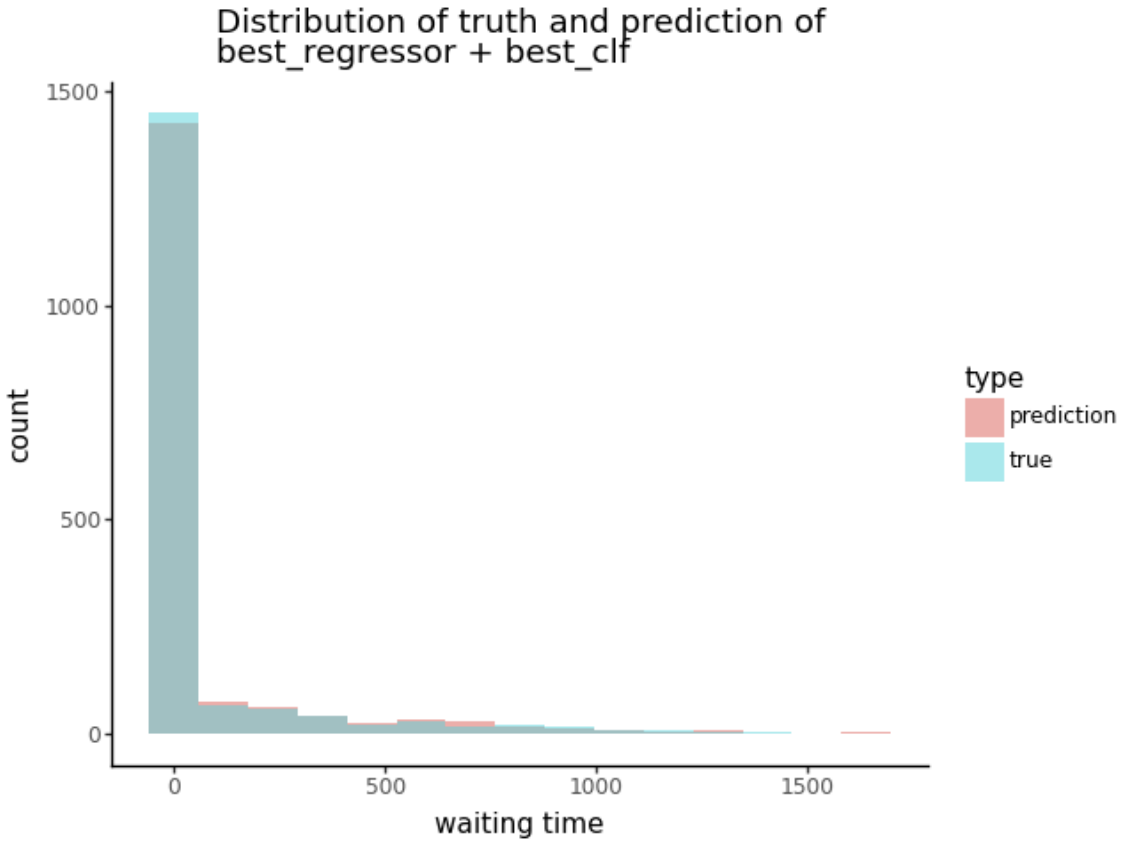

Figure 6.13: Comparison of true and predicted waiting time distribution using best regressor and classifier: After combining the data from the classifier and regressor, the predicted and actual value distributions remain highly similar.

6.5 GNN4AGV–GNN depth exploration

As discussed in the Section 5.3.2, when there are demands for increased network depth, oversmoothing can lead to potential issues in the training of Graph Neural Networks (GNNs).

When optimizing the binary classifier, this phenomenon is particularly evident. Training deeper GNNs becomes more difficult to converge, and for the same number of epochs, deeper networks often exhibit worse performance. This is ultimately reflected in the model optimization results, where the winning classifier has a depth of only 4 instead of 5, 6, or 7.

To explore a solution for over-smoothing based on AGV data, we introduced residual connections and adopted a more complex GRU update function to enhance the

model's update mechanism[30][31]. The effect of this improvement can be observed by monitoring the trend of training loss as epochs progress. The training loss curves of the basic GNN and the GNN with more advanced update functions, including extra residual connections and the GRU update function, are shown in Figure 6.14.

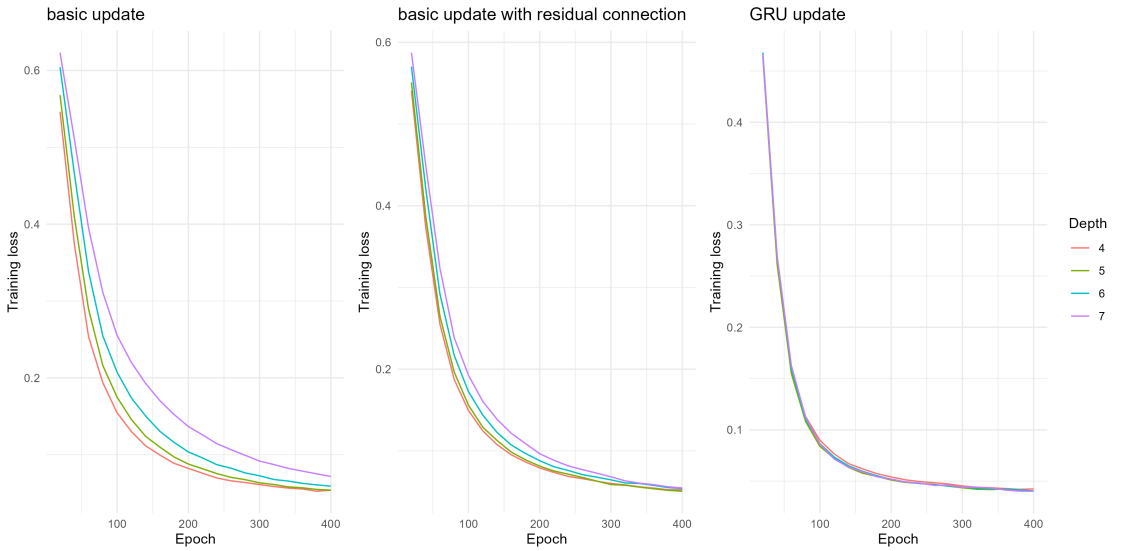

Figure 6.14: Comparison of training loss across different update functions and model depth: When using the basic update function, deeper GNNs require longer training epochs to converge. With the introduction of residual connections and more advanced GRU update function, the differences in loss curves over training epochs are reduced, making the training of deeper GNNs easier to converge.

This result echoes our discussion in Section 5.3.2. If, potentially more complex AGV system designs require deeper GNNs due to a broader traffic propagation range, introducing residual connections or more advanced update functions can help improve convergence, making the model training process more seamless.

7
Discussion

This chapter begins with a discussion of the results from Chapter 6 that are related to the core research question, then analyzes the limitations and shortcomings of this thesis, and finally concludes with corresponding future research directions.

7.1 Hierarchical framework evaluation

7.1.1 Results evaluation

First, at the binary classifier level, the model demonstrates strong predictive performance, as evidenced by both the winning macro F1 score and the confusion matrix. In the confusion matrix, our classifier does not exhibit any noticeable bias, with similar prediction accuracy for zero blocking segments and non-zero blocking segments.

Although this binary classifier primarily serves as a component of the framework, contributing to regression-based predictions, it also provides valuable insights on its own. In certain layout design scenarios, just identifying which segments may have wait time associated with them at all (i.e. which segments are non-zero-blocking-segments), could assist the Application Engineer in directing his or her attention to areas in the layout that merit further scrutiny before a simulation. in other words, it could be possible that helping the Application Engineers focus on a smaller subset of segments would enable them to iterate on the layout design more efficiently.

As for whether GNN4AGV has achieved its core research objective—directly predicting segment-level waiting time in seconds—this question undoubtedly requires an answer from the performance of the regressor. As seen in results, during model optimization, the winning MAE was 119. In the context of our dataset, this translates to an waiting time prediction error of almost two minutes. Considering that the total simulation duration in our dataset is 30 minutes, such an error is not negligible. More importantly, in the actual operation of the hierarchical framework, the classifier may misclassify some non-zero blocking segments, causing the regressor to generate insensible regression predictions for these segments, potentially leading to an even greater prediction error.

Although this phenomenon can partly be attributed to limitations in dataset size and other factors, for the best regressor, we still need to investigate whether the

framework has truly learned meaningful patterns within the AGV system. This aspect is crucial for this exploratory project.

We evaluate the regression prediction values through distribution analysis, as shown in Figure 6.7, 6.10 and 6.13. The results indicate that, both during the training phase and in the framework's actual operation, the distribution of predicted values closely matches the distribution of true values. Therefore, the hierarchical framework has not generated predictions that fall outside reasonable ranges or are distributed unreasonably, suggesting that the GNN model has still learned meaningful patterns.

On the other hand, for the ranking of waiting times, we examined Figure 6.9 and 6.12. During cross-validation, most of the folds showed relatively high scores (greater than 0.5), but some folds had lower scores. This suggests that the framework is capable of providing highly accurate rankings in most layouts, while the rankings in certain layouts are completely unreliable. This phenomenon is similar to the error distribution shown in Figure 6.8 and 6.11. The reasons behind this include not only the need for further improvement in the framework's prediction accuracy but also the limitations of the dataset. A discussion of these limitations will be provided in the next section.

In summary, although the hierarchical framework we have developed is not yet ready for production, it has undeniably learned patterns related to congestion and traffic within the AGV system, demonstrating promising potential.

7.1.2 Limitations

Although the current Hierarchical framework has successfully learned important patterns in the AGV system, its performance in waiting time prediction is still insufficient for real-world applicability at this point. To some extent, this also highlights certain limitations of this thesis.

The first limitation lies in the dataset. As mentioned in Section 2.3.1, as a deep learning project, a dataset with only 48 samples inevitably leads to overfitting issues during GNN training. Moreover, the dataset size may not be sufficient to reveal rich patterns within the AGV system. In this case, it is challenging to mitigate overfitting by further splitting the dataset and monitoring the changes separately, especially in the training of the regression model. Although we used a large number of epochs, no clear overfitting threshold was observed, further indicating the insufficiency of the data. Deep learning models require sufficiently large datasets to learn all domain patterns as comprehensively as possible.

Furthermore, the layout we used represents only a small part of a real AGV system layout. As a result, each layout contains only a limited number of segments, which may not fully capture all possible scenarios encountered in real-world AGV operations. In our dataset, the total number of segments/nodes is 1500. In comparison,

the smallest dataset for node-level tasks in Stanford University's GNN benchmark dataset contains 132,534 target nodes. This significant difference in data scale suggests that our dataset may be insufficient to support a comprehensive learning of AGV operational patterns by the GNN.

On the other hand, GNN4AGV also has its own limitations. Returning to the original research context, our primary motivation for adopting GNN was to accelerate the layout design process by providing waiting time predictions. However, in layout design, other key evaluation metrics exist beyond waiting time, such as order delivery rate. The core objective of an AGV system is to complete assigned orders on time. Thus, if vehicles can fulfill their assigned tasks within a given layout as scheduled, the layout can be considered acceptable. However, this information has not yet been incorporated into the current GNN4AGV modeling framework.

7.2 Future works

Future work can focus on two main aspects, corresponding to the two limitations mentioned previously: on one hand, improving the predictive accuracy of the hierarchical framework, and on the other hand, continuously refining the objectives of GNN4AGV to more comprehensively support layout design process for AGV systems.

For the former, in the future GNN4AGV project, we should generate and simulate more layouts, especially layouts that are more similar to real-world ones, and try to incorporate more features into the GNN modeling. Additionally, the layouts in this thesis only include two types of traffic rules. Implementing more traffic rules that are used in real AGV systems would help narrow the gap between the training set and real AGV systems. On the other hand, this thesis does not introduce significant customized innovations at the GNN algorithm level, and the GNN models used are relatively popular choices. A potential avenue to strengthen GNN4AGV is to draw more domain-specific knowledge from the AGV domain and integrate it into the theoretical design of the GNN's aggregator and update function.

For the latter, this is a more open-ended direction. Instead of building a regression model that is "outstanding" from a scientific research perspective, we should always prioritize whether the model's output meets the actual needs of the layout design process. Therefore, continuously adjusting the prediction objectives and modeling scope of GNN4AGV based on domain knowledge to better align with real-world requirements will be of greater value.

Bibliography

[1] NextSmartShip. (2023). AGV robots for automation in warehousing. Retrieved from `https://tinyurl.com/42rvdzwx`.

[2] Dehghan, A., Cevik, M., & Bodur, M. (2023). Dynamic AGV Task Allocation in Intelligent Warehouses. *arXiv preprint arXiv:2312.16026*.

[3] Thylén, N. (2025). *Design of Automated Guided Vehicle Systems Considering the Human, Technical, and Work Organisational Subsystems* (Doctoral dissertation, Chalmers University of Technology).

[4] Hamilton, W. L. (2020). Graph representation learning. *Morgan & Claypool Publishers*.

[5] Waikhom, L., & Patgiri, R. (2023). A survey of graph neural networks in various learning paradigms: methods, applications, and challenges. *Artificial Intelligence Review, 56*(7), 6295-6364.

[6] Zhang, X., Wang, L., Helwig, J., Luo, Y., Fu, C., Xie, Y., ... & Ji, S. (2023). Artificial intelligence for science in quantum, atomistic, and continuum systems. *arXiv preprint arXiv:2307.08423*.

[7] Chen, W., & Moqvist, S. (2024). Generative AI for Molecular Simulations.

[8] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems, 32*(1), 4-24.

[9] Wiberg, M., & Lauri, J. (2023). Supervised Learning to Evaluate Road Networks for Automated Guided Vehicles.

[10] Jepsen, T. S., Jensen, C. S., & Nielsen, T. D. (2020). Relational fusion networks: Graph convolutional networks for road networks. *IEEE Transactions on Intelligent Transportation Systems*, 23(1), 418-429.

[11] Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

[12] Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[13] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017, July). Neural message passing for quantum chemistry. In *International conference on machine learning* (pp. 1263-1272). PMLR.

[14] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks.

[15] Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems, 30*.

[16] You, J., Ying, Z., & Leskovec, J. (2020). Design space for graph neural networks. *Advances in Neural Information Processing Systems, 33*, 17009-17021.

[17] Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., & Welling, M. (2018). Modeling relational data with graph convolutional networks. *In The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15* (pp. 593-607). Springer International Publishing.

[18] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

[19] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4700-4708).

[20] Li, G., Muller, M., Thabet, A., & Ghanem, B. (2019). DeepGCNs: Can GCNs go as deep as CNNs?. In *Proceedings of the IEEE/CVF international conference on computer vision (pp. 9267-9276)*.

[21] PyTorch Geometric. (2023). Heterogeneous Graphs. Retrieved from `https://pytorch-geometric.readthedocs.io/en/stable/notes/heterogeneous.html`

[22] Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2018). How powerful are graph neural networks?. *arXiv preprint arXiv:1810.00826*.

[23] Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., & Grohe, M. (2019, July). Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence (Vol. 33, No. 01, pp. 4602-4609)*.

[24] Scikit-learn: Cross-validation: evaluating estimator performance. Available at: `https://scikit-learn.org/stable/modules/cross_validation.html`

[25] Scikit-learn: Leave One Out. Available at: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeaveOneOut.html`

[26] Sasaki, Y. (2007). The truth of the F-measure. *Teach Tutor Mater*, 1(5), 1-5.

[27] Myers, J. L., Well, A. D., & Lorch Jr, R. F. (2013). *Research design and statistical analysis*. Routledge.

[28] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *arXiv preprint arXiv:1502.01852*.

[29] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.

[30] Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

[31] Chung, J., Gulcehre, C., Cho, K., Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY