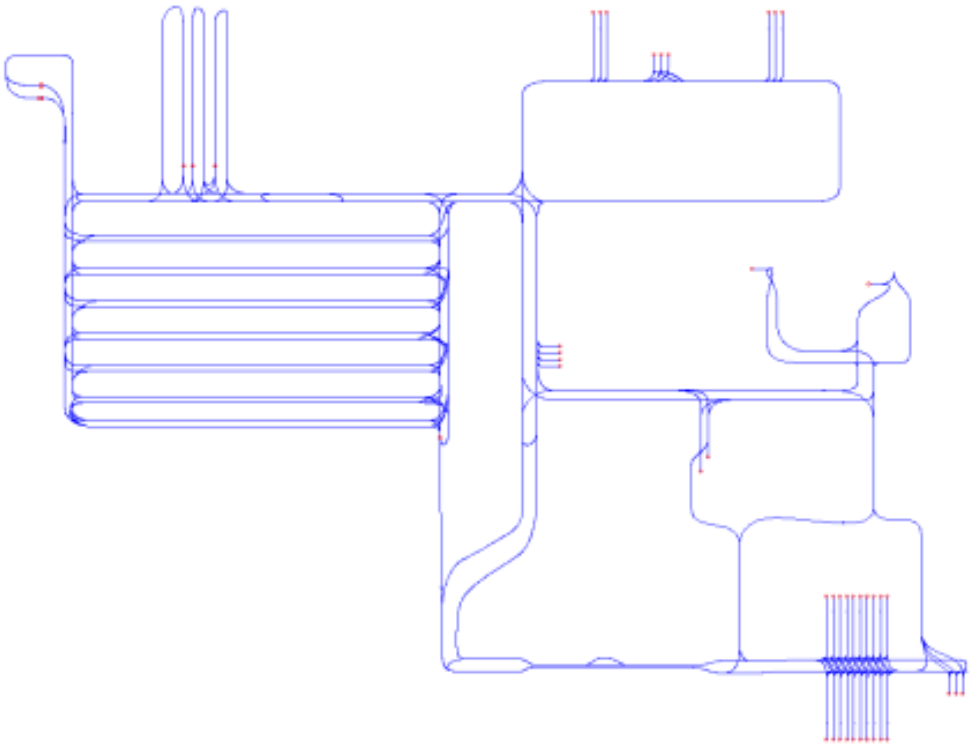
CHALMERS
UNIVERSITY OF TECHNOLOGY



Analyzing Simulation Dynamics in AGV Systems for Improved Traffic Prediction Using Graph Neural Networks

Data Augmentation, Simulation Constraints, and Generalization Challenges in Using Machine Learning for Traffic Predictions In AGV Systems

Master's Thesis In Complex Adaptive Systems And Systems, Control and Mechatronics

Hamza Habanakeh, Nikolai Gustafsson

DEPARTMENT OF ELECTRICAL ENGINEERING

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

Master's thesis 2025

Analyzing Simulation Dynamics in AGV Systems for Improved Traffic Prediction Using Graph Neural Networks

Data Augmentation, Simulation Constraints, and Generalization Challenges in Using Machine Learning for Traffic Predictions In AGV Systems

Hamza Habanakeh, Nikolai Gustafsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Division name
Chalmers University of Technology
Gothenburg, Sweden 2025

Analyzing Simulation Dynamics in AGV Systems for Improved Traffic Prediction Using Graph Neural Networks
Data Augmentation, Simulation Constraints, and Generalization Challenges in Using Machine Learning for Traffic Predictions In AGV Systems
Hamza Habanakeh, Nikolai Gustafsson

Supervisor: Rasmus Åkerlund, Kollmorgen Automation AB
Examiner: Jonas Sjöberg, Department of Electrical Engineering

Master's Thesis 2025
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A demo layout representing the complexity of a real-world AGV system.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Analyzing Simulation Dynamics in AGV Systems for Improved Traffic Prediction Using Graph Neural Networks
Data Augmentation, Simulation Constraints, and Generalization Challenges in Using Machine Learning for Traffic Predictions in AGV Systems
Hamza Habanakeh, Nikolai Gustafsson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Automated Guided Vehicle (AGV) systems play a critical role in modern industrial automation, but designing efficient AGV layouts is time-consuming due to the reliance on extensive simulations. This thesis explores the use of Graph Neural Networks (GNNs) to predict congestion and waiting times within AGV layouts, with the goal of accelerating the design process. Building on previous research, we use a hierarchical GNN model that combines classification and regression to estimate segment-level waiting times. To overcome data scarcity and simulation inconsistency, we construct a graph-based dataset from real AGV systems and apply targeted data augmentation focused on traverse time. Simulation consistency and appropriate simulation durations are carefully analysed to ensure data reliability. Our results show that while the classifier performs robustly, surpassing variation baselines set by the augmentation average, the regressor faces challenges in accurately modelling continuous waiting times. The study highlights key limitations, including the absence of AGV count and fleet management logic in the input features. Nonetheless, the findings demonstrate the potential of GNNs for layout evaluation and provide a foundation for more generalizable, traffic prediction tools in AGV system design.

Keywords: Graph Neural Networks, Automated Guided Vehicles, congestion prediction, layout optimization, data augmentation, machine learning.

Acknowledgements

First and foremost, we would like to extend our sincere thanks to Rasmus Åkerlund for his invaluable supervision throughout this thesis. His frequent, hands-on guidance and deep expertise in AGV systems, machine learning, and academic research were essential to the progress and outcomes of this work.
We would also like to thank Jonas Sjöberg for his role as examiner and supervisor, and for providing constructive feedback during the process.
A heartfelt thank you to the management and all employees at Kollmorgen for not only hosting our research and generously sharing your expertise, tools, and resources, but also for making us feel truly welcomed and continuously cared for throughout our time with the company.

Hamza Habanakeh, Nikolai Gustafsson, Gothenburg, May 2025

Contents

List of Figures

List of Tables

1
Introduction

Automated Guided Vehicles (AGVs) are crucial for industrial automation, enhancing efficiency and safety across manufacturing, logistics, and healthcare. They reduce human labor and errors, improving performance and scalability. Research shows that well-designed AGVs lead to cost savings and higher productivity. For example, optimization techniques in flexible manufacturing systems improve efficiency, and a case study demonstrates how AGVs increased productivity and reduced labor costs in the automotive industry [1], [2]. AGVs also improve worker satisfaction by automating repetitive tasks, allowing employees to focus on more complex work. The AGV market is projected to reach over USD 7.97 billion by 2031, highlighting their growing importance in smart logistics [3].

1.1 Background

The key to effective and smooth warehouse operation is customizing the AGV road network called layout, as the physical size and structure of existing warehouses often limit how much goods can be moved around in the system. The layout determines the vehicles' movement space and includes segments (roads) and points (points connecting segments). A more detailed explanation of layouts in AGV systems will be provided in a later section 2.1.

There are many potential sources of waste in an AGV system. In this context, waste means inefficiencies or activities that do not add value to the material handling process. Examples of this include AGVs standing still without a task, congestion, or poor path planning that leads to longer travel times. These problems slow down the system and use up resources like energy, time, and vehicle capacity without improving performance. Waiting time is a metric used to measure these inefficiencies, simulations are used to reveal waiting times before deploying a system.

Simulation is a key component in the development and deployment of AGV systems. It allows for safe testing of layout designs, AGV behaviour, and control logic without affecting real-world operations. By simulating different scenarios, engineers can validate system performance, identify potential issues such as deadlocks or congestion, and optimize parameters like speed, routing, and task allocation. The simulation tool used to perform these simulation is however not flawless. There is no clear research on the consistency of the tool, i.e. whether identical inputs result in identical waiting times for all segments in the system. It is unclear how long the simulation

needs to run for the waiting times to accurately reflect which segments in the AGV system are prone to long-term congestion. Due to limited resources, it is important to find the shortest possible simulation time that still captures these long-term congestion patterns.

The traditional way for designing a layout for an AGV system is to base the process on domain experience. The initial layout design is then iteratively simulated and improved based on simulation results, the process can be seen in Figure 1.1. However, the simulation process is highly time consuming.

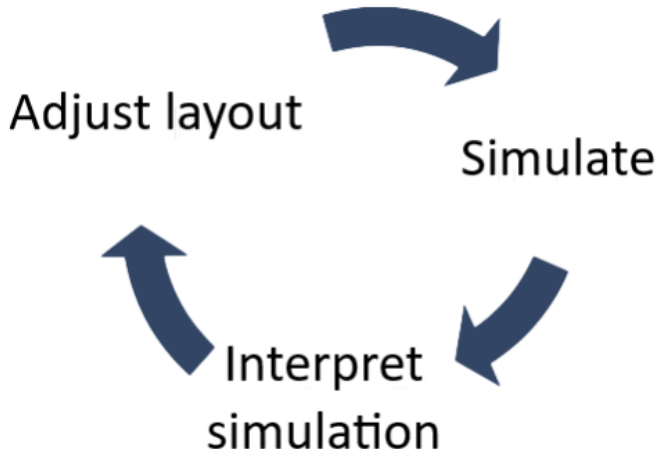


Figure 1.1: Layout design is conducted through a three-stage iterative process

Recent advancements in machine learning, particularly Graph Neural Networks (GNNs), have opened new pathways for predicting traffic dynamics more efficiently than the time-consuming simulations. Three previous studies at Kollmorgen have made progress in demonstrating the capabilities of GNN in this application. These studies will be described in detail in later section. Despite these contributions, there has not been clear analysis on whether the data used is sufficient for any conclusions about the applicability of GNN in full-scale AGV systems to be drawn. Challenges also remain in training and testing GNN models on more diverse data and large scale AGV systems, which limits their applicability to real use in AGV systems deployment. This gap under-scores the need for further research into the area.

1.2 Previous Research

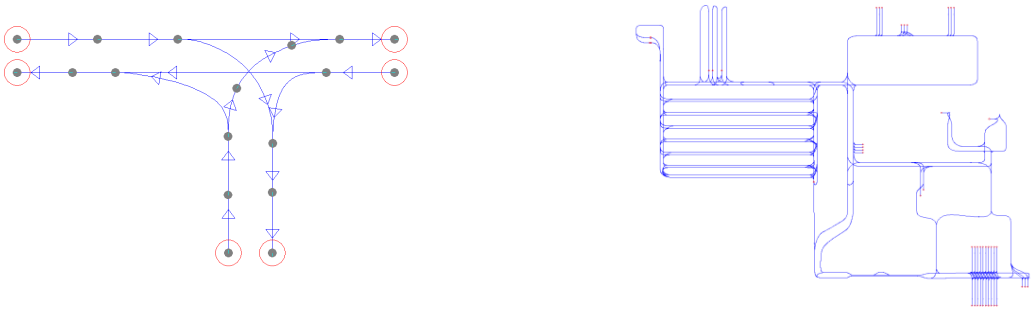
This thesis builds upon previous work conducted at Kollmorgen that explores the use of Graph Neural Networks (GNNs) for modeling Automated Guided Vehicle (AGV) systems. These works introduced graph-based representations of AGV layouts, proposed suitable features, and evaluated different learning objectives for predicting layout performance.

Lauri and Wiberg (2023) proposed a method for evaluating AGV layouts using supervised learning on both primal and dual graph representations [9]. They formulated a regression task with the target variable Mean Total Relative Frequency (MTRF), which aggregates simulation-based metrics, that reflect the effeciency of the flow of the orders in the system. Their approach demonstrated the feasibility of using GNNs for layout-level prediction and introduced a set of geometric and traffic-related features. However, their work was limited by a small dataset, which hindered model generalization. The applicability of their research is also limited because a graph-level prediction gives no insight about which segments in the layout are causing reduced performance in the AGV system. This is addressed in later research and in this thesis by using node-level predictions.

Velayutham (2024) extended this work by shifting the focus from graph-level prediction to node-level prediction [10]. Instead of using a single scalar metric, he framed the task as a node-level classification problem, predicting waiting time categories for each segment in the layout. The model incorporated a multi-relational dual graph structure with blocking rules to better capture realistic AGV systems. This approach improved spatial resolution in the evaluation and enabled the identification of specific problem segments. However, due to the classification-based formulation, the model was not sensitive to small layout improvements unless they caused a class shift in the segment, which reduced its usefulness for continuous optimization.

Zhang (2025) addressed this limitation by introducing a hierarchical model that first classifies nodes into zero or non-zero waiting time, followed by a regression step to predict continuous waiting times for the non-zero class [8]. This approach increased the resolution of predictions and allowed for finer feedback on layout changes. Zhang used a similar input structure as in previous work but investigated additional architectural choices and emphasized the need for larger, more representative datasets and further refinement of AGV-specific features. These recommendations form a central part of the foundation for the current thesis.

All the previous thesis used AGV systems that only contain one T-intersection as dataset to train the GNN models. Whereas a real AGV system is usually significantly larger and more complex, see Figure 1.2. This is being addressed in this thesis by only including real AGV systems in the dataset. The previous thesis have also not considered the previously mentioned concerns about the simulation tool. This thesis thoroughly investigates these concerns regarding simulation tool inconsistency and the choice of simulation duration.



(a) Example from the T-intersection data set in previous research.

(b) One of the AGV systems used in this thesis.

Figure 1.2: Visual comparison of data sets used in previous research at Kollmorgen, see [9], [10] and [8], and the data set used in this thesis. To our knowledge, this is the first time GNNs are applied to full scale AGV systems to predict congestion patterns.

1.3 Contributions

This study contributes to the development of methods for AGV layout evaluation by leveraging Graph Neural Networks (GNNs) and graph augmentation. We establish a baseline for simulation inconsistency by analysing the resulting waiting times from repeated simulation runs with identical inputs. This baseline serves as a reference for evaluating both the quality of GNN predictions and the effects of data augmentation.

To support model generalization, we propose a feature-level graph augmentation strategy that perturbs segment traverse times while preserving the underlying topology. We evaluate whether this method introduces sufficient data diversity beyond the inherent inconsistency of the simulation tool.

Based on Zhang's results, see Section 1.2, our approach applies a hierarchical GNN architecture comprising a classifier for detecting zero waiting time segments, and a regressor for predicting waiting times in non-zero waiting time segments. We contribute by evaluating this architecture's performance on augmented layouts of real AGV systems.

Additionally, we investigate the relationship between simulation duration and the quality of target values produced. Using stationarity tests, we determine a shorter simulation time that reliably captures relevant system behavior.

Together, these contributions support the use of machine learning as a scalable alternative to exhaustive simulation, with the goal of accelerating AGV layout design workflows. The contributions also guide future research to address current key challenges in developing such a model.

1.4 Thesis Outline

This thesis is structured as follows. Chapter 1 introduces the background and context of the work, includes a summary of previous research, and describes the contributions of the thesis. Chapters 2, 3, and 4 explain the main topics: AGV systems, simulation of AGV systems, and the use of graph data structures to represent AGV systems. These chapters also include the methodology used in the research.

Chapter 5 dives into the topic of GNN, and explains how it is used in this thesis, including architecture, parameter choices and evaluation metrics. Chapter 5 also includes an explanation of viable data augmentation methods and introduces the method used in this thesis. Chapter 6 presents the outcomes of the research. In Chapter 7, the results are discussed, possible limitations are addressed, and suggestions for future work are given. Chapter 8 concludes the thesis by summarizing the main findings.

2

AGV Systems

This chapter explains the structure of AGV systems by describing layout properties, the role of traffic rules, and how layout design impacts congestion.

2.1 Layout as a Virtual Road Network

A layout is a virtual road network representing the warehouse structure. It consists of one-way segments and points that connect those segments, as well as functional locations such as load stations and unload stations. An example showing how complex the layout can be is shown in Figure 2.1.

Segments carry the following attributes:

- Direction
- Allowed traverse speed for AGVs
- Spatial coordinates that define the precise path the AGV should follow

The term *traverse time* refers to the time it takes for an AGV to move through a segment, incorporating both spatial information and allowed speed. Each segment also specifies whether the AGV should move forward or in reverse, an essential feature to accurately control forklift operations, where reverse maneuvers are often required in narrow aisles or during pallet handling.

Each point in the layout can have several attributes. For example, a point may be configured to block access if another AGV is currently located on a connected segment, helping to prevent collisions or traffic congestion (more on these traffic rules in Section 2.3.

Load- and unload stations are special types of points that AGVs use to access the goods that need to be transported. These points are the points used as start- and end points in an order flow (explained in Section3.1).

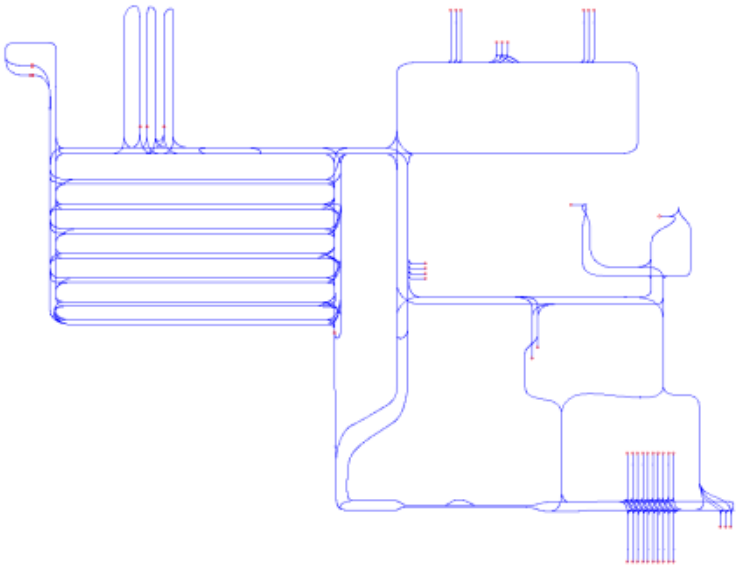


Figure 2.1: A layout representing the complexity of a real-world AGV system

2.2 Traffic Congestion

Congestion in AGV systems can occur when multiple vehicles attempt to access conflicting segments or intersections simultaneously, or as a result of restrictive traffic rules that block movement in key areas. If not addressed, such congestion can propagate through the network, leading to delays (waiting times) and reduced system efficiency. To mitigate these issues, a fleet management system coordinates AGV movement by assigning routes and tasks, enforcing traffic rules, and prioritizing actions to prevent deadlocks and maintain flow.

However, congestions in an AGV system are still dependant on the layout design. A system with functioning fleet management and traffic rules implementation can still be suboptimal and prone to congestion if designed poorly.

To optimize the layout design, areas in the layout that frequently cause congestion need to be identified. The simulation tool helps identifying these areas by revealing waiting times per segment over simulation time. A GNN capable of predicting segment wait times would help identify areas of high congestion more rapidly.

2.3 Traffic Rules

There are four main traffic rules used in AGV systems: point and segment blocking, occupied cluster, Ackpoint cluster, and order priority rules. Typically, these traffic rules exist to prevent collisions but can be used for other purposes, one example is illustrated in Figure 2.2.

Figure 2.2: Illustration of a traffic rule: On the left, two vehicles risk colliding due to unrestricted movement. On the right, a traffic rule is applied that blocks segment (2,3) when a vehicle occupies point 4. This rule successfully prevents the collision by prohibiting access to segment (2,3).

In theory, point and segment blocking prevents AGV1 from entering segment A because segment B is occupied by AGV2. This rule applies even if segment A is completely free, and regardless of the distance between segments A and B. Essentially, the blocking mechanism ensures that AGVs do not enter segments or points that could lead to conflicts due to nearby occupied segments or points.

3
Simulation Of AGV Systems

This chapter describes the inputs and outputs in AGV system simulations, with focus on how the order flow and layout are used as inputs to generate segment-level waiting times as outputs. The chapter introduces methodology for evaluating simulation consistency. It also includes assessing the stationarity of segment-level congestion trends and defining a simulation duration that balances shorter simulation time that still reflects long-term congestion patterns of the system.

3.1 Inputs and Outputs: Order Flow and Layout as inputs, and waiting times as outputs

An order flow is a list of tasks meant to simulate a certain number of orders being carried out each hour, with the same pattern repeating every hour. Each task/order requests an AGV to move from one station to another. The order flow, together with the layout information, is used in Kollmorgen's software to simulate the whole system and record waiting times for each segment in the system over time.

3.2 Simulation-Software Consistency

The consistency of the simulation software refers to its ability to produce similar outputs when provided with identical inputs. In the context of AGV system simulations, this means that repeated simulations of the same layout, with the same initial conditions and order flow, should result in similar waiting times per segment (see Section 4.2.2). While some degree of inconsistency is expected due to inherent stochastic elements in the simulation process (e.g. dynamic routing, or event timing), a consistent simulation tool should minimize this noise and yield results that are sufficiently stable to represent the same congestion patterns in all simulations.

Measuring the inconsistency of the simulation software is important in order to establishes a baseline for the GNN performance. The GNN is trained to replicate the results of simulation software, and cannot be expected to outperform the simulation software itself in replicating its own results in different runs.

To assess the consistency of the simulation software, each AGV system is simulated three times with the exact same settings. The results of every pair of runs are compared, and the evaluation metrics Mean Absolute Error (MAE) and F1-score

are calculated. These metrics will later be explained in detail in Section 5.3.3 and Section 5.3.2. These same metrics are used to evaluate the performance of the GNN, allowing for the previously mentioned baseline to be achieved.

3.3 Interpreting Segment Waiting Times in AGV Simulations

3.3.1 Segment Waiting Times and Congestion Patterns

In the simulation, the waiting time for each segment depends on how long the system is simulated. Segment waiting times generally increase over time, but segments prone to high congestion increase more rapidly. Initial conditions may cause temporary congestion and rapid increases in waiting times in areas not otherwise prone to congestion. Identifying long-term congestion requires confirming that the system's behavior has stabilized. Meaning waiting times are caused by recurring congestion that will continue in the long term. In time series terminology, this corresponds to the concept of *weak stationarity*.

3.3.2 Stationarity in Time Series

In time series analysis, the assumption of stationarity plays a crucial role in both the theoretical development and practical application of many statistical models. A stochastic process $\{X_t\}$ is said to be *weakly stationary* (or second-order stationary) if it satisfies two key conditions [7]:

1. The mean function $\mu_X(t) = \mathbb{E}[X_t]$ is constant over time, i.e.,

$$\mu_X(t) = \mu \quad \text{for all } t.$$

2. The autocovariance function $\gamma_X(t+h, t) = \mathrm{Cov}(X_{t+h}, X_t)$ depends only on the lag h and not on the specific time point t, i.e.,

$$\gamma_X(t+h, t) = \gamma(h) \quad \text{for all } t, h.$$

These conditions imply that the statistical properties of the process, the mean, variance, and autocovariance are invariant over time. In particular, the variance is constant because $\gamma(0) = \mathrm{Var}(X_t)$ is independent of t. Weak stationarity is less restrictive than strict stationarity (which requires invariance of all joint distributions) which rely on consistent second-order properties for estimation and forecasting.

3.3.3 Augmented Dickey Fuller Test

To determine whether a time series is stationary, the Augmented Dickey Fuller (ADF) test is employed. The test addresses the problem of unit roots in autoregressive models, which are indicative of nonstationarity. The null hypothesis H_0 states that the time series has a unit root, implying nonstationarity, while the alternative hypothesis H_1 suggests stationarity.

The ADF test involves estimating a regression of the form:

$$\Delta X_t = \phi_0 + \phi_1 X_{t-1} + \sum_{i=1}^{p-1} \phi_{i+1} \Delta X_{t-i} + \varepsilon_t \tag{3.1}$$

where ΔX_t is the first difference of the series, and lagged differences are included to account for higher-order serial correlation. The key parameter of interest is ϕ_1, which reflects the presence of a unit root. The test statistic is a modified t-ratio comparing the estimated ϕ_1 to its standard error.

Due to nonstandard asymptotic behavior under the unit root null hypothesis, the test statistic is compared to critical values derived from the Dickey Fuller distribution. If the test statistic is less than the critical value (e.g., -2.86 at the 5% level), the null hypothesis is rejected, suggesting the series is stationary. [7]

Commonly in python implementations a p value is calculated and compared to the Dickey Fuller distribution automatically, thus just giving if the unit root hypothesis is rejected or not, making it simple to use.

3.4 Choice of Simulation Time

The choice of simulation time used when creating target values for the GNN comes with a trade-off. Shorter simulation times allow for simulating more systems and creating a larger data set, but might be too short to result in waiting times that reflect the long-term congestion patterns of the system, as explained in Section 3.3. In order to determine a shorter, yet sufficient simulation time. For each AGV system, three extended simulations are performed to ensure that the long-term behavior of the system is captured. Based on consultations with domain experts, a 14-hour simulation is considered abundant to reflect the long-term behavior of the systems.

To ensure that a shorter simulation time is representative of the long-term behavior, the results of the extended simulation need to show:

1. **Stationary Rate of Change**: The rate of change of cumulative wait times, more precisely defined below, is stationary indicating that a shorter simulation does not cause the data to be less representative of the system's long-term behavior. The stationarity will be determined using ADF test, defined in equation (3.1).
2. **Representation of Non-Zero Waiting Times segments**: The shorter simulation time needs to include most of the non-zero waiting time segments that appear in the extended simulation. The threshold for this is set to 90% of the non-zero waiting time segments. Specifically how the simulation time is calculated based on the threshold is shown in equation (3.4).

The cumulative blocking time $W(t)$ is defined as the sum of all completed blocking durations across all segments, up to time t. A single segment $s_i \in \mathcal{S}$ may experience multiple blocking intervals during the simulation. Let $\mathcal{B}_i$ be the set of blocking intervals for segment s_i, where each interval $b_{ij} \in \mathcal{B}_i$ has a start and end time t_{ij}^{start}

and t_{ij}^{end}, respectively.

The total blocking time for segment s_i up to time t is given by:

$$\begin{aligned} w_i(t) &= \sum_{\substack{b_{ij} \in \mathcal{B}_i \\ t_{ij}^{\text{end}} \leq t}} \left(t_{ij}^{\text{end}} - t_{ij}^{\text{start}} \right) \\ W(t) &= \sum_{s_i \in \mathcal{S}} w_i(t) \end{aligned} \tag{3.2}$$

This formulation accounts for all completed blocking intervals across all segments, where each interval contributes to the total only if it ends on or before time t. Blocking intervals that are ongoing or unresolved (e.g., due to deadlocks) are not included until they are completed, resulting in a plateau in the cumulative blocking time curve during such events.

To quantify the rate of change in the system's cumulative wait time over time, we apply the following forward-looking finite difference formula:

$$R_\tau = \frac{W_{\tau+k} - W_\tau}{\Delta t} \tag{3.3}$$

In this expression, R_τ represents the estimated rate of change at time step τ, calculated over a forward-looking window of k time steps. The cumulative wait times of all segments W_τ are assumed to be sampled at constant intervals Δt, allowing for a simplified and uniform approximation of the derivative.

Formally, let N_{wait} denote the total number of segments with non-zero waiting time in the full simulation, and let $N(t)$ represent the cumulative number of such segments observed up to time t. The minimum required simulation time, t_{90}, is then defined as:

$$t_{90} = \min \left\{ t \;\middle|\; \frac{N(t)}{N_{\text{wait}}} \geq 0.90 \right\} \tag{3.4}$$

This ensures that the shorter simulation duration still includes a representative majority of the waiting behavior seen in the extended run, supporting the validity of comparing system performance based on shorter simulation time.

4

AGV Systems as Graph Data Structures

This chapter presents the graph-based representation of AGV systems. It covers basic graph concepts, the structure of directed multi-relational graphs, an overview of the provided AGV systems, and how layouts, order flows, and waiting times are represented as nodes, edges, and attributes in the graph format.

4.1 Graph Data Structures

The graph data structure is an effective method for representing data with relational information. It consists of a set of nodes, with connections between different nodes represented by edges. Both nodes and edges can include various attributes.

Graphs can be either directed or undirected. In a directed graph, edges have a direction, indicating a one-way relationship between nodes. Conversely, undirected graphs have edges that represent bidirectional relationships.

Graphs are extensively used in various applications such as social networks, road maps, and network routing algorithms. They enable efficient representation and manipulation of complex relational data.

Formally, a graph $G = (V, E)$ is defined as a set of nodes V and a set of edges E. An edge connecting $u \in V$ to $v \in V$ is denoted as $(u, v) \in E$ [6].

4.1.1 Directed Multi-Relational Graphs

Directed multi-relational graphs are a sophisticated extension of traditional graph structures, designed to represent complex relationships between nodes. In these graphs, edges not only have a direction but can also represent multiple types of relationships between the same pair of nodes.

In a directed multi-relational graph, each node represents an entity, and directed edges indicate the direction of the relationship between entities. Unlike simple directed graphs, multi-relational graphs allow multiple edges between the same pair of nodes, each edge representing a different type of relationship. These relationships

can be categorized and labeled to provide more detailed information about the connections

Formally, a directed multi-relational graph $G = (V, E, R)$ consists of:

- A set of nodes V.
- A set of directed edges E, where each edge (u, v, r) connects node u to node v with a relationship type r.
- A set of relationship types R.

Each edge (u, v, r) indicates that there is a directed relationship of type r from node u to node v.

Directed Multi-Relational graphs will be used to represent the AGV systems as graph data structures. More about this representation and how its implemented in Section 4.2.

4.2 Representation of AGV Systems as Graph Data Structures

The AGV Systems are represented as directed multi-relational Graphs in which segments are nodes, and transition relations and blocking relations as edges. The following section explains this representation in detail, and it is used as input data and target values for a GNN.

4.2.1 Representing Layouts and Order Flows

Based on previous research conducted at Kollmorgen, the inputs to the GNN, which consist of information about layouts and order flows, are represented as multi-relational directed graphs.

In these representations, each segment in the layout is depicted as a node in the graph, with three attributes. Active usage, which shows how often a segment is used during the simulation based on the order flow. Another is blocking probability, which shows how likely it is that a vehicle will get blocked within a specific segment, based only on first order relations to other segments and points. Lastly Traverse time, which refers to the time it takes for an AGV to move through a segment, incorporating both spatial information and allowed speed. These attributes are calculated using an internal tool at Kollmorgen.

The nodes are connected by two types of directed edges: transition edges and blocking edges. A transition edge from node u to node v indicates that segment u's endpoint in the layout is segment v's start point, meaning the segments are directly connected. Conversely, a blocking edge from node u to node v signifies a traffic rule that prevents AGVs from traversing segment u if segment v, or its endpoint, is occupied by another AGV.

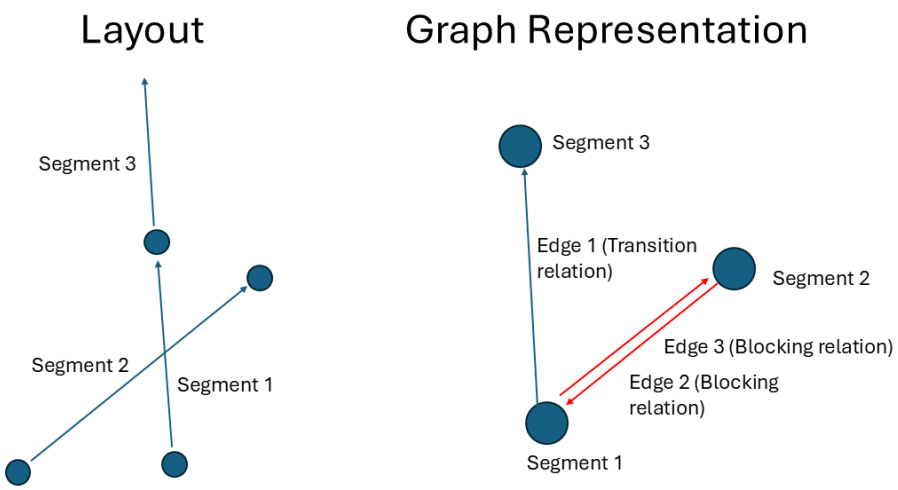


Figure 4.1: Representation of Transition and Blocking Relations

Description: The figure illustrates a layout with three segments (Segment 1, Segment 2, Segment 3) on the left side and their corresponding graph representation on the right side. In the graph representation:

- **Segment 1** connects to **Segment 3** via an edge labeled "Edge 1 (Transition relation)".
- **Segment 2** blocks **Segment 1** which is indicated by a blocking relation "Edge 2 (Blocking relation)"
- **Segment 1** blocks **Segment 2** which is indicated by a blocking relation "Edge 3 (Blocking relation)" line.

This visual representation aids in understanding how different segments are interconnected through transitions and how certain segments can block others based on specific traffic rules.

4.2.2 Target Values: Waiting times

The target values are the waiting times for each segment in the layout. These are represented as node attributes in the graph data structure. Given that the layouts used for training will be simulated for different amounts of time, Depending on the outcome of the simulation time analysis, the target values are normalized to waiting times per simulated hour.

4.3 System Setup and Assumptions

Three AGV systems were provided by Kollmorgen in order to conduct this research, this means that the results can not be seen as general but the methodology will be tested on these systems. The AGV systems represent the behavior of a normally

functioning AGV system. To simplify the analysis and focus on core functionalities, certain complex traffic rules were intentionally omitted. The order flows in the systems were designed and tested by the domain experts to capture relevant system dynamics. For more information about order flows and layouts in AGV systems, see Section 3.1.

For confidentiality and commercial sensitivity, the systems have been anonymized to allow its inclusion in this report. The following table gives an overveiw of the complexity and scale of the systems.

System	Nr. Segments	Nr. Stations	Nr. AGVs	Nr. Active Usage Segments
1	643	28	20	391
2	478	38	5	325
3	2988	695	10	2678

Table 4.1: Provided AGV systems from Kollmorgen

5

Graph Neural Networks in AGV System

This chapter introduces GNNs and explains their use for predicting waiting times in AGV systems. It covers node-level classification and regression, GNN concepts such as permutation invariance and message passing, and aggregator functions. The chapter also presents loss functions, data augmentation techniques, the hierarchical GNN framework used in this work, and the evaluation methods applied to measure model performance.

5.1 Graph Neural Networks

A GNN uses machine learning approaches to make predictions about an input graph. The predictions could be graph-level predictions, or node-level predictions. The predictions that are relevant to this thesis are node-level classification and node-level regression (since the segments whose waiting time we are to predict are represented as nodes, see Section 4.2.1).

Node classification makes use of each node's embedded attributes, as well as relational attributes and neighbors attributes to make a prediction on the likelihood that the node belongs to a certain class. In the case of this thesis, this approach is used to classify whether a node (i.e. a segment) is a zero-waiting time node or not, more on this topic later.

Similarly, Node-level regression makes use of each node's embedded attributes, as well as relational attributes and neighbors attributes. With the difference being that the regressor makes predictions about a node's attribute, such as waiting time in this instance.

5.1.1 Permutation Equivariance and Invariance

The main advantage of a GNN, compared to other neural networks such as Convolutional Neural Networks (CNNs), is that GNNs rely on relational information rather than positional information. In mathematical terms, GNN models can be permutation equivariant or invariant.

A function f is permutation invariant if the output of the function remains the same for all permutations of the input. Conversely, a function f is permutation equivariant if the output of f is permuted in a consistent manner when the input is permuted.

In mathematical terms, this can be expressed as:

$$f(PAP^T) = f(A) \quad \text{(Permutation Invariance)}$$

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

where P is a permutation matrix.

5.1.2 Neural Message Passing in GNNs

GNNs rely on message passing to utilize the attributes of the nodes and their relations, effectively representing the graph's relational structure. The message passing starts with a hidden embedding for each node based on its features. This embedding is then updated based on the embeddings of neighboring nodes. The message passing process can be done iteratively[6, pp. 54].

Message passing in GNNs involves the exchange of information between nodes through their edges. Each node aggregates information from its neighbors to update its own embedding. This process captures the dependencies and relationships within the graph, allowing the GNN to learn meaningful representations[6, pp. 54-55].
The message passing process typically consists of the following steps:

1. **Initialization:** Each node v is initialized with a hidden embedding $h_v^{(0)}$ based on its features x_v.
2. **Aggregation:** At each iteration t, each node v aggregates messages from its neighbors $\mathcal{N}(v)$. The aggregated message $m_v^{(t)}$ is computed using an aggregation function AGG.
3. **Update:** The node's embedding $h_v^{(t)}$ is updated using an update function UPDATE, which combines the node's previous embedding $h_v^{(t-1)}$ and the aggregated message $m_v^{(t)}$.
4. **Iteration:** Steps 2 and 3 are repeated for a fixed number of iterations or until convergence.

Let $h_v^{(t)}$ be the hidden embedding of node v at iteration t, and x_v be the initial features of node v. The message passing process can be mathematically defined as follows:

- **Initialization:**
$$h_v^{(0)} = x_v$$
- **Aggregation:**
$$m_v^{(t)} = \text{AGG}\left(\{h_u^{(t-1)} : u \in \mathcal{N}(v)\}\right)$$
where $\mathcal{N}(v)$ denotes the set of neighbors of node v.

- **Update:**

$$h_v^{(t)} = \text{UPDATE}(h_v^{(t-1)}, m_v^{(t)})$$

The aggregation function AGG and the update function UPDATE can vary depending on the specific GNN architecture. Common choices for AGG include sum, mean, and max, while UPDATE is often implemented as a neural network layer such as a fully connected layer or a recurrent neural network.

Consider a basic GNN with sum aggregation and a neural network update function. The message passing process can be defined as:

$$m_v^{(t)} = \sum_{u \in \mathcal{N}(v)} h_u^{(t-1)}$$

$$h_v^{(t)} = \sigma\left(W \cdot [h_v^{(t-1)} \| m_v^{(t)}]\right)$$

where σ is an activation function, W is a trainable weight matrix, and $\|$ denotes concatenation. After the last message-passing layer, *post-processing layers* may be applied; these are additional neural network layers that operate on the final node embeddings to produce predictions.

Activation Functions: Activation functions introduce non-linearity into the model, enabling it to capture complex patterns in the data. Common examples include ReLU, sigmoid, and tanh. In this thesis, the Parametric Rectified Linear Unit (PReLU) is used. PReLU is defined as:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0, \\ ax, & \text{if } x < 0, \end{cases}$$

where a is a learnable parameter that allows the model to adaptively control the slope for negative inputs. This flexibility can improve convergence and performance compared to a fixed ReLU. [17]

5.1.3 Aggregator Functions In GNN

Different types of GNNs are mainly defined by their aggregator function. The choice of the aggregator function determines how information from neighboring nodes are passed in the message passing phase. In the previously mentioned example, where simply a sum function is used, nodes with high number of neighbors will result in high embedding magnitude relative to other nodes. In some cases, this can be unstable and misrepresentative of the graph. In this chapter, a selection of popular choices of aggregator functions will be presented. Based on the results of Boshens work [8], the aggregators that performed best are GraphSAGE for the classifier part, and Basic GNN for the regressor part.

5.1.3.1 GraphSAGE

GraphSAGE (Graph Sample and Aggregation) is a framework for inductive representation learning on large graphs. It leverages node feature information and efficiently generates embeddings for previously unseen data. There are multiple variants of aggregator functions for the GraphSAGE architecture. In this thesis, the mean variant is considered.

The mean aggregator function in GraphSAGE works by taking the mean of the neighboring nodes' embeddings to update the target node's embedding. This approach helps to mitigate the issue of high embedding magnitudes caused by nodes with many neighbors, providing a more stable and representative embedding.

The mean aggregator function in GraphSAGE can be described as follows:

$$h_v^{(k)} = \sigma\left(W \cdot \text{MEAN}\left(h_v^{(k-1)}, \{h_u^{(k-1)} : u \in \mathcal{N}(v)\}\right)\right)$$

where σ is an activation function, W is a trainable weight matrix.

5.1.4 Loss Functions for Training GNN models

In order to train the GNN on a set of training data, a training loss is calculated for every epoch in the training phase. This loss is back-propagated to the trainable weight matrix and updates the model parameters via gradient descent. The choice of loss function depends on the specific task (e.g., node classification, relation prediction, or graph classification). In the case of this thesis, the tasks are node-level classification and node-level regression.

5.1.4.1 Cross-Entropy Loss

In node-level classification, *cross-entropy loss* is the most commonly used loss function. Cross-entropy is preferred because it quantifies the difference between the predicted probability distribution and the true class distribution, penalizing confident incorrect predictions more heavily than less certain ones. This encourages the model to produce well-calibrated probability outputs and make correct class predictions with high confidence [11, 12].
The cross-entropy loss for multi-class classification is defined as:

$$\mathcal{L}_{\text{CE}} = -\sum_{i \in \mathcal{Y}_{\text{train}}} \sum_{c=1}^{C} y_{ic} \log(\hat{y}_{ic}) \tag{5.1}$$

where:

- $\mathcal{Y}_{\text{train}}$ is the set of nodes used for training,
- C is the number of classes,
- $y_{ic} \in \{0, 1\}$ is the ground truth label indicating whether node i belongs to class c,

- $\hat{y}_{ic}$ is the predicted probability that node i belongs to class c, usually obtained via a softmax function.

This formulation ensures that the loss is minimized when the predicted probability is close to 1 for the correct class and close to 0 for the incorrect ones.

5.1.4.2 Mean Squared Error Loss

In node-level regression, the most commonly used loss function is the *Mean Squared Error (MSE)*. MSE is preferred because it directly measures the average squared difference between the predicted values and the actual target values. It is a natural choice for regression problems where the goal is to predict continuous values, as it penalizes larger errors more heavily than smaller ones and is differentiable, making it suitable for gradient-based optimization [15, 12].
The MSE loss is defined as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{|\mathcal{Y}_{\text{train}}|} \sum_{i \in \mathcal{Y}_{\text{train}}} (y_i - \hat{y}_i)^2 \tag{5.2}$$

where:

- $\mathcal{Y}_{\text{train}}$ is the set of nodes used for training,
- y_i is the true continuous target value for node i,
- $\hat{y}_i$ is the predicted value for node i.

This formulation ensures that the loss is minimized when the predicted values are as close as possible to the true target values across all training nodes. The squaring of the error emphasizes larger discrepancies, which helps in driving the model to prioritize minimizing larger prediction mistakes.

5.1.5 Inference of a Trained GNN Model

Once a GNN model has been trained, inference involves applying the learned parameters to new input data in order to produce predictions. The structure of the graph (adjacency matrix A) and node features X are provided to the model, which then performs message passing for a fixed number of layers T, using the parameters $\Theta = \{W^{(t)}\}_{t=1}^{T}$ learned during training.
Let $H^{(0)} = X$ denote the initial node feature matrix. For each layer $t \in \{1, \ldots, T\}$, the node embeddings are updated as:

$$H^{(t)} = \sigma\left(\text{AGG}\left(A, H^{(t-1)}\right) W^{(t)}\right) \tag{5.3}$$

where:

- $H^{(t)} \in \mathbb{R}^{N \times d_t}$ is the matrix of node embeddings at layer t,
- $\sigma(\cdot)$ is a non-linear activation function (e.g., PReLU),
- $\text{AGG}(\cdot)$ denotes the neighborhood aggregation function,
- $W^{(t)}$ is the weight matrix for layer t,
- N is the number of nodes in the graph, and d_t is the embedding dimension at layer t.

After the final message-passing layer T, the output $H^{(T)}$ is mapped to predictions.

5.1.5.1 Node Classification:

For classification into C classes, the prediction for node i is given by:

$$\hat{y}_i = \text{softmax}\left(H_i^{(T)} W_{\text{cls}}\right) \tag{5.4}$$

where $W_{\text{cls}} \in \mathbb{R}^{d_T \times C}$ is the classification weight matrix and $H_i^{(T)}$ is the final embedding of node i.

5.1.5.2 Node Regression:

For regression tasks, the prediction for node i is:

$$\hat{y}_i = H_i^{(T)} W_{\text{reg}} + b_{\text{reg}} \tag{5.5}$$

where $W_{\text{reg}} \in \mathbb{R}^{d_T \times 1}$ and b_{reg} are regression parameters.
During inference, no parameter updates are performed; the weights Θ, W_{cls}, and W_{reg} are fixed, and only the forward pass through the network is executed to compute $\hat{y}_i$.

5.2 GNN construction

The structure of the GNN will be completely based on the work of previous master thesis at Kollmorgen. The best-performing GNN will be used on the new data set and evaluated.

5.2.1 Hierarchical Graph Neural Network Framework

Predicting waiting times for segments in the layout is formulated as a node-level regression problem. However, previous research has faced a problem of data imbalance where the model becomes biased towards predicting values close to zeros. Previous research has investigated multiple strategies to address this issue, with the most recent and effective approach described in Zhang's thesis [8]. This method, referred to as the *Hierarchical Graph Neural Network Framework*, is adopted in the present work due to its demonstrated ability to handle the zero-dominant target distribution.

The framework decomposes the prediction task into two sequential stages:

1. **Classification:** Determine whether each segment has a zero or non-zero waiting time.
2. **Regression:** For the segments classified as having non-zero waiting time, predict the actual waiting time value.

These stages are implemented using two separate node-level GNN models, one classifier and one regressor, both trained independently. The classifier is trained as a binary node classification model, while the regressor is trained as a node-level regression model.

5.2.1.1 Inference Phase in Hierarchical Framework

During inference (Figure 5.1), the process proceeds as follows:

1. The full set of segment nodes in the layout is passed through the classifier.
2. Segments predicted as having zero waiting time are assigned a final prediction of zero.
3. Segments predicted as having non-zero waiting time are passed to the regressor to estimate the waiting time value.

The final prediction for the entire layout is obtained by combining the zero predictions from the classifier with the continuous predictions from the regressor. Description of how trained models make predictions is presented in Section 5.1.5.

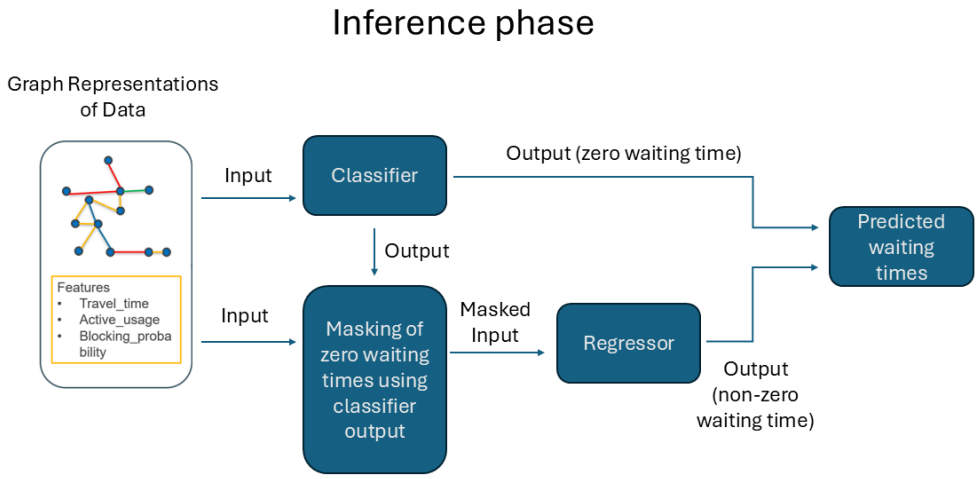


Figure 5.1: Inference phase of the Hierarchical Graph Neural Network Framework

5.2.1.2 Training Phase in Hierarchical Framework

Unlike inference, the regressor's training phase (Figure 5.2) does not depend on the classifier's output. Instead, ground truth labels are used to identify the non-zero waiting time segments, which are then used to train the regressor. This approach ensures that the regressor learns from correct positive samples rather than being influenced by classification errors during training.

Both models receive the same graph-structured input features for each node, such as travel time, active usage, and blocking probability. Backpropagation is performed separately for the two models, using:

- *Cross-Entropy Loss* (defined in Section 5.1.4.1) for the classifier.
- *Mean Squared Error (MSE) Loss* (defined in Section 5.1.4.2) for the regressor.

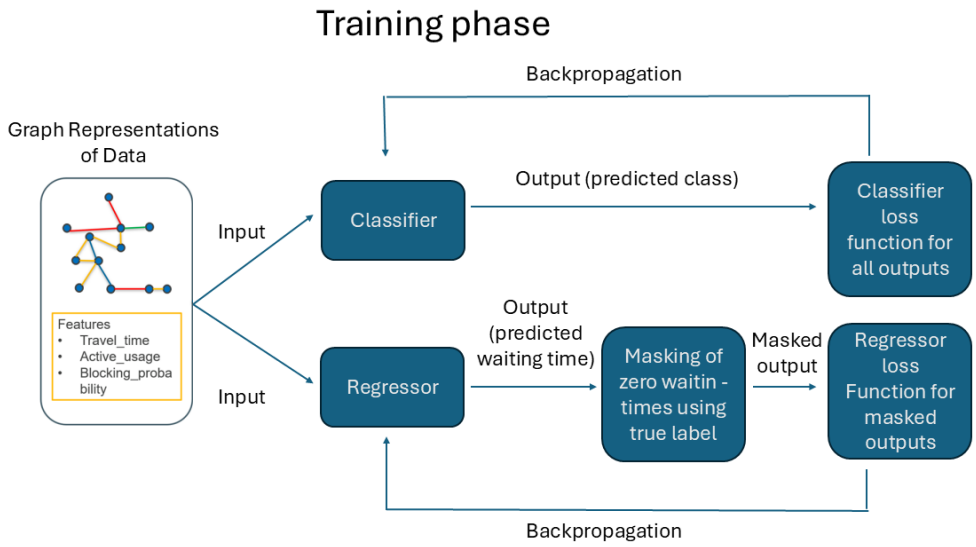


Figure 5.2: Training phase of the Hierarchical Graph Neural Network Framework

The hyper-parameters for each model are summarized in Tables 5.1 and 5.2. These parameters are all defined in Section 5.1.

M_p	Post	Hidden Size	Activation	Aggregator	Loss Function
4	1	32	PReLU	GraphSAGE	Cross-Entropy Loss

Table 5.1: Classifier settings. M_p denotes the number of message-passing layers. Post is the number of post-processing layers after message passing. Hidden Size is the dimensionality d of the node embeddings.

M_p	Post	Hidden Size	Activation	Aggregator	Loss Function
5	1	32	PReLU	Basic GNN	Mean Squared Error

Table 5.2: Regressor settings. M_p denotes the number of message-passing layers. Post is the number of post-processing layers after message passing. Hidden Size is the dimensionality d of the node embeddings.

5.3 Validation and Evaluation

The GNN models will be evaluated based on prediction accuracy. To ensure generalization, evaluation methods such as hold-one-out cross validation will be used. Prediction results will also be averaged over multiple runs to make sure that the results are representative.

To rigorously evaluate the performance of the developed AI models, a combination

of classification and regression metrics is employed. The choice of metrics is based on the nature of the prediction task and is intended to provide a comprehensive assessment of each model's strengths and limitations.

5.3.1 Leave-One-Out Cross-Validation

To evaluate the performance of the Graph Neural Network (GNN) model, we apply a leave-one-out cross-validation (LOOCV) strategy. This approach is a case of k-fold cross-validation where k equals the total number of layouts in the dataset. It is a recognized and widely used method for evaluating machine learning models, particularly when working with small datasets [4].

In this setup, the model is trained and evaluated over k iterations. In each iteration, one layout is held out as the validation set, while the remaining k - 1 layouts are used for training. This process is repeated until every layout has been used once as the validation sample. As a result, each layout contributes to both training and validation, ensuring a comprehensive and robust assessment of the model's generalization performance.

LOOCV provides a reliable estimate of model performance by systematically testing on each individual data point while training on the rest. However, despite its advantages, LOOCV can lead to high variance in the results if the model is sensitive to fluctuations in the training data. It is also computationally intensive, as the model must be trained once for each layout. [4]

To further improve the reliability of the evaluation and mitigate the effects of randomness in neural network training (such as weight initialization and dropout), the entire leave-one-out procedure is repeated multiple times, three runs with different random seeds. The final evaluation results are then obtained by averaging the performance metrics across all runs.

5.3.2 Classification Metrics

For classification tasks, we utilize the *confusion matrix* and the *F1 score* as primary evaluation tools. The F1 score is the harmonic mean of precision and recall, and it is particularly well-suited for scenarios where class distributions are imbalanced or where both false positives and false negatives carry a cost. Unlike accuracy, which can be misleading when one class dominates, the F1 score captures the balance between identifying relevant instances and minimizing incorrect predictions. This makes it a more informative metric for evaluating the generalization ability of the model in tasks where class imbalance or uneven prediction difficulty is present [5].

The confusion matrix provides a detailed breakdown of prediction outcomes in terms of:

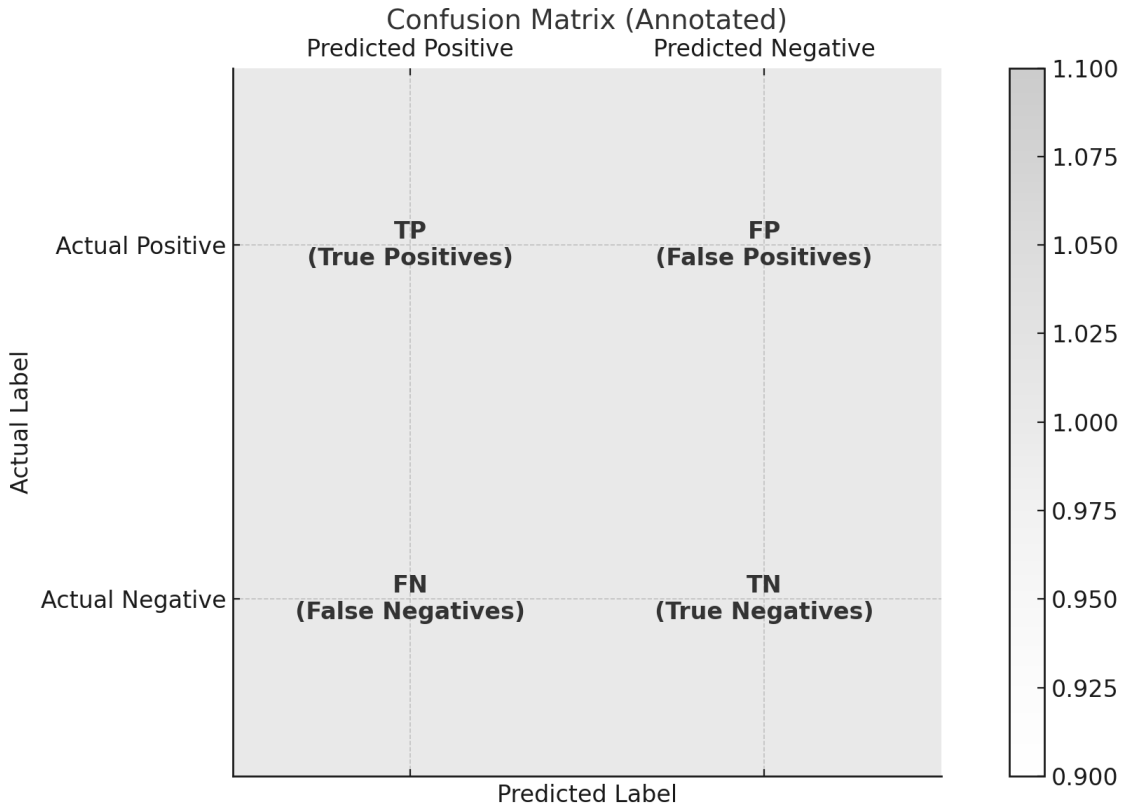


Figure 5.3: Confusion matrix

- **True Positives (TP):** Correctly predicted positive instances
- **False Positives (FP):** Incorrectly predicted positive instances
- **True Negatives (TN):** Correctly predicted negative instances
- **False Negatives (FN):** Incorrectly predicted negative instances

From this, we derive precision, recall, and the F1 score:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.6}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.7}$$

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.8}$$

These metrics enable a more nuanced evaluation of model performance than accuracy alone, especially when the dataset contains unequal class representation.

5.3.3 Regression Metrics

For regression tasks, we assess prediction quality using *Mean Absolute Error (MAE)*:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{5.9}$$

Here, y_i denotes the true target value (waiting time of segment i per hour of simulation), $\hat{y}_i$ the predicted value, and n the total number of segments with waiting time, in both predictions and true values.

5.4 Graph Data Augmentation Techniques for GNNs

Data augmentation is a well-established technique in domains like computer vision, where transformations such as image flipping or rotation help improve generalization. However, graphs are fundamentally different: their non-Euclidean structure and relational nature make conventional augmentation methods inapplicable. This has led to the development of specialized augmentation techniques tailored for graph-structured data, particularly for Graph Neural Networks.

In the literature, augmentation techniques are typically classified into two main categories: topology-level augmentation and feature-level augmentation. These categories reflect whether the augmentation operates on the structure of the graph or on the attributes associated with its components. [16]

5.4.1 Topology-Level Augmentation

Topology-level augmentation involves modifying the structure of the graph, such as its nodes, edges, or subgraph composition. These methods generate alternative structural views of the original graph, which can help models learn representations that are invariant to specific structural perturbations. Common topology-level techniques include:

- **Edge perturbation**, which involves randomly adding or removing edges in the graph.
- **Node dropping**, where certain nodes and their incident edges are temporarily removed.
- **Subgraph sampling**, in which smaller portions of the graph are extracted for training purposes.
- **Graph coarsening**, which reduces the graph's complexity by merging nodes or edges.

While these methods can introduce useful diversity, they must be applied with caution in domains where structural changes can violate domain-specific constraints. For example, altering layout connectivity in an industrial system could lead to invalid states such as deadlocks or disconnected components. [16]

5.4.2 Feature-Level Augmentation

Feature-level augmentation operates on the attributes of nodes or edges without altering the graph's topology. These techniques aim to introduce variation in the semantic content of the graph while maintaining its structural integrity. Representative feature-level methods include:

- **Feature masking or dropout**, where random elements of the feature vectors are zeroed out.
- **Noise injection**, which adds stochastic perturbations (e.g., Gaussian noise) to the feature values.
- **Feature scaling or transformation**, which systematically adjusts feature magnitudes.
- **Feature shuffling**, where features are randomly redistributed among graph elements.

Feature-level augmentations are particularly advantageous in scenarios where structural consistency is critical. They allow for simulation of realistic variations without compromising the validity of the graph. [16]

5.5 Implementation of Data Augmentation

The input data can be divided into several variables that can be augmented: topology of the layout, traffic rules, traverse time, and order flows. The topology of the layout and traffic rules are represented in the connectivity of the graph, while traverse time and order flows are represented in node features. More details about these representations can be found in section 4.2.1.

Although augmenting the topology of the layout or traffic rules could be beneficial, it may lead to deadlocks in the system if not done carefully. This challenge makes automatic augmentation of these variables impractical for this thesis.

Traverse time depends on multiple relevant factors that can vary within a layout. Simply augmenting segment traverse times represents various possible changes that might occur during real layout editing processes. Changes such as altering a segment's shape, length, or travel speed all result in modifications to traverse time within graph representation. Therefore, augmenting traverse times is chosen as an effective method to mimic micro-changes within a layout.

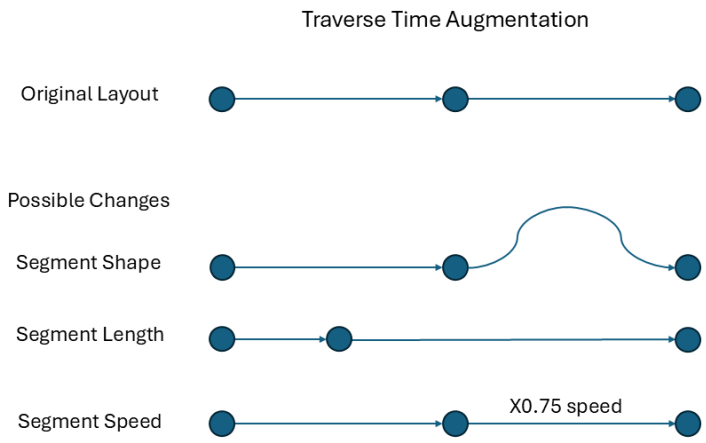


Figure 5.4: Traverse Time Augmentation.

Description: The figure illustrates different types of possible changes during layout editing that can be represented in traverse time augmentation: segment shape alteration, segment length modification, and segment speed adjustment.

Augmentation is applied to the layouts before being represented as graph data. A C# script performs this augmentation by modifying the traverse times of a subset of the segments in the layout. Specifically, the script takes two inputs: a fixed percentage of segments to augment and a constant augmentation factor.

In this thesis, 20% of the segments are randomly selected and augmented. For each selected segment $e_i \in \mathcal{E}_{\text{aug}} \subset \mathcal{E}$, its original traverse time t_i is scaled by a constant augmentation factor $\alpha = 0.2$, such that the new augmented traverse time t_i' is given by:

$$t_i' = t_i \cdot \alpha, \quad \forall e_i \in \mathcal{E}_{\text{aug}} \tag{5.10}$$

where:

- $\mathcal{E}$ is the set of all segments (edges),
- $\mathcal{E}_{\text{aug}} \subset \mathcal{E}$ is the set of segments selected for augmentation, with $|\mathcal{E}_{\text{aug}}| = 0.2 \cdot |\mathcal{E}|$,
- t_i is the original traverse time for segment e_i,
- $\alpha = 0.2$ is the augmentation factor.

The values 20% and $\alpha = 0.2$ are arbitrary chosen to analyze the change caused in waiting times in the system by this type of augmentation.

To analyze whether the selected augmentation method results in data diversification that is larger than the diversification caused by the inconsistency of the simulation tool, MAE and F1-score are used. These metrics measure the difference of the results between an original layout and its augmentations, as well as between different

augmentations of the same layout. This ensures that the augmentation method used has real impact on the dataset created.

6
Results

In the following section, the results of the methods applied on the three systems are presented. This includes the results of analyzing the simulation tool and assessing its consistency, the diversification of the data caused by augmentation and the simulation time chosen for each system based on analyzing extended simulations. Lastly, the performance of the GNN, divided into classifier part and regressor part.

6.1 Variation Across Identical Simulation Runs

In the following section, the results of analyzing the consistency of the simulation tool provided by Kollmorgen are presented. These simulations are run for an extended period of time, 14 hours. The outputs of a simulation are the labels of the segments, zero waiting time segments or non-zero waiting time segments, and the total waiting time for each segment. MAE here represents the mean average error of segment waiting time between two simulation runs of the same layout with the same order flow. F1-score represents the consistency of the final labels of the segments.

System 3 entered a deadlock during simulation, making it invalid for this analysis. Results that show this are presented in 6.3.

System	Runs Compared	MAE (s/h*segment)	F1-score
1	1 and 2	0.98	0.93
1	1 and 3	7.56	0.88
1	2 and 3	8.23	0.93
2	1 and 2	6.81	0.98
2	1 and 3	7.66	0.98
2	2 and 3	1.33	1.00
3	1 and 2	N/A	N/A
3	1 and 3	N/A	N/A
3	2 and 3	N/A	N/A

Table 6.1: The table shows the MAE values calculated according to equation 5.9, comparing the hourly waiting times between repeated simulations of the same layout. On average, the hourly wait time differs by approximately 1–10 seconds between runs. No results are reported for System 3, as all three runs failed to complete due to simulation issues identified as deadlocks. The results from system 3 can be seen in Figure 6.3

6.2 Effect of Augmentation on Simulation Outputs

In the following sections, the results of the augmentation analysis are presented. First, the results of comparing the simulation outputs of an original layout with the the simulation outputs of its augmentations. Similar to the previous section, MAE represents the mean average error of segment waiting time, here averaged from comparing the original layout to all of its three augmentations. F1-score represents the similarity of the final labels of the segments, also averaged from comparing the original layout to all three augmentations. The second part is the result of comparing the simulation outputs of different augmentations with each others using the same metrics. This is averaged from comparing all combinations of two different augmentations of the same layout (three augmentations give three different combinations for comparison)

System	MAE (s/h*segment)	F1-score
1	189,53	0.52
2	28.83	0.71
3	N/A	N/A

Table 6.2: Similarity between an layout and its augmentations, values are averaged from comparing the results from three simulation runs of an layout to all three of its augmentations. MAE calculated according to equation (5.9). This table is intended to show that augmentations introduce substantially more variation than the simulator's inherent randomness: the MAE values here are clearly higher, and the F1-scores are lower than those presented in Table 6.1

A comparision between Table 6.2 and Table 6.1 prove that the augmentation method used causes data diversity that is larger than the diversity caused by simulation software error, this confirms that the augmentation method performs as intended.

System	MAE (s/h*segment)	F1-score
1	393.60	0.34
2	46.33	0.63
3	N/A	N/A

Table 6.3: Similarity between augmentations of the same layout, averaged. MAE calculated according to equation (5.9). This table is intended to show that there is substantial variation not only between a layout and its augmentations, but also among the augmentations themselves. The magnitude of these differences indicates that the augmentations introduce diversity beyond what can be attributed to simulation software error seen in Table 6.1

Table 6.3 is intended to show that there is variation not only between a layout and its augmentations, but also among the augmentations themselves. The magnitude of these differences indicates that the augmentations introduce diversity beyond what

can be attributed to simulation software error seen in Table 6.1. This outcome is consistent with the intended purpose of the augmentation process, namely to generate new layouts that enrich the dataset.

6.3 Choice of Simulation Time

In the following section, the results of the extended simulations to determine a representative simulation time are presented.

Table 6.4 shows results from long simulation runs used to find sufficient simulation time for each system. The 90 percent threshold minute, calculated with Equation (3.4), approximates how long it takes to capture most of the system behavior. Equation (3.1) is used to check if the system is stationary. The results show that systems 1 and 2 are stationary, while System 3 is not in any run, due to it entering a deadlock.

System	Run	90 percent threshold minute	ADF test
1	1	431	Stationary
1	2	406	Stationary
1	3	450	Stationary
2	1	128	Stationary
2	2	194	Stationary
2	3	191	Stationary
3	1	N/A	non-stationary
3	2	N/A	non-stationary
3	3	N/A	non-stationary

Table 6.4: Results of extended simulation runs used to determine a representative simulation time for each system. The 90 percent threshold minute indicates the simulation time required to capture most of the system dynamics, computed according to Equation (3.4). The ADF test Equation (3.1) assesses whether the system is stationary during the simulation. The results show that the required simulation time varies significantly between systems and that only Systems 1 and 2 are stationary, while System 3 is non-stationary across all runs.

In Figures 6.1, 6.2 and 6.3. Sub-figure a) shows a plot calculated according to equation (3.2), b) shows a plot calculated according to equation (3.3) and c) shows plot calculated according equation (3.4).

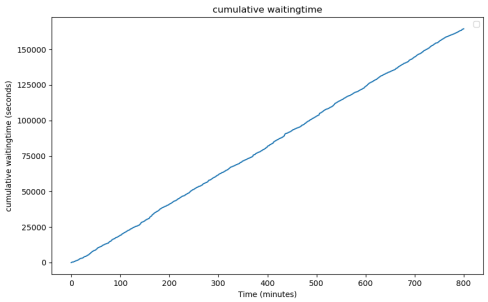
In Figure 6.1, System 1, run 1, The plots illustrate that System 1 requires a simulation time approx. 400–450 minutes to reach the 90 % threshold, and that it is stationary according to the ADF test. This emphasizes that system behavior can be captured in a significantly shorter time than the extended run of 14 hours.

In Figure 6.2, System 2, run 1, System 2 achieves the 90% threshold approx. 130–190 minutes and is also stationary per the ADF test. This shows that the time needed

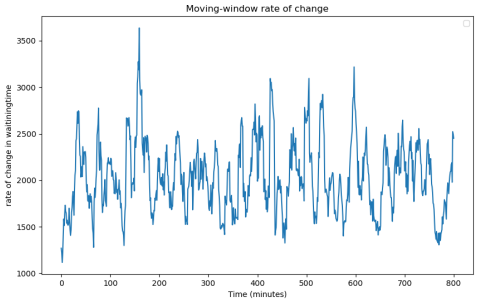
to capture system behavior is system dependent, as the resulting time for System 2 to reach the 90% threshold is significantly shorter than that needed for System 1 shown in 6.1.

In Figure 6.3, from plot (a) and (b) we can tell that System 3 entered a deadlock during simulation. Waiting time from a segment is only added to the total when a vehicle leaves the segment, when all vehicles are stuck in the system, the waiting time accumulating on the segments is never added to the total waiting time in the system. This leads to the flat-lining seen in both plot (a) and (b). The system was verified by Kollmorgen with a shorter simulation before being provided as a functioning system to be used in this thesis. This highlights the need of extended verification of the systems used in the data set.

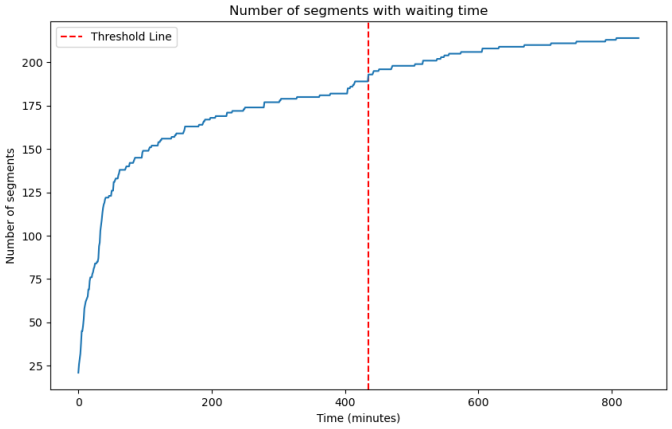
One plot for each system is shown here to illustrate the key results. Plots from the other two runs for each system are provided in the appendix.



(a) Cumulative wait-time over time. Computed according to (3.2).

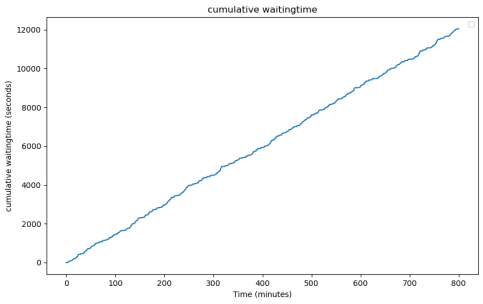


(b) Rate of change per minute (3.3) with windowsize of 10 min, ADF test shows stationarity.

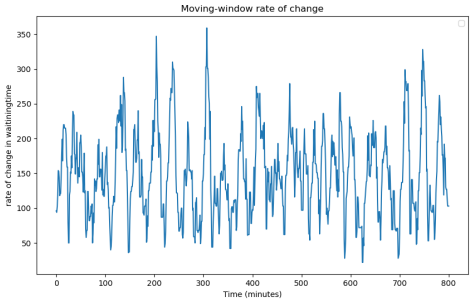


(c) Number of segments with non-zero waiting time over time. The red dotted line marks the point in time when 90 percent of all segments with wait time have received their wait time, calculated according to equation (3.4).

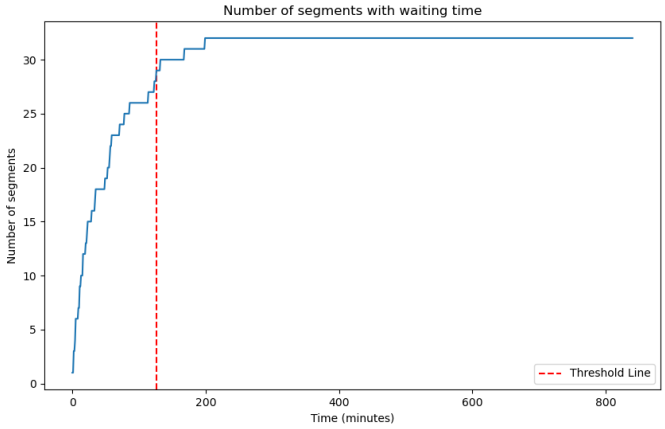
Figure 6.1: System 1, run 1, The plots illustrate that System 1 requires a simulation time approx. 400–450 minutes to reach 90 % of its dynamics, and that it achieves stationarity according to the ADF test. this emphasizes that system behavior can be captured in a significantly shorter time than the extended run of 14 hours.



(a) Cumulative wait-time over time. Computed according to (3.2).

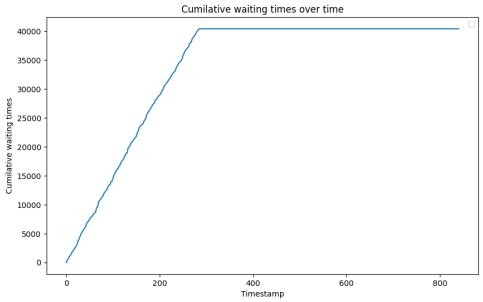


(b) Rate of change per minute (3.3) with windowsize of 10 min, ADF test shows stationarity.

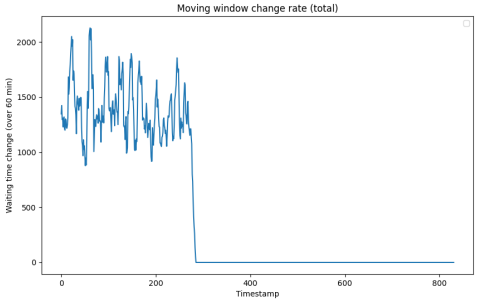


(c) Number of segments with non-zero waiting time over time. The red dotted line marks the point in time when 90 percent of all segments with wait time have received their wait time, calculated according to equation (3.4).

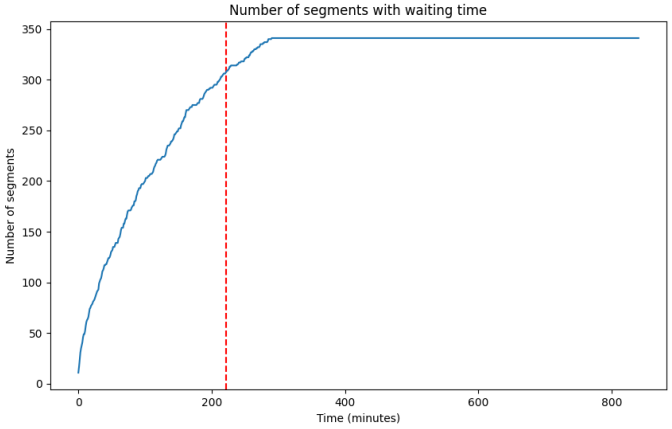
Figure 6.2: System 2, run 1, System 2 achieves the 90% threshold approx. 130–190 minutes and is also stationary per the ADF test. This shows that the time needed to capture system behavior is system dependent, as the resulting time for System 2 to reach the 90% threshold is significantly shorter than that needed for System 1 shown in Figure 6.1.



(a) Cumulative wait-time over time. Computed according to (3.2).



(b) Rate of change per minute (3.3) with windowsize of 10 min, ADF test shows non-stationarity.



(c) Number of segments with non-zero waiting time over time. The red dotted line marks the point in time when 90 percent of all segments with wait time have received their wait time, calculated according to equation (3.4).

Figure 6.3: System 3, from plot (a) and (b) we can tell that System 3 entered a deadlock during simulation. Waiting time from a segment is only added to the total when a vehicle leaves the segment, when all vehicles are stuck in the system, the waiting time accumulating on the segments is never added to the total waiting time in the system. This leads to the flat-lining seen in both plot (a) and (b). The system was verified by Kollmorgen with a shorter simulation before being provided as a functioning system to be used in this thesis. This highlights the need of extended verification of the systems used in the data set.

6.4 Performance of the GNN

In the following section, the results of training and evaluating the GNN, with the chosen architecture and parameters, on the new data set implementing data augmentation and the results from the simulation time analysis.

6.4.1 Classifier Performance

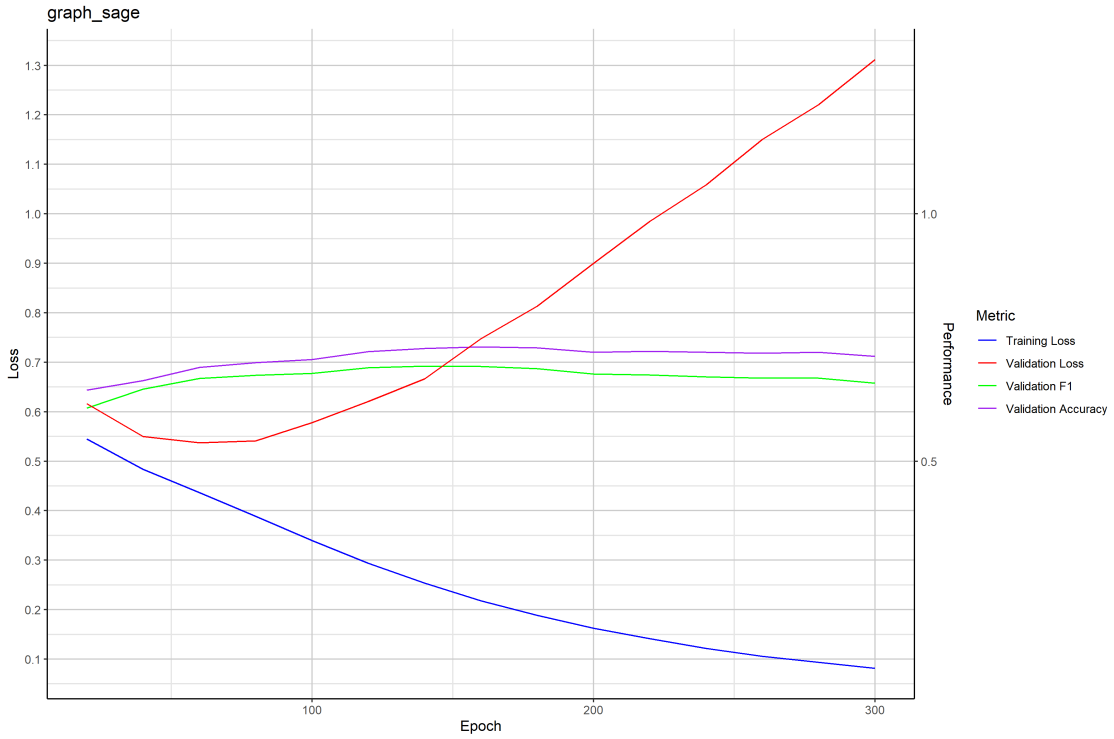


Figure 6.4: Classifier performance when trained and evaluated on System 1 including its layout augmentations. The F1-score improves to 0.69, well above the augmentation baseline in Table 6.3, showing that the model learns patterns beyond the average of the given input data.

The GNN achieved its highest classification performance on System 2 with layout augmentations, reaching an F1-score of 0.82 seen in Figure 6.5. In comparison, System 1 reached an F1-score of 0.69. This difference is in line with the patterns seen in the augmentation analysis in Table 6.3. The average F1-score between different augmentations of System 2 is 0.63, while for System 1 it is only 0.34. This means that the augmentations for System 2 are more similar to each other than those for System 1, which makes the classification task easier for the GNN.

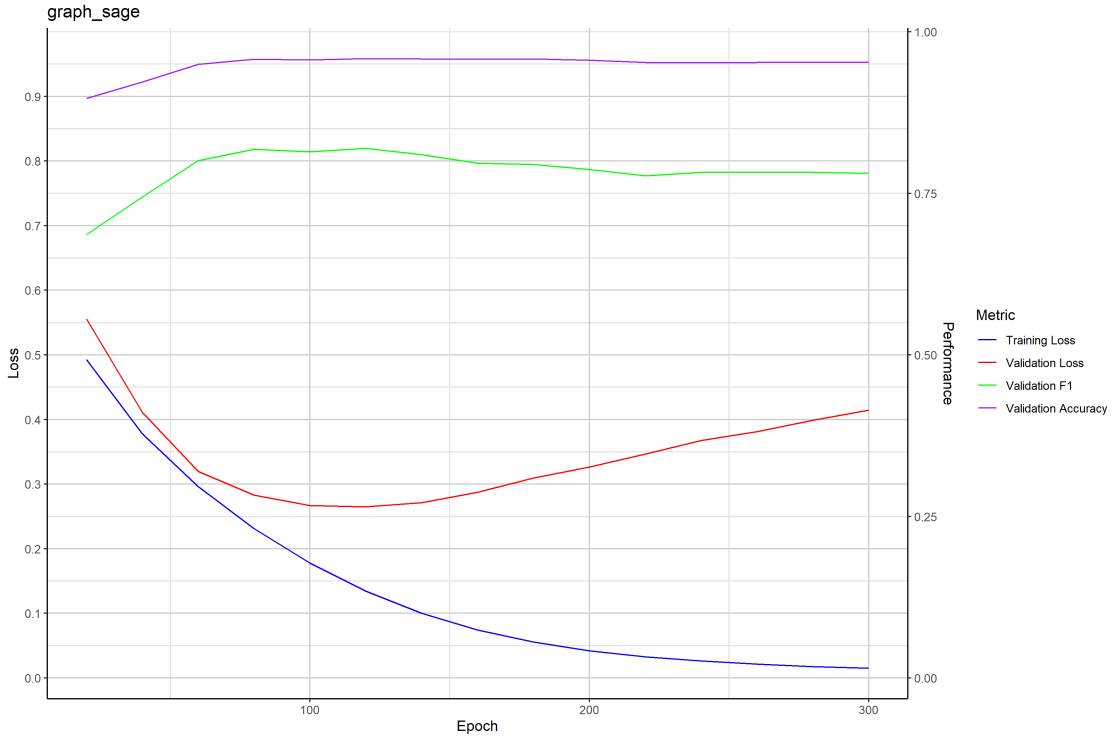


Figure 6.5: Classifier performance when trained and evaluated on System 2 including its layout augmentations. The highest F1-score achieved here is 0.82, matching the fact that System 2 augmentations are more similar to each other than in System 1.

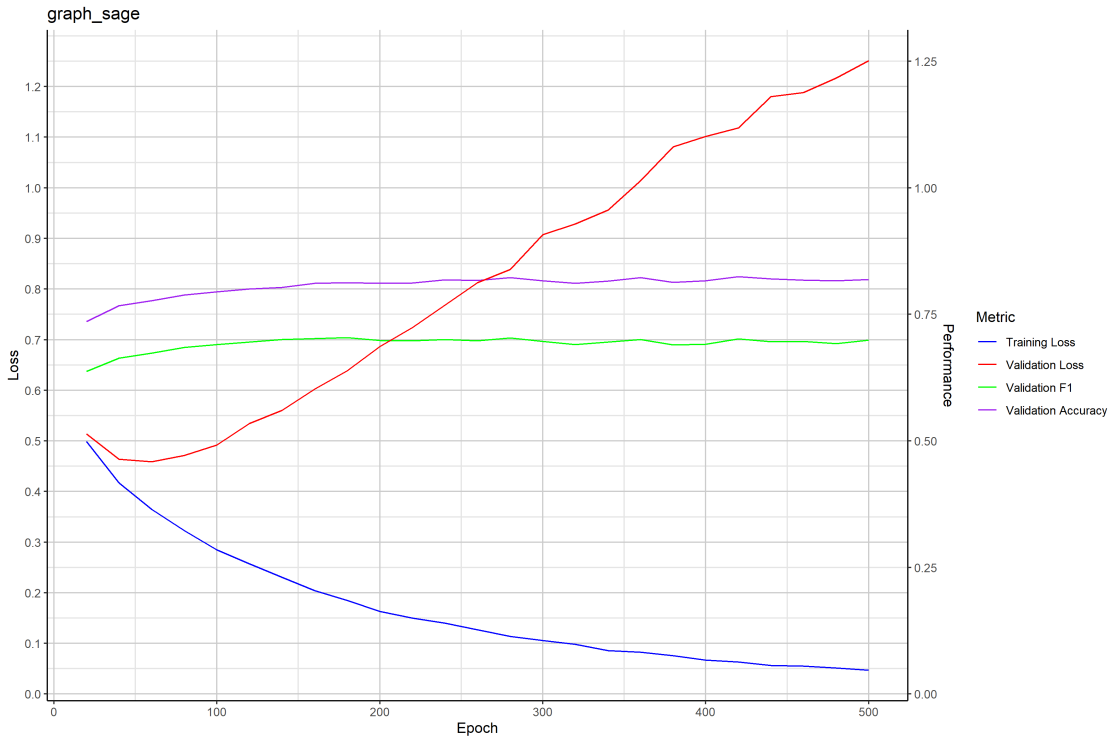


Figure 6.6: Performance of the classifier when trained and evaluated on both system 1 and system 2, including respective layout augmentations.

When training on both systems together, the results in Figure 6.6 show that the highest F1-score reached is 0.71. Indicating that the model is able to maintain similar performance when trained and evaluated on multiple systems and their augmentations, to its performance on single systems with augmentations.

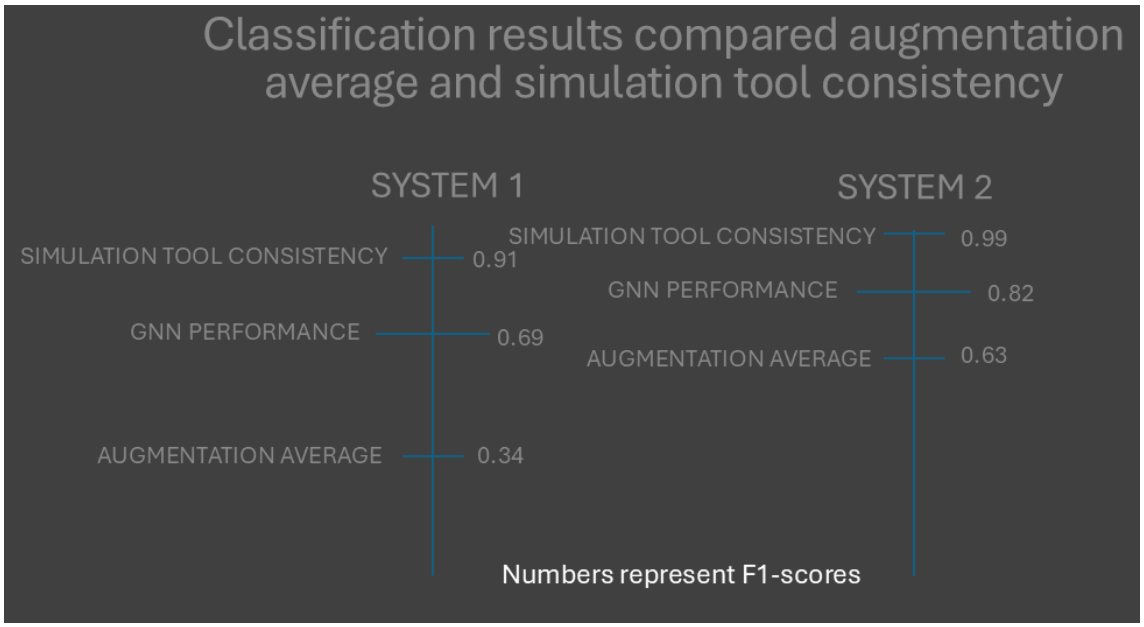


Figure 6.7: Comparison of classifier F1-scores for each system, taken from Figures 6.4 and 6.5, and its augmentations against augmentation averages in Table 6.3 and simulation software consistency in Table 6.1. The GNN outperforms augmentation average baselines in both single-system cases. The GNN performance does not reach the score of the simulation tool consistency.

In both cases seen in Figure 6.7, the GNN outperforms the baseline set by the similarity of the layout augmentations. For System 1, the model improves from a baseline of 0.34 to 0.69, and for System 2 from 0.63 to 0.82. This shows that the GNN is able to learn meaningful patterns from the input features, rather than just replicating the average output from the training data. However, the Figure also shows that the F1-scores are still significantly lower than the simulation tool inconsistency. Meaning the current GNN model cannot reliably replicate the results from the classification simulation tool.

6.4.2 Regressor Performance

The following figures Figure 6.8 and Figure 6.9 display of the true waiting times distributions in each system. The next three figures show the performance of the regressor on each system and the combination of the two valid systems (System 1 and System 2)

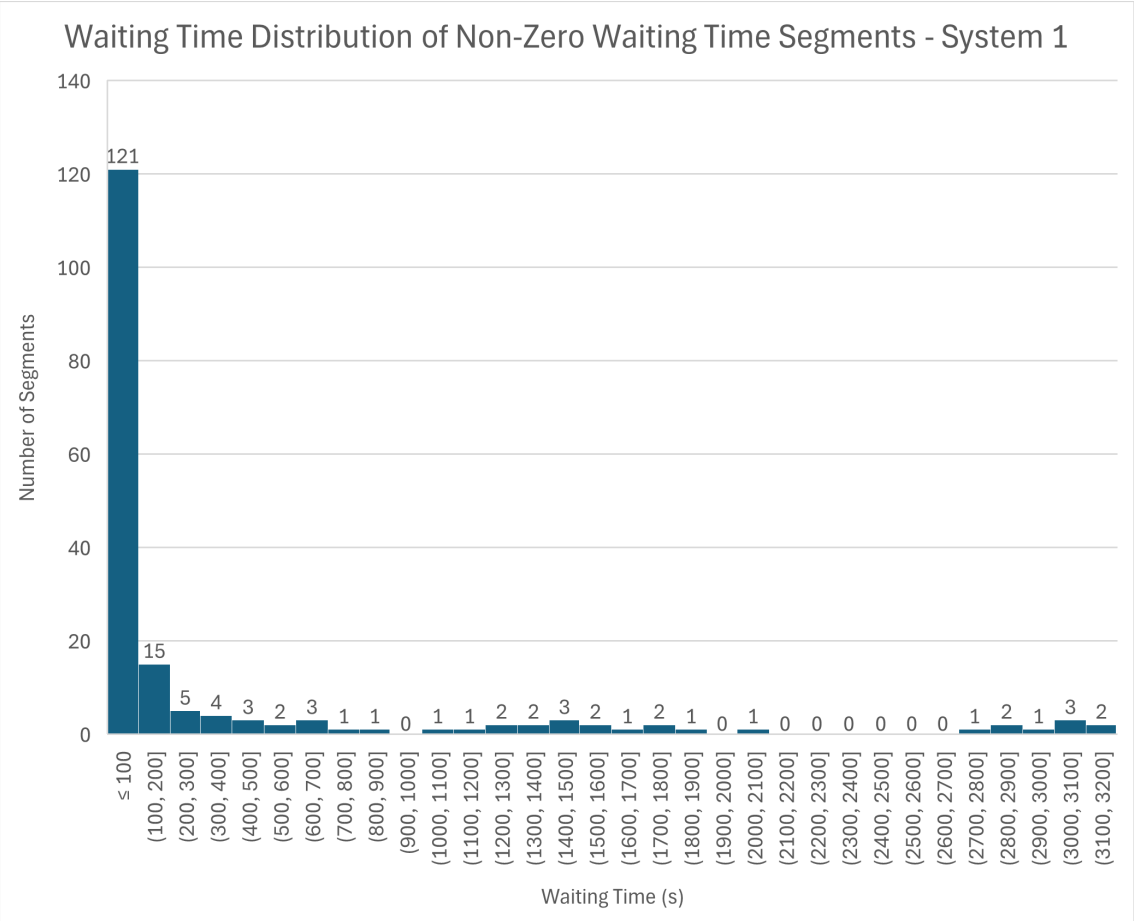


Figure 6.8: Distribution of true waiting times for non-zero waiting time segments in one augmentation of System 1. Most values are below 100 seconds, with a few extreme outliers above 2700 seconds.

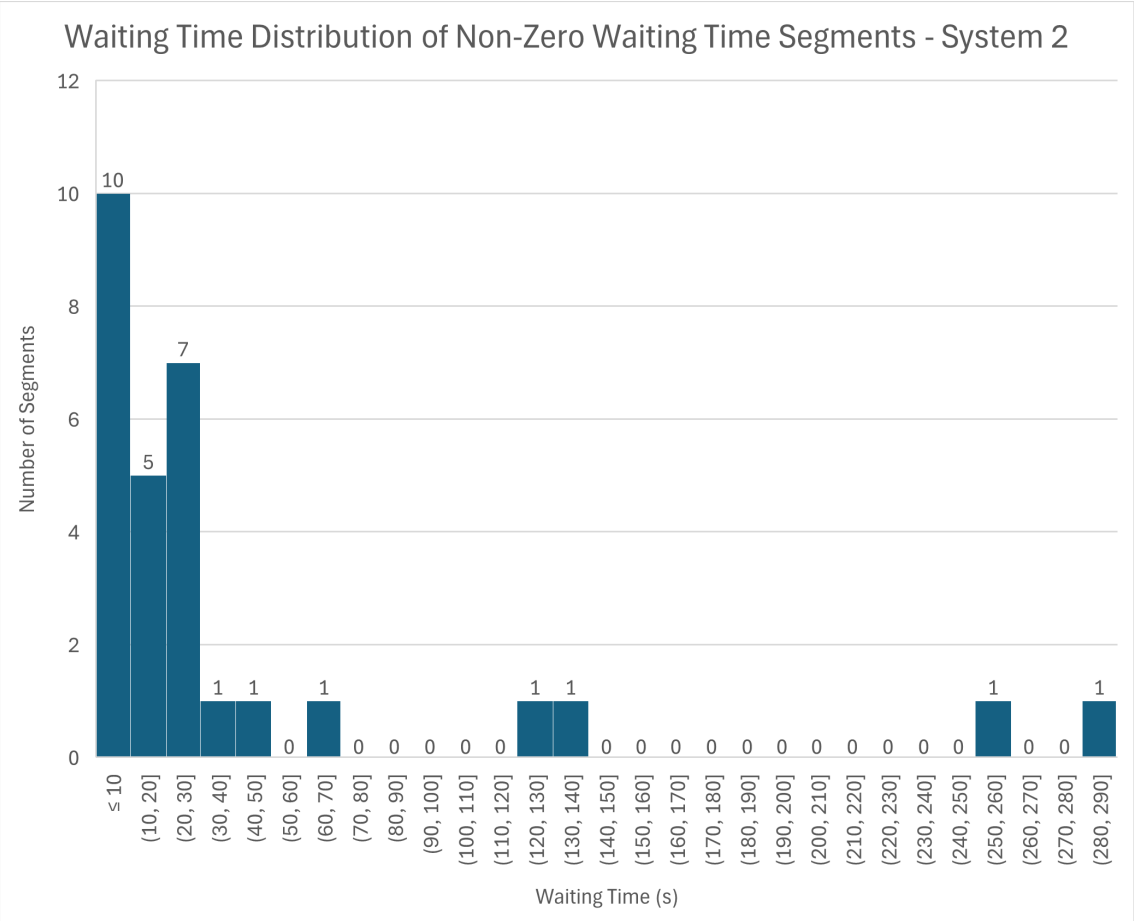


Figure 6.9: Distribution of true waiting times for non-zero waiting time segments in one augmentation of System 2. Compared to System 1 seen in Figure 6.8, the distribution is narrower with fewer extreme outliers.

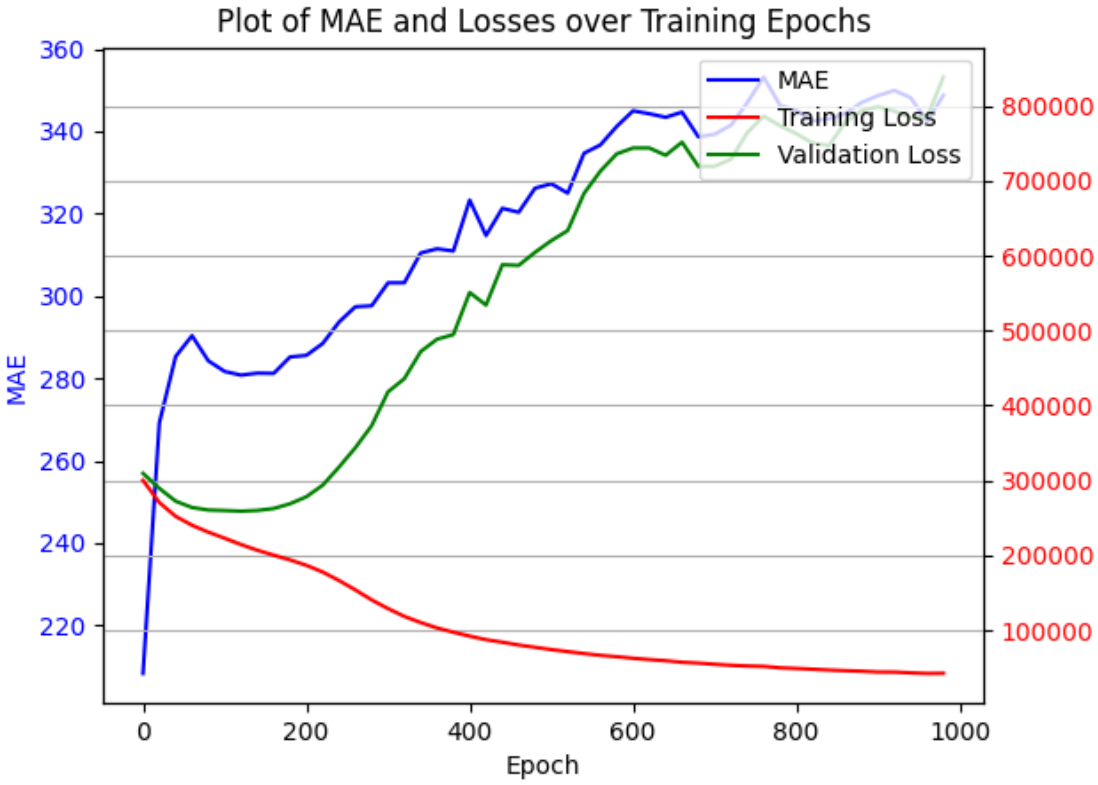


Figure 6.10: Regressor performance when trained and evaluated on augmentations of System 1. MAE initially rises despite falling losses, then decreases in the middle stage, and rises again toward the end, showing overfitting.

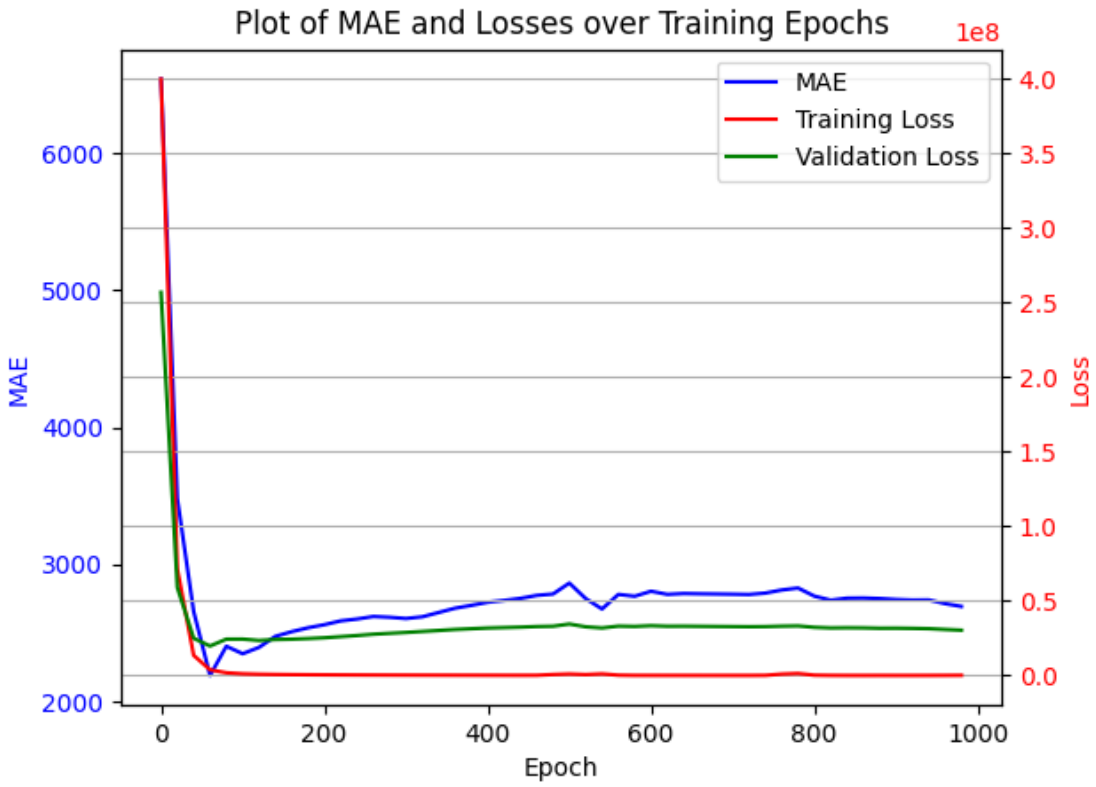


Figure 6.11: Regressor performance when trained and evaluated on augmentations of System 2. Losses and MAE drop quickly at first but remain well above the augmentation baseline, suggesting poor fit to the system's patterns.

Figure 6.12: Regressor performance when trained and evaluated on augmentations of both System 1 and System 2. The results of this show strange behavior where training loss is much lower than validation loss, and has some spikes in some epochs without a clear reason.

For System 1, the training curve in Figure 6.10 shows an initial stage where MAE is low but increases slightly, while training and validation losses decrease. This matches the waiting time distribution in Figure 6.8, where most values are short and only a few are extreme outliers. In the middle stage, both MAE and losses decrease, indicating improved predictions. Toward the end, MAE and validation loss rise while training loss continues to drop, showing overfitting. The lowest MAE is slightly below the augmentation baseline.

For System 2, the results in Figure 6.11 show high initial losses and MAE that drop quickly in early epochs. However, the final MAE remains much higher than the augmentation baseline. This is consistent with the distribution in Figure 6.9 and suggests the model does not capture the system's data patterns effectively.

When trained on both systems together, Figure 6.12 show strange behavior where training loss is much lower than validation loss, and has some spikes in some epochs without a clear reason.
Overall, the GNN performs poorly for both systems and the combined case.

7
Discussion

This chapter presents the findings from the experiments and analysis. It examines the effects of simulation noise, evaluates data augmentation, discussed the choice of simulation time, and evaluates the GNN's predictive capabilities. Finally, it identifies limitations and outlines areas for future improvement.

7.1 Simulation-Software Consistency

The results in Table 6.1 show variation in the simulation outputs of different runs of the same system. This proves that the simulation software has stochastic noise. The type of the noise and its distribution is hard to determine from only three samples of simulations for every system. However, the results give some insights about the data used to train the GNN model.

Given that the simulation software is used to create data to train the GNN model, the GNN model performance can at best be expected to have less error than the errors caused by the inconsistency of the simulations.

The magnitude of the MAE caused by the simulation software is similar for both System 1 and System 2 which is seen in Table 6.1, however, the F1-score is noticeably higher in System 2. A possible explanation for this is the smaller size of System 2 in terms of number of segment, leading to higher concentration of non-zero waiting time segments. Which allows the segments to be well explored in all runs.

7.2 Augmentation Analysis

The results from Table 6.2 prove that the augmentation method used causes data diversity that is larger than the diversity caused by simulation software error seen in Table 6.1. This indicates that the augmentation method provides the GNN model with beneficial data to generalize relations between inputs and outputs. The augmentation method is however limited to only one of the node attributes that are included in the graph representation of the layout. This means that other node relations, like transition relations and blocking relations, have only the diversity provided by the original layouts of the systems available. While node attributes like active usage and blocking probability are only varied based on traverse times and not the other information incorporated in those metrics. A more representative approach would be to implement some augmentation for all attributes and relations,

however, augmenting order flows, transition relations or blocking relations poses a risk of creating a deadlock in the system. Therefore, such augmentations cannot be automated and require domain specific expertise to be applied without creating deadlocks, which is out of the scope for this thesis.

Another potential improvement is to optimize the values chosen for the augmentation of traverse time. In Equation (5.10), alpha and theta can be tuned for maximal variation of simulation results, providing more diversity of data.

7.3 Simulation Time Analysis

The extended simulations of both systems seen in Figures 6.1 (b) and 6.2 (b) show that longer simulation time results in more segments having non-zero waiting times. However, the increase is the highest at the start of the simulation and then goes towards a constant value gradually. Meaning that segments that are the most prone to congestion, which are the most important segments, should appear within a short time frame and thus be detected by the 90% threshold method and thus included into the GNN.

Figure 6.1 (a) and Figure 6.2 (a) show the cumulative wait time increasing steadily over time, indicating consistent system activity throughout the simulation period. This confirms that the system does not enter an idle or locked state prematurely and continues to exhibit realistic AGV behavior.

In Figure 6.1 (b) and Figure 6.2 (b), the rate of change of the cumulative wait time is plotted using a forward-looking window of 10 minutes according to Equation 3.3. The curve fluctuates and does not display a trend, suggesting that the rate of change is stationary. This is further confirmed by the results of the ADF stationarity test indicating that the process is weakly stationary. Consequently, a shorter simulation would still be representative of the long-term system behavior.

Figure 6.1 (c) and Figure 6.2 (c) show the number of segments with non-zero waiting times accumulated over time. The red dotted line marks the point where 90 percent of all segments with waiting time have occurred. For System 1, this threshold is reached well before the 14-hour mark, suggesting that most of the relevant waiting time behavior is captured early in the simulation. More specifically for System 2 around 200 minutes and for System 1 around 450 minutes.

The shorter simulation times allow for the inclusion of more augmentations for each system, given the limited time and resources in this thesis. However, it could be said that the method used implicates that there is still some data loss when the shorter simulation time is chosen, however the importance of this loss could be considered small compared to the total amount. This is in line with the expected trade off between quantity and quality of the data produced when choosing a simulation time. The analysis provides an estimation of a sufficient simulation time, and a better

understanding of the behavior of the given systems.

As shown in Figure 6.3, the system's behavior deviates from that of a properly functioning AGV system. In Sub-figure (a), the cumulative wait time curve levels off, indicating a blocking event that causes the system to stall. This plateau suggests a flaw in how wait time is calculated. Intuitively, wait time should continue to rise rapidly over time if deadlock occurs. However, in the current implementation, a blocking period is only recorded if it has an end time. As a result, if the system enters a deadlock and no end time is captured, the cumulative wait time remains unchanged. Given that the method of this thesis are limited to functioning AGV systems without deadlocks, the simulation output of System 3 is not suitable for use as input to the GNN framework.

7.4 Evaluation and Analysis of GNN Performance

In this section, the performance of the GNN is discussed.

7.4.1 Classification Performance and Generalization Insights

The GNN showed the best classification performance on System 2 with its layout augmentations, reaching an F1-score of 0.82, While the F1 score on System 1 layout only reached 0.69. This can be explained by the difference of F1-scores in the augmentation analysis of the two systems. The layout augmentations of System 2 had an average F1-score of 0.63 when compared to each others, while the layout augmentations of System 1 had 0.34 for the same comparison. This means that the layout augmentations for System 2 are more similar that than the layout augmentations of System 1, making it easier for the GNN to predict correct labels. In both cases, the GNN performed considerably higher than the baseline set by the similarity (F1-score) of the layout augmentations, indicating that the GNN model is capable of drawing conclusions from the inputs rather than just using the average of the outputs from the training data.

7.4.2 Regression Performance and Training Dynamics

The performance of the regressor on System 1, System 2 and the combination of System 1 and System 2 showed no signs of convergence towards meaningful waiting time predictions.

When the GNN is run on System 1, there seems to be three stages in the training. Initially, a low MAE that is increasing, while validation and training losses are decreasing. This can be explained by the distribution of the waiting time values in this system. Figure 6.8 shows an example of the distribution in one the augmentation of System 1. The distribution shows that most non-zero waiting time segments have a waiting time of less than 100 seconds, while some few segments have relatively very high waiting time (over 2700 seconds). Initial predictions might be close to the majority of the segments while being very far off from the few outliers. This

results in relatively low MAE while being much more penalized in the validation and training losses that are calculated in terms of MSE. In the second stage, the MAE starts to decrease while training and validation losses are still decreasing. This stage indicates real learning in the model, where predictions are improved over epochs in a way that generalizes from training data to validation data. In the third stage, the training loss keeps decreasing while MAE and validation loss start increasing. This indicates that the model has started overfitting. The lowest MAE achieved is moderately less than the MAE resulted from comparing the augmentations of the system as in table 6.3. While this might indicate that the GNN is succeeding in modeling the relation between inputs and outputs, it is in this case most likely only a result of making predictions that are close to the majority of the segments without adapting to outliers.

In the regression run on System 2, the initial MAE and validation and training losses are very high, and then quickly decrease. This indicates that the GNN is capable of adapting somehow to the data, but the MAE is still very high in comparison to the result of comparing the augmentations of the system as in table 6.3.

The performance of the regressor was also trained and evaluated on augmentations of both System 1 and System 2 at the same time. The results of this show strange behavior where training loss is much lower than validation loss, and has some spikes in some epochs without a clear reason. The MAE and validation loss decrease in unsteady pattern and drop significantly around epoch 900. In general, the MAE stays over 1000 which is significantly higher than the averages in the augmentations comparisons of the systems in table 6.3. These strange patterns can be explained by the model being too simple for modeling the relationship between inputs and outputs from multiple systems at the same time, resulting in underfitting.

7.5 Improvement Areas

A key part of an AGV system is the number of vehicles to deployed in the system. In this research, this number is not represented as a raw input for GNN. This makes the input information incomplete for accurately predicting waiting times, as the number of vehicles in the system has significant impact on the congestion in the AGV system. This has arguably little impact when the GNN is trained and evaluated on one system with its layout augmentations, as the number of AGVs is the same for all the layouts and therefore would not be a meaningful input. However, when training and evaluating on multiple systems, including the number of AGVs is key information as some difference in waiting times could be directly attributed to the difference in number of AGVs.

To save time, a decision was made to rely on domain experts at Kollmorgen to provide a representative order flow compatible with the given layout. While this approach was practical in the short term, it introduces limitations that require further investigation. Specifically, issues such as deadlocks and the broader impact of order flow on simulation outcomes need to be explored. For the AI model to make

fair and accurate predictions, it must be provided with inputs like order flow. Although this data is embedded in the active usage, it remains unclear how effective this embedding is. A more raw representation of the order flow could potentially be more beneficial for the model. The number of AGVs in the system is also arguably embedded in the active usage. This number is assumed to be optimal based on calculations from the provided system analysis software. However, the optimal number is not always an integer and can differ from the actual number used in the simulations. This number could be input explicitly to the model, providing more accurate representation of the system.

Another important factor that hasn't been addressed yet is the role of the fleet management system. This system is crucial for coordinating AGV operations, and it can greatly affect how predictable and efficient the entire setup appears. By controlling task assignments, routes, and conflict resolution, the fleet management system adds a layer of decision-making that can either reduce or worsen problems like deadlocks and inefficiencies. Overlooking this aspect can lead to an incomplete understanding of how the system behaves as a whole. That's why future work should closely examine how the fleet management logic interacts with both the order flow and AGV behavior. This interaction likely has a big impact on the system's overall stability and performance.

8

Conclusion

The results confirm that the simulation tool introduces measurable stochasticity, defining an upper bound for GNN prediction accuracy. Despite this, the proposed augmentation method, applied via traversal time perturbation, successfully increased data diversity and improved generalization in classification tasks.

Simulation duration analysis showed that shorter runs can suffice, provided the systems are stationary. However, system deadlocks, as seen in System 3, prevent reliable data generation, underscoring the need to extensively verify simulation stability before use.

While previous research has demonstrated GNNs ability to predict congestion patterns in toy examples of AGV systems, representing individual intersections, our research indicates that significant challenges remain in order to scale to more realistic examples functioning AGV systems. For our data set, we observe that the classification part somewhat generalizes, outperforming similarity-based baselines, but still not reaching the consistency of the simulation tool. In contrast, regression performance was weak, particularly across different systems, highlighting the difficulty of modeling continuous outcomes with limited features.

Key limitations include small dataset only consisting of 2 working systems, the exclusion of system-level parameters (e.g., AGV count), reliance on fixed expert-designed order flows, and the omission of some fleet management behaviour, all of which restrict generalization.

Future work should extend augmentation to additional features, include global system inputs, and improve regression modeling through architectural or training enhancements.

In summary, this thesis demonstrates feasibility of using GNNs to detect layout inefficiencies and establishes a methodological basis for more scalable, data-driven AGV system design.

Bibliography

[1] Gómez, R., Pérez, J., and Ruiz, S. (2021) 'Optimization techniques for AGVs in flexible manufacturing systems,' *Journal of Science Review*. Available at: https://www.indexacademicdocs.org/pdf/1592/72243/994588 [Accessed: May 27, 2025].

[2] Wang, L., and Li, Y. (2020) 'Implementing an AGV system to transport finished goods to the warehouse,' *International Journal of Engineering and Technology*, 9(5). Available at: https://www.researchgate.net/publication/340060714_Implementing_an_AGV_System_to_Transport_Finished_Goods_to_the_Warehouse [Accessed: May 27, 2025].

[3] Global Trade Magazine (2023) 'The future of automated guided vehicles in smart logistics.' Available at: https://www.globaltrademag.com/the-future-of-automated-guided-vehicles-in-smart-logistics/ [Accessed: May 27, 2025].

[4] Ravichandiran, V. (2021) *Leave-One-Out Cross-Validation (LOOCV)*. Data Aspirant. Available at: https://dataaspirant.com/leave-one-out-cross-validation-loocv/ [Accessed: April 3, 2025].

[5] Srinivas, K., Ramesh, M., and Rajesh, K. (2023) 'Comparative Analysis using Various Performance Metrics in Imbalanced Data', *International Journal of Advanced Computer Science and Applications (IJACSA)*, 14(6). Available at: https://thesai.org/Downloads/Volume14No6/Paper_116-Comparative_Analysis_using_Various_Performance_Metrics.pdf [Accessed: April 3, 2025].

[6] W. L. Hamilton, *Graph Representation Learning,* Reprint of original ed., Morgan & Claypool, 2020.

[7] Brockwell, P. J., and Davis, R. A. (2016) *Introduction to Time Series and Forecasting*, 3rd ed., Springer, New York.

[8] Zhang, B. (2025) *Graph Neural Networks for Mobile Robots: A Systemic GNN Design Solution for Traffic in AGV Systems*. Master's thesis, Department of Mathematical Sciences, Chalmers University of Technology.

[9] Lauri, J., and Wiberg, M. (2023) *Supervised Learning to Evaluate Road Networks for Automated Guided Vehicles: Using Graph Neural Networks to Estimate the Efficiency of Different Road Network Designs*. Master's thesis, Department of Electrical Engineering, Chalmers University of Technology.

[10] Velayutham, S. (2024) *Traffic Prediction in Automated Guided Vehicular Systems using Graph Neural Networks: Using Relational Graph Neural Networks to Model Congestion in AGV Systems*. Master's thesis, Department of Com-

puter Science and Engineering, Chalmers University of Technology and University of Gothenburg.

[11] R. Miao, K. Zhou, Y. Wang, N. Liu, Y. Wang, and X. Wang, "Rethinking Independent Cross-Entropy Loss For Graph-Structured Data," *arXiv preprint arXiv:2405.15564*, 2024. Available at: https://arxiv.org/abs/2405.15564.

[12] K. P. Selvam, "Graph Neural Network: In a Nutshell," *Karthick.ai Blog*, 2024. Available at: https://karthick.ai/blog/2024/Graph-Neural-Network/.

[13] IKI (2024) 'AGVs and AMRs: What's the Difference?', *IKI Blog*, 26 February. Available at: https://www.iki.com/blog/2024/02/26/agvs-and-amrs/#:~:text=AGVs%20typically%20work%20using%20guidance%20systems%20like,surroundings%2C%20plan%20routes%2C%20and%20make%20real%2Dtime%20decisions. [Accessed: May 30, 2025].

[14] AGV Network (2023) 'AGV Sensors – The eyes and ears of mobile robots.' Available at: https://www.agvnetwork.com/mobile-robot-sensors-agv-amr [Accessed: May 30, 2025].

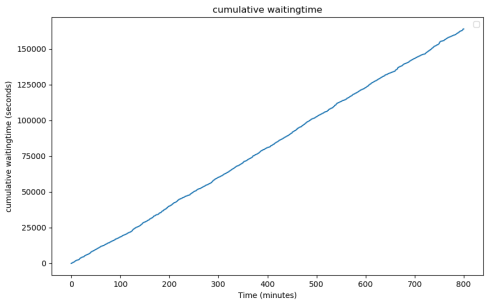
[15] DataCamp, "Loss Functions in Machine Learning Explained," *DataCamp Tutorials*, 2023. Available at: https://www.datacamp.com/tutorial/loss-function-in-machine-learning.

[16] Adjeisah, M., Zhu, X., Xu, H., and Ayall, T. A. (2022) 'Towards data augmentation in graph neural network: An overview and evaluation', *Information Fusion*, [Online]. Available at: https://doi.org/10.1016/j.inffus.2022.11.014 [Accessed: April 3, 2025].

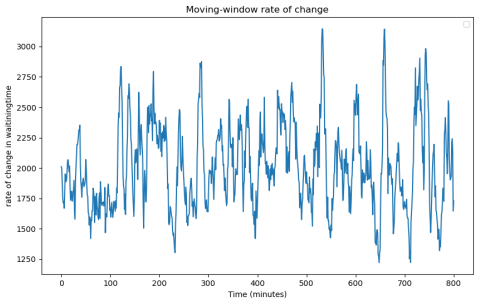
[17] He, K., Zhang, X., Ren, S., and Sun, J. (2015) 'Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,' *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034. Available at: https://doi.org/10.1109/ICCV.2015.123 [Accessed: August 14, 2025].

A

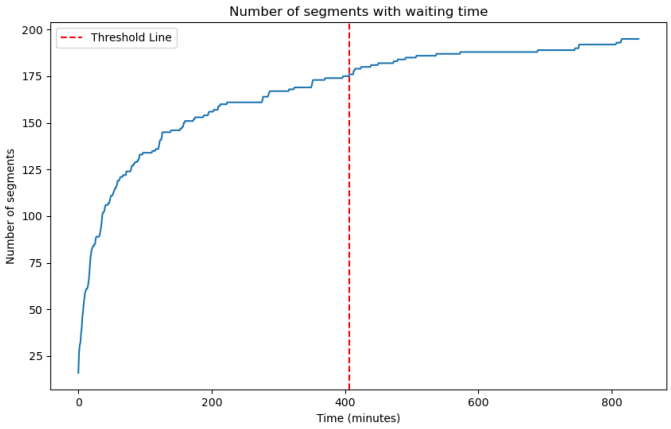
Appendix 1



(a) Cumulative wait-time over time. Computed according to (3.2).

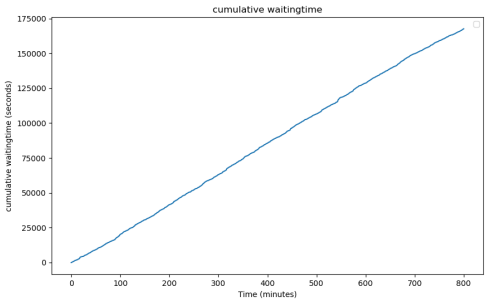


(b) Rate of change per minute (3.3) with windowsize of 10 min, ADF test shows stationarity.

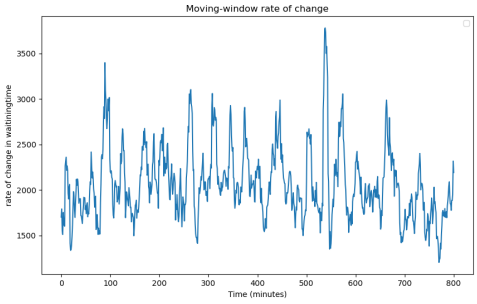


(c) number of segments with non-zero waiting time over time. The red dotted line marks the point in time when 90 percent of all segments with wait time have received their wait time, calculated according to equation (3.4).

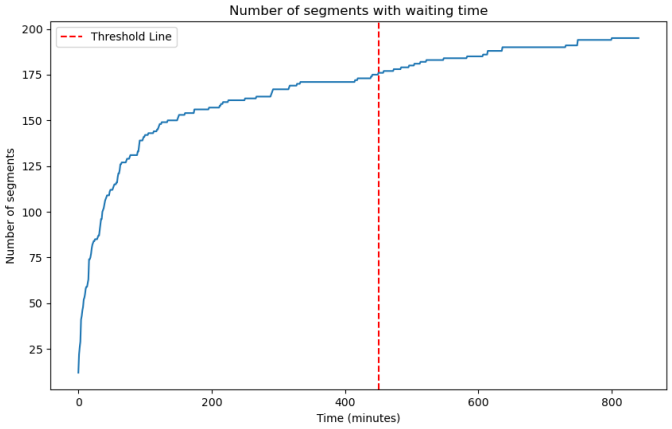
Figure A.1: System 1, run 1, The plots illustrate that System 1 requires a simulation time approx. 400–450 minutes to reach 90 % of its dynamics, and that it achieves stationarity according to the ADF test. This emphasizes the need for extended simulation time depending on system behavior.



(a) Cumulative wait-time over time. Computed according to (3.2).

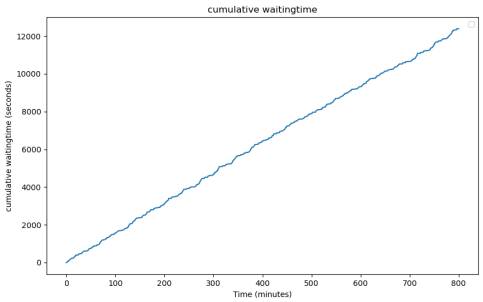


(b) Rate of change per minute (3.3) with windowsize of 10 min, ADF test shows stationarity.

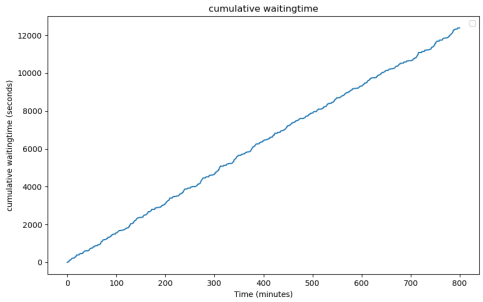


(c) number of segments with non-zero waiting time over time. The red dotted line marks the point in time when 90 percent of all segments with wait time have received their wait time, calculated according to equation (3.4).

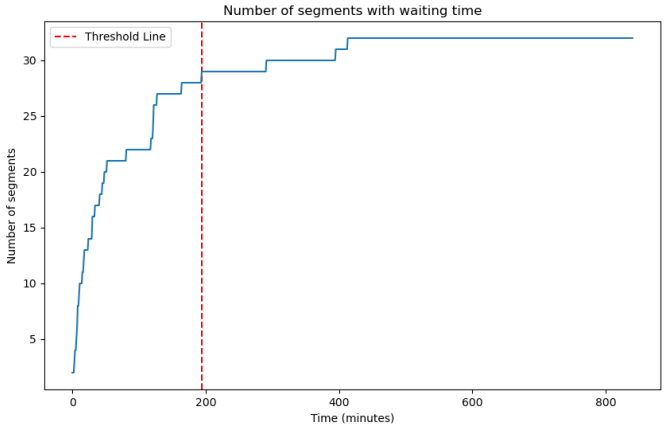
Figure A.2: System 1, run 1, The plots illustrate that System 1 requires a simulation time approx. 400–450 minutes to reach 90 % of its dynamics, and that it achieves stationarity according to the ADF test. This emphasizes the need for extended simulation time depending on system behavior.



(a) Cumulative wait-time over time. Computed according to (3.2).

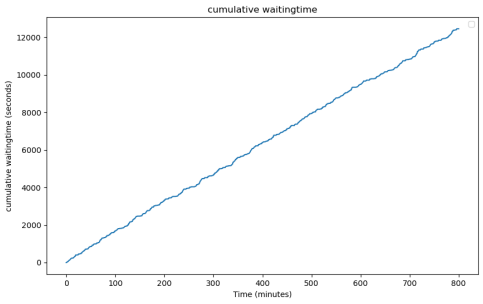


(b) Rate of change per minute (3.3) with windowsize of 10 min, ADF test shows stationarity.

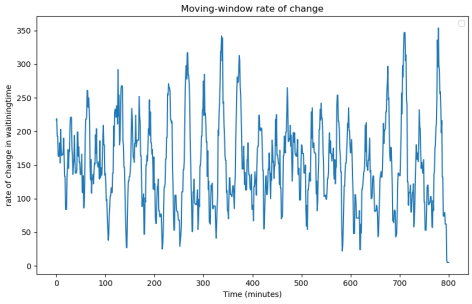


(c) number of segments with non-zero waiting time over time. The red dotted line marks the point in time when 90 percent of all segments with wait time have received their wait time, calculated according to equation (3.4).

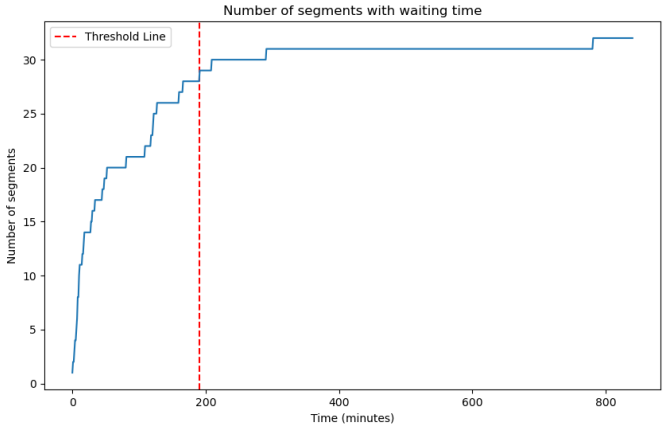
Figure A.3: System 2, run 1, System 2 achieves 90% of its dynamics approx. 130–190 minutes and is also stationary per the ADF test. This demonstrates how system structure affects both convergence rate and stability.



(a) Cumulative wait-time over time. Computed according to (3.2).

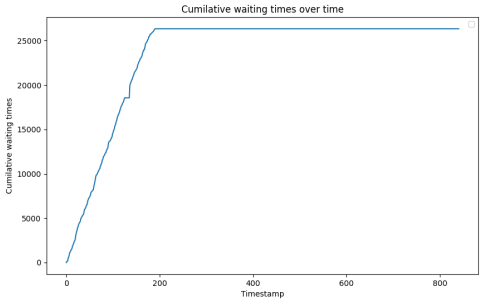


(b) Rate of change per minute (3.3) with windowsize of 10 min, ADF test shows stationarity.

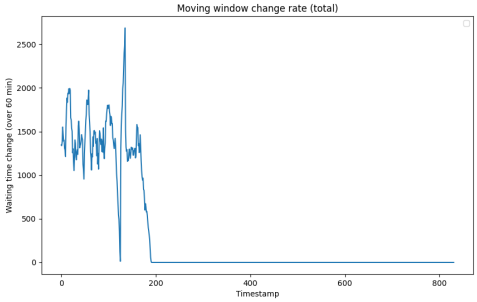


(c) number of segments with non-zero waiting time over time. The red dotted line marks the point in time when 90 percent of all segments with wait time have received their wait time, calculated according to equation (3.4).

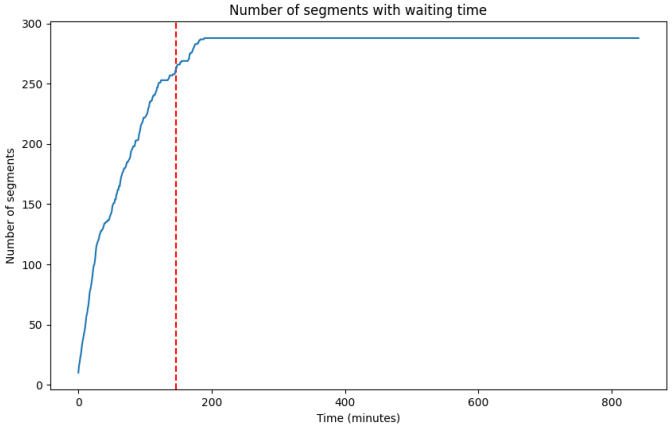
Figure A.4: System 2, run 1, System 2 achieves 90% of its dynamics approx. 130–190 minutes and is also stationary per the ADF test. This demonstrates how system structure affects both convergence rate and stability.



(a) Cumulative wait-time over time. Computed according to (3.2).

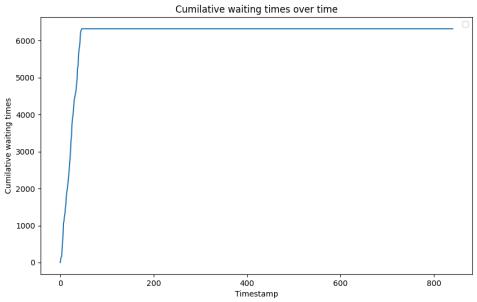


(b) Rate of change per minute (3.3) with windowsize of 10 min, ADF test shows non-stationarity.

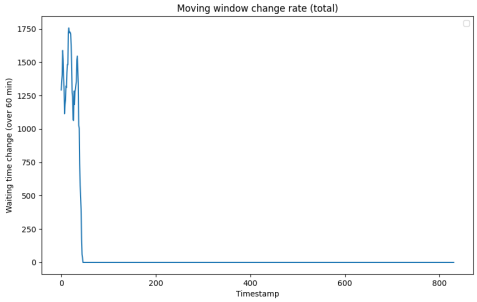


(c) number of segments with non-zero waiting time over time. The red dotted line marks the point in time when 90 percent of all segments with wait time have received their wait time, calculated according to equation (3.4).

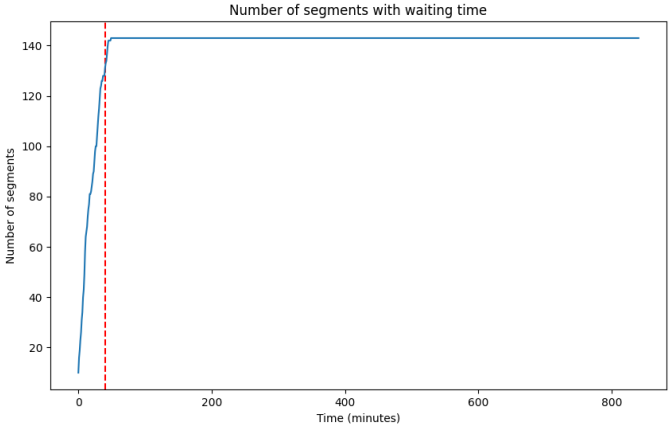
Figure A.5: System 3, run 1, Despite using the same evaluation methods (a–c), System 3 fails to reach a clear threshold and remains non-stationary throughout all simulation runs. This highlights that not all system configurations converge and that simulation stability is highly dependent on system setup.



(a) Cumulative wait-time over time. Computed according to (3.2).



(b) Rate of change per minute (3.3) with windowsize of 10 min, ADF test shows non-stationarity.



(c) number of segments with non-zero waiting time over time. The red dotted line marks the point in time when 90 percent of all segments with wait time have received their wait time, calculated according to equation (3.4).

Figure A.6: System 3, run 1, Despite using the same evaluation methods (a–c), System 3 fails to reach a clear threshold and remains non-stationary throughout all simulation runs. This highlights that not all system configurations converge and that simulation stability is highly dependent on system setup.

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

CHALMERS
UNIVERSITY OF TECHNOLOGY