CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# Traffic Prediction in Automated Guided Vehiclular Systems using Graph Neural Networks

## Using Relational Graph Neural Networks to Model Congestion in AGV Systems

Master's thesis in Computer science and engineering

SHIVNESHWAR VELAYUTHAM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

# Traffic Prediction in Automated Guided Vehiclular Systems using Graph Neural Networks

Using Relational Graph Neural Networks to Model Congestion in AGV Systems

SHIVNESHWAR VELAYUTHAM

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Traffic Prediction in AGV Systems using Graph Neural Networks
Using Relational Graph Neural Networks to Model Congestion in AGV Systems
SHIVNESHWAR VELAYUTHAM



Supervisor: Elad Schiller, Department of Computer Science and Engineering
Advisor: Rasmus Åkerlund, Kollmorgen Automation AB
Examiner: Elad Schiller, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Traffic Prediction in AGV Systems using Graph Neural Networks
Using Relational Graph Neural Networks to Model Congestion in AGV Systems
SHIVNESHWAR VELAYUTHAM
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

# Abstract

Automated Guided Vehicle (AGV) systems play a critical role in modern warehouses by automating the movement of goods along set paths. This thesis investigates how to predict traffic and congestion within these systems, focusing on estimating wait times at various points in the network. We apply Graph Neural Networks (GNNs), specifically Relational Graph Convolutional Networks (RGCNs) and Relational Graph Attention Networks (RGATs), to forecast traffic conditions.

Our approach leverages a unique dataset of 50 different warehouse layouts, incorporating detailed vehicle movement simulations and traffic rules defined by fleet managers. This dataset allows us to model complex interactions and constraints affecting vehicle flow in indoor warehouses.

The results demonstrate that RGCNs effectively predict and classify wait times, achieving an F1 score of 0.94 with a quick inference time of 0.01 seconds. These findings enhance the planning and management of AGV systems by providing accurate predictions of traffic conditions, facilitating better design adjustments and reducing delays.

Keywords: Traffic, Graph Neural Networks, Automated Guided Vehicle, Relational Graph Convolutional Networks, Relational Graph Attention Networks, F1 score, dual graphs, multi-relational graphs, blocking, congestion.

# Acknowledgements

I would like to express my deepest gratitude to all those who have supported and guided me throughout the course of this thesis.

First and foremost, I would like to thank Kollmorgen for providing the opportunity to conduct my research within their esteemed organization. The resources, working space, and support provided by the company were instrumental to the success of this work. I am particularly grateful to Rasmus Åkerlund, whose guidance, insights, and encouragement were invaluable throughout the entire process. I would also like to extend my thanks to the entire team at Kollmorgen, whose collaboration and assistance contributed to the completion of this thesis.

I owe special thanks to my academic supervisor, Elad Michael Schiller, whose constant support and feedback were crucial to the development and completion of this thesis.

Shivneshwar Velayutham, Gothenburg, 2024-09-11

# Contents

# List of Figures

# List of Tables

# Glossary

- **AGV** - Automated Guided Vehicle
- **MLP** - Multi-Layer Perceptron
- **CNN** - Convolutional Neural Network
- **RNN** - Recurrent Neural Network
- **GNN** - Graph Neural Network
- **GCN** - Graph Convolutional Network
- **GAT** - Graph Attention Network
- **RFN** - Relational Fusion Network
- **RGCN** - Relational Graph Convolutional Network
- **RGAT** - Relational Graph Attention Network

# 1

# Introduction

Automated Guided Vehicle (AGV) systems [1] have emerged as indispensable components of modern warehouses, streamlining the transportation of goods with efficiency and precision. These systems, governed by a fleet of mobile robots, navigate predefined routes within warehouse environments, executing tasks with minimal human intervention. Figure 1.1 shows the type of mobile robots used in AGV systems. Central to the functionality of AGV systems is the intricate network of roads, which serves as the backbone for vehicle movement. This road network is called the AGV graph and comprises nodes and edges, where nodes correspond to specific locations or waypoints within the warehouse, and edges represent the paths or routes that AGVs can traverse between these nodes. This work aims to improve AGV systems, specifically focusing on predicting traffic and congestion.

Figure 1.1: Automated Guided Vehicles

## 1.1 Problem Definition

This work focuses on predicting traffic in AGV systems, a crucial element for improving the performance of these systems. In the context of AGV systems, the wait time indicates the average duration an AGV is obstructed from traversing an edge due

to traffic or congestion. The traffic distribution refers to the wait time experienced by AGVs at each edge of the warehouse. These metrics help quantify the system's traffic and behavior.

The core of this investigation revolves around a classification problem. The input dataset includes various parameters, such as AGV graph topology, itinerary specifications, traffic regulations set by fleet managers, and other relevant factors. The goal is to categorize the severity of wait times at each edge within the AGV system. These wait times are divided into distinct classes representing different levels of delay, allowing for a nuanced understanding of traffic dynamics.

It is essential to emphasize that this problem is a supervised learning task. The training dataset consists of historical instances of AGV graphs, along with their corresponding itinerary specifications. Additionally, the traffic distribution is derived from simulations. By leveraging this rich dataset, we aim to develop predictive models capable of identifying patterns and accurately extrapolating future traffic distributions.

The choice of supervised learning for this problem justified due to several key factors such as:

- **Labeled Data Availability:** The availability of labeled data, including historical AGV graphs, itinerary specifications, and traffic distributions makes supervised learning a natural fit, as it relies on such labeled examples for training.
- **Clear Prediction Objective:** The goal is to categorize the severity of wait times at different edges in the AGV system, which involves predicting known target classes (levels of delay). Supervised learning is highly effective when there is a specific prediction outcome, making it a suitable choice for this task.
- **Predictive Accuracy:** Supervised learning allows models to be trained to minimize prediction errors by leveraging labeled examples.

## 1.2 Research Questions

The main research question investigates whether Graph Neural Networks (GNNs) can accelerate the planning process of AGV system design by accurately predicting the traffic distribution of the designed AGV network. Key aspects of this research question include:

- What is the best way to represent the various types of data related to AGV systems to enable effective learning by GNNs?
- Can GNNs be used to predict the traffic distribution of AGV systems, thereby accelerating the planning process, and how accurate are these predictions?

## 1.3 Related Work

The most famous example of using GNNs in predicting traffic conditions is the case of Google Maps [2]. The work did considerably well in predicting the estimated time to arrive in real world transportation networks.

A substantial body of research utilizing GNNs for traffic prediction, with notable studies including [3]–[6]. Specifically, the work by Cui et al. [3] and by Yu et al. [4] employ Graph Convolutional Networks (GCNs), while the study by Guo et al. [5] and Zheng et al. [6] utilize Graph Attention Networks (GATs). Our study employs both GCNs and GATs, drawing on insights from these foundational works. However, there is a critical distinction in our research focus: whereas the aforementioned studies address traffic prediction in outdoor road networks, our work concentrates on predicting traffic within indoor road networks, such as those traversed by AGVs in warehouses. This shift to indoor environments introduces unique characteristics and data considerations that differentiate our work from existing studies on outdoor traffic networks. Outdoor road networks are relatively fixed and not prone to changes, whereas indoor road networks are more dynamic and subject to modifications, as they are often designed and altered by AGV engineers.

Jepsen et al. [7] proposed a Relational Fusion Network (RFN) to predict traffic on real world roads and our study draws significant inspiration from their innovative work. The RFNs approach to handling road networks through the use of dual graphs has been particularly influential. By leveraging dual graphs, the RFN effectively captures both the spatial and relational aspects of road networks, addressing the limitations of traditional GCNs in this context. In our study, we build upon this dual graph concept to further enhance the representation and analysis of complex networks. Specifically, we extend the idea by incorporating additional features and relationships, allowing our models to better generalize across different types of networks.

While existing applications employ GNNs to predict traffic conditions, to the best of our knowledge, there is no literature addressing the specific use case outlined in this work. The distinctive aspect of this work lies in its consideration of not only the graph structure representing the AGV network but also the traffic rules enforced by the fleet manager coordinating the movement of the AGVs within said network. Consequently, a key challenge arises in determining how to effectively integrate the traffic rules within the framework of GNNs. This aspect represents an open question, highlighting this research endeavor's unique and unexplored dimension.

Recently, Lauri and Wiberg [8] trained a GNN model on 50 variants of a T-intersection, i.e., three-way junctions. They predict the overall performance regarding the percentage to which the throughput requirements were met. In detail, they aimed to predict a single number, the average ratio of order fulfillment. The order fulfillment ratio equals the predicted throughput divided by the required throughput. Thus, Lauri and Wiberg's work framed the problem as an end-to-end graph-level machine-learning task. Note that the problem formulated by Lauri and Wiberg tries to characterize the system in one single number and thus can possibly lose relevant in-

formation. We are interested in calculating the entire traffic distribution to facilitate a better-informed decision by the user. So, we divide the problem into two parts. The first part estimates the traffic distribution, and the second part is to derive key performance indicators from this traffic distribution.

## 1.4 Our Approach

We focus on the first problem for this study, where we will be predicting the traffic distribution or, more precisely, the wait time for each edge in the graph. This approach has several benefits.

1. The target of the dataset has a richer representation that can more precisely discriminate between different simulation results. This can make it easier for the network to learn the relevant patterns, i.e., how the design of the graph affects the simulation results.
2. The network output is more meaningful to humans since it would be easier to understand what is being predicted and how that translates to the actual AGV system behavior.
3. The prediction will be more actionable since it will pinpoint where the traffic situation is unfavorable in the system and, thus, where the layout design might need to be adjusted. This would help the application engineer find the issues faster and more efficiently address design flaws.

## 1.5 Our Contribution

We address the problem of predicting traffic distribution and congestion propagation within AGV systems. This problem is important because traditional simulation-based methods for calculating traffic distribution are computationally intensive and time-consuming. By employing Machine Learning, particularly GNNs, the study aims to accelerate this process, thereby reducing costs and improving the efficiency of designing and managing AGV systems. Accurate prediction of traffic distribution allows for better-informed decisions, optimizing the flow of goods, minimizing delays, and enhancing the overall productivity of warehouse operations.

We present a novel solution using GNNs to predict and classify traffic distribution in AGV systems. Specifically, we utilize Relational Graph Convolutional Networks (RGCNs) and Relational Graph Attention Networks (RGATs) to classify wait times.

In the context of GNNs for traffic prediction, we are the first to incorporate traffic rules as part of the input processed by the model. The traffic rules limit what movement is allowed or not for each vehicle given the position of the other vehicles and is a key aspect of the AGV system design, and may have a critical impact on its performance. By incorporating these traffic rules, represented as blocking relationships in our multi-relational graph model, we more effectively capture the complex interactions and constraints within the AGV system. This approach advances the state

of the art by providing a mechanism that enables GNNs to model the relationship between the layout data, including the traffic rules, and the resulting distribution of congestion, significantly enhancing such machine learning models applicability to the AGV domain.

The specialized dataset [9] used in our study addresses the unique characteristics of indoor AGV traffic. The dataset contains a unique collection of 50 T-intersection layout variants, each with 25 to 120 nodes and a node-to-edge ratio of approximately one. Figure 1.2 shows an example T-intersection layout. Each layout variant is simulated for 30 minutes to collect detailed vehicle movement statistics. The size of the dataset and simulation time is somewhat limited given that this study is a starting point and a precursor to further studies in this subject. The dataset used in our study is unique because it includes graphical data, traffic simulations, and traffic rules generated by the fleet manager. These traffic rules, represented as blocking relationships, specify when an edge cannot be traversed due to the presence of a vehicle on another edge.

This study investigates the effectiveness of MLPs, RGCNs, and RGATs for predicting traffic distribution and congestion in AGV systems. MLPs were initially selected to assess whether simpler neural networks could solve our problem. However, due to the complex and multi-relational nature of AGV system graphs, which include both spatial connections and blocking rules generated by fleet managers, we opted for RGCNs and RGATs. These GNN architectures are well-suited for handling multi-relational graphs, allowing them to capture intricate dependencies and interactions within the AGV network.

The results demonstrate that GNNs, specifically RGCNs, effectively predict and classify wait time severity in AGV systems with high accuracy as described in Table 6.4 where RGCN scored an average F1 score across all classes of 0.94. These metrics indicate robust classification capabilities, allowing GNNs to assess congestion levels at each segment of AGV networks accurately and quickly with an inference time of 0.01 seconds. The studied methods, which use GNNs, enable proactive decision-making in AGV network planning and traffic management strategies and help reduce costs associated with delays.

Our results can assist AGV system designers and warehouse managers in improving the planning and design of AGV networks. By accurately predicting traffic and congestion levels within AGV systems, our approach provides these professionals with valuable information to identify potential bottlenecks and optimize the layout of the AGV network before it is implemented. This insight helps reduce delays and enhance the efficiency of goods movement.

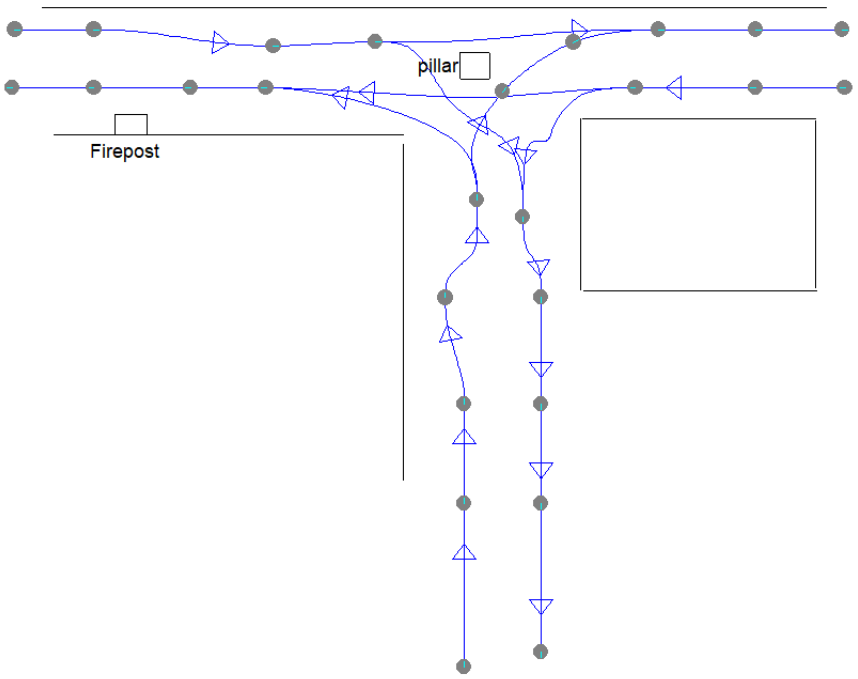


Figure 1.2: T-intersection layout

# 2

# Background

In the background section of this paper, we delve into the foundational principles of AGV Systems and GNNs.

## 2.1 AGV Systems

An Automated Guided Vehicle (AGV) system [1] includes a fleet of mobile robots that automatically transport goods in a warehouse. These vehicles can move on the warehouse's floor according to given specifications or paths. Using these specifications, a road network is designed. The network abstracts away the low-level mobility details required for traversing the warehouse safely. This is done by using fixed virtual roads as the edges of the AGV graph that represent the road network.

For our particular context for this study, these AGV systems have a server responsible for centralized fleet management. The fleet manager assigns vehicles to incoming orders and dispatches the vehicles along the shortest path to their destination. The fleet manager also has a set of traffic rules to prevent collisions and maintain traffic flow.

## 2.2 Traffic in AGV Systems

If two vehicles try to occupy the same place, the fleet manager avoids a collision by making them slow down or stop. This gives rise to congestion, which can cascade and spread throughout the road network, causing delays as vehicles wait for each other. Assuming a constant rate of delivery assignments and vehicles, an AGV system will converge to a steady-state or state of equilibrium around which the traffic will oscillate. For this study, we will be focusing on predicting the steady-state traffic distribution. The steady-state traffic distribution is defined as the average wait time for vehicles at each edge due to traffic or congestion.

## 2.3 Graphs

Graphs [10] are fundamental mathematical structures that serve as a versatile tool for modeling pairwise relationships between objects. They are ubiquitous in various

fields such as computer science, biology, social sciences, and engineering, where they help in understanding and solving complex problems involving networked systems.

### 2.3.1 Basic Definitions

A graph $G$ is formally defined as an ordered pair $G = (V, E)$, where:

- $V$ is a set of vertices (also known as nodes or points). These vertices represent the entities in the system being modeled.
- $E$ is a set of edges (also known as links or arcs). Each edge is a pair of vertices and represents the relationship between these vertices.

An edge going from node $u \in V$ to node $v \in V$ is denoted as $(u, v) \in E$. In many cases, we will be concerned only with simple graphs, where there is at most one edge between each pair of nodes, no edges between a node and itself, and where the edges are all undirected. Formally, this means $(u, v) \in E \iff (v, u) \in E$.

### 2.3.2 Adjacency Matrix Representation

A convenient way to represent graphs is through an adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$. To represent a graph with an adjacency matrix, we order the nodes in the graph so that each node indexes a particular row and column in the matrix. The presence of edges is then represented as entries in this matrix:

$$A[u, v] = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases}$$

For undirected graphs, the adjacency matrix $A$ is symmetric, meaning $A[u, v] = A[v, u]$. For directed graphs, the matrix is not necessarily symmetric, reflecting the direction of the edges.

### 2.3.3 Weighted Graphs

In some cases, graphs can have weighted edges, where the entries in the adjacency matrix are arbitrary real values rather than binary indicators $\{0, 1\}$. In a weighted graph, the weight of an edge $(u, v)$ is represented by a real number $w(u, v)$, and the adjacency matrix $A$ is defined as:

$$A[u, v] = \begin{cases} w(u, v) & \text{if } (u, v) \in E \\ 0 & \text{otherwise} \end{cases}$$

For example, in a protein-protein interaction graph, a weighted edge might indicate the strength of the association between two proteins, with higher values representing stronger associations. The adjacency matrix in this context provides a compact and efficient way to store and manipulate the information about the graph structure and the relationships between nodes.

### 2.3.4 Graph Properties

Graphs exhibit various properties that can be analyzed to understand the structure and dynamics of the system they represent.

#### 2.3.4.1 Degree of a Vertex

The degree of a vertex in an undirected graph is the number of edges incident to it. Formally, the degree of vertex $v$, denoted $\deg(v)$, is the number of edges $e = \{v, u\}$ for all $u \in V$. In directed graphs, we distinguish between the in-degree (number of incoming edges) and out-degree (number of outgoing edges) of a vertex. The in-degree of vertex $v$ is denoted by $\deg^-(v)$, and the out-degree by $\deg^+(v)$.

#### 2.3.4.2 Paths and Cycles

A path in a graph is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that each adjacent pair $(v_i, v_{i+1})$ is connected by an edge. If the graph is directed, the direction of the edges must be respected. A cycle is a path that starts and ends at the same vertex, with all other vertices distinct. Cycles in undirected graphs do not consider the direction, while directed cycles do.

#### 2.3.4.3 Connectedness

A graph is connected if there is a path between any pair of vertices. For directed graphs, we use the term strongly connected to indicate that for every pair of vertices $u$ and $v$, there is a directed path from $u$ to $v$ and a directed path from $v$ to $u$. A graph that is not connected is called disconnected, and it consists of multiple connected components.

## 2.4 Multi-Relational Graphs

Multi-relational graphs, also known as multi-edge graphs or heterogeneous graphs, are an extension of simple graphs that allow multiple types of relationships (edges) between the same set of nodes. These graphs are particularly useful in representing complex systems where entities are connected by different types of interactions or relationships.

### 2.4.1 Basic Definitions

A multi-relational graph $G$ is defined as a triplet $G = (V, E, R)$, where:

- $V$ is a set of vertices (nodes).
- $E$ is a set of edges, where each edge $e \in E$ is a triplet $(u, \tau, v)$ with $u, v \in V$ and $\tau \in R$.
- $R$ is a set of relation types. Each relation type $\tau$ represents a different kind of interaction between the nodes.

In a multi-relational graph, an edge $(u, \tau, v)$ denotes that there is a relationship of type $\tau$ from node $u$ to node $v$.

### 2.4.2 Adjacency Tensor Representation

To represent a multi-relational graph, we use an adjacency tensor $\mathcal{A} \in \mathbb{R}^{|V| \times |R| \times |V|}$. This tensor extends the concept of an adjacency matrix to multiple dimensions, capturing the different types of relations. The entries in the adjacency tensor are defined as:

$$\mathcal{A}[u, \tau, v] = \begin{cases} 1 & \text{if there is an edge of type } \tau \text{ from } u \text{ to } v \\ 0 & \text{otherwise} \end{cases}$$

For weighted multi-relational graphs, the entries in the adjacency tensor can be real-valued, representing the strength or weight of the relationship:

$$\mathcal{A}[u, \tau, v] = \begin{cases} w(u, \tau, v) & \text{if there is an edge of type } \tau \text{ from } u \text{ to } v \\ 0 & \text{otherwise} \end{cases}$$

### 2.4.3 Examples and Applications

Multi-relational graphs are used in various domains to model complex relationships between entities:

- **Knowledge Graphs:** In knowledge representation, entities (vertices) are connected by different types of relationships (edges) such as "is a", "has part", "is located in", etc. For example, in a knowledge graph, an edge $(Einstein, works\_at, Princeton)$ represents the fact that "Einstein works at Princeton."
- **Drug-Drug Interaction Graphs:** In pharmacology, drugs (vertices) can have multiple types of interactions (edges) such as causing different side effects when taken together. For instance, an edge $(DrugA, side\_effect1, DrugB)$ indicates that taking DrugA and DrugB together can cause a specific side effect.
- **Social Networks:** In social network analysis, individuals (vertices) can be connected by multiple types of relationships such as friendship, collaboration, or familial ties. For instance, an edge $(Alice, friends, Bob)$ represents a friendship, while $(Alice, colleagues, Bob)$ represents a professional relationship.
- **Biological Networks:** In biology, genes or proteins (vertices) can interact through various types of interactions such as activation, inhibition, or binding. For example, an edge $(GeneA, activates, GeneB)$ denotes that GeneA activates GeneB.

## 2.5 Multi-Layer Perceptrons

A Multi-Layer Perceptron (MLP) [11] is a class of feedforward artificial neural networks (ANN). An MLP consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function [12]. MLP utilizes a supervised learning technique called backpropagation [13] for training.

Formally, let us denote the input to the network by $\mathbf{x} \in \mathbb{R}^d$, where $d$ is the number of features. The MLP is structured as follows:

- **Input Layer:** The input layer consists of $d$ neurons, each representing one feature of the input vector $\mathbf{x}$.
- **Hidden Layers:** Each hidden layer consists of $n_i$ neurons, where $i$ denotes the $i$-th hidden layer. The neurons in the hidden layers are connected to the neurons in the previous layer with associated weights.
- **Output Layer:** The output layer consists of neurons that represent the predicted output. For a classification task, this typically includes one neuron per class.

The output of each neuron in the hidden and output layers is computed as follows:

$$z_j^{(l)} = \sum_{i=1}^{n_{l-1}} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}, \tag{2.1}$$

$$a_j^{(l)} = \sigma(z_j^{(l)}), \tag{2.2}$$

where $z_j^{(l)}$ is the weighted sum of the inputs to the $j$-th neuron in layer $l$, $w_{ij}^{(l)}$ are the weights, $a_i^{(l-1)}$ are the activations from the previous layer, $b_j^{(l)}$ is the bias term, and $\sigma$ is the activation function. Common activation functions include the sigmoid function, hyperbolic tangent function (tanh), and rectified linear unit (ReLU).

The objective of training an MLP is to minimize the loss function, which measures the discrepancy between the predicted output and the true output. For example, in a classification task, the cross-entropy loss is often used:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log(\hat{y}_{ic}), \tag{2.3}$$

where $N$ is the number of samples, $C$ is the number of classes, $y_{ic}$ is a binary indicator (0 or 1) if class label $c$ is the correct classification for sample $i$, and $\hat{y}_{ic}$ is the predicted probability of sample $i$ being in class $c$.

Training the MLP involves optimizing the weights and biases using gradient-based optimization algorithms such as stochastic gradient descent (SGD) or its variants like Adam [14]. The gradients of the loss function with respect to the weights are computed using the backpropagation algorithm.

In summary, MLPs are a fundamental building block in the field of deep learning, providing the foundation for more complex architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

## 2.6 Node Classification in Graphs

Node classification is a fundamental task in graph-based learning where the goal is to predict the labels of nodes based on the graph structure and node features. This task is crucial in various applications, including social network analysis, knowledge graph completion, and biological network analysis.

### 2.6.1 Problem Definition

Given a graph $G = (V, E)$ where $V$ is the set of nodes and $E$ is the set of edges, and a subset of nodes $V_L \subset V$ with known labels, the objective of node classification is to predict the labels for the remaining nodes $V_U = V \setminus V_L$. Each node $v \in V$ may also have associated feature vectors $x_v$.

Formally, the problem can be defined as follows:

- **Input:** A graph $G = (V, E)$, feature vectors $\{x_v\}_{v \in V}$, and labels $\{y_v\}_{v \in V_L}$ for a subset of nodes $V_L$.
- **Output:** Predicted labels $\{\hat{y}_v\}_{v \in V_U}$ for the unlabeled nodes $V_U$.

### 2.6.2 Unique Characteristics of Node Classification

At first glance, node classification appears to be a straightforward variation of standard supervised classification. However, there are important differences. The most critical difference is that the nodes in a graph are not independent and identically distributed (i.i.d.). In traditional supervised machine learning, we assume that each datapoint is statistically independent from all other datapoints and that they are identically distributed. These assumptions do not hold for node classification, where nodes are part of an interconnected structure.

### 2.6.3 Exploiting Graph Structure

The key insight behind many successful node classification approaches is to leverage the connections between nodes. Concepts such as homophily and structural equivalence play a significant role:

- **Homophily:** This is the tendency for nodes to share attributes with their neighbors. For example, people tend to form friendships with others who share the same interests or demographics. Based on homophily, machine learning models aim to assign similar labels to neighboring nodes in a graph.
- **Structural Equivalence:** This concept suggests that nodes with similar local neighborhood structures will have similar labels. For instance, in social networks, individuals with similar roles or functions might have similar connections.
- **Heterophily:** In some cases, nodes may preferentially connect to nodes with different labels. An example is gender, which exhibits heterophily in many

social networks.

### 2.6.4 Supervised and Semi-Supervised Learning

Node classification often blurs the boundaries between supervised and semi-supervised learning:

- **Supervised Learning:** In standard supervised learning, we have labeled training data and unseen test data. However, in node classification, we usually have access to the entire graph, including all the unlabeled test nodes. This availability allows us to use the structure and features of test nodes during training.
- **Semi-Supervised Learning:** Due to the availability of the full graph during training, node classification is often referred to as semi-supervised learning. In semi-supervised learning, models combine labeled and unlabeled data during training. Despite this, traditional semi-supervised learning still assumes i.i.d. data, which does not hold for node classification. Thus, the term is used with the understanding that it applies differently in graph contexts.

## 2.7 The Graph Neural Network Model

In this section, we introduce the Graph Neural Network (GNN) [15] framework, a encoder model that generates node representations dependent on the graph's structure and node features.

The primary challenge in developing deep learning models for graph-structured data lies in the fact that conventional deep learning architectures, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), are not directly applicable to graph data. CNNs are designed for grid-structured inputs like images, while RNNs are suited for sequence data like text. To address this, GNNs define a new deep learning architecture suitable for graphs.

### 2.7.1 Permutation Invariance and Equivariance

A critical requirement for neural networks operating on graphs is permutation invariance (or equivariance). This property ensures that the network's output is not dependent on the arbitrary ordering of nodes. Mathematically, a function $f$ that takes an adjacency matrix $A$ as input should satisfy:

$$f(PAP^\top) = f(A) \quad \text{(Permutation Invariance)} \tag{2.4}$$

$$f(PAP^\top) = Pf(A) \quad \text{(Permutation Equivariance)} \tag{2.5}$$

where $P$ is a permutation matrix. Permutation invariance means the function's output is unaffected by the permutation of the adjacency matrix's rows and columns, whereas permutation equivariance means the output is consistently permuted in line with the permutation of the input.

### 2.7.2 Neural Message Passing

The core idea of GNNs is neural message passing, where nodes exchange and update vector messages through neural networks [16]. In each iteration of message passing, a hidden embedding $h_u^{(k)}$ for each node $u \in V$ is updated based on the information aggregated from its neighborhood $N(u)$:

$$h_u^{(k+1)} = \text{UPDATE}^{(k)}\left(h_u^{(k)}, \text{AGGREGATE}^{(k)}\left(\{h_v^{(k)}, \forall v \in N(u)\}\right)\right) \tag{2.6}$$

The initial embeddings ($h_u^{(0)}$) are set to the input node features ($x_u$). After $K$ iterations, the final node embeddings $z_u$ are given by $z_u = h_u^{(K)}$. This framework ensures permutation equivariance by design, as the AGGREGATE function operates on sets.

### 2.7.3 Node Features

Unlike shallow embedding methods, the GNN framework requires node features $x_u$ for each node $u \in V$. In graphs lacking inherent node features, features can be derived from node statistics or one-hot indicator vectors. However, the use of one-hot vectors renders the model transductive, limiting its ability to generalize to unseen nodes.

### 2.7.4 Motivations and Intuitions

The intuition behind GNNs is that each iteration aggregates information from a node's local neighborhood, expanding the context with each iteration. After $k$ iterations, a node embedding encodes information from its $k$-hop neighborhood, capturing both structural and feature-based information. This local aggregation is akin to convolutional operations in CNNs but applied to graph neighborhoods rather than spatial patches.

### 2.7.5 The Basic GNN Model

The basic GNN message passing model can be described as:

$$h_u^{(k)} = \sigma\left(W_{\text{self}}^{(k)} h_u^{(k-1)} + \sum_{v \in N(u)} W_{\text{neigh}}^{(k)} h_v^{(k-1)}\right) \tag{2.7}$$

where $W_{\text{self}}^{(k)}$ and $W_{\text{neigh}}^{(k)}$ are trainable weight matrices, and $\sigma$ denotes an elementwise non-linearity (e.g., ReLU). This framework can be simplified by adding self-loops to the input graph, merging the update and aggregation steps:

$$h_u^{(k)} = \text{AGGREGATE}\left(\{h_v^{(k-1)}, \forall v \in N(u) \cup \{u\}\}\right) \tag{2.8}$$

This self-loop approach can reduce overfitting but may limit expressivity.

### 2.7.6 Generalized Neighborhood Aggregation

Neighborhood aggregation can be further enhanced through normalization techniques, such as average or symmetric normalization, to stabilize the aggregation process and account for varying node degrees. Additionally, set pooling approaches based on permutation-invariant neural networks [17] or Janossy pooling [18] can improve aggregation by incorporating more sophisticated permutation-sensitive functions.

### 2.7.7 Neighborhood Attention

Applying attention mechanisms [19] to neighborhood aggregation allows for weighted sums of neighbor embeddings, where attention weights indicate the importance of each neighbor. This approach, exemplified by Graph Attention Networks (GATs) [20], computes attention weights as:

$$\alpha_{u,v} = \frac{\exp\left(\mathbf{a}^\top [W\mathbf{h}_u \parallel W\mathbf{h}_v]\right)}{\sum_{v' \in N(u)} \exp\left(\mathbf{a}^\top [W\mathbf{h}_u \parallel W\mathbf{h}_{v'}]\right)} \tag{2.9}$$

where $\mathbf{a}$ is a trainable attention vector and $\parallel$ denotes concatenation. This attention mechanism can be extended with multiple attention heads, akin to the transformer architecture [21], enhancing the model's capacity to capture diverse aspects of the neighborhood information.

## 2.8 Multi-Relational Graph Neural Networks

In this section, we delve into the realm of multi-relational graph neural networks (GNNs), which are tailored to handle graphs with diverse types of relations or edges. We begin by discussing Relational Graph Convolutional Networks (RGCN) as a foundational approach, followed by a detailed exploration of Relational Graph Attention Networks (RGAT), which represents a more recent advancement in the field.

### 2.8.1 Relational Graph Convolutional Networks (RGCN)

Relational Graph Convolutional Networks (RGCN) were proposed as an early approach to address the challenges posed by multi-relational graphs [22]. RGCN extends the basic graph convolutional network (GCN) architecture to accommodate multiple relation types.

#### 2.8.1.1 Multi-Relational Aggregation

In RGCN, the aggregation function is augmented to accommodate multiple relation types by specifying a separate transformation matrix per relation type:

$$m_{N(u)} = \sum_{\tau \in R} \sum_{v \in N_\tau(u)} \mathbf{W}^\tau \cdot h_v \tag{2.10}$$

where $\mathbf{W}^\tau$ are learnable weight matrices specific to relation type $\tau$, and $N_\tau(u)$ represents the neighbors of $u$ connected by relation type $\tau$.

#### 2.8.1.2 Parameter Sharing

To mitigate the increase in parameters due to separate transformation matrices for each relation type, RGCN employs parameter sharing schemes. One such approach is basis sharing, where all relation matrices are defined as linear combinations of basis matrices.

#### 2.8.1.3 Extensions and Variations

RGCN architecture can be extended in various ways, including attention mechanisms and feature concatenation strategies. These extensions enable RGCN to effectively model complex multi-relational graphs, making it a versatile framework for various applications.

### 2.8.2 Relational Graph Attention Networks (RGAT)

Relational Graph Attention Networks (RGAT) [23] represent an extension of the Graph Attention Network (GAT) architecture [20], tailored for multi-relational graphs. In RGAT, attention mechanisms are employed to adaptively aggregate information from neighbor nodes based on their importance.

#### 2.8.2.1 Attention Mechanism

Given a node $u$ and its neighbors $v \in N(u)$ connected by relation type $\tau$, the attention mechanism computes attention coefficients $e^\tau_{u,v}$ for each neighbor $v$ as follows:

$$e^\tau_{u,v} = \text{LeakyReLU}(\mathbf{a}^\tau \cdot [\mathbf{W}^\tau \cdot h_u || \mathbf{W}^\tau \cdot h_v]) \tag{2.11}$$

where $h_u$ and $h_v$ are the embeddings of nodes $u$ and $v$ respectively, $\mathbf{W}^\tau$ are learnable weight matrices specific to relation type $\tau$, $[\cdot||\cdot]$ denotes concatenation, and $\mathbf{a}^\tau$ are attention parameters to be learned.

#### 2.8.2.2 Aggregation with Attention

The aggregated message for node $u$ considering attention is computed as:

$$m_{N(u)} = \sum_{\tau \in R} \sum_{v \in N_\tau(u)} \text{softmax}(e^\tau_{u,v}) \cdot h_v \tag{2.12}$$

where $N_\tau(u)$ represents the neighbors of $u$ connected by relation type $\tau$. The softmax function is applied to normalize attention coefficients, ensuring that the aggregation is weighted according to the relevance of each neighbor.

#### 2.8.2.3 Parameter Sharing and Extensions

Similar to RGCNs, parameter sharing schemes and extensions can be applied to RGAT to improve scalability and generalization. For instance, basis sharing approaches can be utilized to reduce the number of relation-specific parameters, enhancing model efficiency.

RGAT offers a flexible and effective framework for modeling multi-relational graphs, leveraging attention mechanisms to adaptively capture relational dependencies. By dynamically focusing on relevant neighbors, RGAT enhances the expressive power of graph neural networks in handling complex graph structures.

# 3

# Dataset

A dataset of 50 T-intersection layout variants [9], with 25 to 120 nodes and a node-to-edge ratio of approximately one, is simulated for a total of 30 minutes or 1800 seconds to collect vehicle movement statistics. For each edge, the total time that a vehicle needs to drive the edge but cannot due to being blocked by another vehicle, known as the wait time, will be recorded from the simulation. This recorded wait time is then categorized into different severity levels to create discrete classes. The neural network's task is to predict the severity class of the wait time for each edge. For each layout variant, the goal is to classify the severity of the wait time at each edge using a GNN, thereby framing the problem as an edge-level classification task. Figure 3.1 and Figure 3.2 shows an example T-intersection and how congestion is visualized in the dataset.

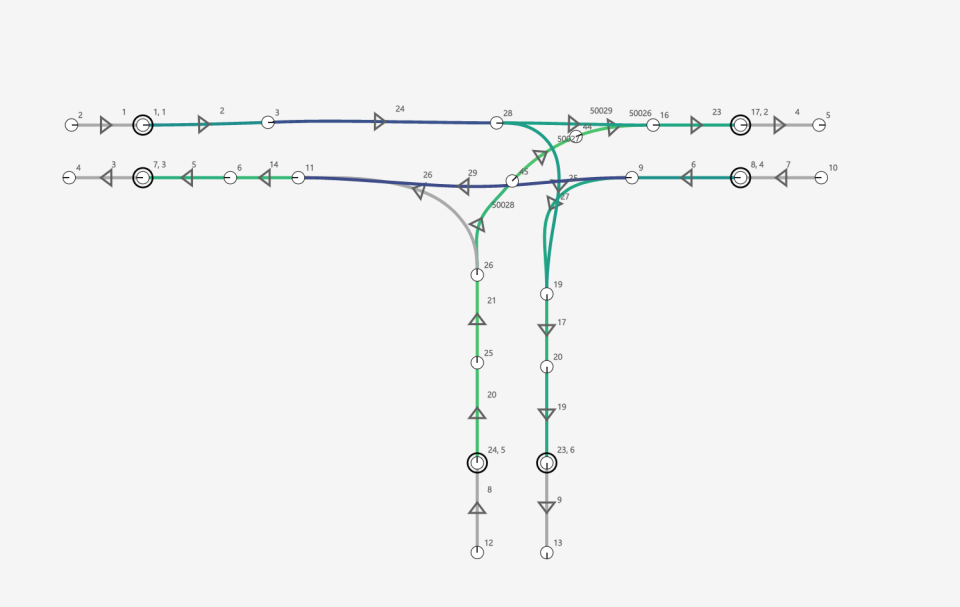

Figure 3.1: T-intersection data in dataset

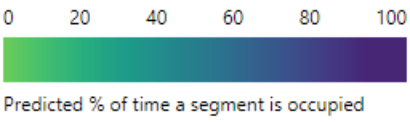


Figure 3.2: Color bar of Figure 3.1

## 3.1 Class Distribution

The recorded wait times are categorized into discrete severity levels to facilitate edge-level classification. The predefined categories based on wait time durations are as follows:

1. Class 1: Wait time less than 50 seconds (0 seconds <= wait time < 50 seconds)
2. Class 2: Wait time less than 400 seconds (50 seconds <= wait time < 400 seconds).
3. Class 3: Wait time less than 800 seconds (400 seconds <= wait time < 800 seconds).
4. Class 4: Wait time 800 seconds or more, up to the duration of the entire 30-minute simulation (wait time >= 800 seconds and <= 30 minutes).

| **Class** | **Number of Instances** | **Percentage (%)** |
|---|---|---|
| Class 1 | 1443 | 82.46 |
| Class 2 | 169 | 9.66 |
| Class 3 | 80 | 4.57 |
| Class 4 | 58 | 3.31 |

Table 3.1: Class Imbalance

After categorizing the wait times into these classes, the distribution across the dataset variants reveals an imbalance among the classes as described in Table 3.1. To tackle the class imbalance in the dataset, a weighted loss function is applied during neural network training. This method adjusts the contribution of each class to the overall loss based on its frequency. Classes with fewer instances are given higher weights, prioritizing their importance in the training process. This approach ensures the model learns effectively from all severity classes, regardless of their distribution in the dataset.

## 3.2 Data and Features

The section describes the types of data that are available in our dataset.

### 3.2.1 Graphical data

The graphical data include the nodes and edges present in the graph. It also include edge features such as edge length, the amount of time it takes to travel the edge and the driving speed.

### 3.2.2 Traffic data

The traffic analysis provides two essential edge features as input to the GNN. The first feature, known as “active usage,” measures how frequently an edge is utilized.

The second, termed “initial wait probability,” represents the likelihood of waiting to traverse an edge, without considering the effect of congestion propagation.

It is important to acknowledge that the primary objective of the GNN is to calculate wait times while factoring in the effects of congestion propagation, referred to as steady-state wait time. As congestion permeates the system and stabilizes into a steady-state, the initial wait probability alone may not entirely encapsulate the behavior of the AGV system. This limitation becomes evident when comparing wait times derived from simulations. Therefore, while the initial wait probability provides valuable insights, it is essential to recognize its inherent constraints in fully representing the system's dynamics.

### 3.2.3 Blocking rules

The traffic rules mainly consists of blocking rules generated by an internal software dedicated to AGV systems analysis. These rules, based on the drawn graph, dictate instances where an edge cannot be traversed due to the presence of a vehicle at another edge. These blocking rules are represented as an edge-to-edge relationship.

# 4

# Methods

This section outlines the data representation and models used to address the research questions. We review established methods, with a particular focus on GNNs such as RGCNs and RGATs, which have proven effective in traffic prediction for outdoor road networks. However, the novelty of our approach is the application of these methods to a unique dataset representing AGV systems within indoor warehouse environments. Unlike static outdoor road networks, AGV networks are dynamic and frequently modified, presenting distinct challenges for traffic prediction. Our research introduces an innovative application of RGCNs and RGATs to this specialized dataset, which includes detailed AGV network layouts, simulation results, and traffic rules from fleet managers. By applying these GNN techniques to such a dataset, our study demonstrates how these models can predict traffic of AGV systems in warehouse environments.

## 4.1 Data representation

We describe how the data is represented as multi-relational dual graphs.

### 4.1.1 Dual Graphs

Dual graphs offer a clever solution to a persistent challenge in Graph Neural Networks (GNNs): the inability to perform edge-level classification directly. GNNs traditionally excel at node or graph-level classification but lack a native mechanism for edge-level classification. Therefore, we explain how the literature addresses this challenge by reimagining the graph structure itself.

The concept of dual graphs involves a transformation where edges and nodes switch roles. This transformation generates a new representation of the original graph,

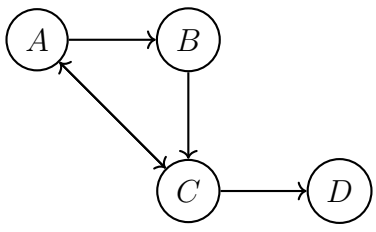


Figure 4.1: An example graph

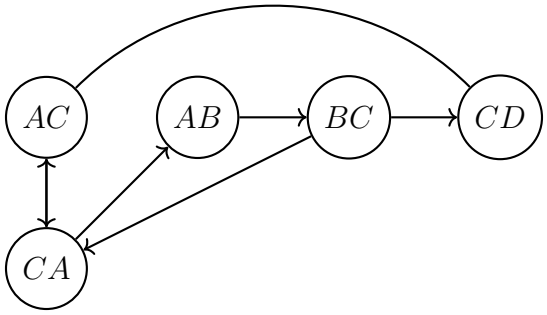


Figure 4.2: The dual graph of Figure 4.1

wherein edges become nodes and nodes become edges. This inversion provides a fresh perspective on the graph, enabling edge-level classification that was previously inaccessible to traditional GNNs.

Jepsen et al. defines the concept of dual graph representation in [7]. Figure 4.1 illustrates a typical graph, while Figure 4.2 showcases its corresponding dual representation. The efficiency of dual graphs has also been demonstrated in prior research, such as the study conducted by Lauri and Wiberg [8]. We leverage dual representation to overcome the limitations of conventional GNNs and study edge-level characteristics and relationships, which are crucial for our edge-level classification task. Next, we will discuss the edge-to-edge relationships that this dual representation allows.

### 4.1.2 Multi-Relational Graph

As we have discussed in Section 3.2.3, we have two edge-to-edge relationships. One indicating real world connection between two edges in a AGV graph in a warehouse and two being a blocking relationship between two edges where one edge can be blocked by the other when an AGV is presently on them. Since we have two types of relationship, we use a multi-relational graph as the best representation of the data so that the machine learning model would be able to learn effectively.

## 4.2 Models

We began by training a MLP model to determine the necessity of GNNs and to evaluate the performance improvement GNNs could offer over MLPs on graphical data. RGATs and RGCNs were selected as the GNN models due to their robust capability to operate on multi-relational graphs.

### 4.2.1 MLP

We firstly train a MLP for classifying the severity of wait time for each edge. MLP is chosen as a baseline to compare with GNN models and to see how well GNN models do better compared to MLPs which do not take into account the graphical structure of data. Thus, we can identify if GNNs are essential to our research question.

A MLP is a type of feedforward artificial neural network consisting of an input layer, one or more hidden layers, and an output layer. Each neuron in the hidden layer, applies a nonlinear activation function to its inputs. This nonlinearity allows MLPs to model complex relationships between inputs and outputs, enabling the network to capture patterns that a linear model could not.

MLPs are trained using backpropagation, a supervised learning technique that adjusts the weights and biases to minimize a loss function, such as cross-entropy loss in classification tasks as used in this thesis. As described in Section 3.1, there is a class imbalance, so a weighted cross-entropy loss function is used. This loss function weights the loss based on the frequency of each class, ensuring that each class is treated with equal importance despite its frequency. Backpropagation calculates the gradient of the loss function with respect to each parameter by applying the chain rule, enabling efficient gradient-based optimization methods. Common methods include stochastic gradient descent (SGD), which updates weights using the gradient of the loss for a single or small batch of samples, and Adam, which is used in our case adapts the learning rate for each parameter for faster convergence. By processing input vectors through weighted sums and activation functions, MLPs can effectively learn from data and serve as the foundation for more advanced neural network architectures. To learn more about MLPs, you can read Section 2.5.

When utilizing MLPs, graphical data poses a unique challenge due to its inherent structure, which consists of nodes and edges. MLPs are designed to handle tabular data, where each record (row) is independent of others and can be represented by a fixed number of features (columns). To adapt graphical data for MLPs, the graph is transformed into a tabular format by focusing on its edges and their associated features. Specifically, each edge in the graph is treated as a separate record in the table. This transformation process effectively linearizes the graph, allowing the MLP to process it. However, this method abstracts away the topological structure of the graph, capturing relationships between nodes only in terms of the edges and their features.

In developing our MLP model, we systematically explored various configurations to identify the optimal architecture based on the evaluation criteria outlined in Section 5.3. Given that our model requires three input features per edge and four output classes for multi-class classification, we focused on identifying the right number of hidden layers and neurons.

We began by evaluating models with a single hidden layer and incrementally increased the number of hidden layers to assess performance. Specifically, we tested configurations with 1, 2, and 3 hidden layers. Our results indicated that performance did not improve beyond a single hidden layer, suggesting that additional layers did not contribute to better outcomes for this task.

For each hidden layer configuration, we varied the number of neurons in the hidden layer, exploring values in increasing powers of two: 2, 4, 8, 16, 32, and 64. We observed that increasing the number of neurons beyond 8 did not yield significant improvements in model performance.

The optimal configuration emerged as a model with a single hidden layer consisting of 8 neurons. The learning rate for this model was set to 0.00015. The architecture of the selected MLP model is illustrated in Figure 4.3.

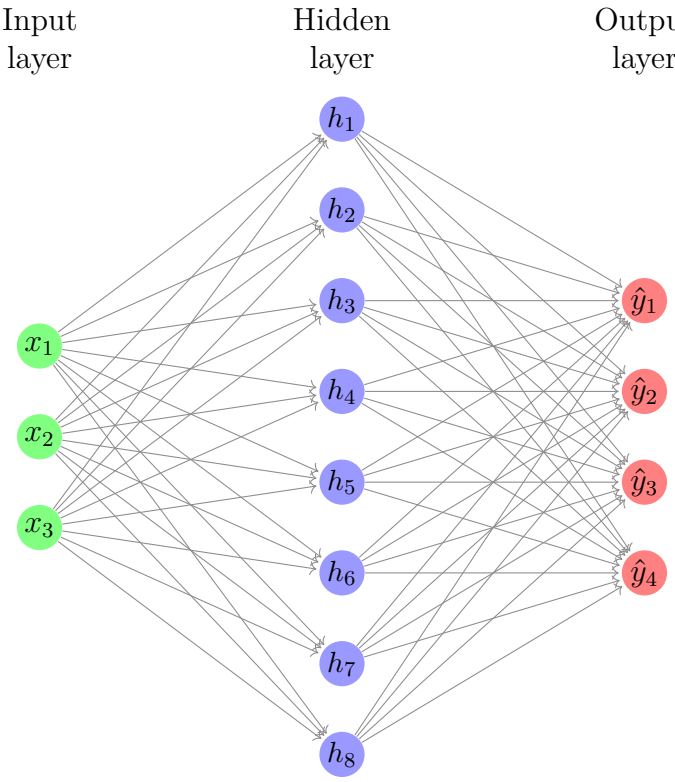


Figure 4.3: MLP Model

### 4.2.2 GNN

With our baseline MLP model established for comparison, we now evaluate the performance of GNNs on our dataset. We employ two prominent GNN architectures: GCNs and GATs. To handle multi-relational graphs, we extend these models to their multi-relational counterparts: RGCNs and RGATs. These models are adapted to work with the multi-relational dual graphs described in Section 4.1.

GNNs extend the concept of convolutions to graph-structured data, where the core operations involve aggregating information from a nodes neighbors and updating its representation. In a generalized GNN convolutional layer, each node $u$ updates its embedding $h_u^{(k+1)}$ based on aggregated information from its neighborhood $N(u)$ and its current state $h_u^{(k)}$. Mathematically, this is expressed as:

$$h_u^{(k+1)} = \sigma\left(W^{(k)} \cdot \text{AGGREGATE}\left(\{h_v^{(k)}, \forall v \in N(u)\}\right)\right), \tag{4.1}$$

where AGGREGATE represents a function that combines neighbor embeddings, $W^{(k)}$ is a learnable weight matrix, and $\sigma$ denotes a non-linear activation function. To learn more about GNNs, you can read Section 2.7.

The multi-relational dual graphs employed in our study are directed graphs, where the direction of an edge represents the path taken by an AGV from the source node to the destination node. Specifically, traffic moves from the source to the destination node, while the effects of congestion propagate in the reverse direction from the point of congestion back to the source. This reverse flow is essential for classifying wait times due to traffic congestion. It makes sense for information to flow from the target node (where congestion is experienced) to the source node (where the vehicle originates), as this accurately reflects how congestion at the destination node impacts overall wait times and traffic conditions affecting the source node. Furthermore, the direction of edges in the multi-relational graph is also influenced by blocking rules. The blocking rules are described as a source node being blocked by a destination node. When a source node is blocked by a destination node, the congestion effect also flows from the target node back to the source node, reinforcing the need for reverse information propagation.

PyTorch Geometric facilitates this information flow by supporting the propagation of messages from target to source nodes through its message-passing framework. In PyTorch Geometric, the MessagePassing class allows for the specification of custom message aggregation and update functions. By configuring these functions to propagate information in the reverse direction of edge traversal, we ensure that the model accurately reflects the effects of congestion and blocking rules. This setup allows the GNN to learn how congestion at a destination node impacts conditions at the source node, thereby enhancing the model's ability to predict wait times and traffic conditions.

In our setup, the GNN models process the dual graph version, where nodes and edges are inverted. The task for the GNNs is node classification, where each node in the input dual graph has 3 features. The GNN outputs a graph where each node is represented by a vector of 4 features, corresponding to the probabilities of belonging to one of four classes.

Consistent with our MLP training approach, we use the same loss function a weighted cross-entropy loss to address the class imbalance discussed in Section 3.1. The training of the GNN models is carried out using the Adam optimizer, following a gradient-based optimization approach.

#### 4.2.2.1 Selecting the Number of Hidden Layers and Neurons in GNNs

Selecting the optimal number of hidden layers and the appropriate size of hidden units in GNNs is pivotal for effectively capturing the complex dependencies within graph-structured data. The number of hidden layers in a GNN influences the depth of the model's ability to propagate information through the graph. Increasing the number of layers allows the network to aggregate features from nodes at increasingly distant neighborhoods, thereby enhancing its capacity to capture higher-order relationships and intricate structural patterns. However, deeper architectures also introduce the risk of over-smoothing, where node embeddings become overly homogenized, potentially leading to diminished performance [24]. In our experiments, we explored a range of hidden layers from 1 to 5. Given the size of our graphs, we

anticipated that using 6 or more hidden layers would likely lead to over-smoothing and homogenized node embeddings, impairing model performance. The size of the hidden layers, defined by the number of units, affects the model's representational capacity. Larger layers can capture more detailed features but may increase computational complexity and overfitting risk. Thus, we evaluate hidden layer sizes using increasing powers of two: 2, 4, 8, 16, 32, 64, 128, and so forth until performance either plateaus or begins to decline for too long.

#### 4.2.2.2 RGCN

When constructing a RGCN, we utilize the relational variant of the GraphConv operator described by Morris et al. in [25] and implemented by Pytorch Geometric. This advanced layer extends the standard GraphConv operation to handle heterogeneous graphs with multiple edge types, allowing the network to learn from various types of relationships between nodes.

The GraphConv layer operates by aggregating information from a node's neighbors and updating the node's representation based on this aggregated information. Specifically, it computes a weighted sum of the features of neighboring nodes, with the weights determined by the connectivity structure of the graph. This aggregation is followed by a linear transformation and a non-linear activation function, which enables the model to capture and learn complex patterns in the node features.

In the case of RGCN, this process is adapted to handle different types of edges. Each type of edge is processed through separate weight matrices, allowing the network to learn distinct transformations for each relationship type. The results from these separate transformations are then combined to update the node's feature representation. This allows RGCNs to effectively model graphs where nodes are connected by various kinds of relationships, enhancing the network's ability to understand and leverage the multi-relational structure of the data. To learn more about RGCN, you can read Section 2.8.1.

Our best performing RGCN model is described in Figure 4.4 with 3 relational GraphConv layers each of size 32 and trained on our dataset with a learning rate of 0.0001.

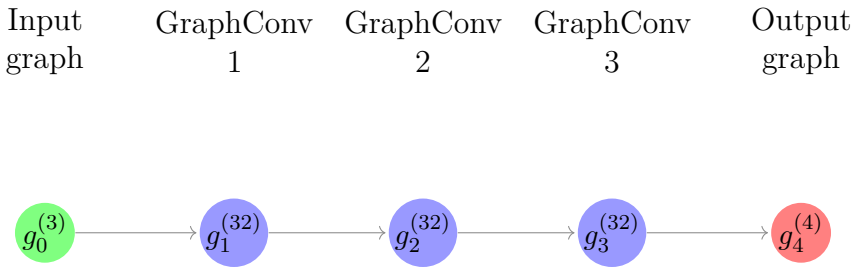


Figure 4.4: RGCN Model

#### 4.2.2.3 RGAT

When building an RGAT, we employ the relational version of the GATConv operator as described by Velickovic et al. in [20] and implemented in Pytroch Geometric.

This advanced layer enhances the traditional GATConv by incorporating relational attention mechanisms to manage graphs with multiple edge types.

The standard GATConv layer operates by computing attention scores between a node and its neighbors to dynamically weigh the importance of their features. This is achieved through a self-attention mechanism where each node calculates attention coefficients based on the similarity of its features with those of its neighbors. These coefficients are normalized using a softmax function, ensuring that the importance scores sum to one. To learn more about neighbourhood attention, you can check out Section 2.7.7.

In RGAT, this process is adapted to handle multiple types of relationships. For each edge type, separate attention mechanisms and weight matrices are used to compute distinct attention scores. This allows the network to learn different importance levels for neighbors depending on the edge type connecting them. The attention scores are then normalized and applied to aggregate the features of neighbors for each edge type. Finally, the aggregated features from all edge types are combined, typically by summing or concatenating, and passed through a linear transformation and activation function to update the node's feature representation. To learn more about RGAT, you can read Section 2.8.2.

Our best performing RGAT model is described in Figure 4.5 with 3 relational Graph-Conv layers each of size 32 and trained on our dataset with a learning rate of 0.0001.

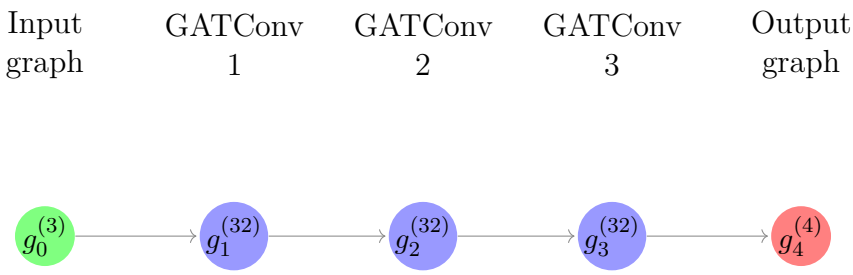


Figure 4.5: RGAT Model

# 5

# Evaluation

This section reviews methods for assessing GNN performance in classifying AGV system wait times.

## 5.1 Refined research questions

The following are the research questions this master thesis is trying to answer:

1. How can Graph Neural Networks effectively predict and classify wait time severity at each segment within an AGV system?

2. To what extent can these classifications by Graph Neural Networks enhance the efficiency of planning AGV network designs?

3. How long does it take to train Graph Neural Network models for predicting AGV system wait times?

4. How much memory and computation are needed to deploy Graph Neural Network models for on-demand applications in AGV systems?

## 5.2 Evaluation Environment

This section discusses the environment setup and methodologies employed to evaluate the multi-class classification model. The evaluation process is critical to ensure the model's effectiveness, generalization capability, and robustness. We detail the hardware and software configurations, evaluation metrics, and analysis techniques used throughout the evaluation phase.

### 5.2.1 Hardware Configuration

The hardware specifications used for training the machine learning models include a Dell Latitude 5410 laptop, equipped with an Intel(R) Core(TM) i7-10610U CPU, 32 GB of RAM, and 1 TB of storage. The GPU isn't utilized.

### 5.2.2 Software Configuration

The software configuration for training the machine learning models on the specified hardware includes Windows 11 as the operating system, and Python 3.11 as the programming language. Key libraries utilized are Pytorch 2.2, NumPy, Pandas, and Matplotlib. The development environment used is PyCharm.

## 5.3 Evaluation metrics

Evaluation of the model in multi-class classification involves key metrics as proposed by Sokolova and Lapalme [26] are described below.

### 5.3.1 Precision

Precision measures the proportion of true positive predictions among all instances classified as positive by the model. It is calculated as the ratio of true positives to the sum of true positives and false positives.

### 5.3.2 Recall

Recall, also known as sensitivity, quantifies the proportion of true positive predictions captured by the model relative to all actual positive instances. It is calculated as the ratio of true positives to the sum of true positives and false negatives.

### 5.3.3 F1 Score

The F1 score is the harmonic mean of precision and recall, providing a balanced measure of the model's accuracy. It is calculated as the weighted average of precision and recall, emphasizing the balance between precision and recall

### 5.3.4 Confusion Matrix

A confusion matrix is a table that summarizes the performance of a classification model by tabulating the true positive, true negative, false positive, and false negative predictions across all classes. It provides a comprehensive overview of the model's performance, facilitating the identification of classification errors and misclassification.

### 5.3.5 Accuracy

Accuracy measures the proportion of total correct predictions (both true positives and true negatives) among all predictions made by the model. It is calculated as the ratio of the sum of true positives and true negatives to the total number of instances.

### 5.3.6 Specificity

Specificity, also known as the true negative rate, quantifies the proportion of actual negatives that are correctly identified by the model. It is calculated as the ratio of true negatives to the sum of true negatives and false positives.

### 5.3.7 Fall-Out

Fall-Out, also known as the false positive rate, measures the proportion of actual negatives that are incorrectly classified as positive by the model. It is calculated as the ratio of false positives to the sum of false positives and true negatives.

### 5.3.8 Model Size

Model size denotes the storage space required for the trained model.

### 5.3.9 Training Time

Training time measures the duration required to train a machine learning model on a given dataset. It is influenced by model complexity and dataset size and directly impacts development cycles and resource planning.

### 5.3.10 Inference Time

Inference time refers to the speed at which a trained model can make predictions on new data.

## 5.4 Experiments

Our experiments begin with the utilization of the specified dataset detailed in Section 3. This dataset is used to train the various models described in Section 4.2 and evaluate their performance against our evaluation criteria outlined in Section 5.3. During these experiments, the most efficient model is selected based on its ability to improve performance and understand the strengths and limitations of different architectures.

## 5.5 Hypothesis

We expect MLPs to perform poorly on tasks involving graph-structured data. MLPs, designed primarily for tabular data, are likely to struggle with capturing the intricate relationships and topology inherent in graphs. Conversely, GNNs are designed to excel in understanding complex graph structures, making them far superior to MLPs for tasks that require comprehension of graph-based relationships.

RGCNs might outperform RGATs due to their robustness to noise. RGCNs aggregate information uniformly from neighboring nodes, which dilutes the impact of any

single noisy node and ensures more stable representations. Unlike RGATs, which can be sensitive to noise due to their reliance on attention scores that may overemphasize or underemphasize certain nodes, RGCNs maintain noise insensitivity by treating all neighboring nodes equally. This uniform aggregation makes RGCNs inherently more resilient to noisy data, leading to more reliable performance in noisy environments.

# 6

# Results

In exploring effective models for classifying severity wait times, we found that MLPs were inadequate, prompting our shift to GNNs to leverage the inherent graph structure in the data. Our analysis focused on comparing RGCN and RGAT for their performance in this classification task. After rigorous experimentation, RGCN emerged as the superior choice over RGAT for several reasons:

- **Performance:** RGCN consistently achieved higher F1 scores as shown in Table 6.4 across all severity classes compared to RGAT in Table 6.6.
- **Training Efficiency:** RGCN demonstrated shorter training times as shown in Table 6.5 compared to RGAT in Table 6.7. The simpler computational nature of RGCN allows for quicker convergence during training.
- **Inference Speed:** In addition to faster training, RGCN exhibited lower inference times as shown in Table 6.5 compared to RGAT in Table 6.7 due to its computational simplicity. This efficiency is essential for deploying models in on-demand systems such as AGV production environments, where timely decision-making is crucial.

Based on our hypothesis as described in Section 5.5, the experimental results confirmed that MLPs performed poorly on tasks involving graph-structured data, aligning with our expectation that MLPs are not well-suited for capturing intricate relationships and topology inherent in graphs. Conversely, GNNs, specifically RGCNs and RGATs, demonstrated superior performance. As hypothesized, RGCNs outperformed RGATs due to their robustness to noise and uniform aggregation of information from neighboring nodes.

## 6.1 Models

The performance of the trained models was evaluated using a set of key metrics as described in Section 5.3. Table 6.1 describes the various classes and it's corresponding number of instances and it's percentage.

### 6.1.1 MLP

Table 6.2 summarizes precision, recall, F1 score, specificity, and fall-out for each class, while Figure 6.1 shows the confusion matrix detailing predictions across all

| Class | Number of Instances | Percentage (%) |
|---|---|---|
| <50s | 1443 | 82.46 |
| <400s | 169 | 9.66 |
| <800s | 80 | 4.57 |
| >=800s | 58 | 3.31 |

Table 6.1: Class Percentage

classes. Table 6.3 presents the accuracy, model size, training time, and inference time for the trained models.

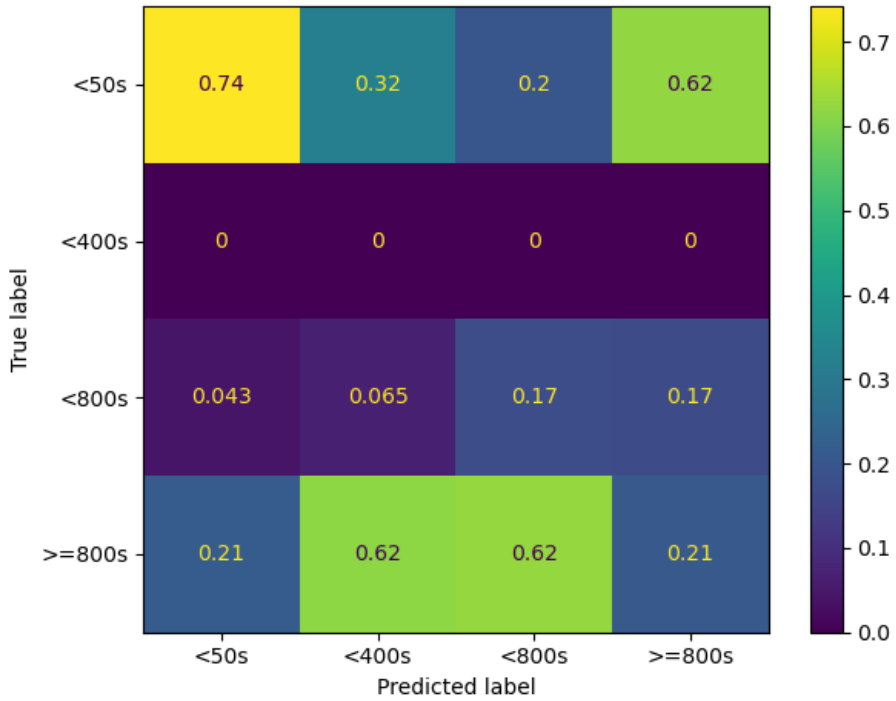


Figure 6.1: MLP Confusion Matrix

| Class | Precision | Recall | F1-score | Specificity | Fall-Out |
|---|---|---|---|---|---|
| <50s | 0.91 | 0.74 | 0.82 | 0.93 | 0.07 |
| <400s | 0.00 | 0.00 | 0.00 | 0.86 | 0.14 |
| <800s | 0.14 | 0.17 | 0.16 | 0.94 | 0.06 |
| >=800s | 0.03 | 0.21 | 0.04 | 0.95 | 0.05 |

Table 6.2: MLP Classification report

Based on the F1-scores presented in Table 6.2, the MLP model demonstrates limited effectiveness in classifying all classes, showing satisfactory performance only for Class 1 with an F1 score of 0.82, which constitutes approximately 82.46% of the dataset. However, MLP has shown poor performance with F1 scores of 0.0, 0.16 and 0.04 in classifying every other class, including critical edge cases. Thus highlighting its inadequacy for classifying graphical data in this context.

| Metric | Value |
|---|---|
| Accuracy | 0.62 |
| Model Size | 1 KB |
| Training Time | 2 minutes |
| Inference Time | 0.002 seconds |

Table 6.3: MLP metrics

### 6.1.2 RGCN

Table 6.4 summarizes precision, recall, F1 score, specificity, and fall-out for each class, while Figure 6.2 shows the confusion matrix detailing predictions across all classes. Table 6.5 presents the accuracy, model size, training time, and inference time for the trained models.

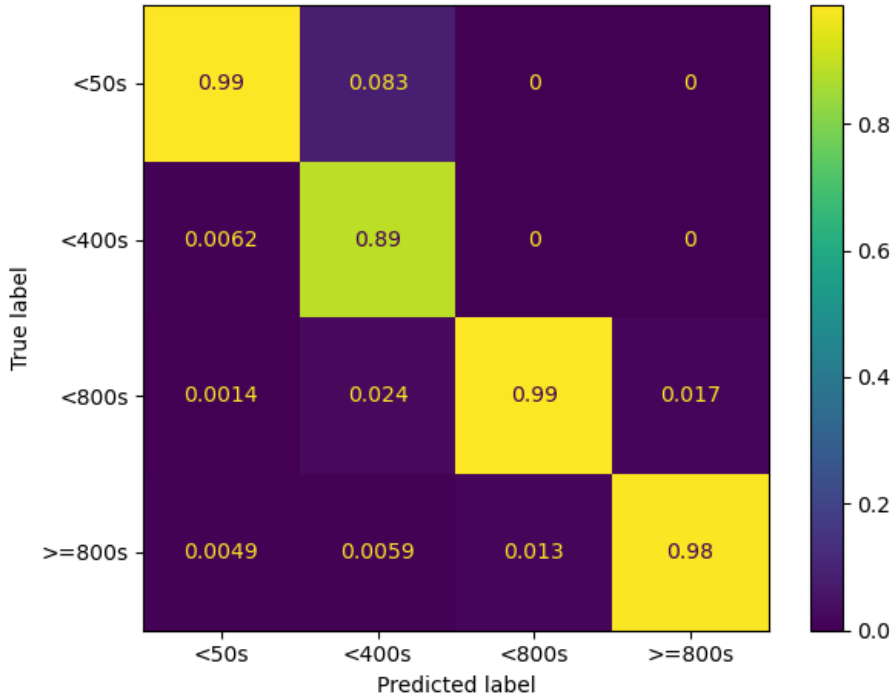


Figure 6.2: RGCN Confusion Matrix

| Class | Precision | Recall | F1-score | Specificity | Fall-Out |
|---|---|---|---|---|---|
| <50s | 0.99 | 0.99 | 0.99 | 0.94 | 0.06 |
| <400s | 0.94 | 0.89 | 0.91 | 0.98 | 0.02 |
| <800s | 0.92 | 0.99 | 0.95 | 0.99 | 0.01 |
| >=800s | 0.86 | 0.98 | 0.92 | 0.99 | 0.01 |

Table 6.4: RGCN Classification report

Based on the F1-scores presented in Table 6.4, the RGCN model exhibits strong performance in classifying all classes with an average F1 score of 0.94, including edge

| Metric | Value |
|---|---|
| Accuracy | 0.97 |
| Model Size | 35 KB |
| Training Time | 4 minutes |
| Inference Time | 0.01 seconds |

Table 6.5: RGCN metrics

and extreme cases. It achieves high accuracy across the dataset, demonstrating its capability to classify graphical data in this context.

### 6.1.3 RGAT

Table 6.6 summarizes precision, recall, F1 score, specificity, and fall-out for each class, while Figure 6.3 shows the confusion matrix detailing predictions across all classes. Table 6.7 presents the accuracy, model size, training time, and inference time for the trained models.

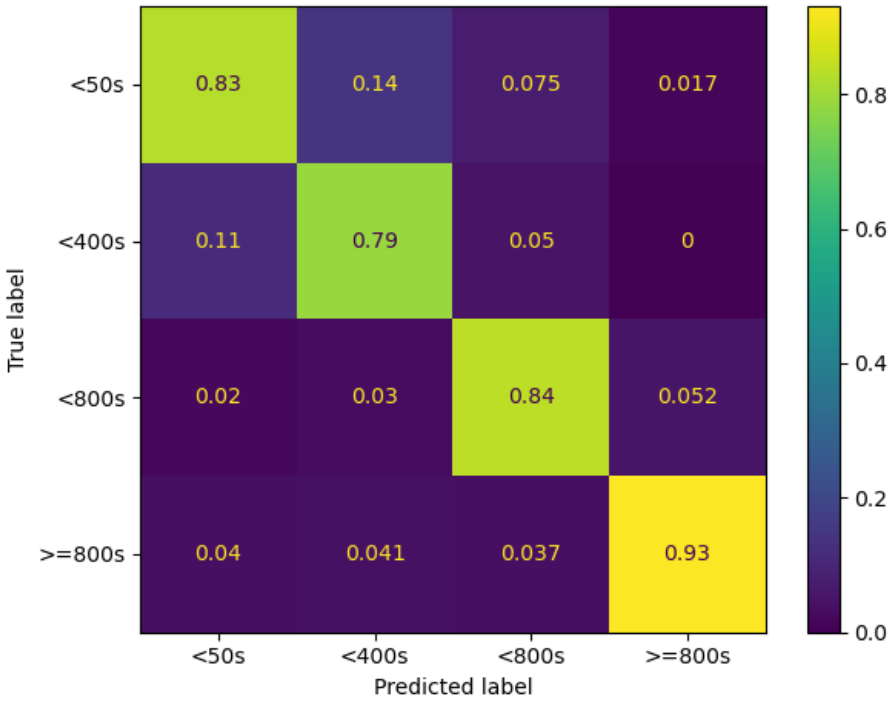


Figure 6.3: RGAT Confusion Matrix

Based on the F1-scores presented in Table 6.6, the RGAT model demonstrates moderate performance in classifying all classes with an average F1 score of 0.695.

| Class | Precision | Recall | F1-score | Specificity | Fall-Out |
|---|---|---|---|---|---|
| <50s | 0.97 | 0.83 | 0.89 | 0.5 | 0.5 |
| <400s | 0.44 | 0.79 | 0.56 | 0.97 | 0.03 |
| <800s | 0.64 | 0.84 | 0.73 | 0.99 | 0.01 |
| >=800s | 0.45 | 0.93 | 0.60 | 0.99 | 0.01 |

Table 6.6: RGAT Classification report

| Metric | Value |
|---|---|
| Accuracy | 0.82 |
| Model Size | 20 KB |
| Training Time | 6 minutes |
| Inference Time | 0.02 seconds |

Table 6.7: RGAT metrics

# 7

# Conclusion

This study demonstrated the effectiveness of using GNNs to predict traffic and congestion within AGV systems. We utilized a comprehensive dataset consisting of 50 different warehouse layouts, including detailed simulations of vehicle movements and traffic rules defined by fleet managers. This dataset allowed us to model the complex interactions and constraints that impact vehicle flow in indoor warehouse environments.

Our research showed that multi-relational dual graphs are the ideal way to represent data for modeling traffic in AGV systems. The multi-relational aspect enabled us to capture different types of relationships between nodes, such as paths and traffic regulations. Meanwhile, the dual graph structure helps capture both the spatial layout and the relational dynamics of vehicle movements and traffic rules.

Our study demonstrates that RGCNs are highly effective in predicting and classifying wait times, achieving a notable F1 score of 0.94 on our dataset with a swift inference time of 0.01 seconds. While these models demonstrate impressive performance on our dataset, their effectiveness on different road networks not included in the data is still to be evaluated. Nonetheless, these results indicate that RGCNs have the potential to greatly enhance AGV system planning and management by delivering accurate traffic predictions. This enhanced predictive capability can lead to more informed design adjustments and reduced delays in AGV system design. The current state-of-art method of predicting traffic in AGV systems is running simulations for more than 30 minutes but our method classifies traffic in less than a second thus dramatically reducing delays in AGV system design.

## 7.1 Discussion

Our work on traffic prediction in AGV systems contributes to the broader field of traffic management in dynamic and complex environments. By applying GNNs, we have accelerated the planning process and reduced the computational resources needed compared to traditional simulation-based methods. This demonstrates the potential of machine learning techniques in improving the design and operation of automated systems in logistics and beyond. Although our study focused on T-intersection layouts in warehouse environments, where GNNs effectively modeled and predicted congestion, our methods can be applied to a broader range of problems and benefit more stakeholders.

### 7.1.1 Expanding Beyond T-Intersection Layouts

While our study centered on T-intersection layouts, our principles and techniques are adaptable to other warehouse configurations. Different layouts, such as cross intersections or complex networked paths, introduce unique traffic prediction and management challenges. GNNs are highly versatile and can be applied to graphs of any topology due to their ability to process information through node and edge relationships. They operate by iteratively aggregating and transforming features from neighboring nodes, making them effective for any type of graph structures and thus road networks or AGV layouts. By refining our GNN models to accommodate these varied configurations, we can enhance the versatility and applicability of our approach, offering robust solutions across diverse warehouse environments.

The main implications of using a larger and more varied set of road networks in the dataset are:

- **Improved model generalization:** A larger dataset with more diverse AGV layouts and traffic scenarios would likely enable the GNN models to learn more robust and generalizable features, potentially leading to better performance on unseen configurations.
- **Reduced overfitting risk:** More training examples can help reduce the risk of overfitting, especially for complex GNN architectures.
- **Increased computational requirements:** Training on a larger dataset requires more computational resources and time, which can slow down the research process and limit the number of experiments that can be run.

### 7.1.2 Applications in Other AGV Systems

The methodologies we developed are not confined to warehouse settings. AGV systems are increasingly being utilized in various sectors, including hospitals, airports, and manufacturing plants, where vehicles' efficient and reliable movement is critical. For instance, in hospitals, AGVs are used to transport medical supplies and equipment. By applying our traffic prediction models, these systems can improve operational efficiency. Beyond individual warehouses, our methodologies can be scaled to larger logistics networks, improving the flow of goods across supply chains and minimizing bottlenecks.

# Bibliography


[1] T. Owen, "*Automated Guided Vehicle Systems*, edited by R.H. hollier, IFS (publications) ltd, bedford, u.k., 02 1987 (price č39)," *Robotica*, vol. 6, no. 2, p. 169, 1988. DOI: 10.1017/S0263574700004161. [Online]. Available: https://doi.org/10.1017/S0263574700004161.

[2] A. Derrow-Pinion, J. She, D. Wong, *et al.*, "Eta prediction with graph neural networks in google maps," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, ser. CIKM 21, ACM, Oct. 2021. DOI: 10.1145/3459637.3481916. [Online]. Available: http://dx.doi.org/10.1145/3459637.3481916.

[3] Z. Cui, K. Henrickson, R. Ke, and Y. Wang, "Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 11, pp. 4883–4894, 2020. DOI: 10.1109/TITS.2019.2950416. [Online]. Available: https://doi.org/10.1109/TITS.2019.2950416.

[4] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, J. Lang, Ed., ijcai.org, 2018, pp. 3634–3640. DOI: 10.24963/IJCAI.2018/505. [Online]. Available: https://doi.org/10.24963/ijcai.2018/505.

[5] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, AAAI Press, 2019, pp. 922–929. DOI: 10.1609/AAAI.V33I01.3301922. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.3301922.

[6] C. Zheng, X. Fan, C. Wang, and J. Qi, "GMAN: A graph multi-attention network for traffic prediction," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, AAAI Press, 2020, pp. 1234–1241. DOI: 10.1609/AAAI.V34I01.5477. [Online]. Available: https://doi.org/10.1609/aaai.v34i01.5477.

[7] T. S. Jepsen, C. S. Jensen, and T. D. Nielsen, "Relational fusion networks: Graph convolutional networks for road networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 418–429, Jan. 2022, ISSN: 1558-0016. DOI: 10.1109/tits.2020.3011799. [Online]. Available: http://dx.doi.org/10.1109/TITS.2020.3011799.

[8] J. Lauri and M. Wiberg, "Supervised learning to evaluate road networks for automated guided vehicles," Available at https://odr.chalmers.se/server/api/core/bitstreams/190e0890-ac60-4b60-93f4-1cb912160bbe/content, M.S. thesis, Chalmers Univeristy of Technology, Gothenburg, Sweden, 2023.

[9] S. Velayutham, *Agv traffic prediction dataset*, https://github.com/Shivneshwar/traffic-prediction-agv-dataset, Github, 2024.

[10] J. L. Gross, J. Yellen, L. W. Beineke, and R. J. Wilson, "Introduction to graphs," in *Handbook of Graph Theory*, ser. Discrete Mathematics and Its Applications, J. L. Gross and J. Yellen, Eds., Chapman & Hall / Taylor & Francis, 2003, pp. 1–55. DOI: 10.1201/9780203490204.CH1. [Online]. Available: https://doi.org/10.1201/9780203490204.ch1.

[11] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015. DOI: 10.1016/J.NEUNET.2014.09.003. [Online]. Available: https://doi.org/10.1016/j.neunet.2014.09.003.

[12] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=Hkuq2EkPf.

[13] R. Hecht-Nielsen, "Theory of the backpropagation neural network," *Neural Networks*, vol. 1, no. Supplement-1, pp. 445–448, 1988. DOI: 10.1016/0893-6080(88)90469-8. [Online]. Available: https://doi.org/10.1016/0893-6080(88)90469-8.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980.

[15] Z. Liu and J. Zhou, *Introduction to Graph Neural Networks* (Synthesis Lectures on Artificial Intelligence and Machine Learning). Morgan & Claypool Publishers, 2020, ISBN: 978-3-031-00459-9. DOI: 10.2200/S00980ED1V01Y202001AIM045. [Online]. Available: https://doi.org/10.2200/S00980ED1V01Y202001AIM045.

[16] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, 2017, pp. 1263–1272. [Online]. Available: http://proceedings.mlr.press/v70/gilmer17a.html.

[17] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Decem-*

*ber 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, *et al.*, Eds., 2017, pp. 3391–3401. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/f22e4747da1aa27e363d86d40ff442fe-Abstract.html.

[18] R. L. Murphy, B. Srinivasan, V. A. Rao, and B. Ribeiro, "Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs," *CoRR*, vol. abs/1811.01900, 2018. arXiv: 1811.01900. [Online]. Available: http://arxiv.org/abs/1811.01900.

[19] D. Soydaner, "Attention mechanism in neural networks: Where it comes and where it goes," *Neural Comput. Appl.*, vol. 34, no. 16, pp. 13 371–13 385, 2022. DOI: 10.1007/S00521-022-07366-3. [Online]. Available: https://doi.org/10.1007/s00521-022-07366-3.

[20] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *CoRR*, vol. abs/1710.10903, 2017. arXiv: 1710.10903. [Online]. Available: http://arxiv.org/abs/1710.10903.

[21] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

[22] M. S. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," *CoRR*, vol. abs/1703.06103, 2017. arXiv: 1703.06103. [Online]. Available: http://arxiv.org/abs/1703.06103.

[23] D. Busbridge, D. Sherburn, P. Cavallo, and N. Y. Hammerla, "Relational graph attention networks," *CoRR*, vol. abs/1904.05811, 2019. arXiv: 1904.05811. [Online]. Available: http://arxiv.org/abs/1904.05811.

[24] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=S1ldO2EFPr.

[25] C. Morris, M. Ritzert, M. Fey, *et al.*, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, AAAI Press, 2019, pp. 4602–4609. DOI: 10.1609/AAAI.V33I01.33014602. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33014602.

[26] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, 2009. DOI: 10.1016/J.IPM.2009.03.002. [Online]. Available: https://doi.org/10.1016/j.ipm.2009.03.002.