PhD-FSTM-2022-023
The Faculty of Sciences, Technology and Medicine

DISSERTATION

Defence held on 23/02/2022 in Esch-Sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Tieu Long MAI
Born on 22 July 1989 in city of Khanh Hoa, Vietnam

MACHINE LEARNING IN THE DESIGN SPACE EXPLORATION OF TSN NETWORKS

Dissertation defence committee

Prof. Dr Yves LE TRAON, Chairman
Professor, Université du Luxembourg (Luxembourg)

Prof. Dr Pere BARLET-ROS, Vice-chairman
Professor, UPC Barcelona Tech (Spain)

Prof. Dr Nicolas NAVET, Dissertation Supervisor
Professor, Université du Luxembourg (Luxembourg)

Dr TingTing HU, Dissertation Co-supervisor
Université du Luxembourg (Luxembourg)

Dr Fabien GEYER, Member
Technical University of Munich (Germany)

UNIVERSITY OF LUXEMBOURG

Machine Learning in the Design Space Exploration of TSN networks

Tieu Long Mai

Faculty of Science, Technology and Medicine (FSTM)

Abstract

Real-time systems are systems that have specific timing requirements. They are critical systems that play an important role in modern societies, be it for instance control systems in factories or automotives. In recent years, Ethernet has been increasingly adopted as layer 2 protocol in real-time systems. Indeed, the adoption of Ethernet provides many benefits, including COTS and cost-effective components, high data rates and flexible topology. The main drawback of Ethernet is that it does not offer ”out-of-the-box” mechanisms to guarantee timing and reliability constraints. This is the reason why time-sensitive networking (TSN) mechanisms have been introduced to provide Quality-of-Service (QoS) on top of Ethernet and satisfy the requirements of real-time communication in critical systems. The promise of Ethernet TSN is the possibility to use a single network for different criticality levels, e.g, critical control traffic and infotainment traffic sharing the same network resources.

This thesis is about the design of Ethernet TSN networks, and specifically about techniques that help quantify the extent to which a network can support current and future communication needs. The context of this work is the increasing use of design-space exploration (DSE) in the industry to master the complexity of designing (e.g. in terms of architectural and technological choices) and configuring a TSN network. One of the main steps in DSE is performing schedulability analysis to conclude about the feasibility of a network configuration, *i.e.*, whether all traffic streams satisfy their timing constraints. This step can take weeks of computations for a large set of candidate solutions with the simplest TSN mechanisms, while more complicated TSN mechanisms will require even longer time.

This thesis explores the use of Artificial Intelligence (AI) techniques to assist in the design of TSN networks by speeding up the DSE. Specifically, the thesis proposes the use of machine learning (ML) as an alternative approach to schedulability analysis. The application of ML involves two steps. In the first step, ML algorithms are trained with a large set of TSN configurations labeled as feasible or non-feasible. Due to its pattern recognition ability, ML algorithms can predict the feasibility of unseen configurations with a good accuracy. Importantly, the execution time of an ML model is only a fraction of conventional schedulability analysis and remains constant whatever the complexity of the

network configurations.

Several contributions make up the body of the thesis. In the first contribution, we observe that the topology and the traffic of a TSN network can be used to derive simple features that are relevant to the network feasibility. Therefore, standard and simple machine learning (ML) algorithms such as k-Nearest Neighbors are used to take these features as inputs and predict the feasibility of TSN networks. This study suggests that ML algorithms can provide a viable alternative to conventional schedulability analysis due to fast execution time and high prediction accuracy. A hybrid approach combining ML and schedulability analyses is also introduced to control the prediction uncertainty.

In the next studies, we aim at further automating the feasibility prediction of TSN networks with the Graph Neural Network (GNN) model. GNN takes as inputs the raw data from the TSN configurations and encodes them as graphs. Synthetic features are generated by GNN, thus the manual feature selection step is eliminated. More importantly, the GNN model can generalize to a wide range of topologies and traffic patterns, in contrast to the standard ML algorithms tested before that can only work with a fixed topology. An ensemble of individual GNN models shows high prediction accuracies on many test cases containing realistic automotive topologies. We also explore possibilities to improve the performance of GNN with more advanced deep learning techniques. In particular, semi-supervised learning and self-supervised learning are experimented. Although these learning paradigms provide modest improvements, we consider them promising techniques due to the ability to leverage the massive amount of unlabeled training data.

While this thesis focuses on the feasibility prediction of TSN configurations, AI techniques have huge potentials to automate other tasks in real-time systems. A natural follow-up work of this thesis is to apply GNN to multiple TSN mechanisms and predict which mechanism can provide the best scheduling solution for a given configuration. Although we need distinct ML models for each TSN mechanism, this research direction is promising as TSN mechanisms may share similar feasibility features and thus transfer learning techniques can be applied to facilitate the training process. Furthermore, GNN can be used as a core block in deep reinforcement learning to find the feasible priority assignment of TSN configurations. This thesis aims to make a contribution towards DSE of TSN networks with AI.

Keywords

Real-Time Systems, Time-Sensitive Networking, Schedulability Analysis, Feasibility Test, Design-Space Exploration, Artificial Intelligence, Machine Learning, Deep Learning, Graph Neural Network.

Publications included in this thesis

- [1] Navet, N. and **Mai, T.L.** and Migge, J., Using machine learning to speed up the design space exploration of Ethernet TSN networks, *Technical report, University of Luxembourg* 2019.

- [2] **Mai, T.L.** and Navet, N. and Migge, J., On the use of supervised machine learning for assessing schedulability: application to Ethernet TSN, *Proceedings of the 27th International Conference on Real-Time Networks and Systems* (RTNS 2019).

- [3] **Mai, T.L.** and Navet, N. and Migge, J., A Hybrid Machine Learning and Schedulability Method for the Verification of TSN Networks, *Proceedings of the 15th IEEE International Workshop on Factory Communication System* (WFCS 2019).

- [4] **Mai, T.L.**, and Navet, N., Deep learning to predict the feasibility of priority-based Ethernet network configurations, *ACM Transactions on Cyber-Physical System* 2021.

- [5] **Mai, T.L.**, and Navet, N., Improvements to Deep-Learning-based Feasibility Prediction of Switched Ethernet Network Configurations, *Proceedings of the 29th International Conference on Real-Time Networks and Systems* (RTNS 2021).

Acknowledgments

The PhD program at the University of Luxembourg is a remarkable chapter of my life. During this period, I have learned what is science and how scientific knowledge is acquired. I have gained expertise in real-time systems, particularly Time-Sensitive Networking (TSN), through learning-by-doing. I am also very proud of my accumulative knowledge about Artificial Intelligence (AI) and the state-of-the-art research studies conducted with AI.

I am very grateful for this achievement. I would like to express my sincere appreciation to Prof. Nicolas Navet, my supervisor, who believed in me and my skills. His profound knowledge about real-time systems is the most important factor for the development of my skills. His abundant experience not only in science but also in training has helped me to overcome countless difficulties during the program and thrive as a PhD student. He is my great mentor and I look forward to having more opportunities to collaborate with him after graduation.

Words cannot express my gratitude to Dr. TingTing Hu, my co-supervisor, for all the valuable discussions and encouragement during my PhD. The ritual morning coffee breaks in the lab are great chances for me to exchange ideas and improve my research skills. Besides the formal training, she also actively share information about Luxembourg's daily life, which is very useful for me when I was moving to the country.

I wish to express warm thanks to Prof. Yves Le Traon, who has shared comprehensive feedback about my research studies during the last CET meetings. I am thankful to Prof. Pere Barlet-Ros from UPC Barcelona Tech and Dr. Fabien Geyer from the Technical University of Munich, who have valuable comments about my thesis when serving on the defense committee.

I would like to thank my labmates and all staff members at the Department of Computer Science (FSTM, University of Luxembourg) who have been kindly supported me with their responsibilities.

Especially, I want to send my deepest thank to my family for all their love and support.

Contents

Abstract iii

Contents **ix**

List of Figures **xiv**

List of Tables **xxi**

Acronyms **xxvi**

1 Introduction **1**

1.1 Context and Motivation 1

1.1.1 Adoption of Ethernet to real-time communications 2

1.1.2 Time-Sensitive Networking (TSN) 3

1.1.3 Design-space exploration of Ethernet TSN networks 4

1.1.4 Artificial Intelligence for DSE of TSN networks 7

1.2 Research questions 9

1.3 Contribution 10

1.4 Thesis Organization 11

2 Background and Preliminaries **13**

2.1 Real-time Ethernet TSN communication 13

2.1.1 Definition of Ethernet TSN network 14

2.1.2 General System Model 15

2.1.3 Performance evaluation of Ethernet TSN networks 16

2.2 TSN QoS mechanisms for timing 18

2.3 Artificial Intelligence and Machine Learning 21
2.3.1 Introduction to Artificial Intelligence 22
2.3.2 Machine learning 23
2.3.3 Graph Neural Network 26
2.3.4 Machine learning in networking 29
2.3.5 Performance criteria 31
2.4 Discussion 32

3 On the Use of Supervised Machine Learning to TSN Networks 35

3.1 Introduction 36
3.2 Ethernet TSN model and design problem 39
3.2.1 Designing and configuring TSN networks 39
3.2.2 Traffic characteristics and network topology 40
3.3 Predicting feasibility with supervised learning 42
3.3.1 Supervised learning applied to network classification 42
3.3.2 Feature engineering and feature selection 43
3.3.3 The k-NN classification algorithm 44
3.4 Experiments with k-NN 45
3.4.1 Performance criteria and evaluation technique 45
3.4.2 Experimental setup 46
3.4.3 Experimental results 50
3.5 Comparison with schedulability analysis 52
3.6 Insights into when machine learning fails 56
3.6.1 Predictive power of the selected features 56
3.6.2 Analysis of wrong predictions 58
3.7 Hybrid verification strategy combining machine learning and schedulability analysis 60
3.7.1 Principles 60
3.7.2 Experimental results 61
3.7.3 Trade-offs between accuracy and computation times 64
3.8 Multi-stage hybrid approach with an approximate analysis 65
3.9 Related work 66

3.10 Discussion and perspectives 67

4 Application of Graph Neural Network to TSN Networks 71

4.1 Introduction 72

4.1.1 Context of the study 72

4.1.2 Research questions 73

4.1.3 Contributions of the chapter 73

4.1.4 Organisation of the paper 75

4.2 System model 75

4.2.1 Configuring the stream priorities 75

4.2.2 Topology and traffic characteristics 76

4.3 Predicting network feasibility with graph neural network 77

4.3.1 GNN: a promising tool for networking 77

4.3.2 GNN model structure 79

4.3.3 Computational steps in a general GNN model 81

4.3.4 Encoding of TSN configurations as a graph 82

4.3.5 The GNN model for TSN and associated learning algorithm . . . 85

4.3.6 Ensembles of graph neural networks 88

4.4 Experimental setup 89

4.4.1 Data sets 89

4.4.2 Parameters of the GNN model 92

4.5 Experimental results 94

4.5.1 RQ3-1: Insight into when ML approaches based on manual feature selection fail 95

4.5.2 RQ3-2: Accuracy of GNN models 98

4.5.3 GNN models in the design space exploration of in-vehicle networks 102

4.6 Related works 105

4.7 Conclusion and future work 106

5 Improvements to the Graph Neural Network Model 109

5.1 Introduction 110

5.2 Preliminary 112

5.2.1 System model 112
5.2.2 Graph Neural Network 112
5.2.3 The base GNN model 112
5.3 Improving the training set and the GNN model 114
5.3.1 Experiments 1 and 2: Improving the training set 115
5.3.2 Improving the GNN models 116
5.4 Expanding the GNN model and the training set 121
5.4.1 Expanding the GNN model 121
5.4.2 Expanding the training set 123
5.4.3 Analysis of the experimental results 124
5.4.4 Execution times of training and testing 127
5.5 Application of semi-supervised learning 128
5.5.1 Introduction to semi-supervised learning 128
5.5.2 Performance of GNN with semi-supervised learning 129
5.6 Self-supervised learning (SSL) for GNN model 131
5.6.1 Introduction to SSL 131
5.6.2 Enhancing the GNN model with SSL 134
5.6.3 Experimental results and discussions 136
5.7 Related works 137
5.8 Conclusion and future work 138

6 Conclusion **139**
6.1 Summary of the thesis work 139
6.2 Future research directions 141
6.2.1 Further enhancement of GNN model 141
6.2.2 Diversify application of GNN in TSN networks 142
6.2.3 Final thoughts 143
6.3 On the use of our works in the industry 144
6.3.1 Large training set and Active learning 144
6.3.2 Domain expertise 145
6.3.3 Advanced GNN block 146
6.4 Discussion and perspective 146

Bibliography **149**

List of Figures

1.1 Time-Sensitive Networking (TSN) is a set of mechanisms on top of standard Ethernet that ensure new forms of QoS such as timing predictability. 3

1.2 The product development process for an automotive electronics system (inspired from Figure 8.5 in [6]) requires the definition of the hardware system, functional architecture, and communication. A combination of options generates a large set of possibilities. The designers not only need to find the optimal solutions but also guarantee specific requirements. In this thesis, we assume that the hardware system has been fixed and thus the design space includes possible combinations of design options, each can be configured by one or more TSN mechanisms to satisfy QoS in real-time communications. 5

1.3 In the design-space exploration of TSN networks, machine learning (ML) is applied to quickly predict the feasibility of TSN configurations without performing schedulability analysis. The downside is that ML can lead to false positives. Therefore, the designers need to verify the set of selected candidate solutions with schedulability analyses to ensure that false positives do not happen. 7

1.4 Artificial intelligence and sub-domains relevant to this thesis. 8

2.1 Example of a TSN network with two switches (S1, S2) and three end-nodes (E1, E2, E3). Each full-duplex link is considered as two uni-directional links. For instance, the full-duplex link connecting end-node E1 and switch S1 is made up of link l_{e1}^{s1} from E1 to S1, and link l_{s1}^{e1} from S1 to E1. In a switch, there are ingress ports that connect to incoming links and egress ports that connect to outgoing links. Each egress port has a buffer to store packets waiting to be sent. All traffic flows (not shown in the figure) have to be sent from one end-node to other end-node(s) and forwarded by one or more switches. This figure will be re-used in Chapter 4. 14

2.2 Example of egress port in TSN switch inspired by [7]. Traffic flows are grouped into 8 classes at most, each class associated with a waiting queue. The CBS scheduling mechanism is applied to one or several classes. With TAS, gates are implemented on all queues, and the time-triggered schedule to open and close the gates is saved in a "gate control list". 18

2.3 A model for feasibility prediction of TSN configurations that includes three steps from left to right: the generation of training samples of the training set (e.g., TSN configurations with known feasibility in our context), the feature engineering step that consists in the selection or the synthesis of features from the training samples, and the training step that use ML algorithms to learn patterns that can classify, *i.e.* draw boundary, between classes in the training samples based on their features. This figure is re-used in Chapter 3. 23

2.4 The structure of a GNN model applied to TSN network engineering. The GNN model first encodes a configuration as a graph, while components in the network are corresponding nodes in the graph. Topology and routing in the network are represented by edges between nodes. Each node and possibly each edge has an embedding, which is recurrently updated by artificial neural networks that aggregate embeddings of its neighbor nodes. The global attribute vector is aggregated from all embeddings and thus contains synthetic features of the configuration, which is used for the prediction step. This figure is reused in Chapter 4 with more details provided. 27

3.1 Network topology used in the experiments (topology from [7]). 41

3.2 During the generation of the training set, TSN configurations are labeled as feasible or non-feasible, respectively represented by blue triangles and red stars. The labels are the feasibility derived by precise analysis. The ML algorithm learns from the values of features the patterns needed to predict the feasibility of configurations. A border between the feasible and non-feasible groups is virtually drawn by the ML algorithm to classify unseen configurations. 43

3.3 The curves represent the percentage of non-feasible TSN configurations in the entire training set (y-axis): solid green for FIFO, dashed orange for Manual, dotted red for CP8, and dash-dotted blue for Preshaping. The x-axis is the maximum load over all links. The number of non-feasible cases is clearly correlated to the maximum load, which statistically increases along with the larger number of flows. In addition, more complex TSN solutions lead to a lower number of non-feasible configurations, as better scheduling solutions can be derived. 48

3.4 The prediction accuracy of k-NN and the size of the training set are shown on the y-axis and x-axis, respectively. The accuracy and the training set size linearly grow together, then the accuracy reaches a plateau when the training set size is over 3250. The non-deterministic properties of TSN configurations in the testing set cause a small fluctuation in the accuracy, less than 0.11%, in the range 3250 to 5200 training samples. These results are with Preshaping scheduling solution and the number of nearest neighbors k for k-NN is 20. . . 49

3.5 Boxplots shows the variance in the prediction accuracies of k-NN when there are changes in the payload of critical flows with respect to the training data. The variations in either positive or negative directions are between 10% and 90% of the original payload. Each box represents the results of 9 experiments with a granularity of change of 10%. Each horizontal line is the prediction accuracy of k-NN when the traffic patterns of the testing set are identical to the training set. The horizontal line is therefore the baseline accuracy of the box with the same color. 53

3.6 Time to determine feasibility with k-NN, precise and approximate analysis. The y-axis is the accuracy (%) and the x-axis is the total execution time in seconds in log scale. The results correspond to testing sets of size 10^3, 10^5, and 10^6, respectively in yellow, blue, and red. TSN configurations have a number of flows ranging from 50 to 350 and apply the Manual scheduling solution. The results of k-NN are represented by squares, while points are for approximate analysis and triangles for precise analysis. When the testing set size grows, the running time of k-NN only increases slightly. The running time of schedulability analyses is linear with the number of configurations. . . 55

3.7 The topology used in the second part of this chapter. The unicast flow highlighted in the figure is from ECU1 to ECU6. 56

3.8 Histograms (frequency of appearance) of feasible configurations (blue) and non-feasible configurations (red) with respect to each feature, counted on the training set containing 4000 samples. The histograms suggest that k-NN is able to distinguish two well-distinct groups, however, the classifier can make wrong predictions in the overlapping areas (purple). 57

3.9 Predictions of k-NN for Preshaping scheduling solution on 3D plots (different viewpoints). The testing set contains 1000 unseen configurations. Green represents true positives, red true negatives, and black wrong predictions, which mostly happen in the mixed area between feasible and non-feasible groups. 59

3.10 The x-axis is running times measured on an Intel 8700 with 6 cores, and the y-axis is the accuracies (%) with the Manual scheduling solution, for a testing set containing 10^3, 10^5, and 10^6 configurations. Results of k-NN, the hybrid model, and Precise analysis are respectively represented by squares, points, and triangles. The threshold in the hybrid model is set to achieve 99% accuracy. Each method provides a useful trade-off between performance and computing time, except with the testing set containing 10^3 configurations that Precise analysis is a dominant technique. 62

4.1 The Graph Neural Network (GNN) model takes raw data from a TSN configuration and encodes the configuration as a graph. The components in the network (flows, links, queues) are nodes in the graph, filled in gray color. Each node is represented by an initial embedding vector that captures the characteristics of the corresponding component. For instance, the packet payload of a flow can be scaled in the range [0, 1]. The embeddings are updated through recurrently learning to exploit complex relations in the network. A global attribute vector representing the whole graph is aggregated from embeddings of all nodes. It is used by a simple neural network dedicated to feasibility prediction. 80

4.2 We extend Figure 4.1 to give more details on neural networks that constitute the GNN model. Embeddings of nodes and the global attribute of the graph (encoding a TSN configuration) are updated through three loops with two neural networks f_v and f_u, respectively. Finally, the neural network f_{pred} uses the global attribute to predict the feasibility of the configuration. 83

4.3 Details steps to model a TSN configuration as a graph in the GNN model. We use several colors in the figures: yellow for end-nodes (only in (a)) and links, blue for switches (only in (a)) and queues, green for flows. For the sake of simplicity, the encoding process is separated into steps, however, only the final graph is used in the GNN model. 85

4.4 Topologies used in the testing sets (captures from RTaW-Pegase). 93

4.5 Prediction accuracy (%) of k-NN, GB, and GNN with two ensemble techniques on all data sets, while outliers are removed. Boxplots are the variance in the performance of the 32 individual GNN models, which increases from left to right corresponding to how dissimilar the testing sets are from the training set. 99

4.6 Visualization of the variance in the performance of 32 individual GNN models. The prediction accuracies of each model are shown by points with a distinct shape and color. The thick lines in blue and green are the accuracies of the ensembles with majority voting and GB, respectively. A GNN model that performs well on some testing sets can also have a poor performance on other data sets. 101

4.7 We compare the predictions of GNN to true feasibility values on 4 testing sets with distinct topologies. x-axis and y-axis are respectively the numbers of flows and the proportion of feasible configurations in the data sets. The color areas represent the 95% confidence interval. 104

5.1 Some improvements applied to the base GNN model (a): global pooling layer (b), and multiple channels (c). 113

5.2 In Experiment 1 and Experiment 2, resp. denoted as E11 and E21, we use two training sets that do not only have star-shaped topologies as in the base GNN model. This change leads to better accuracies in some testing sets, however, the accuracies decrease in testing sets #1 to #5 that have star-shaped topologies. 116

5.3 In Experiment 3, denoted as E31 and E32, we evaluate the influence of using ELU activation function. E31 is the base GNN model that uses ELU, while E32 has a similar setup as E31 but uses the focused training set. E21 uses the focused training set and not ELU, shown for comparison. E31 and E32 have higher prediction accuracies than the models that do not use ELU, except on testings sets #6, #9, and #11. 118

5.4 In Experiment 4, we evaluate the influence of using a global pooling layer, trained with star topologies (E41) and the focused training set (E42). The global pooling layer does not lead to improvement in the performance, however, it suppresses the final neural network and thus makes the GNN model simpler. 119

5.5 In Experiment 5, we evaluate the influence of using multiple channels at the first two layers of the GNN model, trained with star topologies (E51) and the focused training set (E52). This technique does not increase the prediction accuracies overall, while it requires more computation resources. 120

5.6 In Experiment 6, we evaluate the influence of scaling up the size GNN model: the number of training epochs is increased from 20 to 50 (E61), the number of layers is increased from 3 to 5 (E62), the size of embedding is expanded from 16 to 32 (E63), and finally, E64 combines the larger embedding and more training epochs. 122

5.7 In Experiment 7, we evaluate the influence of larger training sets made up of 20000 samples instead of 10000 and use it to train the improved and expanded GNN models (E71 and E72, resp.). Both models can generalize better with the larger training set, except the test case #13 that is discussed in §5.4.3. . . 123

5.8 Four variant GNN models show variations in predictions accuracies, expressed in %, with the based GNN model. 125

5.9 Testing set #13 with Volvo topology: frequencies in histograms of the number of individual GNN models that predict as feasible (in the ensemble). It is counted on each TSN configuration and summed up for 1000 configurations in the testing set. Bin 0 is the case that all individual GNN models in the ensemble predict a configuration as non-feasible, and bin 32 for feasible. . . . 126

5.10 Performance of GNN with semi-supervised learning, compared to supervised learning. There are 10,000 labeled samples and 100,000 unlabeled samples in the training set. Although Semi-supervised learning improves the performance of GNN on many testing sets., it does not achieve its main goal of reducing the number of labeled samples in the training set. 130

5.11 An example of SSL: On the left hand-side, there are two points of view of the same cat's image as positive samples, and a point of view of the dog's image as a negative sample. An encoder is used to generate three embedding vectors. The role of the encoder is to maximize the similarity between embeddings of positive samples while minimizing the similarity between positive and negative samples. The figure on the right hand-side works in the same manner. This figure is from [8]. 132

5.12 Visualization of embeddings that represent TSN configurations on some testing sets. These embeddings are produced by the SSL model after being trained with 100,000 unlabeled configurations. The 3D plots are generated by t-SNE available in Python. 135

List of Tables

3.1 The traffic in the network contains three classes. Each flow has a deadline constraint and its WCTT must satisfy the constraint. The frame size means the payload size of a packet. 41

3.2 Four scheduling solutions have different optimal k* values and different performances. Acc, TPR, TNR and K are the abbreviations for Accuracy, True Positive Rate, True Negative Rates and Kappa statistic, respectively. These results are obtained with a testing set of 1300 unseen configurations with 5-fold evaluation. Column (F) is the proportion of feasible configurations in the training set of 5200 data points corresponding to each scheduling solution. 50

3.3 Summary of experimental results. The approximate analysis leads to the less accurate results for CP8 with 86.31%. This type of schedulability analysis is not available for the Preshaping scheduling solution. We note that neither approximate nor precise analysis produces false positives, which is the case when using ML. 54

3.4 Correlations between the values of the selected features and the feasibility of TSN configurations (0 for feasible and 1 for non-feasible) with the Pearson coefficient. The correlation measured on the Gini index is negative since a high Gini index indicates a less balanced network, that is likely to possess a small number of flows. 58

3.5 The performance of the hybrid model with a threshold from 5% to 40%. The table is divided due to space limitation. The first row is the results with k-NN used alone. Column (#) indicates the number of configurations that undergo precise analysis. All performance metrics: Accuracy (acc), TPR, and TNR are in percentage. There are 1000 unseen configurations in the testing set. 61

3.6 We estimate the number of configurations undergoing precise analysis to achieve 99% accuracy in column #. k* is the optimal number of nearest neighbors that is different for each scheduling solution. 63

3.7 Estimated prediction accuracy (in %) if 10% of the configurations undergo precise analysis in the hybrid model. 63

3.8 Estimation on different testing set sizes (column #) of the speedup factors of the baseline hybrid model compared to precise analysis. The hybrid model is calibrated to achieve 99% accuracy. 64

3.9 Performance of the multi-stage hybrid model with k-NN, approximate analysis, and precise analysis, expressed in percentage. The testing set has 1000 unseen configurations. The first row is the results for k-NN alone. The value k for FIFO is 30 and Manual is 20. Columns (1) and (2) respectively show the number of configurations that undergo approximate analysis and precise analysis. 66

4.1 Topology and traffic patterns for TSN configurations in the training set. Frame size is the payload only. Testing sets may follow different topology and traffic patterns. The requirement is that all flows must satisfy their deadline constraints. 78

4.2 Summary of training, evaluation and testing sets. 91

4.3 Summary of parameters of the GNN model 95

4.4 Prediction accuracies (%) of GNN in a comparison with k-Nearest Neighbors (k-NN), Gradient Boosting (GB) on 13 testing sets. The baseline is the accuracy of a trivial model that always predicts configurations as feasible. Two columns En-MV and En-GB are respectively the accuracies of ensembles with two different techniques: majority voting and Gradient Boosting. The values highlighted in red are the cases conventional ML algorithms have low prediction accuracy. 98

5.1 Summary of all changes in the experiments. 1^{gl} and 4^{gl} denote the use of a global pooling layer with 1 and 4 channels, respectively. Exp. and Abbr. are respectively Experiment and Abbreviation. Imp. and Ext. resp. stand for the Improved model and the Expanded model. E31 and E32 denote Experiment 3, resp. at step 1 or step 2. In other experiments, we also use this naming convention. 114

5.2 A summary of prediction accuracies, expressed in %, of all GNN models in this study. The baseline is the ratio of feasible configurations in the data sets, derived by precise analysis. 121

5.3 Execution times of the ensemble of 32 individual GNN models, compared to schedulability analysis. The first row is the absolute value of running times on a single GPU. While the running time of GNN is a constant on all configurations, it varies in schedulability analysis depending on the complexity of TSN networks. In the second row, the first column is the speedup factor of GNN, corresponding to the smallest and largest factor. The rest is the relative speedup factor compared to the base GNN model. 127

Acronyms

AI Artificial Intelligence. 1, 7, 8, 11–13, 16, 21–24, 30, 106, 141–144, 146

CAN Controller Area Network. 2

CBS Credit-Based Shaper. xv, 18, 20, 80, 81, 143

CP8 Concise Priorities with 8 priority levels. xvi, xxi, 39, 47, 48, 50, 51, 53, 54, 58, 61–65, 74, 139

DL Deep Learning. 24, 25, 29, 77, 138

DSE Design-Space Exploration. 1, 6–10, 14–16, 19, 29, 31, 36, 37, 68, 73, 74, 102, 103, 105, 106, 139, 141

ECU Electronic Control Unit. 4–6, 16, 17, 92, 95

ELU Exponential Linear Unit. xix, 117, 118, 120

FIFO First In First Out. xvi, xxii, 12, 31, 39, 42, 47, 48, 50, 51, 53, 54, 58, 60–64, 66, 74, 139, 140

FNR False Negative Rate. 32, 46

FP False Positive. 32

FPR False Positive Rate. 32, 46, 53, 54

GB Gradient Boosting. xviii, 97, 99–101, 103, 105

Gbps Gigabits per second. 2, 40, 56, 97

GNN Graph Neural Network. xv, xviii–xx, xxii, xxiii, 9–13, 26–32, 71–76, 78–90, 92, 94, 95, 98–107, 109–131, 134, 137–147

k-NN k-Nearest Neighbors. xvi–xviii, xxi, xxii, 9, 35, 36, 38, 42, 44–57, 59–68, 71, 72, 96–99, 106, 134, 136, 139, 140

Mbps Megabits per second. 2

ML Machine Learning. xiv–xvi, xxi, xxii, 7–13, 21, 23–29, 31, 32, 35–38, 42, 43, 45–48, 50, 51, 53–56, 58, 60, 62–64, 66–69, 71–77, 80, 88, 95–102, 105–107, 109–111, 125, 127–129, 131, 137, 139–142, 146

NC Network Calculus. 17, 19, 30, 76

OPA Optimal Priority Assignment. 19, 39, 76, 88, 112, 127, 142

QoS Quality of Service. xiv, 1–6, 11, 13–15, 18, 39, 40, 74, 80

RQ Research Question. 9, 10, 36, 67, 72, 73, 106, 110, 138–141

SSL Self-Supervised Learning. xx, 131–138, 141

ST Scheduled Traffic. 20, 21

TAS Time-Aware Shaper. xv, 18, 20, 80

TNR True Negative Rate. xxi, 31, 45, 50, 51, 53, 54, 61, 66

TPR True Positive Rate. xxi, 31, 45, 50, 51, 54, 61, 66

TSN Time-Sensitive Networking. xiv–xviii, xx–xxiii, 1, 3–21, 23, 25–27, 29, 31, 32, 35–40, 42–53, 55, 58–60, 67–69, 71–80, 82–87, 89–92, 94–96, 98, 100–107, 109–112, 126–128, 130, 131, 134–136, 138–147

WCTT Worst-Case Traversal Time. xxi, 8, 16, 17, 19, 30, 40, 41, 52, 76

Chapter 1

Introduction

Abstract

Ethernet is becoming the prominent layer 2 protocol for communication in critical systems. However, it does not offer mechanisms to guarantee timing and reliability constraints. Time-sensitive networking (TSN) mechanisms have been introduced to provide temporal predictability for Ethernet as required in real-time systems. As TSN provides more options for the design, design-space exploration (DSE) becomes more complex: there is a large set of possible combinations of QoS mechanisms to create candidate solutions.

Schedulability analysis is required to verify the feasibility of TSN configurations, however, it is very compute-intensive and thus it constitutes a bottleneck in DSE. The motivation of this thesis is to use AI techniques for feasibility prediction in order to speed up DSE. In this chapter, we also present the challenges of applying AI in our context. The contributions presented in this thesis to address these challenges are summarized at the end of this chapter.

1.1 Context and Motivation

This section presents the general context of the thesis, *i.e.* the design of Ethernet TSN networks, which leads to the motivation of applying AI techniques to speed up DSE. More details about TSN and AI techniques are given in Chapter 2.

1.1.1 Adoption of Ethernet to real-time communications

Fieldbus protocols such as Controller Area Network (CAN) are widely used for real-time communications in embedded systems. However, there are two main reasons that restrict the use of these protocols in the future. The first reason is the limitation of data transmission rate, for instance, CAN and its modern variants CAN FD and CAN XL offer data rates in the Mbps range (see Chapter 6 of [6]), which is not enough to handle the increasing volume of data generated by the sensors nowadays. Indeed, modern sensors like light detection and ranging sensor (e.g., LIDAR) can generate data streams in Gbps. In addition, infotainment services including audio and video streams are increasingly used in embedded systems. A solution to cope with these communication needs is to deploy several networks and use domain-specific technologies like LVDS networks (Low-voltage differential signaling) for video streams [9]. Other limitations of fieldbus networks is that their topology may not be flexible (e.g., CAN has a bus topology), the maximum number of devices in the network may be limited and the bus length reduced (*i.e.*, 40m for CAN at 1Mbps [6]). As a result, large embedded systems need multiple networks to meet the requirements in real-time communications. For instance, a car may have separate communication networks for power systems, chassis systems, devices in the cabin, infotainment systems and so on.

Ethernet is a promising protocol for modern high-speed networks in mixed-criticality systems. The benefits of Ethernet are the ultra-low communication latencies with data rates up to 10Gbps, its flexible topology, and the availability of cost-effective components such as switches or cables. There is also a large community that maintains and actively updates the set of Ethernet protocols to meet future requirements. With the possibilities of Ethernet, the existence of multiple communication networks become questionable. E.g., can we use Ethernet for both command and control streams and infotainment streams and thus reduce the cost of the communication architecture? The most important Quality-of-Service (QoS) requirement for real-time communications is usually the worst-case timing requirement, namely the worst-case traversal time of any packet must satisfy its deadline constraint. This is not originally supported by the Ethernet protocol, which offers only best-effort delivery. Therefore, how to assure deterministic communication on top of Ethernet protocol is a key question for the adoption of Ethernet for real-time communication.

1.1.2 Time-Sensitive Networking (TSN)

Figure 1.1 illustrates the rationale for TSN, that is built on top of Ethernet protocol to meet QoS requirements.

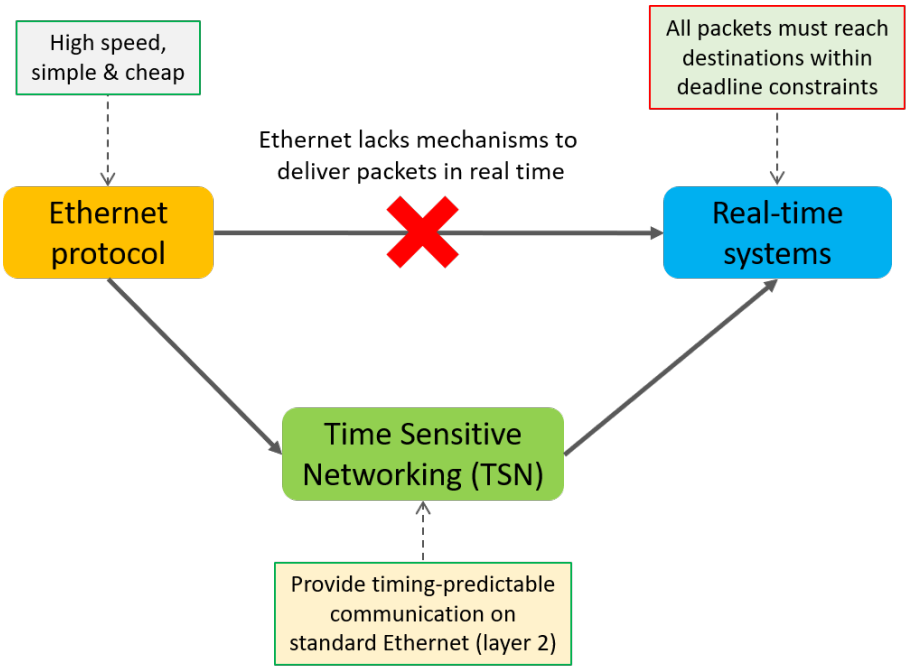


Figure 1.1: Time-Sensitive Networking (TSN) is a set of mechanisms on top of standard Ethernet that ensure new forms of QoS such as timing predictability.

In addition to transmission delay guarantees, real-time systems may have other QoS requirements such as throughput, reliability, and security that cannot be satisfied by the standard Ethernet technology. To address these demands, the IEEE802.1 Time-Sensitive Networking Technical Group has developed new technologies and standardized more than 20 individual TSN standards (see [10–12] and chapter 2 for a survey of ongoing works related to TSN). Key features of TSN include:

- Time synchronization: devices can share a common understanding of time, aka a global clock.
- Traffic scheduling: to meet various timing requirements packets transmitted over the network can be scheduled with various scheduling policies, such as priority-based scheduling, credit-based shaper, time aware shaper, etc. This also gives a lot of flexibility to designers.

- Fault tolerant communication: the possibility to define redundant paths for network streams of high criticality to tolerate transient and permanent failures, which is essential for mission-critical networks.

The use of one or several TSN QoS mechanisms with adequate parameters, that is a *network configuration*, should ensure that the network is *feasible*. That means the configured network satisfies all QoS requirements such as latency, throughput, or reliability. Only the worst-case timing requirement will be considered in this thesis, *i.e.* a TSN configuration is feasible if all packets satisfy their deadline constraints. In order to determine whether a TSN configuration is feasible, a feasibility test is performed. There are two approaches for a feasibility test: schedulability analysis and simulation. While the second approach is simpler, it does not guarantee to find the worst-case communication latencies, as some pessimistic scenarios may not have been seen during the simulations. On the other hand, the first approach, which is based on a mathematical model, provides upper bounds on communication latencies for all packets. More details about schedulability analysis is provided in Chapter 2. In practice, performing a schedulability analysis is compute intensive, in particular for large networks. Our preliminary work in [1] shows that schedulability analysis, with an optimized implementation, takes on average 471ms to determine feasibility for a configuration having two switches and 500 flows. For larger configurations, the execution time can be up to several minutes.

1.1.3 Design-space exploration of Ethernet TSN networks

Design of real-time communication network

In the following, we consider the embedded electronic system in an automotive to illustrate the use of design-space exploration. In an automotive electronic system, actuators and sensors are connected to Electronic Control Units (ECUs). In a simplified manner, the development process has several phases as shown in Figure 1.2: the definition of hardware, functional architecture, and communication.

- The *definition of hardware* involves architecture design and choice of the technologies, for instance, certain ECUs need to be located near certain devices. In this thesis, we assume that the definition of hardware architecture has been done, *i.e.* the

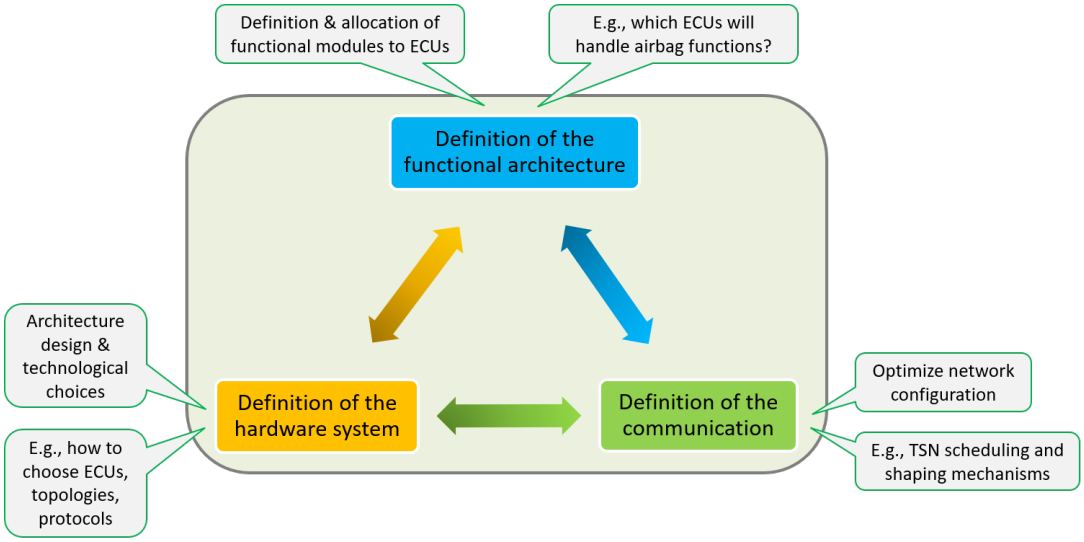


Figure 1.2: The product development process for an automotive electronics system (inspired from Figure 8.5 in [6]) requires the definition of the hardware system, functional architecture, and communication. A combination of options generates a large set of possibilities. The designers not only need to find the optimal solutions but also guarantee specific requirements. In this thesis, we assume that the hardware system has been fixed and thus the design space includes possible combinations of design options, each can be configured by one or more TSN mechanisms to satisfy QoS in real-time communications.

topology of the in-vehicle network is fixed, and therefore we focus on the next phases in the design process.

- The *definition of the functional architecture* involves the definition and allocation of functional modules to ECUs. We note that ECUs need to communicate with each other. For instance, the ECU that reads pressure sensors at tires may have to send messages to the ECU that controls the brakes, the suspension or the dashboard. It is the interactions between ECUs that generate the various types of traffic in the network.

- The *definition of communication* targets the optimization of network configuration. It is important to guarantee that messages sent from one ECU to the others are transmitted within their deadline constraints as many functions are pertaining to the dynamics of the vehicle. Beyond deadline constraints, which serve as the basic requirement in real-time communication, designers may use various metrics to find optimal design solutions, for instance, efficiency or relevant costs [13]. We note that this phase becomes harder when the number of ECUs and the number of functions

increases, as many packets compete for bandwidth and each has different deadline constraints. The combination of these requirements makes the design space of a real-time communication system exponentially large.

New trade-offs with TSN

In the studies of this thesis, it is assumed that the topology of a communication network is fixed, as usually done very early of the design of an automotive embedded system. The definition of functional architecture phase mentioned above will generate a large number of candidate configurations, each having a distinct set of traffic streams created by the allocation of the functions to the ECUs.

During the definition of communication phase, designers can decide to apply one or more TSN mechanisms to a candidate network to guarantee the requirements on the communication (see Chapter 2 for a formal definition of network feasibility). The designers will choose among the feasible network configurations the best ones, based on predefined metrics, for the next steps. Numerous TSN mechanisms and all the possible ways to use them in a combined manner give thus more room for designers to improve QoS. However, the complexity of the TSN mechanisms very diverse, thus requiring potentially very different computation times to verify the feasibility of a candidate network. For instance, let us compare two approaches to priority allocation: manually group traffic into classes based on domain expertise, and search for optimal assignment. In the first approach, the verification of network feasibility requires only the execution time of a single schedulability analysis. On the other hand, the second approach usually requires many steps to derive the final result, leading to multiple executions of schedulability analysis. Although the second approach leads to a much higher proportion of feasible networks in practice, it requires a larger computation time compared to the first approach. This is the example of typical performance-execution time trade-off in DSE.

In summary, TSN gives many options to the designers at the expense of a larger design space, since there are many scheduling policies that can be applied to a candidate network configuration. Ideally, designers will explore all TSN mechanisms for candidate networks and choose the best configurations based on the predefined metrics. However, it is not possible due to the computation time required for schedulability analysis, which may take

weeks to test a medium-size set of candidate configurations (see experiments in [1]). This delay becomes a bottleneck in the DSE, which calls for machine learning to speed up the verification of TSN configurations.

1.1.4 Artificial Intelligence for DSE of TSN networks

Artificial Intelligence (AI) includes a set of techniques that can be used to build a program that can make prediction with a certain degree of accuracy. Research of AI is categorized into sub-domains, in which machine learning (ML) is probably the most active branch at this time. ML algorithms are well-known for their ability to learn from data to recognize patterns, as a result, a trained ML model can be used to make predictions.

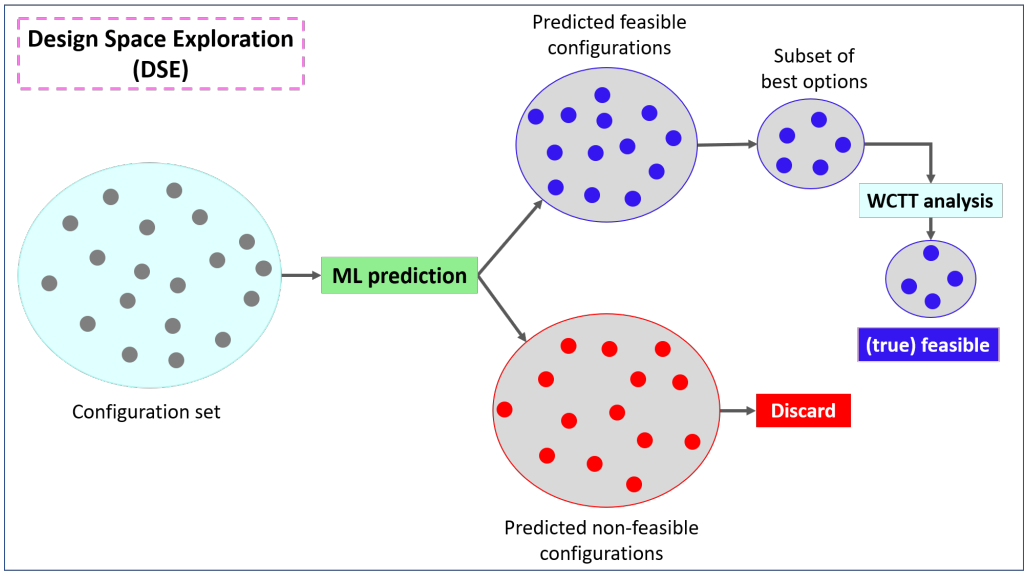


Figure 1.3: In the design-space exploration of TSN networks, machine learning (ML) is applied to quickly predict the feasibility of TSN configurations without performing schedulability analysis. The downside is that ML can lead to false positives. Therefore, the designers need to verify the set of selected candidate solutions with schedulability analyses to ensure that false positives do not happen.

ML is therefore a promising tool to speed up DSE of TSN networks, as illustrated by the process depicted in Figure 1.3: the trained ML model is applied to predict the feasibility of candidate configurations. Only the predicted feasible configurations are considered, while the configurations that are predicted non-feasible are discarded. As ML algorithms will occasionally lead to false predictions, the subset of best candidate solutions will undergo schedulability analysis. The result is a guaranteed delay bound,

a.k.a. worst-case traversal time (WCTT) for each traffic stream in the configurations, and therefore the true feasibility of configurations in the subset. Since only a fraction of the candidate configurations undergo time-consuming worst-case analysis, the ML approach potentially provides a large speedup factor for DSE and allows designers to explore other dimensions in the design of Ethernet TSN networks. However, it is important to emphasize that ML can only be an effective alternative to schedulability analysis if the prediction accuracy is sufficiently high, which requires a number of challenges to be successfully met.

The first challenge is to decide which ML model to use in our context. ML algorithms can be classified as conventional algorithms and deep learning models based on neural networks. The first class includes simple ML algorithms based on statistical inference that usually executes rapidly. The limitation of these algorithms is that they may not have enough expressive power to model complex networks and thus may lead to insufficient prediction accuracy. In addition, these algorithms require in input features that represent the networks, and the algorithms learn from these features to recognize patterns. These features are manually chosen or ”engineered” based on domain expertise, however, they may not be able to convey all the information about TSN configurations. For instance, it is not clear how to choose features that represent the routing of the packets in the network. Without such information, the rate of wrong prediction might be very high. On the other hand, deep learning models outperform the conventional ML model in many applications due to their ability to capture all information and generate synthetic features, but their development may require more advanced AI techniques, and the training time might be a hindrance to their application.

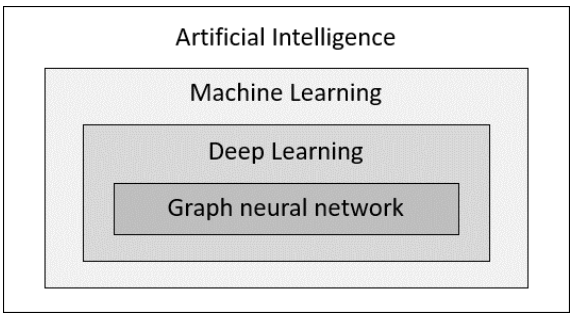


Figure 1.4: Artificial intelligence and sub-domains relevant to this thesis.

The second challenge to overcome is that topologies and traffic characteristics vary

between TSN configurations. What we expect from an ML model is that, after being trained with a certain number of sample configurations, it is able to predict with high accuracy the feasibility of configurations with unseen topologies or unseen traffic patterns. The generalized ability to correctly predict the feasibility for a wide range of TSN configurations requires a powerful ML model that can encode not only the network devices but also the network topology, the traffic characteristics and the routing. In recent years, graph neural network (GNN), a deep learning paradigm tailored for graph-based data, has proven to have the potential to solve difficult problems in networking such as routing optimization [14] and performance prediction [15]. However, in order to be a good candidate to model TSN networks, GNN needs to be specialized to tackle the constraints of real-time communications. In addition, it is well-known that ML requires exposure to as many cases as possible in order to achieve good performance. This implies domain knowledge and extensive trial and error experiments to generate a sufficiently comprehensive training set for the ML model.

These challenges in the use of ML for TSN feasibility prediction, and real-time networks at large, have not been solved yet by the real-time system research community to the best of our knowledge. This motivates us to explore these research questions, and possibly open up the possibility to use ML as an alternative to schedulability analysis to speed up DSE of Ethernet TSN networks.

1.2 Research questions

In this thesis, we address these main research questions (RQs):

RQ1: We first ask whether ML algorithms can be used to predict the feasibility of TSN networks in an efficient manner in terms of time and accuracy. In order to answer this research question, we intend to use traditional ML algorithms such as k-Nearest Neighbors (k-NN), while domain expertise is leveraged in the feature engineering step.

RQ2: We pose the question of how to control the risk of wrong predictions when applying ML algorithms. Since the outcomes of an ML model is not perfect, can we combine ML with reliable schedulability feasibility tests to mitigate this issue ?

RQ3: If the answer to RQ1 is positive, we then ask whether ML models can generalize to TSN networks that are different from those in the training set. In case traditional ML algorithms do not attain this ability, should we harness specific neural networks models like GNN to achieve this ability to generalize?

RQ4: Can we apply recent advanced deep learning techniques such as self-supervised learning to improve the performance of the ML models?

1.3 Contribution

This thesis aims to develop a viable alternative to schedulability analysis in the early phase of DSE using ML models that can predict the feasibility of TSN configurations. Advanced ML techniques are then applied to improve the performance of the proposed ML models. In particular, the application of deep learning with GNN boosts the generalization ability of ML to a wider range of topologies and traffic patterns. Since the prediction of ML is not perfect and thus false conclusions can be made, we also propose methods that effectively control its uncertainty. As a result, the proposed ML models achieve high prediction accuracy and a drastic speedup factor compared to schedulability analysis. The main contributions of the thesis are as follows:

- To the best of our knowledge, we are the first group that uses ML algorithms to predict the feasibility of TSN configurations [2]. A combination of conventional ML algorithms and domain expertise in manual feature selection provides a high prediction accuracy on TSN configurations with fixed topology and various traffic patterns.

- We propose a hybrid model combining ML prediction and schedulability analysis [3] to control the uncertainty of ML. The hybrid model relies on an estimate of the confidence of the prediction. Schedulability analysis is triggered if the confidence is lower than a user-chosen threshold, reducing thus the risk of false prediction with ML.

- We develop a graph neural network (GNN) to predict the feasibility of TSN configurations with various topologies. Being trained with star-shaped topologies, one of

the most popular topologies in industry, the GNN model can generalize to diverse unseen topologies in 13 distinct test cases. We also notice that although GNN is a powerful ML model, it may return wrong predictions for some configurations. We discover that the set of TSN configurations mispredicted with one GNN model does not totally overlap with the set mispredicted with other models. Therefore, an ensemble of individual GNN models with different initialization is used as a fault-tolerance mechanism to minimize probability of wrong conclusion for a TSN configuration.

- Finally, we also explore possibilities to improve the performance of the GNN model. Based on insights into the training data, we create a new training set that is more diverse in the topologies and focuses on configurations that are hard to predict. Neural networks at the core of the GNN model are also enlarged and more advanced deep learning techniques are applied, which contribute to an improvement of the overall prediction accuracy on various test cases.

1.4 Thesis Organization

This thesis is organized in 6 chapters as follows:

- Chapter 1 - **Introduction**. This chapter introduces the context and motivation of the studies in the thesis, that aim to use AI techniques to speed up design-space exploration of TSN networks. There are many challenges for the application of AI to real-time systems, for instance, how to use data from TSN networks and which AI model should be chosen. Therefore, we raise four research questions to address these challenges. The end of this chapter summarizes the main contributions of the thesis, which are the result of the studies that answer research questions by giving solutions for possible challenges.

- Chapter 2 - **Background and Preliminaries**. In this chapter, we introduce real-time communication networks and the need for TSN mechanisms when using Ethernet with QoS requirements. Artificial Intelligence, specifically Machine Learning, is

also introduced. The applications of ML and advanced deep learning techniques in networking are reviewed as well.

- Chapter 3 - **On the Use of Supervised Machine Learning to TSN Networks**. The first study in this chapter is the application of ML to predict the feasibility of TSN configurations with fixed topology. Several TSN mechanisms are considered, including FIFO, scheduling with different priority levels for different traffic classes, and shaping on some traffic classes. The latter part of the chapter presents a hybrid approach combining ML and schedulability analysis to control the uncertainty of ML.

- Chapter 4 - **Application of Graph Neural Network to TSN Networks**. In this chapter, we apply graph neural network (GNN) to predict whether there exists a priority assignment that would lead to the feasibility of a TSN configuration. The distinction between the study in this chapter and Chapter 3 is that the GNN model is trained on random star-shaped topologies but it can predict with good accuracy the feasibility of networks with unseen topologies and traffic patterns.

- Chapter 5 - **Improvements to the Graph Neural Network Model**. This is an effort to improve the performance of the GNN model by applying more advanced deep learning techniques. A more diverse training set is generated. We also enlarge the training set and the neural networks in the GNN model, which significantly increases the prediction accuracy. In this chapter, we also report on efforts to apply promising AI directions to our problem, namely semi-supervised learning and self-supervised learning. Although the applications of these techniques do not lead to positive results, they are promising directions to mitigate the requirement of very comprehensive training set size in deep learning.

- Chapter 6 - **Conclusion**. This concluding chapter highlight potential directions to expand this line of work in the future. Practical skills related to the development of the GNN model are also mentioned in this chapter, and we discuss how to ease the application of the ML models developed in this thesis in the industry.

Chapter 2

Background and Preliminaries

Abstract

This chapter presents background information related to the studies conducted during the thesis. We first introduce the general "system model" considered for real-time Ethernet communication and TSN configurations, while the details of topologies and traffic characteristics will be tailored for each study. In the next section, we introduce TSN QoS mechanisms for timing, which allow for timing-predictable communications on standard Ethernet. The third section introduces a definition of Artificial Intelligence (AI) and Machine Learning (ML). ML will be used in this thesis to predict the feasibility of TSN networks without the execution of schedulability analysis. Classically ML is grouped into conventional ML models that require as inputs features chosen manually, and deep learning models that can generate synthetic features. Graph Neural Network (GNN), an advanced deep learning model tailored for network data is introduced. This section also surveys the applications of ML in networking, in particular the applications of GNN. Finally, we review the performance criteria for ML models that will be used throughout the thesis.

2.1 Real-time Ethernet TSN communication

This section is organized into three parts. The definition of Ethernet TSN networks is presented in the first part, followed by the general system model to generate sample TSN

networks in DSE studies. Performance evaluation of TSN networks, mainly with feasibility tests based on schedulability analysis, is introduced in the third part of the section.

2.1.1 Definition of Ethernet TSN network

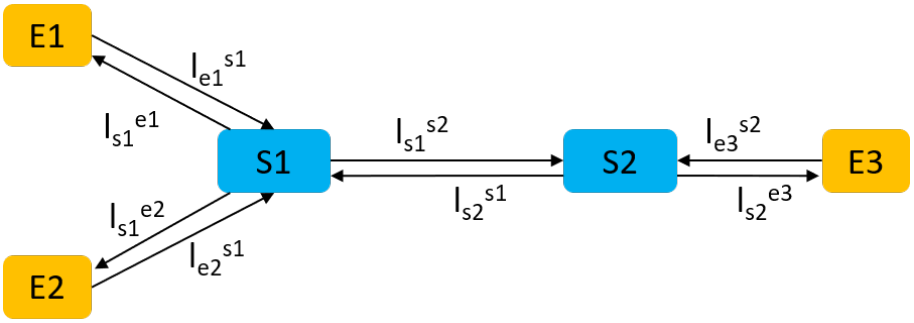


Figure 2.1: Example of a TSN network with two switches (S1, S2) and three end-nodes (E1, E2, E3). Each full-duplex link is considered as two uni-directional links. For instance, the full-duplex link connecting end-node E1 and switch S1 is made up of link l_{e1}^{s1} from E1 to S1, and link l_{s1}^{e1} from S1 to E1. In a switch, there are ingress ports that connect to incoming links and egress ports that connect to outgoing links. Each egress port has a buffer to store packets waiting to be sent. All traffic flows (not shown in the figure) have to be sent from one end-node to other end-node(s) and forwarded by one or more switches. This figure will be re-used in Chapter 4.

In this thesis, we consider real-time communication networks containing network devices, network interfaces and Ethernet switches, supporting TSN scheduling mechanisms. An example of TSN network is shown in Figure 2.1. It includes three end-nodes and two switches that support TSN mechanisms. A full-duplex link connecting an end-node to a switch or two switches is considered as two directional links. Each switch has ingress ports connecting to incoming links and egress ports connecting to outgoing links. There is a buffer at each egress port to store packets waiting for sending. The terms *traffic flow* and *traffic stream* are used interchangeably to refer to the data sent from one sender to one receiver in unicast communication. For each flow, it follows a path from source to destination determined by the routing. The source and destination of flows are end-nodes. A multicast flow with n receivers is thus counted as n individual flows. Each flow in a real-time network typically has a *deadline constraint*, which is the maximum transmission delay allowed for any of the packets of the flow. The adoption of Ethernet protocol for real-time communications is feasible with TSN mechanisms to meet timing QoS requirements.

A network, together with the set of traffic flows transmitted over it, for which one or more TSN QoS mechanisms can be configured to schedule the traffic, is considered as a TSN *configuration*. There can be many configurations derived from the same network with an identical flow set, however different in terms of scheduling parameters. In the DSE of an Ethernet TSN network, a huge set of candidate configurations is usually generated due to the possible combinations of design options (e.g., function allocation) and TSN scheduling solutions.

2.1.2 General System Model

The study of DSE in this thesis requires making assumptions about TSN networks, and a general system model.

Assumptions about TSN networks

The following assumptions are either the norm today in mission-critical TSN networks or required by schedulability analysis (see Section 2.1.3). These assumptions are met by the contributions presented in the next chapters.

- The routing of flows in a TSN network is static. Therefore, all packets of a flow are delivered over the same path.
- The memory at the buffers of switch ports is large enough to guarantee that there is no buffer overflow in the worst case. This property can be checked by schedulability analysis. There is no transmission errors and no packet losses on the links or in the buffers.
- Traffic streams are either periodic, sporadic, or sporadic with bursts. Sporadic means that two successive packets are transmitted at least x ms apart. The third type of traffic stream refers to, for instance, video traffic from a camera that requires to be segmented into a burst of multiple consecutive packets.
- The maximum packet size of any flow is known and does not change during the operation of the network.

- The switching delay is up to 1.3us. Although this value varies between switches, it is in line with the typical latencies of modern Ethernet switches [16].

Topology and traffic patterns

Topology and traffic patterns are needed to generate training TSN configurations. The AI model (see Section 2.3) learns from these configurations to predict the feasibility. Therefore, their characteristics should be realistic and ideally similar to the ones of the TSN networks created during DSE. The following list presents the skeleton for the topologies and traffic characteristics of TSN networks in our work. These characteristics may be tailored for each study and then they will be described in the relevant chapters.

- A topology comprises end-stations, aka ECUs in the automotive, standard Ethernet switches, and full-duplex links. Only ECUs are sources and destinations of flows. Each ECU only connects to one switch, and there is no link redundancy in the topology.

- There are four types of flows: critical flows, audio flows, video flows, and best-effort flows. Critical flows are used for commands and control, e.g. in power and chassis systems. Audio flows transfer audio and video data, e.g. from cameras for driving assistance functions. Due to the amount of data, a video flow is transmitted in bursts of consecutive packets. The proportion of traffic belonging to each traffic type in a configuration varies for each study.

- Each packet of a flow is subject to a deadline constraint. The WCTT of the flow is computed by schedulability analysis and it must be less than the deadline constraint of the flow in order to meet the timing requirement. A multicast flow has only one deadline constraint, which means packets sent to all receivers must arrive within the deadline.

2.1.3 Performance evaluation of Ethernet TSN networks

This thesis solely focuses on timing, which means a configuration is said to be *feasible* if all flows in the network satisfy their deadline constraints. In contrast, a configuration

is *non-feasible* if at least one deadline constraint is not met. A *feasibility test* is needed to determine the feasibility of a TSN configuration. There are two main approaches for feasibility tests: *schedulability analysis* and *simulation*.

In the first approach, the worst-case traversal times (WCTTs), a.k.a. delay bounds, of flows are derived by mathematical models. According to [17], there are three types of analysis [1]: real-time analysis (RTA), network calculus (NC), and use of eligible intervals. Our studies apply NC-based schedulability analysis, supported by the RTaW-Pegase software [18] used during the thesis. The main reason for the use of NC is that it scales very well and tends to execute faster than the other analytic approaches. A summary of NC-based analysis methods for TSN is presented in [19]. The result of the schedulability analysis is pessimistic since the algorithms make conservative assumptions. Hence, an analysis may conclude that a feasible configuration is not feasible, but the opposite case will never happen. There are two classes of schedulability analyses available in Pegase [18,20]:

- *Precise analysis*: WCTTs are derived by a mathematical model whose execution time grows more than linearly [2] with the number of the flows in a configuration. Its result is considered as 100% accurate in our thesis as there is no better performance analysis model. The precise analysis is the default schedulability analysis used during our experiments without further notice.

- *Approximate analysis*: WCTT are computed in linear time but the results are more pessimistic than with the precise analysis. The accuracy of approximate analysis is lower than 100% but, similarly to precise analysis, it never wrongly predicts a non-feasible configuration as feasible.

In the second approach, the feasibility of a TSN configuration can also be derive by extensive simulations, e.g. using varying ECU clock drifts and ECU start times [21]. However, this approach cannot provide formal proof, as simulation does not guarantee that the worst-case scenario has been encountered. Schedulability analysis, as used in this

[1] While the authors in [17] consider machine learning as an analysis technique, the machine learning models actually predict the feasibility of TSN networks rather than performing analysis.

[2] The theoretical complexity is exponential but the developers of the RTaW-Pegase tool may have added local approximation schemes, e.g. linear approximation of stair-case functions, to keep the complexity under control.

thesis, is, to the best of our knowledge, the only trustworthy verification method for the feasibility of TSN configurations.

2.2 TSN QoS mechanisms for timing

The IEEE 802.1 Working group has defined, in more than 25 projects, TSN mechanisms to provide various types of QoS including timing and synchronization, reliability, and resource management [10–12]. This thesis focuses on IEEE 802.1 Timing QoS. Related mechanisms include Priority-based scheduling, Time Aware Shaper (TAS), Credit based Shaper (CBS) and Frame Preemption. An egress port of TSN switch is depicted in Figure 2.2 and its operation mode is explained in the section.

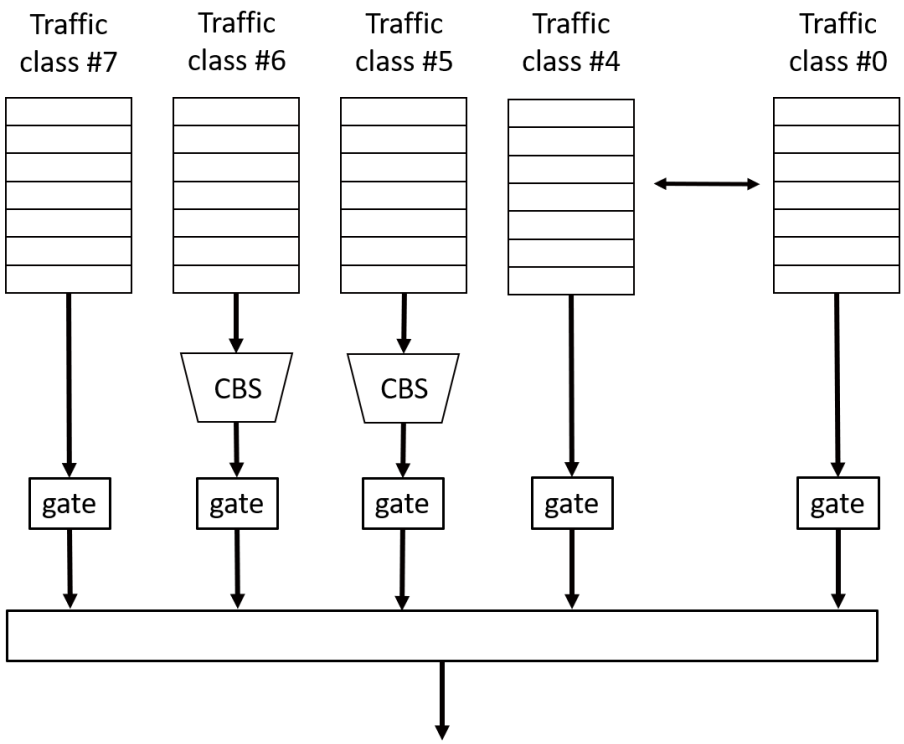


Figure 2.2: Example of egress port in TSN switch inspired by [7]. Traffic flows are grouped into 8 classes at most, each class associated with a waiting queue. The CBS scheduling mechanism is applied to one or several classes. With TAS, gates are implemented on all queues, and the time-triggered schedule to open and close the gates is saved in a ”gate control list”.

Priority-based scheduling

Priority assignment is the simplest method to guarantee delay bounds of traffic in TSN networks. Flows are grouped into traffic classes [22], each class has a priority level as illustrated in Figure 2.2. In other words, a traffic class is a set of traffic flows with the same priority. For a TSN-compliant switch, each egress port has up to 8 queues associated with 8 priority levels, the lowest priority is 0 and the highest is 7. When a frame of a flow arrives at an egress port, it is assigned to the corresponding queue based on its priority. Lower priority traffic must wait until all frames in the higher priority queues are transmitted. The allocations of priorities to flows can be performed either manually or using algorithms such as the Optimal Priority Assignment (OPA) [23]. With manual priority assignment, network designers typically follow heuristics or group according to the functional domain of the flows. For instance, traffic with strong timing constraints, like control data, should be assigned higher priorities than infotainment traffic. In addition, the OPA algorithm guarantees to find the optimal solution [3]. The challenge when applying OPA is the complexity of the algorithm, that it requires the execution of many feasibility tests to determine the priority assignment for a configuration and thus further delays the DSE of TSN networks. Precisely, with n streams, OPA requires at most $n(n+1)/2$ feasibility tests to derive the optimal priority allocation [23].

Credit Based Shaper

Traffic with strict timing constraints should be clustered into classes with higher priorities than traffic that does not affect the safety of the system such as infotainment streams. However, such low priority traffic may also have throughput requirements or soft deadline constraints, *i.e.* missing deadlines should not occur more often than a certain threshold. By definition, low priority traffic is blocked until all frames in the higher priority queues are transmitted. As a result, performance requirements of low priority traffic may not be satisfied, in particular when higher priority flows are bursty video streams. To alleviate this

[3] The study in [24] shows that OPA is optimal for schedulability analysis of periodic and sporadic traffic on AFDX networks. However, we note that OPA is guaranteed optimal if all required assumptions are satisfied. In our thesis, the NC-based WCTT analysis implements the "traffic grouping" optimization [25] that leads the analysis to occasionally violate one of OPA's assumptions. Although OPA is not optimal in our case, we discover in preliminary experiments that it is still very efficient to derive feasible priority allocations.

problem, IEEE 802.1Qav [26] defines the Credit Based Shaper (CBS) that prevents higher priority traffic from monopolizing a link for too long. A transmission credit is assigned to a traffic class applying CBS. When a packet from this class arrives in the queue dedicated to the class, it is transmitted if and only if the credit is positive. The credit reduces during the transmission at a rate called ”send slope”, and frames blocked once the credit becomes negative. The credit is replenished at a rate called ”idle slope” when frames from other classes are transmitted. The CBS is an effective mechanism to allow low priority traffic to get access to the network, and avoid ”starvation”, in particular when CBS is applied to bursty traffic.

Pre-shaping [27] is a simple software-implemented traffic shaper, sometimes considered use as an alternative to CBS, that is specific for bursty traffic. Pre-shaping simply inserts idle times between segmented frames. This strategy allows frames from other traffic flows to be transmitted between the idle time, which not only increases the number of flows that satisfy deadline constraints but also potentially reduces the memory usage at queues of an egress port (see [12]). The main drawback of Pre-shaping is that it creates an overhead, that may not be acceptable beyond a certain number of streams.

Time Aware Shaper

Besides the problem tackled by CBS, another limitation of priorities is that adding new traffic to the network, or changes done to some traffic can cause further delays to an existing flow. It is a severe problem when the traffic has strict deadline constraints. One way to solve this problem is to reserved time slots for the transmission of packets belonging to a certain traffic class. In TSN, the traffic benefiting from these reserved slots is called the *Scheduled Traffic* (ST). IEEE 802.1Qbv [28] defines the Time Aware Shaper (TAS) to support ST. The implementation of TAS on an egress port includes logical gates at each queue, as illustrated in Figure 2.2. Frames in any queues are transmitted only when the gate is open. A Gate Control List (GCL) is used to store the opening and closing schedule of gates. Although TAS is a simple mechanism, constructing the time-triggered schedule is a difficult problem known to be NP-complete.

Related works trace back to the early works on Time-triggered communication networks [29], that propose to divide time into time-slots and assign a slot to each flow.

Later, Satisfiability Modulo Theories (SMT) solver is used to find a solution based on the timing constraints of the traffic [30]. In TSN networks, SMT solver and Optimization Modulo Theories (OMT) are also applied to find the time-triggered schedule to satisfy deadline constraints [31]. Scheduled Traffic provides many benefits, including reduced jitters, and less interferences between classes and it eases composability. However the time slots are allocated on a per traffic class basis, and not a per-flow basis. As a results, interferences between frames of the same traffic class do occur, which results in some jitters and increased communication delays.

Frame Preemption

Exclusive transmission time slots can potentially waste bandwidth if they are not used, *i.e.* if the associated queues are empty when the gates are open. In addition, it requires a ”guard band” to be added to each time slot to guarantee that the transmission of a frame finishes before the closing of the gate. The waste of bandwidth utilization, up to 1522 bytes which is the maximum frame size, can be reduced by removing the guard band. This problem is addressed by IEEE 802.1Qbu [32], which introduces Frame Preemption to be used in combination with ST or without, sometimes as an alternative to ST. The preemption mechanism allows frames from an ”express” traffic class to interrupt, a.k.a preempt, the transmission of frames from a ”preemptible” class. We note that the preemption adds a limited overhead of 24 bytes per preemption (see [33]). By interrupting the transmission of lower priority packets, this mechanism reduces delays of higher priority flows caused by the interference from lower priority frames at each hop in the network. [33] shows that preemption reduces this interference by a factor of 10.

2.3 Artificial Intelligence and Machine Learning

This section is divided into several parts. The first part is the definition of Artificial Intelligence (AI). In the second part, we introduce machine learning (ML) algorithms, which are the means to achieve AI in practice today, then followed by the classification of different machine learning algorithms. The later parts of this section are dedicated to the applications of ML in networking.

2.3.1 Introduction to Artificial Intelligence

Artificial intelligence, or AI, is a field in science and engineering that attempts ”not just to understand but also to build intelligent entities” [34]. The ”intelligent entities” are considered as agents [4] that mimic the intelligence of humans, however, this thesis expands the definition of AI to be any form of biological intelligence. Accordingly, this thesis considers AI as the ability of machines to execute tasks that typically require biological intelligence. Since there is no formal definition of intelligence that is accepted by all researchers at this moment, in this thesis, we accept that *biological intelligence* is the ability of biological brains to perform tasks that seem smart. This thesis avoids arguments on how smart the tasks should be to be considered as intelligent and only concerns about how brains become smart. To the best of our knowledge, the study of Friston et al. provides the most logical theory of the function of biological brains [35]. According to this work, brains continuously learn to predict the future in order to optimize actions when performing tasks, thus achieving a higher chance of surviving. The prediction ability is gained through the recognition of patterns, *i.e.* a better prediction ability means recognizing more patterns. *AI techniques* are techniques that help machines to predict the future with a certain degree of accuracy (see *On Intelligence* [36]). [5]

AI study has a rich history with multiple research directions. The ultimate goal of AI research can be considered to create a general form of AI, aka Artificial General Intelligence (AGI). This is an AI agent that can solve a wide range of problems, just like humans. According to [37] and [38], there are multiple approaches to reach AGI. However, we note that the current status of AI research is still limited, as most AI techniques can only solve a handful of tasks with similar scope. There is a broad range of AI techniques but for specific tasks only, which is called narrow AI. In this thesis, we only focus on machine learning algorithms, the dominant AI approach in recent years, as they have proven to be effective in many practical applications.

[4] An agent is ”just something that acts”, or in more details, an agent is ”anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators” [34]. In our context, agents are implemented in software.

[5] Another popular definition of AI is through the Turing Test [34]: a computer ”acting humanly” so that an observer cannot distinguish the computer from a human. The definition of AI in this thesis has already covered this definition since prediction ability is the key factor to plan and optimize future actions [35], in this case, to deceive the observer.

2.3.2 Machine learning

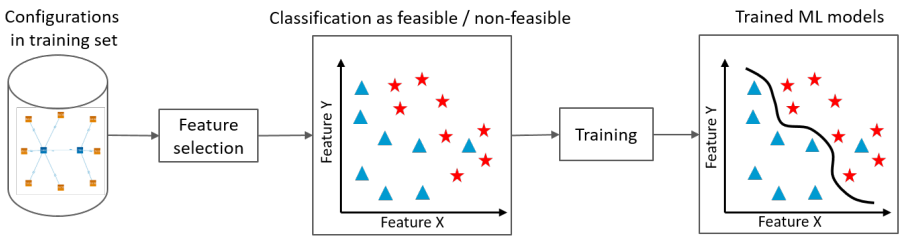


Figure 2.3: A model for feasibility prediction of TSN configurations that includes three steps from left to right: the generation of training samples of the training set (e.g., TSN configurations with known feasibility in our context), the feature engineering step that consists in the selection or the synthesis of features from the training samples, and the training step that use ML algorithms to learn patterns that can classify, *i.e.* draw boundary, between classes in the training samples based on their features. This figure is re-used in Chapter 3.

Definition

”Learning from examples”, or *machine learning* (ML), is a branch of AI, which defines that ”an agent is learning if it improves its performance on future tasks after making observations about the world” [34]. This implies the improvement on the prediction ability of the agent (see the definition of AI above). Accordingly, an *ML algorithm* is an algorithm that learns from observed samples to predict outputs from inputs [39], and an *ML model*, illustrated in Figure 2.3, is ”an expression of an ML algorithm” [40]. In this thesis, a ML model can also be a system made up of multiple ML algorithms.

Generally the key components in a ML model includes: a *training set*, a possible *validation set*, and one or several *testing sets*. The training set contains many training *samples*, a.k.a. training data. For instance, the training set of an ML model for object classification contains images of target objects. In our context, the training set contains TSN configurations. The training samples may have *labels* to indicate their true outcomes. In this thesis, the labels of training TSN configurations is their feasibility: feasible or non-feasible. The validation set contains samples with the same distribution as the training set, and the testing sets contain unseen samples. The roles of data sets are explained in more details in the following sections.

Categories of machine learning algorithms

ML algorithms are categorized as supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, training samples need to be labeled and the supervised ML model is trained to predict labels. Most practical problems fall into this category: training network traffic data is labeled as benign or abnormal in network intrusion detection problems, houses with the selling price, and so on. A supervised learning model can solve the tasks of classification with categorized labels, or regression with labels that have continuous values. On the other hand, data in unsupervised learning is not labeled. Examples of unsupervised ML are clustering algorithms [34] that try to group data into clusters with similar characteristics. However, some unsupervised ML models still need testing sets to evaluate their performance. For instance, autoencoder [41] is an unsupervised ML model that tries to reconstruct the training samples, its testing set thus includes unseen data in order to evaluate the reconstruction error. Finally, reinforcement learning helps an agent to reach a goal by choosing appropriate actions and maximizing long-term rewards, which implies the ability to predict the value of their actions. However, the separation between the three categories is not always clear-cut. In AlphaGo, an AI agent that surpasses humans in the game of Go [42], convolutional networks (a popular supervised learning model for computer vision) is also used in a reinforcement learning model.

ML models can also be grouped into conventional ML models and Deep Learning (DL) models. The key factor to separate the two groups is how the samples are used. Conventional ML algorithms take as inputs the *features*, *i.e.* the characteristics of training samples used for the purpose of prediction, that are chosen by the developer of the model. On the other hand, DL models can take as input the raw data of training samples to generate synthetic features. The first group includes simple ML algorithms that are typically based on statistical inference, supported for instance by the popular ML library Scikit-learn [43], while DL is a set of ML paradigms using artificial neural networks [41]. The advantage of conventional ML algorithms is the simplicity and therefore usually the smaller execution time, however, they lack the ability to model complex systems. On the other hand, DL can exploit more patterns due to its huge computational capability, while the manual feature selection step is removed. The drawback of DL is that it requires more training samples to

reach its full potential and more advanced ML knowledge during the model development.

We note that our problem is to classify TSN configurations as feasible or non-feasible, thus the studies in this thesis only focus on the supervised ML model. The development process of a supervised ML model introduced in the following is applied to both conventional ML models and DL models, except the feature engineering step as DL models can automatically generate features from raw data of the training samples.

Development of a supervised learning model

The development of a supervised ML model usually follows these steps:

1. *Data collection*: In this step, we collect training samples that are relevant to the task of the ML model, *i.e.* from which the algorithm can learn. They can be either collected and selected from existing sources, or generated. In our studies, the task of the ML model is to identify whether unseen TSN configurations are feasible or non-feasible, thus the training samples are artificial TSN configurations that have been labeled by schedulability analysis.

2. *Feature engineering*: With conventional ML models, in this step, we need to select features manually using domain expertise. Although this step is eliminated with DL models, we note that training samples can be of various types, e.g., TSN configurations including topology and routing, while neural networks used for prediction require inputs to be in the forms of vectors or matrices. DL models thus need to be constructed so that they can take raw data of training samples and generate synthetic features in these forms (vectors or matrices).

3. *Training*: In this step, the ML model learns patterns from the training samples to gain the prediction ability, e.g., to classify feasible and non-feasible configurations in our studies. DL models also learn to synthesize useful features in this step.

4. *Validation*: During the training step, we can validate the prediction ability of the ML model with a validation set (see § 2.3.5 for performance evaluation metrics). Although this step is optional, it is important to know whether we should continue the training, which may take a long time for complex ML models like deep learning.

5. *Testing*: the trained ML model is evaluated on unseen data in the testing set to determine its performance. The testing sets need to be specified at the beginning of the development process in order to explicitly define the task of the ML model.

A key element in the development of an ML model is a training set including training samples with similar characteristics as the unseen data that the trained ML model will be applied to. This usually implies extensive trial and effort, and domain expertise, to generate an appropriate training set. Furthermore, a wide range of testing data is crucial to avoid the situation where the ML model returns wrong predictions without the user being aware. However, we note that supervised ML models cannot in general reach the perfect prediction. The reason is that patterns can be considered as results of unseen generators. ML models can recognize patterns by learning how to approximate these generators. In any real-world task, the number of patterns and therefore unseen generators could be infinite. A well-performing ML model can recognize some of the most important patterns, however, it is not possible to recognize all. This also explains why the deep learning model usually performs better than conventional ML algorithms, as it has more computational capability to approximate more generators, *i.e.* it learns more patterns.

2.3.3 Graph Neural Network

General graph neural network model

According to [44], there are four fundamental deep learning paradigms: 1) fully-connected neural network that require independent inputs, 2) convolutional neural network that exploits spatial relations of inputs in images, 3) recurrent neural network for temporal data like time series, and 4) graph neural network (GNN) for graph-based data. Among these paradigms, GNN is the most appropriate model for our problem since computer networks can be encoded as graphs with minimal effort.

The following introduces the general GNN model inspired by [44], while more details are provided in Chapter 4. Without loss of generality, the structure of a GNN model is illustrated in Figure 2.4 with the application on TSN networks. Given a configuration as raw data, the GNN model first encodes the configuration as a graph. The physical and logical components in the network: flows, links, and queues at egress ports of switches are encoded

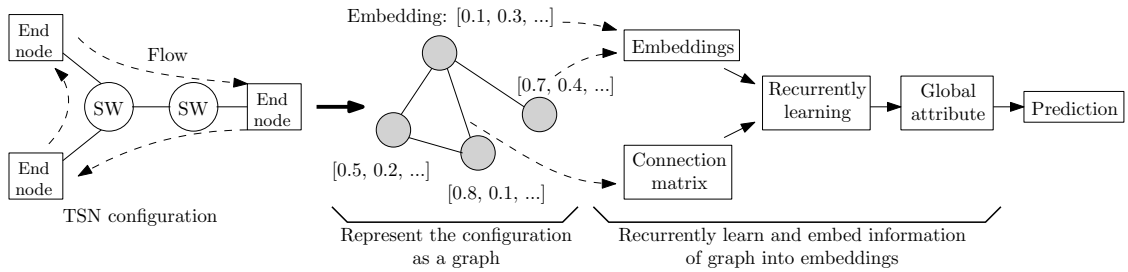


Figure 2.4: The structure of a GNN model applied to TSN network engineering. The GNN model first encodes a configuration as a graph, while components in the network are corresponding nodes in the graph. Topology and routing in the network are represented by edges between nodes. Each node and possibly each edge has an embedding, which is recurrently updated by artificial neural networks that aggregate embeddings of its neighbor nodes. The global attribute vector is aggregated from all embeddings and thus contains synthetic features of the configuration, which is used for the prediction step. This figure is reused in Chapter 4 with more details provided.

as corresponding nodes in the graph. Topology and routing, which connect components in the network, are represented by edges between nodes. Each node and possibly each edge in the graph has an initial representative vector, *i.e.* an *embedding* including characteristics of the corresponding component. For instance, the initial embedding for a node representing a flow includes the period, deadline, etc. of the flow. Accordingly, the encoding step converts a TSN configuration in raw data into embeddings, and into the connection matrix that will be used by ML algorithms.

In the next step, each embedding aggregates information from its neighbors in the graph as well as its current information and applies one or several artificial neural networks to get the updated embedding, *i.e.* recurrent learning. As a result, the GNN model can exploit complex relationships among components in the network, e.g., flows in the network compete for bandwidth at egress ports, and this information is captured by the embeddings. A global attribute vector is aggregated from all embeddings to represent the whole graph. These attributes, a.k.a. features, are synthetic features of the corresponding TSN configuration. The rest of the GNN model is an ML algorithm that uses the global attribute vector to predict the feasibility of the configuration. Although the GNN model in our studies only uses the global attribute to predict the feasibility of TSN configurations, in general, we can also predict the values of nodes and edges in the graph through their embeddings (see examples in [44]). In summary, the most important contribution of the GNN model is to automatically extract features from raw data in graph domains without

the need for domain expertise.

A survey of graph neural networks

One of the earliest efforts to provide a complete deep learning model that can process the data in graph-like domains (computer network, molecular chemistry, cells in biology, and so on) is [45]. In this study, the authors propose a GNN model that encodes the raw data as graphs, then uses a neural networks to exploit underlying relationships among nodes in the graph data. In other words, the GNN model maps the graph-based data to euclidian data required by ML algorithms with minimal loss of information. We note that the study was published in 2008 when deep learning frameworks such as TensorFlow [46] did not exist yet. The authors provided thus a learning algorithm based on backpropagation for the proposed GNN model. Each node in the graph has a representative vector, which is updated by aggregating information from neighbor nodes. This learning method allows information to propagate through the whole graph. In [47], the authors present a more general GNN model to tackle non-euclidean data, in particular graphs.

With the success of the convolutional network in computer vision [48], researchers attempted to extend this deep learning model to graph domains. The graph convolutional network (GCN) introduced in [49] is a typical study in this research direction. In [50], the authors unify the learning algorithms of GNNs in [45,49] and other previous GNN models to formalize the *message passing* algorithm. In this learning algorithm, the representative vector of each node in the graph is updated by aggregating information in messages passing between neighbouring nodes. Recently, attention mechanism [51] has become one of the most effective learning methods in deep learning and has been successfully applied to many domains like computer vision and natural language processing. In [52], Wang et al. introduce a novel GNN model with the self-attention style. The authors in [44] provide an excellent survey of the previous GNN studies and introduce a GNN model that can learn the characteristics of nodes in the graph, edges between nodes, and the global attributes of the graph. This general GNN model applies the message passing learning algorithm and provides options for attention methods as well. To the best of our knowledge, this GNN model is the most comprehensive deep learning model in graph domains. In Chapter 4, we adopt this GNN model without the attention mechanisms and the GNN model specified

for TSN networks is described in more details. In the following, we provide a review of the applications of ML, particularly GNN, in the field of networking, then justify the use of ML to the DSE of Ethernet TSN.

2.3.4 Machine learning in networking

Nowadays, ML is one of the most active research areas with many successful applications in diverse domains. The most well-known applications of ML are in computer vision such as object recognition and segmentation, as well as natural language processing. Researchers have also discovered the usefulness of ML in bioinformatics, autonomous vehicles, and software engineering [53, 54]. With the emergence of deep learning, the application of ML has expanded to the fields that seemed hard to automate. For instance, DL has surpassed human level at the ancient game of Go, and it can predict protein structures with high accuracy [55].

ML is also considered one of the most powerful methods to tackle problems in networking [56, 57]. Many studies focus on the applications of GNN to build network models that can predict network performance like delay or packet loss. We note that while computer networks are graph-based data, most ML prediction algorithms require the inputs to be in tensor-like data types. Therefore, GNN is applied to encode computer networks and return sets of vectors representing the networks. RouteNet [14, 58, 59] is a GNN model that solves this task effectively, particularly in modeling SDN networks. In the RouteNet model, multicast flows are encoded by routing paths, a strategy which has been re-used by many applications of GNN since then. The study in [60] improves the RouteNet model, while the impressive study in [61] is a step toward enhancing the generalized ability of GNN in network modelling. Although GNN facilitates the application of ML to networking, it is worth noting that the development of a GNN model requires advanced knowledge of deep learning, which might be the main obstacle hindering researchers in networking. In [62], the authors proposes a solution to address this issue by introducing IGNNITION, a comprehensive library for fast prototyping of GNNs targeting communication networks.

GNN is also used to address other tasks in networking such as network performance evaluation [15, 63]. The study in [15] applies GNN to predict the throughput of TCP flows, while [63] is its extended version with more details. On the other hand, authors

in [64] apply GNN to Multiprotocol Label Switching (MPLS [65]), in order to speed up the verification of MPLS configurations. Deep learning and GNN are also widely used by some research groups for routing optimization and traffic engineering [66,67]. In [14,59], the authors show the potential of the RouteNet model to optimize routing in SDN. In the studies, multiple routing schemes are generated then GNN is applied to predict the scheme with the best performance. We note that tasks requiring multiple steps to derive the optimal solution, like routing optimization, can also be tackled with reinforcement learning [68], a category of AI that learns to give actions to maximize the cumulated reward. For instance, deep reinforcement learning is used to optimize routing in optical transport networks [69], while reinforcement learning with evolutionary algorithms [70], another AI category, is adopted in [71] to provide better task assignment solutions for fog computing. In [72,73], the authors further automate the routing optimization process with RouteNet, by applying deep reinforcement learning relying on GNN to learn how to choose the most appropriate routing schemes. A similar approach with deep reinforcement learning and GNN for routing optimization is used in [74]. We note that the studies mentioned above limit the scope of reinforcement learning to a control at the level of the network, which possibly leaves room for performance improvement if we could apply reinforcement learning to each devices in the network. In [75], the authors explore the possibility of this research direction by applying multi-agent reinforcement learning to traffic engineering. In this model, each switch in a network is considered as an agent that learns to choose appropriate actions so that it can cooperate with other switches. The objective of this cooperation is to maximize the performance of the whole network.

Graph-based neural networks are also applied to real-time communications [76–79]. In real-time network analysis, Network Calculus (NC [80]) is commonly used to derive deterministic delay bounds for traffic flows, a.k.a worst-case traversal time (WCTT), by searching for the best model of resource contention among these flows. The challenge is that NC needs to search for the best contention model with each tandem of n servers, *i.e.* the queues or packets schedulers on the path of a flow: there are up to $2^{(n-1)}$ possible contention models for each tandem. To facilitate this resource intensive task, DeepTMA in [76] uses GNN to encode the network and learn to provide a good contention model, which leads to a tighter delay bound and shorter execution time. In [79], the authors further

speed up the GNN model in [76] and provide even tighter delay bounds. A remarkable aspect of these applications is that the use of GNN scales to large networks with thousands of flows. We note that the existing applications of GNN to real-time communications only tackle problems for networks using FIFO scheduling policy. On the other hand, our preliminary study [1] unveils the potential of ML algorithms to predict feasibility of TSN networks with other scheduling policies. In this study [1], both supervised and unsupervised learning techniques are used to give more options for designers to satisfy e.g. multi-objective optimization in DSE. Similar methods are applied to a more complex TSN network and achieve impressive performance, which are discussed in Chapter 3 in this thesis.

2.3.5 Performance criteria

In our works, an ML model is said to correctly predict the feasibility of a TSN configuration if this result is the same as the result derived by schedulability analysis. The performance of an ML model is generally evaluated by the *prediction accuracy*. Other metrics are also used to give more details for designers to decide whether the ML model is effective. The followings are definitions of common metrics used in the thesis. Other metrics may also be needed and will be defined along the individual chapters.

- The prediction accuracy, aka overall accuracy (*acc*): the proportion of TSN configurations correctly predicted among a set of TSN configurations being tested. The prediction accuracy is the primary criterion to evaluate the effectiveness of an ML model in our studies.

- The True Positive Rate (*TPR*): the proportion of TSN configurations being correctly predicted feasible over the number of feasible configurations. The meaning of TPR is to measure how effective an ML model is to discover feasible configurations.

- The True Negative Rate (*TNR*): the proportion of TSN configurations being correctly predicted non-feasible over actually non-feasible configurations. The role of TNR is similar to TPR, but its measures the ability of the ML model to discover non-feasible configurations.

- The False Positive Rate (*FP*) a false positive is a non-feasible TSN configuration being wrongly predicted as feasible. FPR is calculated as the ratio of false positives over non-feasible cases. A high FPR can lead to wrong design choices since these TSN configurations seem appropriate while they should actually be discarded.

- The False Negative Rate (*FNR*): a false negative is a feasible TSN configuration that is wrongly predicted as non-feasible. FNR is the proportion of false negatives over feasible cases. False negatives lead to the discard of useful configurations, which is an important factor that needs to be controlled during the development of an ML model.

2.4 Discussion

In this chapter, we have provided the necessary background to comprehend the studies in the thesis. However, many details need to be clarified in the next chapters, starting with the feature engineering step. In the case of conventional ML algorithms, features that are relevant to the feasibility need to be chosen manually, therefore domain expertise in realistic real-time systems is crucial to choosing the best features. On the other hand, knowledge about advanced ML techniques is essential to build deep learning models such as GNN that can synthesize useful features. Second, domain expertise is also important for the generation of the training set used to train an ML model. The reason is that the synthetic training data has to represent the distributions of unseen TSN networks. Finally, a trained ML model has to undergo extensive evaluation before the deployment in the industry, otherwise, it may perform well on some test cases but poorly on others and cause silent failures, *i.e.* wrong predictions without the users possibly knowing about it. The heterogeneity of test cases during the evaluation is a crucial factor to detect this weakness and guarantee the quality of ML predictions.

Chapter 3

On the Use of Supervised Machine Learning to TSN Networks

Abstract

In this chapter, we explore the possibility of using Machine Learning (ML) as a viable verification method to determine the feasibility of TSN configurations, thanks to Machine Learning's ability to learn patterns in topology and traffic characteristics that make systems feasible. This chapter applies k-Nearest Neighbors (k-NN), a simple and standard supervised learning algorithm, to compare its performance in terms of prediction accuracy and execution time against two Network Calculus-based schedulability analyses: precise and approximate analysis. Four TSN scheduling solutions based on priority levels and one of them combined with traffic shaping are considered in the experiments. Results obtained on an automotive network suggest that k-NN can predict the feasibility of TSN configurations with high accuracy, up to 95.9%. k-NN also achieve a speedup factor of 190 over the schedulability analysis over 10^6 realistic TSN networks. However, ML produces false positives, *i.e.* non-feasible configurations wrongly predicted as feasible, which is never the case for schedulability analysis. In the second part of the chapter, we aim to control the false positive rate and therefore the uncertainty of the ML models with hybrid techniques. The proposed hybrid model combines k-NN and precise analysis, the latter being executed when the uncertainty of ML prediction is higher than a threshold. In the case of priorities with 8 classes, the hybrid model produces 46% fewer false positives than

solely using k-NN. By controlling the threshold to achieve a 99% prediction accuracy, the hybrid model still has a speedup factor of 5.7 compared to schedulability analysis. Indeed, ML provides new trade-offs between prediction accuracy and running time that are particularly useful in the context of design-space exploration, while the insights gained from this chapter are an asset when considering the massive adoption of ML to the industry.

The studies in this chapter, come from [1–3], answer research questions **RQ1** and **RQ2** in Section 1.2:

RQ1: We first ask whether ML algorithms can be used to predict the feasibility of TSN networks in an efficient manner in terms of time and accuracy. In order to answer this research question, we use traditional ML algorithms such as k-Nearest Neighbors (k-NN), while domain expertise is leveraged during the processing of inputs for ML.

RQ2: We raise the question of how to control the risk of false predictions when applying ML algorithms. Since the outcomes of an ML model is not perfect, should we combine ML with reliable schedulability feasibility tests?

3.1 Introduction

The development of new cars in the automotive domain, and new electronic systems in general, relies on a embedded communication architecture made up of several networks. The communication architecture has to be able to support the communication needs of many products during the lifetime of the corresponding platform. Each product has multiple different options, which leads to a large number of possible combinations, *i.e.* possible configurations.

Design-space exploration (DSE) is the process in which the designers systematically explore the possibilities to conceive networks that satisfy the desired requirements. There are usually millions of candidate solutions, which exceeds human ability to verify all of them. In addition to the vast number of configurations, TSN mechanisms introduce a new level of difficulty in DSE. On the one hand, the new technical possibilities increase the probability that a TSN network configurations meet the requirements. On the other hand,

it largely increases the size of the configuration space, as well as the complexity of the problem due to the interaction between protocols.

Typically, there are three steps during DSE: creating candidate networks, configuring these networks, and evaluating the performance of the configurations. The last step executes schedulability analysis to derive the feasibility of the configurations. This last step is very compute-intensive, thus limits the possible search space and leads to non-optimal solutions. For instance, with a TSN configuration containing 350 flows and using priority scheduling with three priority levels, schedulability analysis takes 805ms on average [1]. That results in about 224 hours to verify a set of 10^6 configurations. A pioneer DSE assistant algorithm is ZeroConfig-TSN [81,82]. It is available in the RTaW-Pegase TSN toolset [18] that helps the designers to select and configure TSN protocols. However, the execution time of the ZeroConfig algorithm integrates the running times of schedulability analyses for all explored protocols in the worst case, which may not be practical if the configuration set is too large. Before this study, there was no quick method to identify which TSN scheduling solutions guarantee that all flows in the network meet their deadline constraints.

This chapter is made up of two parts. In the first part, we explore the possibility of using ML as a faster verification method in DSE. The experiments are set up to answer two research questions:

1. Can ML be used as a verification tool to identify the feasibility of TSN configurations? In other words, we pose the question of whether ML can provide a viable alternative to schedulability analyses in terms of accuracy and running times.

2. Do ML algorithms have robust performances on different TSN scheduling solutions? For instance, although schedulability analysis accurately derives the feasibility of TSN networks applying priority scheduling or traffic shaping, the running times are not equal by far. Ideally, ML algorithms would have consistent prediction accuracy and running times with all scheduling solutions.

The second part of the chapter is a follow-up work of both the study in [1] and the first part of the chapter. The experimental results show that the wrong prediction percentage of ML techniques, up to 7% for supervised learning and 11% for unsupervised learning

(in [1]) is a hindrance to their application. We improve upon these studies by proposing a hybrid method combining both ML and schedulability analysis. Importantly, this approach offers user-adjustable trade-offs between accuracy and computation time. In addition, the baseline hybrid method is extended with the use of a fast but approximate schedulability analysis before possibly conducting the precise schedulability analysis. This fast analysis serves to identify feasible network configurations in a compute-efficient manner.

This study is limited to the scope that network topology has been set before all communication needs are known, which is often the case in today's real-time network design. Realistic traffic patterns are also defined before the generation of the training set. In this chapter, we intentionally use k-Nearest Neighbor (k-NN), a standard and very simple supervised ML algorithm. The ML model can be implemented with existing libraries, and the training set size is relatively small. To the best of our knowledge, this study is the first application of ML to predict the feasibility of TSN networks. Data and R code used in the experiments of this study are available at `https://github.com/crtes-group-unilux/ML4TSN-Schedulability` as open-source (AGPL V3.0).

The rest of this chapter is organized as follows. Section 3.2 introduces the realistic network topology and traffic statistics that are used to generate TSN configurations. In Section 3.3, we introduce the feature engineering process and k-Nearest Neighbors (k-NN), a supervised ML algorithm. The experimental results of predicting the feasibility of TSN configurations with k-NN are presented in Section 3.4 in a comparison with conventional schedulability analysis. The second part of the chapter includes the next three sections: Section 3.6 analyzes why ML algorithms have wrong predictions in some cases, Section 3.7 proposes the hybrid model combining ML prediction and precise analysis, while Section 3.8 is an extension of the baseline hybrid model with approximate analysis. Section 5.7 is a survey of ML applications in related domains, in particular networking. In Section 5.8, we present the insights gained from the experiments and discuss possible research directions to further improve the performance of ML in feasibility prediction.

3.2 Ethernet TSN model and design problem

In this section, we consider the design of TSN networks involving four scheduling solutions, which provide different Quality of Service. Network topology and supported TSN protocols have been set before the design-space exploration. This is realistic in industrial contexts such as automotive domains: the topology and technologies are chosen in the early phase of the design cycle when they do not entirely know the communication needs. Indeed, some services only become available late in the development cycle, or they are added over the lifetime of the system.

3.2.1 Designing and configuring TSN networks

In this chapter, we apply four scheduling solutions to improve the QoS of TSN networks. They provide different trade-offs between complexity and improvements in terms of increasing in the number of feasible configurations.

1. FIFO scheduling (*FIFO*): There is a single traffic class. All flows are thus in the same traffic class and thus have the same priority level. It is considered the simplest scheduling solution.

2. Priority with manual classification (*Manual*): Flows belong to one of the three traffic classes: command and control (C&C), audio, video. The priority levels of these classes follow the order of C&C $>$ audio $>$ video. This scheduling solution is typically a baseline solution for designers, as the priorities tend to reflect well the relative criticality of the different flows.

3. Concise Priorities with 8 priority levels (*CP8*): Concise Priorities (CP) is an algorithm for priority assignment available in RTaW-Pegase. It relies on the same principles as the Optimal Priority Assignment (OPA) algorithm [83] that has been proven optimal for traffic flows in switched Ethernet network [24]. The main difference between CP and OPA is how to handle non-feasible configurations: CP tries to minimize the number of flows that do not meet deadline constraints, which is not a concern for standard OPA. When applying CP8, traffic classes can contain flows of different functional types, and up to 8 priorities levels can be used if needed.

4. Manual classification with Preshaping (*Preshaping*): Manual priority allocation with three traffic classes is used. However, in addition we apply Preshaping, a software-implemented traffic shaping technique, to all video flows. Images from cameras are segmented into several packets, and Preshaping inserts idle times between consecutive frames that are waiting in the transmission queues. The other characteristics of the flows remain unchanged. The Preshaping technique is available as *Presh* algorithm in RTaW-Pegase [27].

These four scheduling solutions provide two principal QoS strategies: static priority scheduling and, for Preshaping, traffic shaping. They can be implemented with COTS TSN network devices that all support 8 priority levels. Importantly, these scheduling solutions are compatible with standard schedulability analyses, *i.e.* one can derive the feasibility of TSN networks that apply these scheduling solutions without false positives.

3.2.2 Traffic characteristics and network topology

For the experiments, the traffic in a TSN network contains three classes: critical, audio, and video flows. Their characteristics and their proportion are provided in Table 4.1, which is inspired by a case study provided by an automotive OEM [7, 84].

The number of flows in the experiments can vary but the proportion of flow types, as indicated in Table 4.1, is unchanged in the flow generation process. The source and destination of flows are randomly set. Each flow is generated to be unicast or multicast with a probability of 0.5. For each multicast flow, the number of receivers is randomly chosen from two to five. Each flow has a deadline constraint and its WCTT must not excess the deadline. The WCTT is computed with a schedulability analysis in Network-Calculus available in RTaW-Pegase.

The topology used in this study is provided by an OEM in the automotive industry [7]. The network is shown in Figure 3.7. It has 5 switches and 16 end nodes. The data transmission rate between ECU12 and Switch3 is 1Gbps. The other links in the network have a transmission rate 100Mbps.

Table 3.1: The traffic in the network contains three classes. Each flow has a deadline constraint and its WCTT must satisfy the constraint. The frame size means the payload size of a packet.

Audio Streams	• 128 or 256 byte frames • periods: one frame each 1.25ms • deadline constraints either 5 or 10ms • proportion: 7/46
Video Streams	• ADAS + Vision streams • 30*1500byte frame each 33ms (30FPS camera for vision) • 15*1000byte frame each 33ms (30FPS camera for ADAS) • 10ms (ADAS) or 30ms (Vision) deadlines • proportion: 7/46
Command & Control	• from 53 to 300 byte frames • periods from 5 to 80ms • deadlines equal to periods • proportion: 32/46

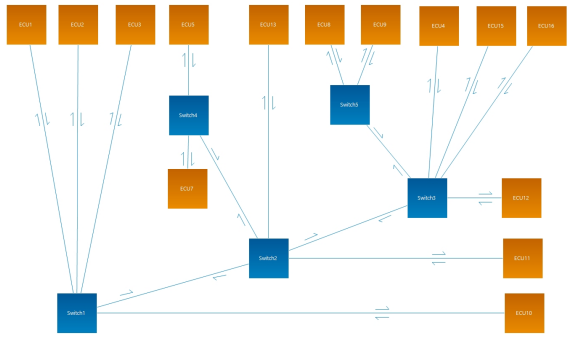


Figure 3.1: Network topology used in the experiments (topology from [7]).

3.3 Predicting feasibility with supervised learning

In this section, we apply supervised ML to network classification, highlighting that feature engineering is a crucial step to choose input features for an ML algorithm. k-Nearest Neighbors (k-NN), a supervised ML algorithm, is described in the second part of the section. k-NN is described in ML textbooks like [85] as a simple but powerful algorithm since it only bases on majority voting: the class of a new data point is identified by the majority of its nearest neighbors in the feature space.

3.3.1 Supervised learning applied to network classification

Based on the dependence on labels for training data, ML algorithms are categorized as unsupervised and supervised learning. The ML algorithms in the first category do not need training data. In our experiments, they cannot directly perform classification with a good accuracy as detailed in our technical report [1]. Therefore, we chose supervised learning as the main approach for the feasibility prediction of TSN configurations. The supervised ML algorithms learn from labeled training samples to predict the potential label of unseen data. In our study, the labels of configurations, either feasible or non-feasible, are derived by schedulability analysis. A configuration is labeled feasible if all deadline constraints are met. The training set contains only labeled TSN configurations as illustrated in Figure 3.2.

A TSN configuration is abstracted by a set of features, aka a feature vector, that summarises the important characteristics of the configuration with respect to classification. Those features will be the inputs of the ML algorithm, and choosing the right features plays a crucial role for the algorithm to perform well on unseen data. In the experiments, we use precise analysis to derive the labels of the training samples, which reduces the false negatives compared to approximate analysis. It should be noted that different scheduling solutions lead to different labels for the same TSN network. As shown in the next sections, complex scheduling solutions such as Preshaping lead to more feasible configurations than simple solution like FIFO.

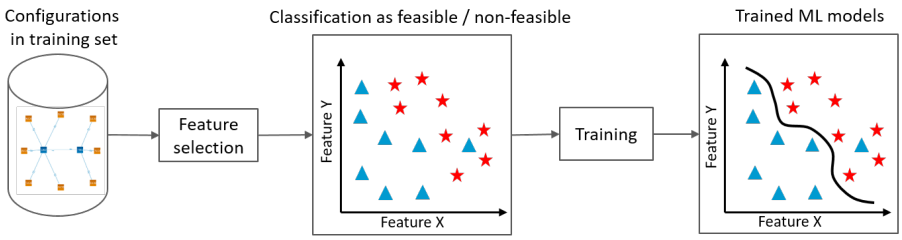


Figure 3.2: During the generation of the training set, TSN configurations are labeled as feasible or non-feasible, respectively represented by blue triangles and red stars. The labels are the feasibility derived by precise analysis. The ML algorithm learns from the values of features the patterns needed to predict the feasibility of configurations. A border between the feasible and non-feasible groups is virtually drawn by the ML algorithm to classify unseen configurations.

3.3.2 Feature engineering and feature selection

Feature engineering is a step to define features that will serve as inputs of an ML algorithm. A feature is a characteristic that captures important information for the ML algorithm to learn how to recognize meaningful patterns. Therefore, this step is crucial for the success of ML applications. In most systems, the number of characteristics and thus possible features is extremely large. We thus need to choose a limited number of features that can summarize the system efficiently for prediction. It should be noted that extraneous features with little predictive power can cause the peaking effect [86], which reduces the efficiency of the ML algorithm. Many studies have discussed feature selection [86,87], which usually relies on domain-specific knowledge possibly couples with algorithms to automate the selection process. Many ML packages offer such functions, for instance, scikit-feature [87]. Nonetheless, feature selection remains difficult when there is a large amount of data, like in genes and proteins in biology.

In the context of the TSN network, the raw data includes the number of flows for each type, the characteristics of flows, link speed and load on the links, etc. This is the information that can simply be observed or derived from the statistics of the network. We know some characteristics that strongly relate to the feasibility of the network. For instance, a bottleneck in the network tends to make it difficult to be feasible. This happens when the maximum load among the links is high or the load is unbalanced, *i.e.* the load on some links is very high compared to the load on other links. Accordingly, the maximum load and the unbalancedness of load among the links can be used as features to predict the

feasibility of a configuration. On the other hand, knowledge about the TSN network can also be applied to remove nonpredictive features.

Through iterative experiments, we have selected the five following features to characterize TSN networks:

- Number of critical flows.
- Number of audio flows.
- Number of video flows.
- Maximum load over all links in the network.
- Gini index: the unbalancedness of load among the links [88].

The number of flows in each class naturally affects the feasibility of a TSN configuration. Critical flows have short deadline constraints, if they have to compete with too many video flows then the configuration is likely to be non-feasible. The two last features, as described above, indicate whether there is one or some bottlenecks in the network. These features have the most predictive power among the known characteristics of a configuration.

3.3.3 The k-NN classification algorithm

Before applying the k-NN algorithm, we need to extract features from the training data. In our study, TSN configurations in the training set are represented by vectors of d features. The application of k-NN has two separate phases: the training phase and the testing phase. In the training phase, k-NN stores all given data in the feature space, *i.e.* the feature vectors representing training TSN configurations. The values of these features should be in the same range to guarantee the same contribution to the final result.

In the testing phase, an unlabeled configuration p with d features is selected and k-NN is asked for the prediction of its feasibility. The algorithm calculates the Euclidean distance from p to all stored data points (all samples in the training set) in the feature space: $dist(p,q) = \sqrt{(p_1 - q_1) + (p_2 - q_2) + ... + (p_d - q_d)}$. The algorithm uses these Euclidean distances to identify the k nearest neighbors of the new data p. The prediction of k-NN is the result of the consensus among the nearest neighbors. If the majority of the

nearest neighbors is feasible, k-NN predicts the unlabeled data feasible, otherwise, the algorithm predicts non-feasible. The optimal number of nearest neighbors k that leads to the best performance of k-NN is unknown. In §3.4.2, we determine this value through iterative search.

One of the advantages of k-NN is its simplicity, which allows its application on a large training set. Indeed, the algorithmic complexity of k-NN is low as it does not actually have a training phase. The k-NN model only stores the training samples in the feature space. The computation is postponed until testing with unseen data. Therefore, the complexity of training k-NN is $\mathcal{O}(1)$, and the complexity of testing with n stored data points is $\mathcal{O}(n)$. For comparison, the testing time of Decision Trees and Support Vector Machines [85], two popular classification algorithms, is a constant for all data but the training complexity in the worst case is polynomial with n samples in the training set. In summary, k-NN has the lowest training complexity among the ML algorithms.

The visualization of labeled TSN configurations (see Figure 4 in [1]) shows that feasible and non-feasible configurations tend to appear well clustered in distinct locations of the feature space. This suggests that if a new configuration is located in an area with a high density of feasible configurations, there is a high chance that it is feasible and vice versa. As k-NN is a majority voting-based algorithm, this ML algorithm is suitable for our feasibility prediction problem. On the other hand, there are some small areas in the feature space where non-feasible configurations are not well separated from feasible configurations. In these areas, k-NN may not produce the correct prediction. Other ML algorithms like SVM (Support Vector Machines, see [85]) that try to draw a clear boundary between two groups also suffer from this limitation. During the development of the ML model in this study, SVM has shown similar performance with k-NN on either correct or wrong predictions.

3.4 Experiments with k-NN

3.4.1 Performance criteria and evaluation technique

In order to evaluate the performance of k-NN to predict the feasibility of TSN configurations, we mainly rely on the prediction accuracy. TPR and TNR are also important to

evaluate the overall effectiveness of ML, while FPR and FNR are used to monitor the failures rates. We are also aware of the risk when the set of TSN configurations being tested is imbalanced. In that case, any ML model can fall into the naive prediction trap. Suppose there is only one true feasible TSN configuration over a set of 1000 non-feasible cases, then an ML model that just naively predicts all cases as non-feasible will achieve an accuracy of 99.9%. To avoid this situation, we use the *Kappa statistic*, aka Kappa coefficient, to measure the reliability of the proposed ML model. The Kappa statistic indicates the "agreement" between correct predictions of ML and true labels, therefore, a high Kappa value discards the possibility that the accuracy comes from chance alone.

The final results of k-NN can also suffer from overfitting, a common phenomenon in ML. It typically happens when an ML model has been overtrained with specific training data and does not generalize to unseen data sets. In order to mitigate this problem, we use the k-fold cross-evaluation technique. The data set is equally divided into k subsets. At each step, one subset becomes the testing test and the other (k - 1) subsets are combined to become the training set. The role of each subset rotates until all subsets have been tested and the prediction accuracy of the ML model is the average over all k testing sets. The k-fold evaluation technique therefore effectively reduces the risk of variability in the testing sets. In the experiments, 5-fold evaluation is applied. The training set contains 5200 labeled configurations (see §3.4.2), and the testing set has 1300 unlabeled configurations.

3.4.2 Experimental setup

The classification approach with k-NN described in §3.3.1 requires a good training set, the generation of which requires to make certain choices.

Networks in the training set

The training set contains TSN configurations that are randomly generated based on the system model, *i.e.* topology and traffic characteristics, specified in §3.2.2. In the generation of a configuration, we need to choose the number of flows in the network. If there are only a few flows, the configurations are most often feasible and the ML model cannot learn the feasibility patterns. On the other hand, too many flows make the configurations likely to be non-feasible and the learning is also ineffective. In addition, the configurations

should be well distributed in terms of the number of flows, otherwise, the trained ML model can be biased. In our study, the number of flows in the configurations is chosen in the range [50, 350] with an interval of 25. That means the values are 50, 75, 100, and so on. This range is deemed sufficiently wide for the needs of multiple applications. The type of flows is chosen randomly in such a way that after the generation of the configurations, the proportion of flow types follows the description in §3.2.2. Some configurations can be overloaded when the load of one or more links exceeds 100% the bandwidth of the links. These configurations are discarded as no feasible TSN solution can be found. With the settings above, the proportion of non-feasible configurations over the entire training set is shown in Figure 3.3. In the figure, the y-axis is the proportion of non-feasible configurations and the x-axis is the maximum load over all links. There are two main perspectives observed from the figures. First, for each TSN solution, starting from a certain point, the number of non-feasible cases increases proportionally with the maximum load. Statistically, the maximum load is higher with more traffic data, thus this result also indicates the correlation between non-feasible configurations and the number of flows. Second, more powerful TSN QoS solutions lead to fewer non-feasible cases with the same maximum load on the links, as their ability to find feasible scheduling solutions is greater. On average, the proportion of feasible TSN configurations in the training set is 8.85%, 22.04%, 52.77% and 55.96% respectively for FIFO, Manual, CP8 and Preshaping.

Size of the training set

The training set size is the single most important factor that contributes to the success of an ML algorithm. This is due to the more training data the ML algorithm exposes, the more feasibility patterns that it can learn to recognize. However, an increase in the performance with this approach comes along with a higher computation time. In order to balance this trade-off, we estimate the number of labeled configurations with the following approach. The training set size is increased from a small value until the improvement in the performance of k-NN is not considerable. The experiment is conducted with Preshaping since this is the most complex scheduling solution and needs the largest training set for the k-NN to be predictive. Figure 3.4 shows that the prediction accuracy plateaus when the training set size reaches 3250. A safety margin is integrated, subsequently, the training set

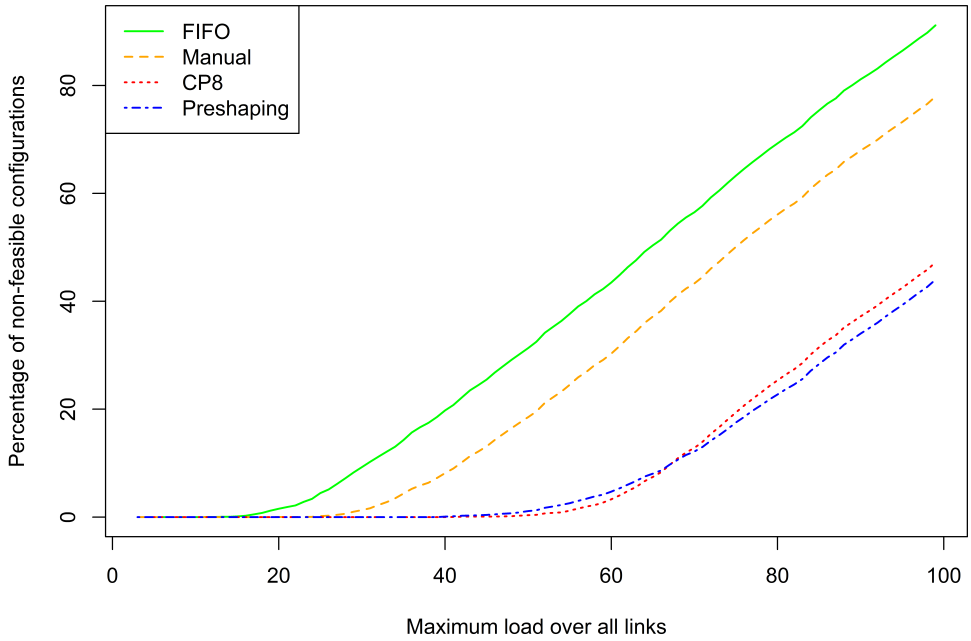


Figure 3.3: The curves represent the percentage of non-feasible TSN configurations in the entire training set (y-axis): solid green for FIFO, dashed orange for Manual, dotted red for CP8, and dash-dotted blue for Preshaping. The x-axis is the maximum load over all links. The number of non-feasible cases is clearly correlated to the maximum load, which statistically increases along with the larger number of flows. In addition, more complex TSN solutions lead to a lower number of non-feasible configurations, as better scheduling solutions can be derived.

has 5200 samples for the rest of the study.

Min-max normalization

As k-NN relies on the Euclidean distances from one data point to other data points in the feature space, a feature with superior value will potentially outweigh the contribution of other features in the final result. On the other hand, we cannot know which features are more important, and thus, a feature with a higher range of value can weaken the ability of k-NN as the ML algorithm will mostly depend on this feature. In order to control this risk and guarantee similar weights for all features, features should be scaled into the same range of values. The two most popular scaling methods are standardization and min-max. While the first method tends to re-scale the distribution of data towards the normal distribution, it is more complex than the second method. The min-max scaling only scales all values of a

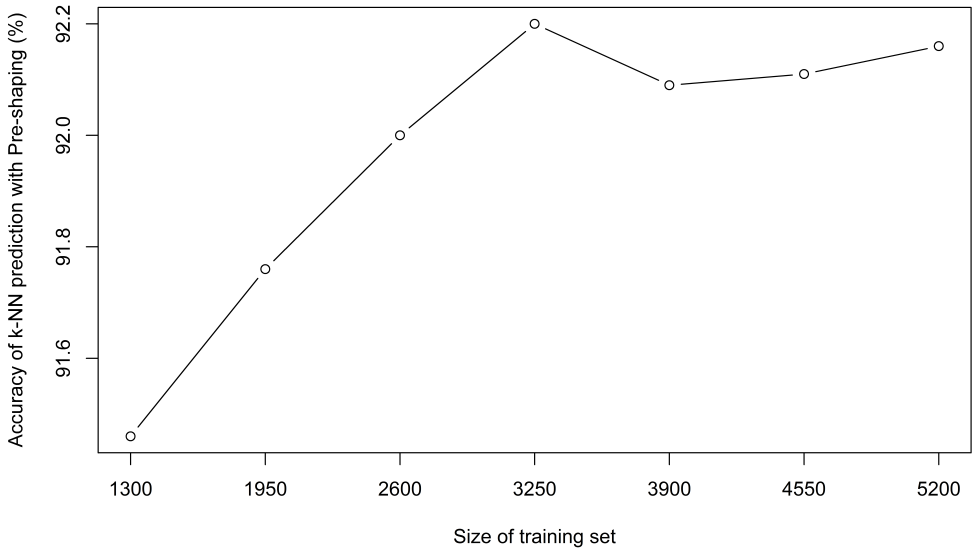


Figure 3.4: The prediction accuracy of k-NN and the size of the training set are shown on the y-axis and x-axis, respectively. The accuracy and the training set size linearly grow together, then the accuracy reaches a plateau when the training set size is over 3250. The non-deterministic properties of TSN configurations in the testing set cause a small fluctuation in the accuracy, less than 0.11%, in the range 3250 to 5200 training samples. These results are with Preshaping scheduling solution and the number of nearest neighbors k for k-NN is 20.

feature into the range [0, 1], corresponding to the minimum and the maximum value of this feature in the training set. In our experiments, we apply min-max scaling to the five features (see §3.3.2) for the sake of simplicity. It is important to emphasize that the min and max values of each feature are only taken from the training set and stored for scaling the testing set, therefore, the values out of range [0, 1] are possibly found in the unseen data.

Parameter k of k-NN

A new data point is labeled based on the labels of its nearest neighbors k. This parameter k is therefore one of the key factors for the k-NN to achieve the best performance. In the experiments, we search for a set of values k ranging from 10 to 100 with an interval of 10 on the training set. The results are the optimal values of k for each scheduling solution (column 1 in Table 3.2). There is a gap between the best prediction accuracy and the accuracy obtained when k is either 10 below or above the optimal value, but it is always

less than 1.17%. This small fluctuation suggests that k-NN is robust to a change in the value k.

3.4.3 Experimental results

In all experiments, computation times are measured on one single core of an Intel I7-8700 CPU. The implementation of the k-NN algorithm used is from R v7.3-14. The schedulability analyses are provided by RTaW-Pegase v3.4.3 and are executed through the Pegase Java library [18].

Prediction accuracy

Table 3.2 shows the performance of k-NN with the optimal values k^* for each scheduling solution. The prediction accuracies are in the range 91.84% to 95.94%. The accuracy of k-NN tend to decrease when scheduling mechanisms increase in complexity. This seems natural as the more complex the TSN QoS mechanism, the harder for the ML algorithm to predict the outcome of applying the mechanism. As a result, the prediction accuracies follow a decreasing trend from FIFO to Manual then Preshaping (Table 3.2). There is an exception with CP8, whose accuracy is slightly better than Manual despite being a more complex scheduling policy.

Table 3.2: Four scheduling solutions have different optimal k* values and different performances. Acc, TPR, TNR and K are the abbreviations for Accuracy, True Positive Rate, True Negative Rates and Kappa statistic, respectively. These results are obtained with a testing set of 1300 unseen configurations with 5-fold evaluation. Column (F) is the proportion of feasible configurations in the training set of 5200 data points corresponding to each scheduling solution.

	k*	Acc(%)	TPR(%)	TNR(%)	K	F(%)
FIFO	40	95.94	72.82	98.16	67.32	8.54
Manual	30	94.23	84.42	97.0	82.74	22.08
CP8	30	94.65	94.88	94.39	89.84	51.38
Preshaping	60	91.84	92.23	91.36	83.38	54.62

A wrong prediction can be either a false positive (non-feasible configuration deemed feasible) or false negative (feasible configuration deemed non-feasible). As shown in Table 3.2, TPR and TNR are significantly different between scheduling solutions. TNR

with FIFO is much higher than TPR, 98.16% over 72.82%, which possibly is due to an imbalance in the training set. Indeed, the percentage of non-feasible configurations in the training set with FIFO is 91.46%, much larger than feasible configurations. The ML algorithm that is trained with more negative training data tends to learn more non-feasible patterns than feasible ones. Since the learning capability of the ML algorithm does not change, the skewness over non-feasible patterns also means the lack of ability to recognize feasible patterns. Consequently, the ML algorithm is likely to conclude a configuration as non-feasible although it is actually feasible. This phenomenon can also be observed with Manual scheduling. On the other hand, the proportion of non-feasible and feasible configurations in the training set is well balanced with CP8 and Preshaping. There are 45.38% non-feasible configurations in the training set with Preshaping, corresponding to TNR 91.36% and TPR 92.23%.

The imbalances between non-feasible and feasible configurations can also happen in the testing set, which may make the ML algorithm produce the right predictions by chance alone. For instance, the prediction accuracy of k-NN with FIFO can be high just because it consistently predicts TSN configurations as non-feasible and the majority of the testing set with FIFO is also non-feasible. To examine this hypothesis, the Kappa statistic, aka Kappa coefficient, is calculated to measure the agreement between true labels and correct predictions. If the value is low, the agreement and thus the high prediction happens by chance. On the other hand, if the Kappa statistic is high, usually $> 60\%$ [89], the hypothesis above is null. The Kappa statistics of four scheduling solutions are shown in Table 3.2. The values range from 67.32% to 83.38%, accordingly, the high prediction accuracy of k-NN is not just by chance but also due to the contribution of the learning.

Robustness of prediction

The training phase of an ML algorithm includes the labor intensive training set generation. The training set is based on an assumption that its distribution will be similar to the testing set. In case the unseen data does not match this assumption, *i.e.* it follows a different distribution than the training set, the ML algorithm needs to have the generalization ability to adapt to new data and have a high prediction accuracy. The ML algorithm that possesses this ability does not need to be retrained each time there is a change in the systems, which

is important to consider in practical applications.

We experimented with changes in the topology and traffic characteristics to answer the question of whether these changes influence the performance of k-NN and therefore whether it can have some degree of generalization. The training set remains unchanged and the modification is only applied to the testing set as follows. The data payload size s of critical flows is set to a random value in the range $[(1-x)\cdot s,(1+x)\cdot s]$ where x is the parameter that indicates the difference from s. Since the variations happen in both directions from the original value s, the average network load does not vary considerably.

We conduct nine successive experiments with the value x from 0.1 to 0.9, where for each experiment the value is increased by 0.1. The testing set size has 1300 data points. Figure 3.5 summarizes the prediction accuracies obtained with optimal k values under the form of boxplots. Horizontal lines are baseline performances for each scheduling solution, *i.e.* prediction accuracies when there is no change in the training set. The results show that a degradation in the performance of k-NN actually happens, however, it is limited. The decrease is less than 2.2% for any values of x on all four scheduling solutions. A preliminary reason is that two features, the maximum load among the links and the Gini index, have embedded information that are robust to variations in the payload size. As a result, k-NN is still predictive with the new testing data.

3.5 Comparison with schedulability analysis

We note that the feasibility derived by the precise analysis is considered 100% accurate in this study as this is the most accurate schedulability analysis in our toolset. In addition, the precise analysis is pessimistic, in the sense that it returns upper bounds of WCTT and therefore, there is no false positive, *i.e.* a non-feasible configuration being wrongly predicted feasible. The downside of this pessimism is that its results will contain a certain number of false negatives (a configuration deems non-feasible but that is actually feasible). However prior works [90,91] suggest that schedulability analysis is tight for FIFO and static priority scheduling, the policies considered in this work. As a result of the analysis being conservative, some TSN configurations predicted feasible but labeled as non-feasible may be counted as false positives. In summary, the pessimism of schedulability

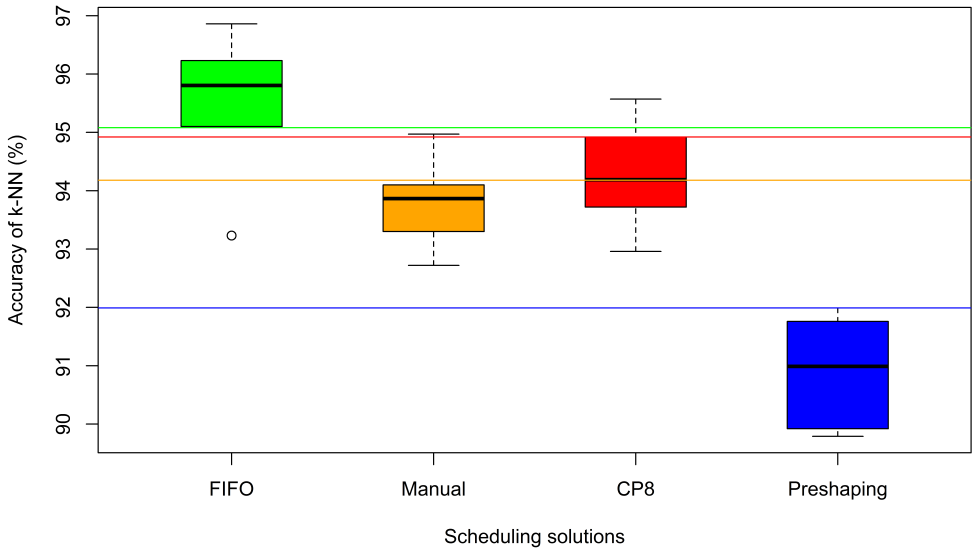


Figure 3.5: Boxplots shows the variance in the prediction accuracies of k-NN when there are changes in the payload of critical flows with respect to the training data. The variations in either positive or negative directions are between 10% and 90% of the original payload. Each box represents the results of 9 experiments with a granularity of change of 10%. Each horizontal line is the prediction accuracy of k-NN when the traffic patterns of the testing set are identical to the training set. The horizontal line is therefore the baseline accuracy of the box with the same color.

analysis introduces a bias in the results, accordingly, FPR is overestimated while TNR is underestimated. False positives are our main concern since they may lead to wrong and unsafe design choices.

In Table 5.2, we summarize the prediction accuracy with k-NN, precise and approximate analysis for the four different scheduling solutions. k-NN is slightly less accurate than the approximate analysis in simple scheduling solutions, FIFO and Manual, with -2.98% and -2.31% respectively. However, k-NN outperforms the approximate analysis on more complex mechanisms such as CP8 ($+8.34\%$). These results suggest that ML is comparable with the approximate analysis and can be an alternative approach in the design space exploration of TSN networks.

In the context of design-space exploration for TSN networks, the computation time of a verification technique is also an important factor besides the accuracy. Figure 3.6 shows the trade-offs between times and accuracy for each technique with the Manual scheduling solution on a PC with a single CPU core. The choice of the Manual scheduling

	k-NN	**Approx. analysis**	**Prec. analysis**
Approach	Sup. learning	Sched. analysis	Sched. analysis
FIFO	Acc.: 95.94% TPR: 72.82% TNR: 98.16% FPR: 1.84%	Acc.: 98.92% TPR: 87.39% TNR: 100% FPR: 0.0%	Acc. :100% TPR: 100% TNR: 100% FPR: 0.0%
Manual	Acc.: 94.23% TPR: 84.42% TNR: 97.0% FPR: 3.0%	Acc.: 96.54% TPR: 84.32% TNR: 100% FPR: 0.0%	Acc. :100% TPR: 100% TNR: 100% FPR: 0.0%
CP8	Acc.: 94.65% TPR: 94.88% TNR: 94.39% FPR: 5.61%	Acc.: 86.31% TPR: 75.35% TNR: 100% FPR: 0.0%	Acc. :100% TPR: 100% TNR: 100% FPR: 0.0%
Preshaping	Acc.: 91.84% TPR: 92.23% TNR: 91.36% FPR: 8.64%	Acc.: NA TPR: NA TNR: NA FPR: 0.0%	Acc. :100% TPR: 100% TNR: 100% FPR: 0.0%

Table 3.3: Summary of experimental results. The approximate analysis leads to the less accurate results for CP8 with 86.31%. This type of schedulability analysis is not available for the Preshaping scheduling solution. We note that neither approximate nor precise analysis produces false positives, which is the case when using ML.

solution is deemed fair to provide a clear picture of the trade-offs. Indeed, FIFO would be too simple solution. On the other hand, CP8 and Preshaping requires multiple steps and includes parameter setting such as priorities and shaping intensity, whose execution times are not easy to control. In contrast, the computation time of Manual solely corresponds the execution of a single schedulability analysis.

The observations above lead to the following insights:

- When the number of configurations tested is small, as 10^3 in Figure 3.6, k-NN algorithm is not competitive with respect to both schedulability analyses since it has less accuracy and executes slower. In other words, the schedulability analyses, precise and approximate analysis, provide Pareto optimal trade-offs between performance and computation. For a medium testing set with 10^5 configurations, the schedulability analyses are also the dominant verification techniques over k-NN.

- In a context where an efficient approximate analysis is not available, such as for Preshaping, the benefit gained from the use of k-NN is considerable even for a small testing set. Since k-NN is very fast compared to precise analysis, its execution time can practically be ignored. The execution of the ML algorithm comes from labelling the 5200 configurations in the training set with schedulability analysis.

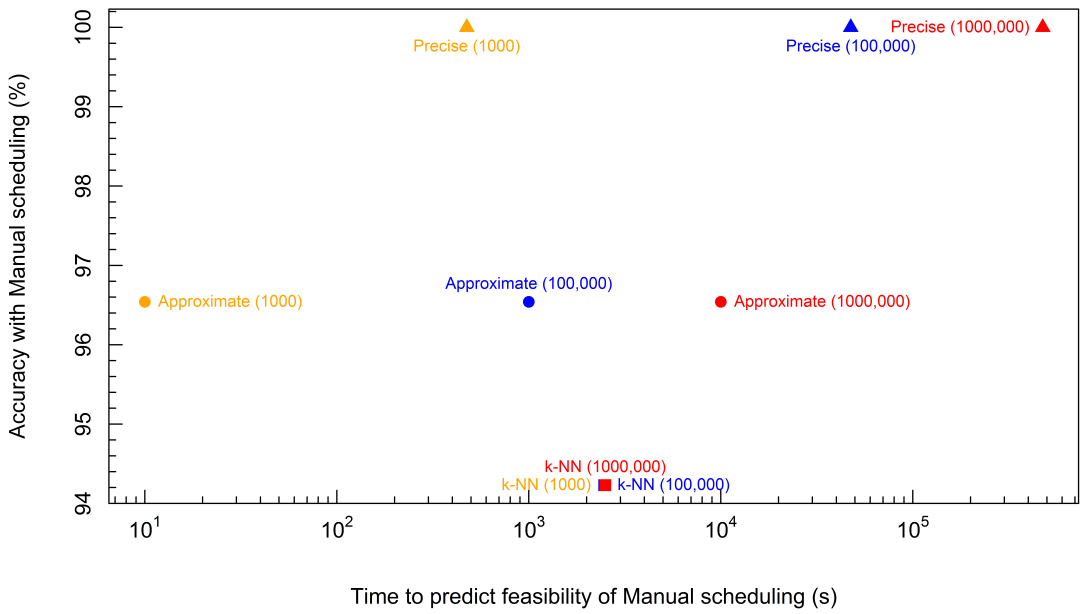


Figure 3.6: Time to determine feasibility with k-NN, precise and approximate analysis. The y-axis is the accuracy (%) and the x-axis is the total execution time in seconds in log scale. The results correspond to testing sets of size 10^3, 10^5, and 10^6, respectively in yellow, blue, and red. TSN configurations have a number of flows ranging from 50 to 350 and apply the Manual scheduling solution. The results of k-NN are represented by squares, while points are for approximate analysis and triangles for precise analysis. When the testing set size grows, the running time of k-NN only increases slightly. The running time of schedulability analyses is linear with the number of configurations.

Consequently, k-NN leads to a shorter computation time than the precise analysis as soon as the number of configurations in the testing set exceeds 5200.

- In the case of large testing sets with 10^6 configurations, all three verification techniques are meaningful in terms of trade-offs between accuracy and computation time. However, precise analysis is not an ideal choice due to its impractical execution times. For instance, with the Manual scheduling solution, precise analysis takes 132 hours while the approximate analysis takes less than 3 hours and only 42 minutes for k-NN. The supervised ML algorithm achieves a speedup factor of 190 over the precise analysis.

It is worth noting that both schedulability analyses are actually very fast. Indeed, the approximate analysis takes only 14.77ms on average to derive the feasibility of a TSN

configuration with 350 flows, and the precise analysis takes 804.52ms [1]. Therefore, other verification techniques such as ML can reach maximal efficiency when applied to a vast amount of configurations or to more complex scheduling solutions.

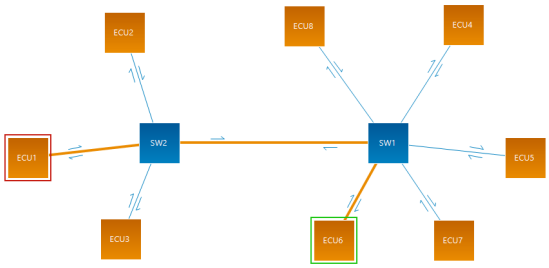


Figure 3.7: The topology used in the second part of this chapter. The unicast flow highlighted in the figure is from ECU1 to ECU6.

3.6 Insights into when machine learning fails

This section and the next two sections belong to the second part of the chapter. In this part, to be able to better explain the experimental results, we intentionally re-use the simple topology from [1] shown in Figure 3.7 that contains two switches connected by a 1Gbps link, and eight nodes connected by 100Mpbs links. The topology is called ”dumbbell” topology. It is the same as the topology as in [92], whose structure is similar to an early Ethernet network prototype of an automotive OEM [21].

In this part, we still use the traffic characteristics detailed in Table 4.1. The training set has 4000 labeled configurations and the testing set has 1000 unseen configurations. Our technical report and the previous part of the chapter show that k-NN suffers from false positives, *i.e.* non-feasible configurations wrongly concluded as feasible. In this section, we analyze the cases of wrong predictions and make proposals on how to choose a method to control the false positive rate.

3.6.1 Predictive power of the selected features

Features are used to capture important information and domain-specific knowledge used by an ML algorithm, so that the algorithm can learn to recognize the useful patterns. In this section, we analyze the features’ predictive ability to try to understand when ML cannot

be trusted. Five features proposed by [1] are re-used: the number of critical, audio, and video flows, the maximum load among the links in the network, and the Gini index to measure the unbalancedness of the load in the links. The predictive power of the features can be estimated by frequency histograms that show whether the feasible and non-feasible configurations are clearly separated by the feature values. For instance, Figure 3.8 shows the frequency histogram of feasibility with Preshaping when the values of five features change. Other scheduling solutions also lead to similar patterns. In Table 3.4, the predictive power of the features with all scheduling solutions are measured with Pearson coefficients, which compute the correlation between the feasibility and the corresponding feature values.

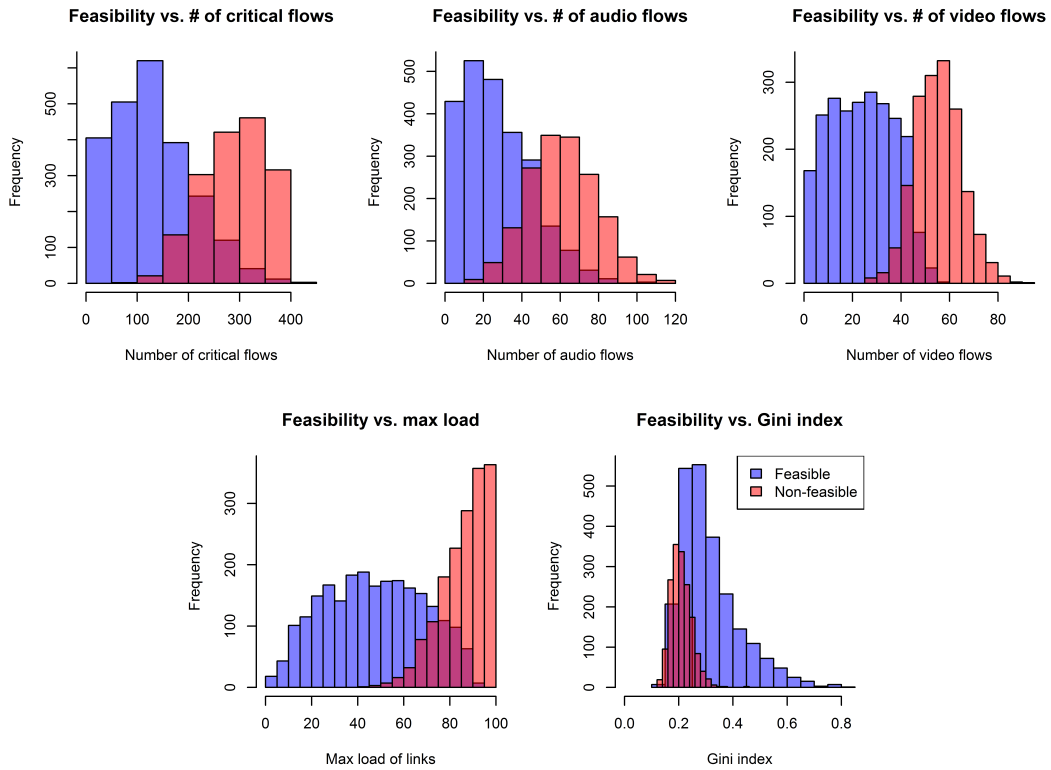


Figure 3.8: Histograms (frequency of appearance) of feasible configurations (blue) and non-feasible configurations (red) with respect to each feature, counted on the training set containing 4000 samples. The histograms suggest that k-NN is able to distinguish two well-distinct groups, however, the classifier can make wrong predictions in the overlapping areas (purple).

We observe from Figure 3.8 and Table 3.4, that the features with the most predictive power are the maximum load, followed by the number of critical flows, then the number of video flows. The dependence of the feasibility on video flows can be explained by the large

bandwidth they required and their tight deadlines. Indeed, in an ADAS, a camera image is made up of 15 consecutive packets but the deadline constraint is 10ms (*i.e.* all packets must have been received within 10ms). In the case of maximum load, it is clear that finding a feasible scheduling solution becomes harder when the load value is high, particularly for the flows routed over the most "crowded" links. The last frequency histogram in Figure 3.8 corresponding to the Gini index shows that non-feasible configurations have a better load balance than feasible configurations. This observation seems counter intuitive, however, it is reasonable from the training set generation point of view. Indeed TSN configurations are generated randomly, thus the more flows, the higher balance in the load of the links. This unbalancedness even decreases exponentially with the increase of flows [93]. As a result, the more balanced the loads, the higher the probability of a configuration to be non-feasible.

Table 3.4: Correlations between the values of the selected features and the feasibility of TSN configurations (0 for feasible and 1 for non-feasible) with the Pearson coefficient. The correlation measured on the Gini index is negative since a high Gini index indicates a less balanced network, that is likely to possess a small number of flows.

Feature	**FIFO**	**Manual**	**CP8**	**Preshaping**
# of critical flows	0.38	0.6	0.73	0.75
# of audio flows	0.35	0.55	0.68	0.69
# of video flow	0.43	0.67	0.79	0.8
Max load	0.5	0.72	0.8	0.73
Gini index	-0.55	-0.67	-0.51	-0.5

Overall, these five features capture useful information to predict the feasibility of TSN networks. However, there are some gray areas in Figure 3.8, which show the difficulty for each features when feasible and non-feasible configurations are not clearly separated, and therefore the prediction can be wrong.

3.6.2 Analysis of wrong predictions

In the histogram corresponding to each feature, the gray area is where the ML algorithm can wrongly predict the feasibility. Consequently, using multiple features can mitigate this problem since the gray area of one feature may not overlap with the gray areas of other features. There are still some areas where the separation is not clear, *i.e.* feasible

and non-feasible configurations are mixed in the feature space. Figure 3.9 highlights these areas, where the black dots represent incorrect predictions while green and red dots represent correct feasible and non-feasible predictions, respectively. In the figure, the dimensions are the number of critical, audio, and video flows. Although the other features are not shown in the figure, they can be determined indirectly through the three features. For instance, the maximum load is higher in the configurations with more flows and they are likely to be non-feasible.

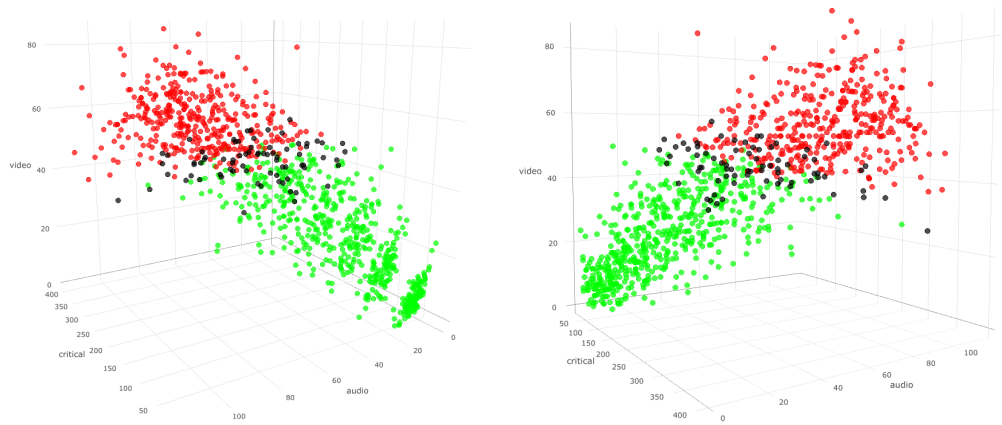


Figure 3.9: Predictions of k-NN for Preshaping scheduling solution on 3D plots (different viewpoints). The testing set contains 1000 unseen configurations. Green represents true positives, red true negatives, and black wrong predictions, which mostly happen in the mixed area between feasible and non-feasible groups.

An observation from Figure 3.9 is that the wrong predictions of k-NN happen more frequently in the mixed area between feasible and non-feasible configuration in the feature space. The use of multiple features also lead to some gray areas, where neither feasible nor non-feasible configurations are dominant. In other words, in the k nearest neighbors, non-feasible neighbors cannot certainly outnumber feasible neighbors and vice-versa. Subsequently, k-NN tends to predict wrongly in these areas.

To the best of our knowledge, none of the voting rules is effective for these gray areas. Support Vector Machines [85] also suffered from similar issues, as we found it hard to determine the boundary between the areas of feasible and non-feasible configurations, defined by the Maximum Margin Hyperplane (MMH). Therefore, in the following sections, we propose a technique that identifies TSN configurations in the gray areas, where k-NN is likely to predict wrongly. We then trigger the schedulability analysis for those

configurations to reduce the incorrect prediction rate.

3.7 Hybrid verification strategy combining machine learning and schedulability analysis

From the prediction accuracy in the experiments and the insights gained in Section 3.6, we perceive the weakness of nearest neighbors algorithms like k-NN that the prediction outcomes cannot be fully trusted for TSN configurations in the gray areas. Therefore, we propose a hybrid approach combining the prediction of k-NN and schedulability analysis as a new verification strategy. It aims to control the wrong prediction rate of ML while still gain in computation speed compared to always executing schedulability analysis. The data sets and the R code implemented in the experiments are available as open-source (AGPL V3.0) at `https://github.com/crtes-group-unilux/ML4TSN-Schedulability`.

3.7.1 Principles

The hybrid approach is conceptually simple. When applying k-NN to predict the feasibility of an unseen configuration, if the majority of its nearest neighbors does not outnumber the minority significantly, the configuration is supposed to be in the gray area where the prediction of k-NN cannot be fully trusted. We then execute a precise analysis to identify the feasibility of the configuration. On the other hand, if the minority is actually small, *i.e.* a clear consensus is achieved among the nearest neighbors of the unseen data, the result of k-NN is trusted. It is worth emphasizing that the proposed hybrid method works with k-NN, it also generalizes to other ML algorithms. The key factor is how to identify the gray area, for instance, the maximum margin hyperplane is a natural choice in SVM.

In order to apply the hybrid method, we need to determine a decision criterion that triggers the schedulability analysis. In this study, it is the proportion of the minority group in the k nearest neighbors. The schedulability analysis is executed if this proportion is higher than a threshold. Various values of the threshold lead to the difference in the performance of the hybrid model, as shown in Table 3.5. The optimal number of nearest neighbors k for k-NN follows the previous study [1]: 30 for FIFO, 20 for Manual, 90 for

Table 3.5: The performance of the hybrid model with a threshold from 5% to 40%. The table is divided due to space limitation. The first row is the results with k-NN used alone. Column (#) indicates the number of configurations that undergo precise analysis. All performance metrics: Accuracy (acc), TPR, and TNR are in percentage. There are 1000 unseen configurations in the testing set.

	FIFO				**Manual**			
Threshold (%)	(#)	Acc	TPR	TNR	(#)	Acc	TPR	TNR
–	**0**	**96.4**	**68**	**98.7**	**0**	**95.5**	**84.49**	**98.03**
5	120	99.6	94.67	100	208	99.4	96.79	100
10	100	99.4	93.33	99.89	164	99	94.65	100
15	86	99.1	90.67	99.78	136	98.4	92.51	99.75
20	68	98.6	86.66	99.57	107	97.8	90.37	99.51
25	47	98.3	82.67	99.57	84	97.3	88.24	99.38
30	39	98.1	80	99.57	61	96.9	87.17	99.14
35	33	97.9	78.67	99.46	38	96.6	86.1	99.02
40	18	97.4	76	99.14	24	96.5	85.56	99.02

	CP8				**Preshaping**			
Threshold (%)	(#)	Acc	TPR	TNR	(#)	Acc	TPR	TNR
–	**0**	**94.7**	**95.39**	**93.89**	**0**	**92.4**	**93.85**	**90.36**
5	319	100	100	100	383	100	100	100
10	258	99.9	100	99.78	297	100	100	100
15	206	99.5	99.82	99.13	239	99.3	99.49	99.04
20	165	98.9	99.26	98.47	190	98.8	98.97	98.55
25	121	98.3	98.89	97.6	151	97.5	97.44	97.59
30	87	97.7	98.15	97.16	124	96.8	97.09	96.39
35	69	97.1	97.05	97.16	91	95.8	96.24	95.18
40	53	96.7	96.68	96.72	62	95.3	95.9	94.46

CP8, and 20 for Preshaping. We note that the optimal values for k are determined when only k-NN is used, which may not be optimal for the hybrid approach.

3.7.2 Experimental results

We observe in Table 3.5 that higher accuracies are achieved when the threshold is low. The reason is that a low threshold means that even with a low number of data points in the minority group, schedulability analysis is still triggered. The prediction accuracy can be controlled by the threshold value. If the value is low enough, as 5% with FIFO and 10% with other scheduling solutions, there is no false positives with the hybrid model. When the

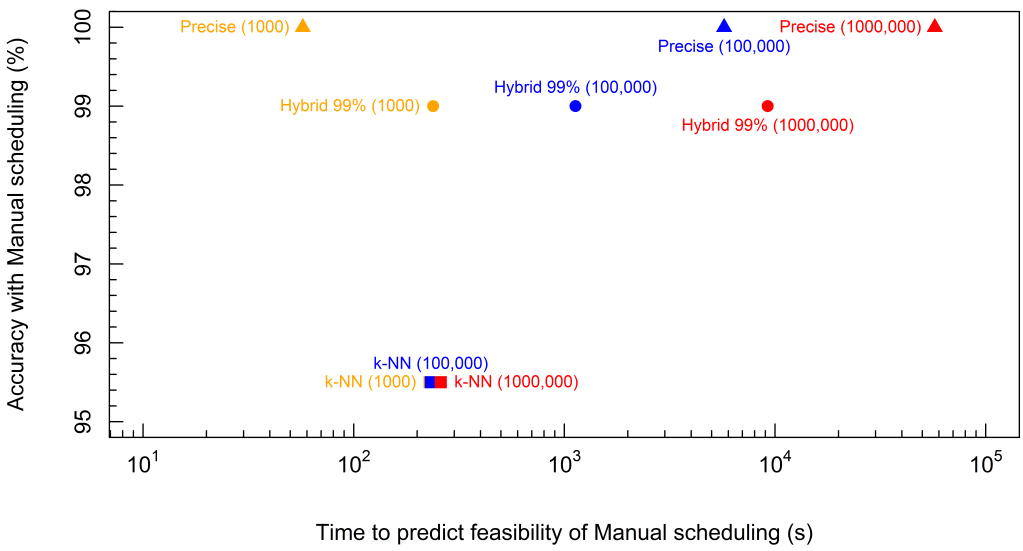


Figure 3.10: The x-axis is running times measured on an Intel 8700 with 6 cores, and the y-axis is the accuracies (%) with the Manual scheduling solution, for a testing set containing 10^3, 10^5, and 10^6 configurations. Results of k-NN, the hybrid model, and Precise analysis are respectively represented by squares, points, and triangles. The threshold in the hybrid model is set to achieve 99% accuracy. Each method provides a useful trade-off between performance and computing time, except with the testing set containing 10^3 configurations that Precise analysis is a dominant technique.

threshold is set to 40%, the hybrid model significantly reduces false positives compared to k-NN: 33% with FIFO, 50% with Manual, 46% with CP8, and 43% with Preshaping. The hybrid model may even reach a 100% maximum prediction accuracy without performing schedulability analysis on all tested configurations. For instance, we only need to conduct the precise analysis on 32% of tested configurations with CP8 and 38% with Preshaping. We also note that although the hybrid approach can effectively control the prediction accuracy, it does not totally rule out the possibility of wrong predictions but only makes them less likely than with ML alone. This can be observed from Table 3.5, as the accuracy with Preshaping improves from 2.9% up to 7.6%.

Table 3.6 shows estimations of how many configurations require schedulability analysis in order for the hybrid model to obtain 99% prediction accuracy. These estimations are derived by linear interpolation from the results in Table 3.5, with the optimal k value for k-NN [1] and k equal to 100 for four scheduling solutions. These estimations suggest that

Table 3.6: We estimate the number of configurations undergoing precise analysis to achieve 99% accuracy in column #. k* is the optimal number of nearest neighbors that is different for each scheduling solution.

	FIFO		**Manual**		**CP8**		**Preshaping**	
k	30*	100	20*	100	80*	100	20*	100
(#)	82	82	164	178	171	176	191	231

when the scheduling solutions are more complex and more powerful (see [1]), we need to conduct more schedulability analyses. In addition, the optimal value k leads to higher accuracies than an arbitrary value, as 100 in our case.

In practice, the accuracy of the hybrid model with a certain threshold also depends on how representative the training data are with respect to the unseen data, since k-NN is a supervised learning algorithm. We can mitigate this problem by estimating the distance from the training data to the tested data, as suggested in [94], to lower the threshold and compensate for the inefficiency of ML. However, our experiment in [1] suggests that k-NN is robust with respect to a variance of the traffic patterns.

Table 3.7: Estimated prediction accuracy (in %) if 10% of the configurations undergo precise analysis in the hybrid model.

	FIFO		**Manual**		**CP8**		**Preshaping**	
k	30*	100	20*	100	80*	100	20*	100
Acc	99.4	99.4	97.65	97.96	97.93	97.92	96.07	95.83

In Table 3.7, we estimate the accuracy of the hybrid model when 10% of tested configurations undergo the precise schedulability analysis. The estimates are also derived by linear interpolation from the results in Table 3.5. There is a decrease in the accuracy of the more complex scheduling solutions. The optimal value of k with solely ML algorithm is still preferable for the hybrid model, except for Manual scheduling where the value of 100 is a better choice. In addition, although more schedulability analyses always lead to a higher accuracy, the threshold that controls the maximum percentage of the precise analysis should be decided based on CPU resources available.

3.7.3 Trade-offs between accuracy and computation times

Figure 3.10 shows for Manual scheduling the trade-offs between accuracy and computation times offered by k-NN, the hybrid model and the precise analysis. The running times are measured on an Intel 8700 CPU with 6 cores 3.2GHz. Other scheduling solutions also lead to similar trade-offs. The speedup factors of the hybrid model (for 99% accuracy) over the precise analysis are shown in Table 3.8 for all scheduling solutions.

Table 3.8: Estimation on different testing set sizes (column #) of the speedup factors of the baseline hybrid model compared to precise analysis. The hybrid model is calibrated to achieve 99% accuracy.

(#)	FIFO	Manual	CP8	Preshaping
10^3	0.24	0.24	0.24	0.24
10^5	8.19	4.9	4.74	4.33
10^6	11.62	5.95	5.71	5.13

We observe from the experimental results that the interest of the different trade-offs between accuracy and running time depends on the size of the testing set:

- For small testing set with 10^3 configurations, both k-NN and the hybrid model are less competitive than the precise analysis in terms of accuracy and running time. It is due to the generation of the training set, as it takes around 3 hours to label the configurations for all four scheduling solutions.

- For medium testing set and large testing set with 10^5 and 10^6 configurations, respectively, the three methods are Pareto optimal and provide a useful trade-off. With Manual scheduling solution, the hybrid model with 99% accuracy has a speedup factor from 4.33 up to 11.62 over the precise analysis. ML alone is the fastest method with a speedup factor 26 over the precise analysis on a testing set containing 10^5 configurations, however, its accuracy is lower with 95.5%.

We note that with more complex mechanisms as CP8 and Preshaping, the speedup factor is smaller than for Manual scheduling. However, the difference between the running time of the two methods, the hybrid model over the precise analysis, in absolute value is higher.

3.8 Multi-stage hybrid approach with an approximate analysis

In the previous sections, the schedulability analysis used in the experiments is always the precise analysis. However, there are different schedulability analyses that vary in the degree of their pessimism and their execution time. For instance with Manual scheduling solution, the approximate analysis is faster 40 times than the precise analysis (see [1] for more details).

The downside of the approximate analysis is that it is more pessimistic than the precise analysis, which causes more false negatives (a feasible configuration that is wrongly predicted non-feasible). However, it never leads to false positives (wrongly predicting a non-feasible configuration as feasible). Therefore, we propose to combine k-NN, the approximate analysis and the precise analysis in a hybrid model containing multiple stages of verification as follows:

1. Use k-NN to predict the feasibility of the tested configurations.

2. Identify the configurations that seem to be in the gray area with the threshold in Section 3.7. Trigger the approximate analysis. If it concludes a configuration as feasible, then the configuration is actually feasible since the approximate analysis never produces a false positive. This configuration is removed out of the set in the gray area.

3. Trigger the precise analysis on configurations that are still in the gray area. This is the final conclusion about the feasibility of these configurations.

This multi-stage approach can be extended with more schedulability analyses, together with the condition that they do not produce false positives.

In Table 3.9, we present the performance of the new hybrid model. There is no result Preshaping scheduling solution, as an approximate analysis for Preshaping is not available. It should be noted that the approximate analysis when applied with CP8 cannot identify any configuration in the set of the gray area as feasible, therefore it did not improve the performance. These results show that the rate of false negatives produced by the approximate analysis should be low enough for the analysis to play a positive role in

Table 3.9: Performance of the multi-stage hybrid model with k-NN, approximate analysis, and precise analysis, expressed in percentage. The testing set has 1000 unseen configurations. The first row is the results for k-NN alone. The value k for FIFO is 30 and Manual is 20. Columns (1) and (2) respectively show the number of configurations that undergo approximate analysis and precise analysis.

	FIFO					**Manual**				
Threshold (%)	(1)	(2)	Acc	TPR	TNR	(1)	(2)	Acc	TPR	TNR
–	**0**	**0**	**96.4**	**68**	**98.7**	**0**	**0**	**95.5**	**84.49**	**98.03**
5	181	137	99.8	97.33	100	208	166	99.4	96.79	100
10	100	64	99.4	93.33	99.89	164	132	99	94.65	100
15	86	53	99.1	90.67	99.78	136	109	98.4	92.51	99.75
20	68	40	98.6	86.67	99.57	107	81	97.8	90.37	99.51
25	57	35	98.4	84	99.57	84	62	97.3	88.24	99.38
30	39	27	98.1	80	99.57	61	44	96.9	87.17	99.14
35	33	23	97.9	78.6	99.46	38	28	96.6	86.1	99.02
40	18	9	97.4	76	99.14	24	18	96.5	85.56	99.02

the overall efficiency. The results in Table 3.9 show a significant reduction of the cases requiring precise analysis for both FIFO and Manual scheduling solutions, compared to the baseline hybrid model in the previous sections. In particular, the new hybrid model reduces the number of executions of the precise analysis by 38% and 19% with FIFO and Manual, respectively.

3.9 Related work

The application of ML in real-time systems has been found as early as 2006 when the authors in [95] use ML to determine loop bounds of worst-case execution time (WCET) in an automatic manner. Another study from this research group uses Bayesian network, a probabilistic graphical model in ML, to predict which instructions are in the cache [96]. A real-time system usually stores instructions in caches to reduce the execution speed of programs, however, it becomes harder to infer the time needed. An accurate prediction on the presence of instructions is thus highly useful to improve the WCET estimate. Later, a study in [97] use Deep Learning in interference analysis of multicore platforms. Due to interference among tasks that compete for shared resources in cache and hardware, the execution time can increase rapidly and become very complex to analyze. This study uses

the model-free ML model to predict the inference for a set of tasks, which is useful in design, timing watchdogs, and schedulability analysis.

More advanced ML techniques such as reinforcement learning (RL) have also been devoted to tackling problems in real-time systems. For instance, RL is a core technique in the learning-based framework to address dynamic voltage and frequency scaling (DVFS) in multi-core processors [98]. The framework optimizes existing DVFS techniques and selects the best ones to intelligently scale voltage and frequency, which leads to lower energy consumption. In [71], the authors use RL-based genetic algorithms to schedule tasks in real-time, therefore it balances loads in edge computers. Deep reinforcement learning, a combination of DL and the RL techniques, is used for multi-core task scheduling in processors [99].

A study that is relevant to the application of ML in this chapter is [100] that implements a framework to generate a large number of artificial test problems. This open-source framework assists in configuring and performing schedulability analysis, which is similar to the commercial toolset RTaW-Pegase [18] applying in the experiments of our study. These frameworks can assess the performance of real data as well as generate synthetic data. Since the development of a real-time system demands extensive experiments, the frameworks are the key to facilitate the design-space exploration.

3.10 Discussion and perspectives

The contributions in this chapter contribute to answer research questions **RQ1** and **RQ2** in Section 1.2 and show that supervised ML such as k-NN can effectively predict the feasibility of TSN networks with realistic topology and traffic patterns. In some contexts, ML outperforms approximate schedulability analysis in terms of prediction accuracy and running times, while the latter performance metric is the main advantage of ML over precise analysis. ML algorithms do not require a large training set and the computing power is modest. These characteristics make them suitable to become a part of the network design toolbox running on a single desktop computer. Further studies are needed to assess the potentials and limitations of this ML-based approach in other applications.

Although the ML model can be a viable verification method for the feasibility of

TSN configurations, it produces a certain number of false positives, which is not the case with schedulability analyses. The hybrid model of ML and schedulability analysis can effectively control the false-positive rate. However, we note that the uncertainty of ML prediction does not disappear, which means all chosen configurations must be tested by precise analysis in the final phase of DSE.

A challenge with supervised ML is the dependence on training data while it is applied to unseen data, and both data sets may have dissimilar patterns. This phenomenon called distribution shift is one of the main reason why ML may fail silently. In the case of k-NN, the wrong predictions can happen when the majority group in the k nearest neighbors does not outnumber the minority group (see Section 3.6). Experiments in this study prove that k-NN is robust to variabilities in the network traffic from the training data. However, we note that network topology is fixed in the experiments. Outside the scope of this study, a change in network topology will make the problem become more difficult and the training set needs to reflect the diversity of topologies.

The application of ML to feasibility prediction, proposed in the study of this chapter, can be extended to several promising directions:

- In this chapter, we applied standard and simple ML algorithms, *i.e.* k-NN, that require a modest training set and computing power. These can be supplied by a single desktop computer to run in a few hours. We expect that ML will achieve better performance if some key factors during the training phase are improved:

    1. The training set: it should be larger and more heterogeneous in terms of topology and traffic patterns.
    2. Feature engineering: additional features, for instance, priorities of flows, are essential for the ML model to access more relevant information regarding the feasibility patterns.
    3. ML algorithm: advanced algorithms such as XGBoost [101] can learn more complex relations among flows, links, and queues in TSN networks.
    4. The uncertainty of ML predictions: an ensemble of individual ML models is a promising fault-tolerant mechanism to mitigate the risk. The simplest ensemble technique is majority voting.

- The generation of the training set is often the most compute-intensive task during the development of an ML application. We note that the CPU time to generate TSN configurations is a tiny fraction of the CPU time to label the configurations. Therefore, a combination of labeled and non-labeled configurations that is several times larger than the current training set can be achieved with only a slight increase in the CPU time. Semi-supervised learning may leverage such a mixed training set to achieve comparable performance with supervised learning while using fewer labeled configurations, leading to a better efficiency.

- The manual feature selection approach in this chapter requires the involvement of domain expertise, with the risk to potentially miss some relevant features. Beyond the current approach, future studies should consider deep learning due to its ability to automatically synthesize features based on raw input data. This is explored in the next chapter of the thesis.

In this chapter, ML is considered as an alternative to schedulability analysis to verify whether traffic flows meet deadline constraints. In many contexts, simulation is the only choice to verify other constraints. For instance, in a network with complex protocols as TCP, throughput constraints also need to be verified. In a simulation, we need to initialize a larger amount of samples with different settings. ML can be trained to provide suggestions about the initialization and sample size in order for the simulation results to be sufficiently robust.

The ML approach in this chapter can be used to develop verification methods for other sorts of real-time systems. One of the bottlenecks requiring assistant algorithms is schedulability analysis across more than one resource, which is very compute-intensive. Besides the realm of feasibility prediction, ML is a promising tool to tackle various problems in real-time networking. For instance, ML is a good candidate to solve the problems regarding resource allocation [99].

Chapter 4

Application of Graph Neural Network to TSN Networks

Abstract

In the previous chapter, machine learning (ML) has been applied to predict whether TSN configurations are feasible without schedulability analysis. However, conventional ML algorithms such as k-NN have drawbacks: we need to extract input features based on domain expertise, and their performance is not consistent when topologies or traffic patterns vary. In this study, we try to overcome these problems with Graph Neural Network (GNN), a deep learning model specified for graph-based data. An important difference between GNN and conventional ML algorithms is that GNN directly gets all relevant information from raw data through the encoding of a TSN configuration as a graph. In addition, the GNN model can exploit complex relations among physical and logical components in the configuration by aggregating and updating representative vectors of the graph. The evaluation of multiple testing sets shows that GNN possesses the ability to generalize to more diverse topologies and traffic patterns. On 13 testing sets with realistic automotive networks that use priorities as Quality of Service mechanism, an ensemble of 32 GNN models achieves a prediction accuracy from 79.3% up to 90%. The GNN model is faster than schedulability analysis by a factor ranging from 77 to 1715 times, which is a key advantage for design space exploration.

This chapter, mostly published in [4], addresses research question **RQ3** in Section 1.2:

RQ3: If the answer for RQ1 is positive, *i.e.* ML can be used to predict the feasibility of TSN networks efficiently, we then ask can ML models generalize the prediction ability to TSN networks that are different from those in the training set. In case traditional ML algorithms do not possess this ability, can specific neural networks models like GNN attain this ability to generalize?

4.1 Introduction

4.1.1 Context of the study

In the previous works, we have used standard and simple ML algorithms [85] such as k-NN classifier and K-means clustering [1,2] to predict the feasibility of TSN configurations. These ML algorithms can serve as approximator to precise schedulability analysis, as their prediction accuracies are higher than 85% in our case-studies. The running times of the ML algorithms are consistent with the complexity of the networks and they provide a high speedup factor on realistic automotive networks [1], which allows designers to explore more use-cases in search space. The conventional ML algorithms however suffer from two main limitations:

1. During the development of the ML model, we need to select network features that seem to be relevant to the feasibility patterns and thus may contribute to higher prediction accuracy. This process is both time-consuming, due to the manual selection, and error-prone, as it relies on domain expertise and experience.

2. The previous works [1,2] only consider networks whose topology is fixed, *i.e.* the topologies in training set and testing set are identical. In addition, the scope of these studies is also limited by the small variability in the traffic patterns, as we assume that transmission patterns of flows in unseen configurations do change considerably from the training data.

These weaknesses of conventional ML algorithms raise the question of their performance when topology or traffic patterns in unseen configurations have significant changes.

If for each new topology, we need to generate a new training set and re-train the ML model, then the resource required for ML can make it less attractive. Another concern is the manual feature selection step that requires experts in networking to be familiar with ML, which incurs more labor costs. These open questions highlight the bottlenecks caused by the simple ML models and call for other approaches to the feasibility prediction problem.

4.1.2 Research questions

In this chapter, we address the challenges above by answering two research questions, considered as two sub-questions of the research question **RQ3** identified in Chapter 1:

RQ3-1: The first question is whether Graph Neural Networks (GNN), a deep learning paradigm, is more powerful than standard and simple ML algorithms (*i.e.* not relying on neural networks) to predict the feasibility of TSN networks with unseen topologies? This is an important question since we do not have to re-train the GNN model for each new topology only if GNN can generalize to heterogeneous topologies and traffic patterns beyond the trained patterns. A practical application is that GNN can be packaged as a pre-trained model in a DSE-assistant toolset.

RQ3-2: Does the GNN model achieve sufficient prediction accuracy to have a meaningful contribution in the early stage of DSE? That also means we have to investigate how to maximize the performance of GNN, for instance, using the ensemble technique. Designers can obtain the results of DSE in hours if the GNN model proves to be a viable verification technique, *i.e.* it can be considered as an alternative to schedulability analysis which takes weeks to derive the feasibility of a large candidate configuration set.

4.1.3 Contributions of the chapter

Through experiments on different realistic TSN networks, we examine the ability of conventional ML algorithms to adapt to unseen topologies. Our first contribution in this chapter is to unveil the main limitation of these algorithms: they do not offer consistent performance when the topologies change compared to the training data. In other words, they are not able to generalize outside the training sets.

This study overcomes the limitation of conventional ML algorithms in feasibility prediction with a novel Graph Neural Network (GNN) model containing neural networks. It is based on the general GNN model used in [44]. Our proposed deep learning model bypasses the manual feature selection step by taking as inputs the raw data from TSN networks. It encodes the network as a graph where physical and logical components in the network (flows, links, queues) are corresponding nodes in the graph. Each node has an embedding vector representing its synthetic features, while a global attribute vector represents the whole graph. Through multiple message-passing iterations, the representative vectors can collect and aggregate complex relations among nodes in the graph. Finally, the global attribute is used by a simple neural network to predict the feasibility of the configuration.

The experiment results prove the ability of the proposed GNN model to generalize beyond the training set, as it can predict the feasibility of TSN networks containing topologies dissimilar from the training data. The GNN model also performs well on networks with totally new traffic characteristics in the testing sets. In addition, GNN automates the feature engineering process, accordingly, it eases the dependence on domain expertise to choose the right features. To the best of our knowledge, this study is the first GNN model to address the verification with schedulability analysis, a major problem in the development of real-time systems.

We note that there are several applications of GNN in networking, for instance, to evaluate the performance of communication networks in [15,76]. However, GNN has not been applied to verify hard deadline constraints, *i.e.* schedulability analysis, as in our study. In the previous chapter, we have considered four scheduling solutions: FIFO, Manual, CP8, and Preshaping. This chapter focuses solely on priority allocation (CP8) as it is a fundamental QoS mechanism. Precisely, the proposed GNN model predicts whether a priority allocation algorithm can derive a feasible scheduling solution. The priority allocation algorithm, described in § 4.2.1, can effectively provide near-optimal solutions in many contexts. However, it requires the execution of multiple schedulability analyses, eight on average, which causes a bottleneck in DSE and thus requires a fast approximator. Although the problem is still in the realm of feasibility analysis, it actually addresses a broader field and thus implies a higher degree of difficulty.

Another main contribution of this chapter is to address the uncertainty of neural networks, which leads to unpredictability in the performance of GNN, and apply the ensemble technique to mitigate the problem. Indeed, GNN models trained with different initializations will not have identical performance. Subsequently, given a set of GNN models, there are possibilities that most of the models correctly predict the feasibility of a configuration while some models fail silently. It is a major risk of applying a neural network-based model in the industry. In this chapter, we apply an ensemble of 32 individual GNN models to mitigate the risk of a single failure, as it can be overridden by a majority of correct predictions. The experimental results show that the ensemble achieves more consistent performance on 13 testing sets, which is very close to the average prediction accuracies of individual GNN models. We further explore two ensemble techniques: the simplest one with majority voting, and a more sophisticated technique with Gradient Boosting Tree. Although these are opposite in the complexity spectrum, the performance does not change significantly and thus majority voting is preferred due to its simplicity.

4.1.4 Organisation of the paper

The rest of this chapter is organized as follows. In Section 4.2, we introduce the system model including topology and traffic patterns for the generation of TSN configurations. Section 4.3 summarizes the general GNN model presented in previous literature and the specialized GNN model for TSN networks. Section 4.4 is the experimental setup and Section 4.5 analyzes the experimental results obtained with an ensemble of 13 individual GNN models. Related works about ML and GNN applications in networking are recapped in Section 5.7. Section 5.8 is a conclusion of the study and possible future research directions.

4.2 System model

4.2.1 Configuring the stream priorities

This chapter focuses on priority allocation to reduce interferences between traffic flows. That means finding a feasible configuration is equivalent to finding a feasible priority

allocation solution, aka priority assignment. The proposed GNN model predicts the outcome of a priority allocation algorithm, *i.e.* whether the algorithm will derive a feasible priority assignment with a chosen schedulability analysis. From the ML perspective, the algorithm provides labels for TSN configurations in the training set, therefore, the GNN model is also appropriate with any priority assignment algorithm as long as the labels can be derived.

Optimal Priority Assignment (OPA) is the most popular priority allocation algorithm, which initially offers the optimal solution for mono-processor systems [83]. Its optimality is also proven for many schedulability analyses [23, 102], particularly periodic or sporadic traffic flows on AFDX network [24]. We note that the schedulability analysis in our context is the Worst-Case Traversal Time (WCTT) analysis based on Network Calculus (NC), which is not compatible with OPA. In other words, OPA is not optimal as the schedulability analysis here implements "traffic grouping" optimization [25] that could violate an assumption of OPA to make it optimal. Although OPA is not actually optimal with the NC-based schedulability analysis, it is still very good in searching for the feasible priority assignment.

In the experiments, we use the Concise Priorities (CP) algorithm provided in the RTaW-Pegase software. It is a variant of OPA, while the difference between the two algorithms is not relevant to our study. Concise Priorities sharply reduces the time consumption of OPA, however, it still bases on the trial and error approach which requires multiple times of verification whether a configuration is feasible. In particular, computing WCTT by the precise analysis can lead to an exponential computational time. To sum up, a vast amount of candidate TSN configurations may take weeks to search for priority assignments, but only a handful of them are considered in the final decision. Thus the design space exploration calls for a swift prediction of whether there exist feasible priority assignments for the configurations before applying CP to the best candidates.

4.2.2 Topology and traffic characteristics

In the experiments, we need to train the GNN model with a set of artificial TSN networks. They are generated with different topologies following the patterns below:

- Each network has a central switch, connected to two to five switches. The network

therefore has a star-shaped topology, aka center-based topology.

- For each switch, there are two to five end-nodes that connect to it.
- The transmission speeds of links in the network are randomly chosen of either 100 or 1000Mbit/s. They are the most common link speeds in real-time networks, for instance, electronic systems in the automotive industry use 100BASE-T1 or 1000BASE-T1.

Traffic characteristics of TSN configurations in the training set follow the patterns in Table 4.1. They resemble common automotive traffic that is also used in previous studies [27, 103, 104]. Each flow in the network has one to three receivers that are chosen randomly. In the experiments, there are 13 distinct testing sets. Compared to the training set, their topology and traffic patterns can be similar or different, which is presented in detail in § 4.4.1.

4.3 Predicting network feasibility with graph neural network

4.3.1 GNN: a promising tool for networking

ML studies have been shifted from conventional algorithms [85], such as k-Nearest Neighbors and Decision Tree, to more advanced neural network-based deep learning (DL) approaches [55]. The reason is due to the ability of deep learning to extract features from raw data in high dimensional space. Accordingly, it eliminates human intervention in manual feature selection and leads to a reduction of time and effort in the development of ML applications. Many deep learning models have been proposed, which can be grouped into four principal paradigms according to [44]:

- *Fully-conntected neural network*: serves general purposes that are similar to conventional ML algorithms. Inputs must be in ordered features.
- *Convolutional neural network*: works with images as object detection and segmentation. Inputs (pixels an image) have spatial relations.

Table 4.1: Topology and traffic patterns for TSN configurations in the training set. Frame size is the payload only. Testing sets may follow different topology and traffic patterns. The requirement is that all flows must satisfy their deadline constraints.

Command & Control	• from 50 to 150 byte frames, with a step of 10 bytes • periods: one frame each [10,20]ms • deadlines constraints: 20ms • proportion: 20/100
Audio Streams	• from 300 to 500 byte frames, with a step of 100 bytes • periods: one frame each 1ms • deadlines equal to 90% of periods • proportion: 20/100
Video Streams	• either 1000 or 1500 byte frames with burst size in [15, 30] • period: one frame every 30ms • deadlines equal to 90% of periods • proportion: 30/100
Best effort Streams	• from 100 to 1500 byte frames, with a step of 100 bytes • period randomly chosen in [100, 200, 250, 500, 750, 1000]ms • deadlines equal to 90% of periods • proportion: 30/100

- *Recurrent neural network*: exploits data series, for instance, LSTM to predict sequences in natural language processing. Inputs in a sequence have temporal relations.

- *Graph neural network (GNN)*: models data in graph structure. There are no spatial or temporal relations among components in a graph, however, they are interconnected and features cannot be extracted in order.

A computer network is fundamentally a graph that contains nodes and switches connected by links. A TSN configuration can also be considered as a graph made up of

primary components: queues in switches, links between switches and nodes, and flows delivered from nodes to nodes through links and switches. In previous chapters, we have manually extracted features from the configurations and use them to predict the feasibility, but the result cannot be generalized to networks with different shapes. The reason is that, while choosing some features, we ignore raw data from the configuration that contains rich and useful information. On the other hand, GNN is a promising candidate to model networks: it is tailored to capture the complex relationships of entities in structured data, thus it can discover meaningful rules that compose the network properties. Therefore, in this chapter we apply GNN to model TSN configurations by deriving synthesis features directly from raw data to predict the feasibility. Inputs of GNN include characteristics of primary components (flows, links, queues) as well as their connections defined by routing.

Since GNN is a standard deep learning paradigm, many concrete models have been proposed that are different in notations, extract structures and implementation methods. In this study, we select the general GNN model proposed by [44] due to its simplicity in the learning algorithm. § 4.3.2 is a summary of the GNN model, and it is specialized for TSN configurations in § 4.3.4.

4.3.2 GNN model structure

A TSN configuration is a data structured that contains characteristics of physical and logical components in the network:

- *flows*, charaterized by maximum packet payload, transmission pattern (period, and burstiness for video frames), and deadline constraints.
- *links*, characterized by transmission speed.
- *queues*, at the egress ports of switches.

We also define topology to connect links with egress queues, and routing to connects flows with other components. A graph model is used to capture the characteristics of components in the configuration as well as their connections. These are translated into nodes in the graph and connections between them, depicted in an overview of GNN in Figure 4.1. Comparing to prior works that also apply GNN to real-time communication

systems [15,64], a novelty of our GNN model is that we only have a single type of node in the graph for all components. This uniqueness makes the graph much simpler, facilitates the development of the model and offers better performance.[1]

The graph model, located at the second block of Figure 4.1, contains nodes and connections between them. Each node is materialized via an *embedding* vector, which is initialized by raw data characterizing the network component it represents. The embeddings, through the learning phase described in § 4.3.3), evolve the values to actually embed more complex information from the graph. A global attribute represented the entire graph is used to predict the feasibility of the corresponding configuration once the learning stops.

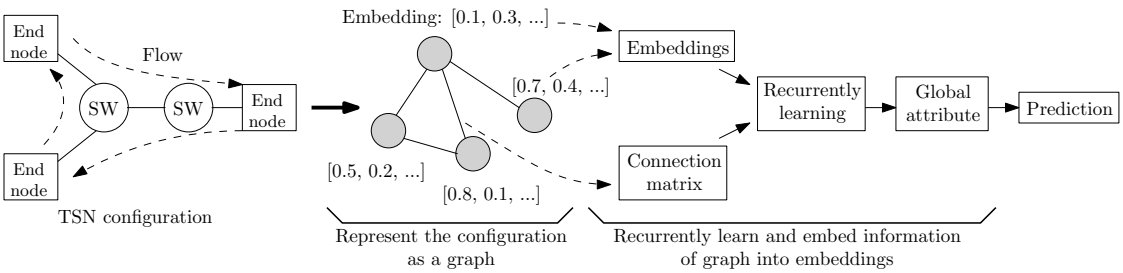


Figure 4.1: The Graph Neural Network (GNN) model takes raw data from a TSN configuration and encodes the configuration as a graph. The components in the network (flows, links, queues) are nodes in the graph, filled in gray color. Each node is represented by an initial embedding vector that captures the characteristics of the corresponding component. For instance, the packet payload of a flow can be scaled in the range [0, 1]. The embeddings are updated through recurrently learning to exploit complex relations in the network. A global attribute vector representing the whole graph is aggregated from embeddings of all nodes. It is used by a simple neural network dedicated to feasibility prediction.

It must be emphasized that in this study, GNN is used to predict whether there is a feasible priority allocation for a TSN configuration, therefore, priority levels are not encoded in the graph. In other words, priority is excluded from raw data of flows. The existence of a feasible solution for priority assignment is used by GNN to learn how to predict the feasibility of the configuration, as in any supervised ML algorithm.

Although this study only focuses on priority allocation, a similar GNN model can be applied to other QoS mechanisms such as Time-Aware Shaper (TAS), Frame Preemption

[1]It is worth noting that we also experiment with a more fine-grained data structure including waiting queues in all end-nodes. However, such detailed data structure does not drive the improvement in the final results. That is because the root of delays is the interference between flows that mostly happens at the egress ports. We also eliminate links from the end-nodes to their first switch since their load is much less than inter-switch links that establish the backbone of the TSN network. Some QoS mechanisms, for instance, end-to-end stream shaping with CBS [26], could take advantage of the data structure with more details.

or Credit-Based Shaper (CBS). Since they are based on predefined priority allocation, the GNN model introduced in this study can be expanded by using priority levels as additional information. Another possible extension would be using multiple queues in the graph, such that each queue carries buffers of a priority level.

4.3.3 Computational steps in a general GNN model

Graph G in the general GNN model is materialized as a collection of embedding vectors (v_i, e_k, u), where v_i: embedding of vertex i aka node i, e_k: embedding of unidirectional edge k, and u: global attribute representing the entire graph. The embeddings are updated through the learning phase including multiple loops provided by Equation 4.1

1. *Update edges*: the embedding of each edge k is updated with a function f_e that takes as inputs the current embedding of e_k, destination node v_{r_k}, source node v_{s_k}, and the global attribute u. The function f_e is identical for all edges in the graph.

2. *Update nodes*: the embedding of node i is updated with a function f_v and its inputs are the current embedding of v_i, the average of incomming edges, and the global attribute u. The function f_v is also shared among edges in the graph.

3. *Update global attribute*: the global attribute u is updated with a function f_u, its inputs includes the current value u, the average of embeddings of all edges as well as all node.

Functions f_v, f_e and f_u are learnable, which are neural networks in this study. The updating process is repeated and only stops when all embeddings converge. This recurrent computational step is crucial to guarantee that GNN can discover complex relationships in

the graph.

$$
\begin{aligned}
&\text{Update edges:} && e'_k = f_e(e_k, v_{r_k}, v_{s_k}, u) \\
&\text{Update nodes:} && e'_i = \sum_{k:r_i=i} e'_k \\
& && v'_i = f_v(\bar{e}'_i, v_i, u) \\
&\text{Update global attribute:} && e' = \sum_k e'_k \\
& && v' = \sum_i v'_i \\
& && u' = f_u(\bar{e}', \bar{v}', u) \\
& && e_k = e'_k; \quad v_j = v'_j; \quad u = u'
\end{aligned}
\tag{4.1}
$$

In Equation 4.1, the symbol $\bar{x}$ denotes an average of embeddings x. For instance, $\bar{e}$ is the average of all edge embeddings. In the following, a specialization of the general GNN model and Equation 4.1 will be applied to evaluate the feasibility of TSN configurations.

4.3.4 Encoding of TSN configurations as a graph

Based on the general GNN model proposed by [44] and summarized in § 4.3.3, as well as the study in [15] that uses GNN in TCP communications, we introduce our GNN model specialized for feasibility prediction of TSN configurations. As depicted in Figure 4.2, the core of our model is a graph that encodes and captures important information about the network. For each TSN configuration, we consider three types of components and each component is modeled by a node with a corresponding embedding vector as follows.

- *flow*: each flow in the configuration is modeled as a corresponding node in the graph. There is no difference between unicast or multicast flow in this step as it is only represented by one node. However, the difference can be found when modeling the connections in the graphs, as the node represented a multicast flow must connect to more than one end-devices. The node embedding is initialized by the information of the flow: packet payload, transmission period, burstiness with video streams, and deadline constraint, while priority level if existed is not included. Since the embedding vector size can be larger than four, the rest of the vector is initialized

by value 0. This technique is also used when initializing embeddings of nodes corresponding to links and queues.

- *link*: Since modern Ethernet switches are full-duplex, we consider each Ethernet link as two separate uni-directional links and therefore modeled by two distinct nodes in the graph. Each node connects to a corresponding queue node. Both nodes embeddings are initialized by the link speed, and the difference between them can only be observed when modeling the connections in the graph.

- *waiting queues*: we consider a switch as a sub-network of ingress and egress ports. The switching delays when transmitting a packet from an ingress port to an egress port are eliminated since they are only a fraction of the end-to-end communication delays. To be precise, the end-to-end delays are in a range of ms but the switching delay [16] is set to maximum 1.3μs in our experiments. As common in most critical systems, the queues at the egress ports are assumed to have enough capacity for the traffic and there is no packet loss. Based on these conditions, waiting queues at the egress ports are modeled by corresponding nodes in the graph. Their initial embeddings are set to 0 as an indication of no message pending.

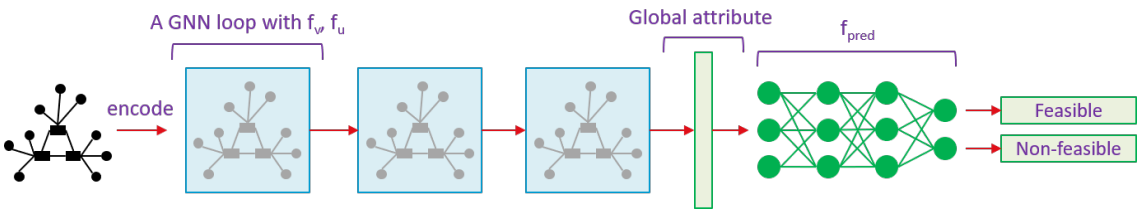


Figure 4.2: We extend Figure 4.1 to give more details on neural networks that constitute the GNN model. Embeddings of nodes and the global attribute of the graph (encoding a TSN configuration) are updated through three loops with two neural networks f_v and f_u, respectively. Finally, the neural network f_{pred} uses the global attribute to predict the feasibility of the configuration.

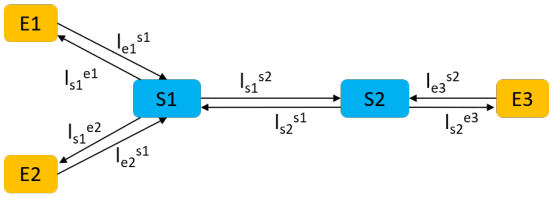
The global attribute representing the entire graph is also initialized by value 0. We model the connections between components in the TSN configuration as an *connection matrix* containing edges between nodes in the corresponding graph. The connection matrix keeps identical over computational steps since the topology of the network is predefined before the training startup. Some values in the matrix are set to non-zero while the rest are 0, as defined in the following.

- *queue-link* edge: if there is a physical connection of a link to an egress port, defined by the topology of the network, an edge between two nodes corresponding to this queue-link pair exists and indicating by value 1 in the connection matrix.

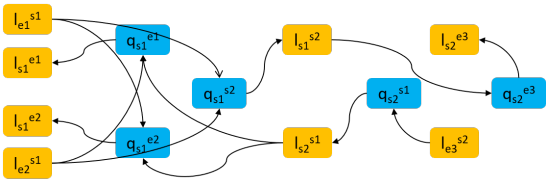
- *queue-flow* edge: if a flow actually goes through an egress port, defined by the routing, value 1 is set to indicate the existence of an edge between two corresponding nodes. This approach also indicates the direction of a uni-directional link, since a full-duplex Ethernet link is modeled by two identical nodes in the graph without any information about the direction in initial embeddings.

- *link-flow* edge: if a flow goes through a link, an edge between two corresponding nodes exists and gets value 1. In particular, if the link connects to the source of the flow, a value 2 is set for this edge to avoid the ambiguity of the flow direction.

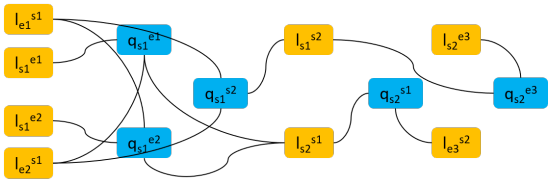
Figure 4.3 depicts the process to model a TSN configuration. The topology of an example TSN configuration is shown in Figure 4.3(a). In Figure 4.3(b), links and queues at the egress port in the topology are modeled as nodes in a graph, while edges between nodes are directional. However, directions of edges are removed in Figure 4.3(c). The reason is that, our GNN model does not only model the explicit structure of a configuration into a graph but also aims to embed the physical relationships among components of the network. If there is a directional edge that projects from a queue node to a link node, then only the link node can "aggregate" information from the queue node. However, in the configuration, the link and the queue mutually affect to each other: if the link speed is slow, then there will be more messages in the queues waiting for sending; on the other hand, the link load increases when there are more packets in the connected queue. In the next three figures, the flows $f1, f2, f3$ are modeled by nodes in the graphs. The connection matrix also add edges that connect these flow nodes to connected queues' and links' nodes, following the predefined routing. Although the graph contains only non directed edges, the direction of any flow is not ambiguous due to the notation of the source in the connection matrix. It should be noted that the entire TSN configuration is actually defined before the modeling process, these six figures only show the separate steps for modeling flows with the sake of simplicity to follow.



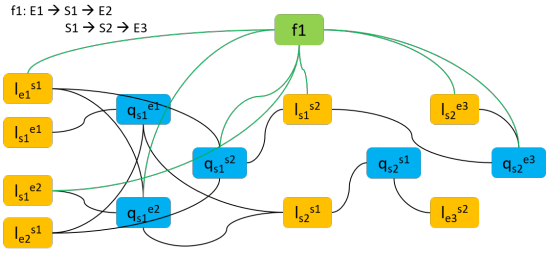
((a)) Sample topology with 3 end-nodes and 2 switches. l_{s1}^{e1} denotes the link from S1 to E1.



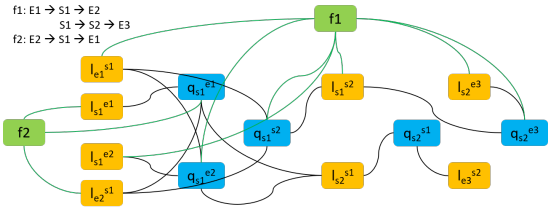
((b)) Modelling of links and queues. q_{s1}^{e1} denotes the waiting queue of S1 on the egress port connected to end-node E1.



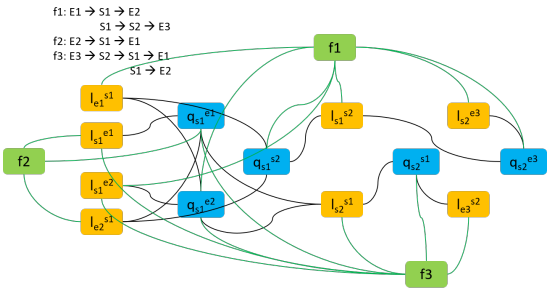
((c)) Unidirectional edges.



((d)) Adding multicast flow $f1$.



((e)) Adding unicast flow $f2$.



((f)) Adding multicast flow $f3$.

Figure 4.3: Details steps to model a TSN configuration as a graph in the GNN model. We use several colors in the figures: yellow for end-nodes (only in (a)) and links, blue for switches (only in (a)) and queues, green for flows. For the sake of simplicity, the encoding process is separated into steps, however, only the final graph is used in the GNN model.

4.3.5 The GNN model for TSN and associated learning algorithm

In this section, we present our GNN model specialized for TSN configuration based on the general GNN model summarize in § 4.3.3. A part of our GNN model, how to model a TSN configuration as a graph, has been introduced in 4.3.4. In the rest of this chapter, we exclusively use the term *GNN model* for our GNN model, except otherwise said.

There are three functions used in the general GNN model to learn and update the embeddings of edges, vertex as well as the global attribute. However, our GNN model only has one type of node in the graph and thus we eliminate the function to update edges. Accordingly, the GNN model uses two functions: f_n to update embeddings of nodes and

f_u to update the global attribute. The symbol u in this context is adopted from the general GNN model proposed by [44], the reader can shift from this reference to our study without any difficulty. We note that although the global attribute embeds useful information about the entire graph, it does not explicitly show the feasibility. Therefore, we use another function f_{pred} that takes the global attributes as input and prediction the feasibility of the corresponding TSN configuration. The function f_{pred} works effectively when the global attribute has finished iterative learning steps and embeds complex relations in the configuration. All functions in our GNN model are deep neural networks with three layers.

In the first step, the GNN model encodes a given TSN configuration as a graph. Each graph contains node embeddings and the global attribute, together with the connection matrix that encodes edges between nodes. The initialization of node embeddings, the global attribute and the connection matrix are explained in § 4.3.4. Algorithm 1 expresses the computation steps to update the GNN model. All embeddings in the graph are assumpted to have the same size l to facilitate implementation of the model. The GNN model, based on the connection matrix of the graph, knows the neighbors of any nodes. An initial embedding v_i of node i is updated by aggregating the embedding of all neighbor nodes to form a new vector v'_i of size l. We then concatenate the aggregated vector v'_i with the current embedding v_i and the global attribute u to be an expansion of v'_i but size $3l$. The neural network f_v takes v'_i as the input, line 7 in Algorithm 1, and the output is an update of v'_i with size l, the new embedding of node i. All nodes in the graph share the same neural network f_v and the operation to update all nodes need to finish before the next step to update the global attribute. We note that the aggregation in our context is element-wise sum (line 7 of Algorithm 1) that is different from the element-wise average in the general GNN model, however, this change has improved the overall performance during the development of our model.

All node embeddings, after being updated, are averaged to be a vector $\bar{v}'$ and concatenated with the current global attribute u to form a new vector of size $2l$. The neural network f_u takes this vector as the input and the output is the update of the global attribute u' with size l, as in lines 9 and 10 in Algorithm 1. Finishing this step, we assign v'_i to be the new node embedding v_i and u' to be the new global attribute u (line 11 in Algorithm 1), and the first loop of the algorithm is terminated. The iteration can happen depending on the

number of loops in the experimental setup, but the computational steps remain identical for all loops.

Algorithm 1 Graph neural network (GNN)

0: **Initialize:** f_v, f_u, f_{pred}, label y
0: **for** each graph **do**
0: **Input:** Initialize all v_i, u, connection matrix
0: **for** each loop **do**
0: **for** each node **do**
0: $v'_i = \sum_{k:neighbor} v_k$ {any nodes connected to node i}
0: $v'_i = f_v(v_i, v'_i, u)$
0: **end for**
0: $v' = \sum_{\text{all nodes}} v'_i$
0: $u' = f_u(\bar{v}', u)$
0: Update $v_i = v'_i, u = u'$
0: **end for**
0: Feasibility prediction: $\hat{y} = f_{pred}(u)$
0: If training, at the end of each mini-batch, update f_v, f_u, f_{pred} based on $loss(\hat{y}, y)$
0: **end for**
0: **return** f_v, f_u, f_{pred} =0

In our experiments, the GNN model is updated with multiple loops. When the final loop finishes, the neural network f_{pred} takes the updated global attribute of size l as the input and results in an output vector of size 2. This vector is the predicted value corresponding to feasible and non-feasible, as in line 13 of Algorithm 1. The given TSN configuration is predicted to be feasible if the first value is larger than the second, otherwise the model returns non-feasible. It is worth noting that the order of the first and the second value with respect to feasible and non-feasible is not critical, however, it must be chosen and fixed before any computational step. In our study, we always apply the order of [feasible, non-feasible] to the output vector.

The training stage is organized with a number of training samples processed in parallel, *i.e.* a mini-batch, to accelerate the learning and increase the generalization of patterns recognized. After each mini-batch, the GNN model compares the predicted values, as in the output vector of size 2, to the true value represent the feasibility of the TSN configuration. The output vector is in the form of one-hot encoding: values [1, 0] for feasible and values [0, 1] for non-feasible. The error in the prediction, aka the *loss*, is computed over all samples in the mini-batch. Since neural networks in the GNN model are continuous

functions, the loss can be backpropagated through gradients and all layers in the neural networks are adjusted toward reducing the prediction error (line 14 of Algorithm 1).

It is worth emphasizing that the GNN model can only be successful in feasibility prediction with enough supporting information. The global attribute, after being updated, should be able to express the interference between flows in the network, which causes delays on links and egress ports. This capability is achieved through the *message-passing* approach: embeddings of nodes in the graph, *i.e.* information about nodes, is passed as messages to neighbor nodes. Gradually, information is prOPAgated to the entire graph and aggregated by the global attribute. Iteration of the learning loop can amplify this effect, however with a cost of gradient vanishing [41] that makes the neural network more difficult to adjust, since the gradient corresponding to prediction error can vanish before backpropagating to all layers.

4.3.6 Ensembles of graph neural networks

Due to the difference in parameter initialization, neural networks with the same structure can also return different results as an unavoidable phenomenon [105]. In our GNN model, made up of three neural networks, such variance can lead to poor performance on some networks. In the worse case, it leads to wrong predictions and the model fails silently without users' acknowledge. We mitigate this risk by adopting the *ensemble* approach, in which the final prediction is a consensus of individual GNN models. In this chapter, two techniques are selected, the first is the simplest ensemble technique while the second conducts the ensemble with the help of an intermediate ML algorithm.

- Ensembles with majority voting: as the name suggests, individual GNN models reach the consensus through a majority voting.
- Ensembles with Gradient Tree Boosting (GTB): GTB is an excellent ML algorithm [85], which shows the best classification results among the other 13 algorithms on 165 data sets [106]. In the ensemble, the predictions of individual GNN models are inputs of GTB, and its output is the final prediction of the model.

To further reduce the variance of the individual GNN models, weight normalization technique is also applied. This technique is introduced in [107] and proved to enhance the

consensus among individual models [108]. We note that the risk of a single model failure reduces proportionally to the number of members in the ensemble model, however with a burden of more computation time. For the sake of simplicity, we do not further explore this trade-off and choose the number of individual models to be 32, as done in [108]. This value balances the run-time overhead and the prediction robustness during the development of the model.

4.4 Experimental setup

The experiment setups in the following sections are designed with the purpose of answering researcher questions in Section 5.1. A training set, evaluation set and several testing sets are needed to conduct experiments. The training set contains networks with center-based topologies, aka star-shaped topologies, which is common in the industry. The evaluation set is generated with the identical setup as the training set, however, it has a different set of networks. While the training set is used to train the GNN model to discover useful patterns, the evaluation set is important to evaluate whether the training set is effective and the model can predict the feasibility of TSN configurations. Through the evaluation, the best trained GNN model is chosen for the testing step. Testing sets contain networks whose distributions of topologies and traffic patterns range from similar to dissimilar from the training set, and some practical networks are expected. The diversity of testing sets is crucial to assess the GNN model, after being trained with a particular class of topology and traffic, can generalize the prediction ability to arbitrary configurations that do not appear in the training set.

In the following, the generation of data sets is described in §4.4.1, and the setup of the GNN model is in §4.4.2.

4.4.1 Data sets

Training and evaluation sets

The training set is generated following the system model, *i.e.* topology and traffic patterns, in § 4.2.2. The system model describes ranges for the number of switches, end-nodes and

number of links between them. Random topologies are generated by obeying these ranges and all have star shapes.

After generating a random topology, we use it to generate one or some TSN configurations based on traffic patterns in § 4.2.2. It raises the question of how many flows should be generated for a configuration, in order words, the maximum number of flows a topology can handle with the given traffic patterns? We choose this value by the following process. The number of flows starts from 20 and increases over steps. During each step, we generate 100 configurations and determine the proportion of overloaded configurations, *i.e.* configurations that the load on one or more links exceeds 100%. When the overloaded proportion reaches 90%, the maximum number of flows is set. The threshold is chosen since above 90%, most non-overloaded configurations are non-feasible as in the empirical observation. This method guarantees that we can choose the number of flows for a configuration that is balanced in a range, from easy to predict as feasible (small number of flows) to hard to predict (moderate number of flows) then easy to predict as non-feasible (large number of flows).

We generate 1000 random star-shaped topologies for the training set. For each topology, 10 non-overloaded configurations are generated (overloaded configurations are discarded). The number of flows is randomly chosen between 20 and the maximum value described above. According, there are 10,000 unique TSN configurations in the training set. On the other hand, the evaluation set has 100 random topologies with 1000 TSN configurations, following the same system model in § 4.2.2.

Testing sets

The testing sets are diverse in terms of topologies and traffic patterns in order to scrutinize the GNN model of whether it generalizes to practical applications. Figure 4.4 shows the topologies used in the testing sets. Some testing sets use standard network architectures: star shape topologies similar to the training set, ring, backbone or ”dumbbell” topologies (used at Daimler [21]). Other testing sets use topologies provided by automotive OEMs: Renault topology [82], FACE topology also from Renault group [103], as well as Volvo topology [104]. In addition, some OEM’s topologies are modified to provide more test cases and further assess the generalization of the GNN model.

Table 4.2: Summary of training, evaluation and testing sets.

Data set	**Topology**	**Traffic statistic**
Training set	Random star-shaped topologies	Training traffic statistic
Evaluation set	Random star-shaped topologies	Training traffic statistic
Testing set 1	Random star-shaped topologies	Training traffic statistic
Testing set 2	Random star-shaped topologies	**Modified** training traffic statistic
Testing set 3	**Modified** random star-shaped topology	Training traffic statistic
Testing set 4	FACE topology	Training traffic statistic
Testing set 5	FACE topology	**Modified** training traffic statistic
Testing set 6	Renault topology	Training traffic statistic
Testing set 7	**Modified** Renault topology	Training traffic statistic
Testing set 8	**Modified** Renault topology	**Modified** training traffic statistic
Testing set 9	**Extended** Renault topology	Training traffic statistic
Testing set 10	Ring topology	Training traffic statistic
Testing set 11	Backbone topology	Training traffic statistic
Testing set 12	Small topology	Training traffic statistic
Testing set 13	Volvo topology	**New** traffic statistic

Table 4.2 summarizes all topologies and traffic patterns used in the testing sets, in a comparison with the training set the evaluation set. Testing set 1 and 2 use random *Random star-shape topologies* similar to the training set and evaluation set. *Training traffic statistics* are the traffic patterns described in Table 4.1 and used by the training set. Each testing set has 1000 random TSN configurations. We also judge how different the testing sets are from the training set and list them in order, as in Table 4.2 when testing set 13 contains topology and traffic patterns unseen in the training set. With respect to the training set, some testing sets have noteworthy changes that are listed below:

- *Testing set 2*: we modify the training traffic statistics by changing the deadlines of best effort streams from 20ms to 10% of periods. The proportion of stream types, critical-audio-video-BE, also changes from 20-20-30-30 as in the training traffic statistics to 10-10-50-50 in this testing set. In other words, the proportion of video streams and best effort streams increase. Since these types of streams have longer deadlines than critical streams and audio streams, configurations are easier to schedule and the feasibility distribution of the test set will shift from the training set.

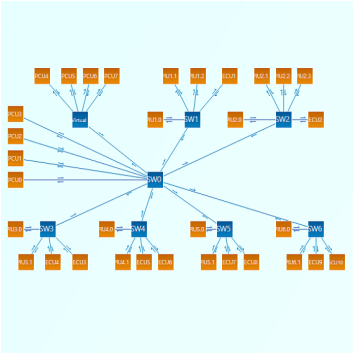
- *Testing set 3*: random centered based topologies are modified by setting the speed of

all links to 100Mbps. While topologies in the training set are mixed of 100Mbps and 1000Mbps links, this modification leads to the violation of timing constraints with fewer flows.

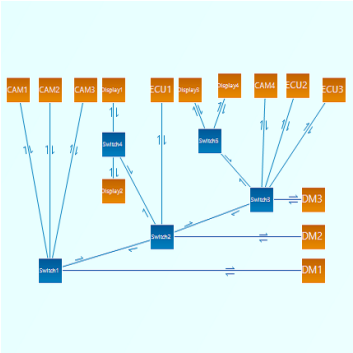
- *Testing set 4*: although FACE topology also has the same star shape, it is much larger than topologies in the training set with more switches and ECUs.
- *Testing set 5*: BE streams increase deadlines from 20ms to 10% of periods, which provide the same effect as the modification in Testing set 2.
- *Testing set 7*: We modify the inter-switch link SW1-SW2 and SW2-SW3 in the Renault topology by increasing their speed from 100Mbps to 1000Mbps, which leads to more feasible configurations.
- *Testing set 8*: We reuse the modified Renault topology in Testing set 7, however, the proportion of stream types in the traffic statistics, critical-audio-video-BE streams, changes to 10-10-50-10 instead of 20-20-30-30. The increase of video streams requires more bandwidth, thus this testing will have fewer feasible configurations than Testing set 7.
- *Testing set 9*: We extend the original Renault topology by adding one more switch and three new end-nodes. The extended topology is more similar to the backbone topology than the center-based topology.
- *Testing set 13*: The Volvo topology uses redundant paths and the traffic patterns are completely different from the training set.

4.4.2 Parameters of the GNN model

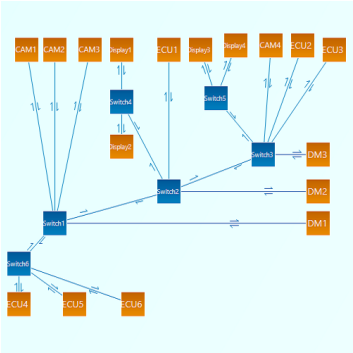
Table 4.3 shows the parameters used to set up the GNN model. These values are chosen considering the practical techniques recommended in deep learning [41,109] as well as our extensive incremental experiments. The GNN model encodes TSN configurations as graphs, materialized by the initial embeddings of nodes and the connection matrix. The size of embedding vectors is 16, and all embeddings are placed into the embedding matrix. The maximum number of nodes in the graph is set to 500. In case there are fewer than 500



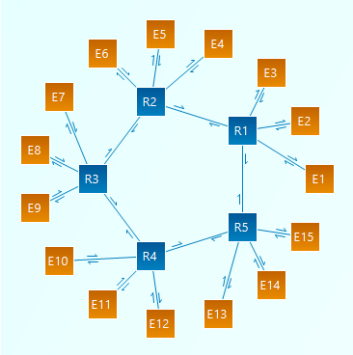
((a)) FACE topology (testing sets 4, 5)



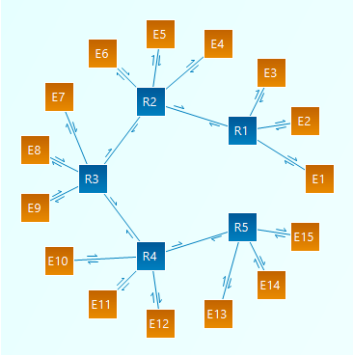
((b)) Renault topology (testing sets 6, 7, 8)



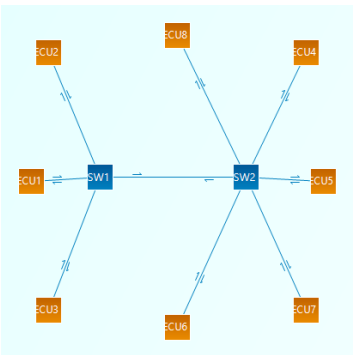
((c)) Extended Renault topology (testing set 9)



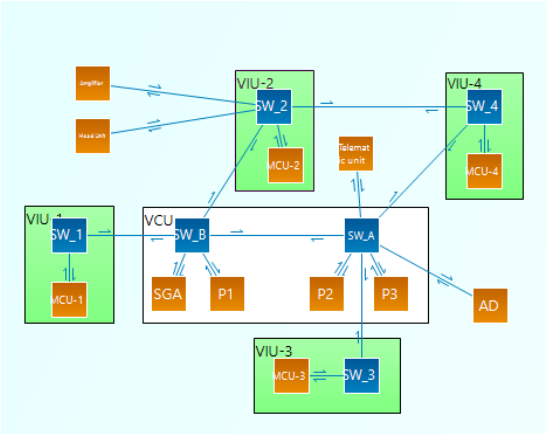
((d)) Ring topology (testing set 10)



((e)) Backbone topology (testing set 11)



((f)) Small topology (testing set 12)



((g)) Volvo topology (testing set 13)

Figure 4.4: Topologies used in the testing sets (captures from RTaW-Pegase).

notes in the graph, we keep the matrices in a fixed size of 500 by padding them with 0s. This is a common technique required in deep learning to process the graphs in parallel and accelerate the training progress.

The structure of three neural networks in the GNN model, f_v, f_u and f_{pred} are identical. Each neural network has an input layer, follows by two hidden layers and finally an output layer. The hidden layers have the same size of 256. Size of the input and output layers are specific to the purpose of each function, detailed in Table 4.3. The GNN model is trained over 500 iterations and a mini-batch of 32 samples is randomly chosen with replacement (from the training set) for each iteration.

The model processes 32 graphs in a mini-batch in parallel. The embeddings of nodes and the global attribute are updated over 3 loops before using for the feasibility prediction. The neural networks in the model are updated with the Adam optimization algorithm, available as an optimizer in TensorFlow libraries [46]. The learning rate is fixed at a value of 0.00001 $\times$ mini-batch size. After every 50 iterations, we pause the training and use the evaluation set to evaluate the prediction accuracy of the model. The best model is saved to be used in the ensemble of GNNs.

We note that in our model, neural networks use the identity activation function in all layers. Although this function is less expressive than non-linear activation functions such as sigmoid, tanh or ReLU, in our experiments it helps to alleviate the gradient vanishing phenomenon and make our model converge faster. Common deep learning techniques like dropout and batch normalization are not used for the sake of simplicity. Besides, the weight normalization technique is applied to reduce the variance in the results of neural networks, as mentioned in § 4.3.6.

4.5 Experimental results

All TSN configurations in the training set, evaluation set and testing sets are generated with the Java library available in the RTaW-Pegase 3.7.5 [18] software, denoted *Pegase library* for short. It runs on the platform Java JDK-13. For each TSN configuration, the feasibility serves as the label used by the GNN model. It is derived with an algorithm available in Pegase library, Concise Priorities (see §4.2.1). In our study, all experiments

Table 4.3: Summary of parameters of the GNN model

Parameters	Description
Size of embedding vectors	16
Number of nodes in the graph	500
Size of embedding matrix	500×16 (with 0 padding)
Size of connection matrix	500×500 (with 0 padding)
# of neural networks (NNs)	3 (same structure)
Size of NN to update node embeddings f_n	$[16 \times 3, 256, 256, 16]$
Size of NN to update global attribute f_u	$[16 \times 2, 256, 256, 16]$
Size of NN to predict feasibility f_{pred}	[16, 256, 256, 2]
# of training iterations	500
Mini-batch size	32
# of loops to train a graph	3
Learning rate	$0.00001 \times$ mini-batch size
Optimization technique	Adam optimization
# of evaluations	20 (one after each 50 training iterations)
Activation function	identity

are conducted on a single computer with Intel Core i9-9900k CPU (8 cores) and 64GB of RAM.

The GNN model and other procedures needed for the experiments are developed in Python 3.7. The traditional ML algorithms such as Gradient Trees Boosting are provided by the standard ML library, Scikit-learn 0.22 [43]. The GNN model uses the most popular deep learning frame, TensorFlow 2.0 [46], to build and train the neural networks. When using TensorFlow, we deactivate the GPU mode and thus only exECUte the model on CPU. This provides a fair comparison in terms of running time with the Concise Priorities algorithm. In addition, two Python libraries Numpy 1.17.4 and Pandas 0.25.3 are used in data processing when modeling TSN configurations.

4.5.1 RQ3-1: Insight into when ML approaches based on manual feature selection fail

In the previous studies, we have explored the possibility to use ML in predicting the feasibility of TSN networks [1–3]. These studies apply conventional ML algorithms that require ordered features as inputs. Five features are chosen manually by domain experts:

the number of critical flows as well as audio flows and video flows, the maximum load of links in the network, and the Gini index [88]. The final feature measures how unbalanced it is among loads of the links. These features can directly be derived from the statistics of the network, which simplifies the development of ML applications. Promising results have been reports when applying this approach.

However, the drawback of the manual feature selection approach is that the statistics do not provide dynamic information about the topology, the routing, and relations of components in the network. Subsequently, such important information is lost and the ML algorithms cannot generalize to networks with different shapes or traffic patterns.

Our study extends the previous works with the new training set described in § 4.4.1. From the standard ML library Scikit-learn [43], we choose the simplest and the most advanced supervised algorithms:

1. k-Nearest Neighbors (k-NN): this algorithm saves all data in the training set into a multi-dimensional feature space. With an unseen sample, the algorithm calculates the Euclidian distance from the new sample to all training data and finds the k neighbors that have the least distances. The feasibility of the new sample bases on the majority of these k neighbors, whether they are feasible or non-feasible.

2. Gradient Tree Boosting (GTB): in our study, we use the term Gradient Boosting (GB) as a short name of this algorithm. It is an ensemble of trees from the basic decision tree algorithm.

The training set contains random centered-based topologies, which allows the conventional ML algorithms to learn how to reason the feasibility of TSN configurations without belonging to a unique topology as in the previous studies. Then the ML algorithms are tested with testing sets different from the training data in either topology types or traffic statistics or both. Accordingly, this extended work explores the generalization of the manual feature selection approach to arbitrary TSN configurations, which has not been done in previous studies.

Analysis of the prediction accuracy of traditional ML algorithms

The experimental results are shown in Table 4.4 with four techniques over 13 testing sets. The baseline is a naive approach that predicts all configurations to be feasible. We should separate the baseline from random guess, which has the probability of correct prediction 50% while the result of the baseline varies depending on the data sets. Column k-NN is the prediction accuracy of the k-NN algorithm and GB for Gradient Boosting. We highlight in red color the values that have high discrepancies with the rest, *i.e.* testing sets 6, 9, 11 and 13. Testing set 13 uses a novel Volvo topology, while Testing sets 9 and 11 use backbone-like topologies. These topology shapes are very different from the star-shaped topologies in the training set. Identical traffic patterns are used in Testing sets 6 and 7, and the topology in Testing set 6 is slightly modified for Testing set 7 (increasing the speed of two links to 1Gbps). However, the prediction accuracies over the two testing sets are very different, which drops significantly in Testing set 6. These results show the limitation of k-NN, GB and the corresponding manual feature selection approach, which cannot generalize to unseen topologies and thus the performance over testing sets is fragile.

Why traditional ML algorithms and the manual feature selection approach have limited prediction accuracy is intricate, but there are some possible reasons. First, the selected features do not provide enough information to model a configuration. For instance, the number of critical, audio and video flows are not sufficient for an ML model to encode complex relations between components in the network (flows, links, queues). Characteristics of flows such as packet payload or period are also not conveyed to the ML algorithms by chosen features.

Second, features may capture ambiguous information. The Gini index is a feature that measures how unbalanced it is among loads of the links, its effectiveness is proven with a fixed topology. However, its value can change significantly with different topologies in 13 testing sets. The center-based topologies usually have well-balanced loads, while the backbone-like topologies are highly unbalanced. Since the training set possesses only center-based topologies, the Gini index becomes a misleading feature in many cases such as Testing sets 9 and 11.

Another reason is the limitation of traditional ML algorithms. The simple computational methods, Euclidean distance in k-NN and tree splitting in GB, do not have the

capability to discover intricate patterns that lead to the feasibility of TSN configurations. In order to overcome these limitations and provide consistent results over heterogeneous testing sets, we need a novel approach like GNN that is able to 1) use the rich and structural raw data derived directly from the configurations, and 2) leverage more advanced and powerful learning paradigms.

Table 4.4: Prediction accuracies (%) of GNN in a comparison with k-Nearest Neighbors (k-NN), Gradient Boosting (GB) on 13 testing sets. The baseline is the accuracy of a trivial model that always predicts configurations as feasible. Two columns En-MV and En-GB are respectively the accuracies of ensembles with two different techniques: majority voting and Gradient Boosting. The values highlighted in red are the cases conventional ML algorithms have low prediction accuracy.

Data sets	**Baseline**	**k-NN**	**GB**	**GNN**		
				min-mean-max	**En-MV**	**En-GB**
Training set	59.0	91.6	92.9	85.9 – **86.6** – 87.3	86.7	87.7
Evaluation set	59.8	92.6	94.0	84.4 – **86.0** – 85.1	85.2	85.4
Testing set 1	57.9	90.3	91.1	85.6 – **86.7** – 87.6	86.9	87.2
Testing set 2	53.9	92.1	91.4	84.4 – **85.3** – 85.9	85.5	85.3
Testing set 3	47.5	93.7	92.1	82.6 – **88.9** – 90.9	89.9	88.8
Testing set 4	62.2	92.6	93.3	87.2 – **89.2** – 90.5	90.0	88.5
Testing set 5	73.3	90.6	91.9	83.9 – **87.8** – 88.8	88.2	88.5
Testing set 6	36.0	**82.9**	**79.7**	83.9 – **87.9** – 90.0	87.9	89.6
Testing set 7	51.4	94.4	93.1	79.6 – **82.4** – 84.9	82.3	80.7
Testing set 8	41.6	93.4	91.9	84.1 – **89.3** – 91.0	89.9	87.7
Testing set 9	22.7	**68.7**	**66.6**	73.6 – **79.4** – 85.7	79.3	81.6
Testing set 10	39.1	90.7	85.7	85.9 – **88.9** – 90.5	88.5	89.3
Testing set 11	22.0	**73.4**	**70.9**	77.0 – **82.3** – 87.6	82.0	84.0
Testing set 12	48.0	94.4	91.3	82.4 – **87.3** – 90.2	87.1	85.5
Testing set 13	57.8	**85.6**	**68.1**	54.3 – **83.6** – 90.1	89.4	89.8

4.5.2 RQ3-2: Accuracy of GNN models

Here we evaluate the ensemble of GNN models in a comparison with traditional ML algorithms. The ensemble includes 32 individual GNN models that have identical setup provided in Table 4.3. The only difference among the individual models is the initialization of neural networks that constitute the models.

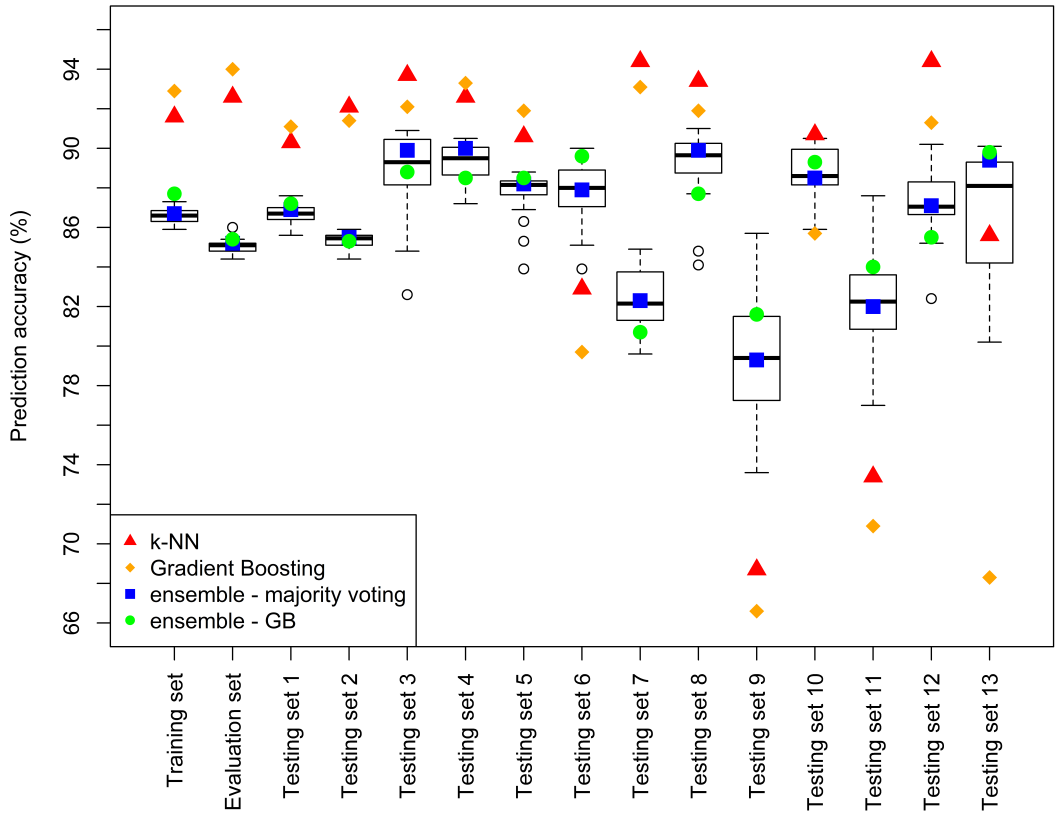


Figure 4.5: Prediction accuracy (%) of k-NN, GB, and GNN with two ensemble techniques on all data sets, while outliers are removed. Boxplots are the variance in the performance of the 32 individual GNN models, which increases from left to right corresponding to how dissimilar the testing sets are from the training set.

The performance of GNN and traditional ML algorithms are represented in Figure 4.5. Box plots show the variance in prediction accuracies of individual GNN models, *i.e.* the robustness of GNNs. Prediction accuracies of the ensemble are represented by blue squares and green dots if applying the majority voting and the GB technique, respectively. Red triangles denoted the accuracies of k-NN and yellow diamonds for the accuracies of GB. Table 4.4 provides the prediction accuracies of all models in detail. We note that two traditional ML algorithms, k-NN and GB, are deterministic algorithms, thus the ensemble techniques are not applicable. Dependent parameters of the two algorithms are set by trial-and-error to achieves the best evaluation accuracy.

The GNN modes have robust performance, shown by the narrow box plots in Figure 4.5, on the first three testing sets. These testing sets contain star-shaped topologies that are similar to the topologies in the training set. On Testing sets 2 and 3, modifying traffic

statistics or topology patterns from the training set, the prediction accuracy does not have considerable difference from Testing set 1. This preliminary evidence suggests that GNN models can perform beyond the scope of the training set. However, we need further analysis to conclude whether GNN models can generalize to arbitrary topologies since topologies in Testing set 3, although modified, still keep star-shaped.

Moving towards the right of Figure 4.5, the performance of GNN models tends to increase the diversity, together with the discrepancy between the training set and testing sets. We observe that GNN models are less sensitive to changes in the topology than the modification in traffic patterns. For instance, compare to Testing set 4, traffic patterns in Testing set 5 are modified and leads to a reduction of 1.4% in the prediction accuracy. Nonetheless, there is no significant drop in the performance of GNN models on any testing sets, even some sets with the unseen shape of topologies or specific traffic patterns, which is contrary to traditional ML algorithms.

On the other hand, the values represented in Figure 4.6 and listed in detail in Table 4.4 also highlight the weakness of GNN, that some individual models perform poorly on one or several testing sets. On Testing set 13, a small proportion of individual models have very low prediction accuracy, see Table 4.4. It is even more severe since we do not know in advance which models will have poor performance on a given testing set. This variance is the characteristic of neural networks that can be mitigated using the ensemble of individual GNN models.

Ensembles of GNNs as a fault-tolerant mechanism

Prediction accuracies of the ensemble have similar values with the average of individual GNN models, as shown in Table 4.4. With majority voting and GB techniques, the accuracies of the ensemble are respectively in range [79.3, 90.0] and [81.6, 89.8] percents. In other words, the ensemble predicts the feasibility correctly at least 79% and up to 90% of TSN configurations in a testing set. Figure 4.6 clearly shows the advantage of using the ensemble technique. In the figure, prediction accuracies of all 32 individual models are graphically represented, which have a large variability on multiple testing sets. The ensemble plays an important role to mitigate this problem, according, it is an effective fault-tolerant mechanism when developing the GNN model.

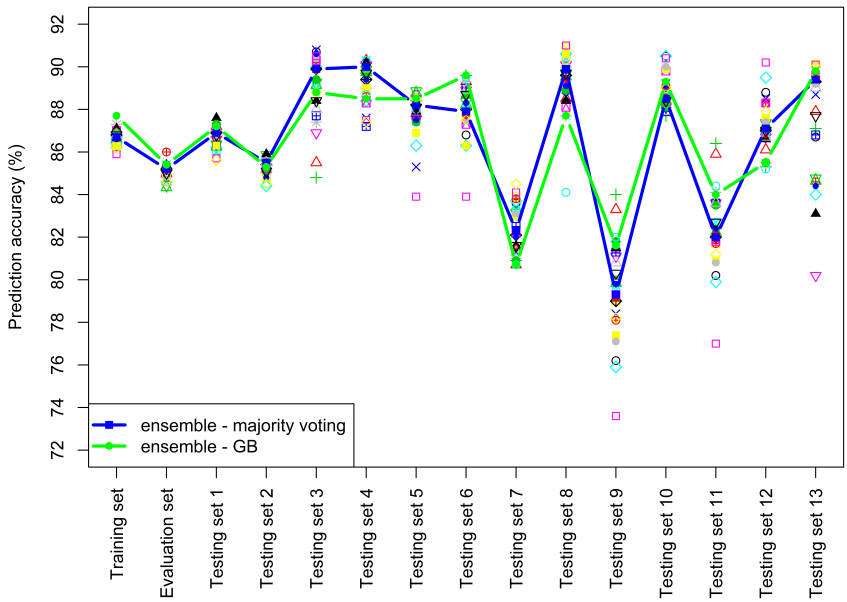


Figure 4.6: Visualization of the variance in the performance of 32 individual GNN models. The prediction accuracies of each model are shown by points with a distinct shape and color. The thick lines in blue and green are the accuracies of the ensembles with majority voting and GB, respectively. A GNN model that performs well on some testing sets can also have a poor performance on other data sets.

Among the two ensemble techniques, GB contributes to better performance than majority voting on 9 over 13 testing sets (see Table 4.4). However, the improvement is not significant with 1.7% at most. We note that deploying the GB ensemble technique is more complicated than majority voting. On the on hand, the final result of majority voting is only a simple consensus on the predictions of 32 individual models. On the other hand, as Gradient Boosting is a supervised ML learning, we need to give it a training set before applying it on testing sets. The training set of GB, in our context, is a collection of the predictions on the training set containing TSN configurations (see 4.4.1). It has 32 features corresponding to the prediction of 32 individual models and contains 10,000 training samples. In summary, two more steps: 1) predicting the feasibility of all TSN configurations in the training set, and 2) training the GB, are needed when applying this ensemble technique, while the performance does not have a clear boost. Accordingly, the simpler ensemble technique with majority voting should be preferred in the next studies.

False positive and false negative prediction rates

The ensemble of 32 individual GNN models mitigate the risk of single model failure, however, there are still some TSN configurations receiving incorrect predictions. They are classified as false positives and false negatives. A false positive is a non-feasible TSN configuration that is wrongly predicted as feasible. False positive ratios in our experiments range from 0.8% to 17.6% over 13 testing sets, and 6.6% on average. If the number of false positives is high, designers may have sub-optimal design options to choose. Although the inconvenience, it does not jeopardize the safety of the real-time system as any chosen TSN configurations must be verified using the schedulability analysis before the final decision.

A false negative is a feasible TSN configuration that is wrongly predicted as non-feasible. This leads to the discard of the configuration during the design phase. False positive ratios range from 0.1% to 19.9% depended on the testing sets, and the average ratio is 6.7%. Since the DSE usually involves a vast number of candidate configurations, some candidates being wrongly rejected have a low impact on the final outcomes. In addition, the rate of false positives can be controlled effectively using a combination of ML and schedulability analysis in a hybrid model, as proposed in [3].

4.5.3 GNN models in the design space exploration of in-vehicle networks

The design of new E/E architectures is increasingly belonging to DSE experiments on virtual platforms [103, 104, 110, 111]. Key drivers for DSE include the complexity of architectures and corresponding technologies, design time and cost effectiveness. Recently, the emerging of software update over-the-air also introduces new requirements. Other industrial domains such as Industry 4.0 witness similar requirements evolving, and should also benefit from the development of DSE techniques.

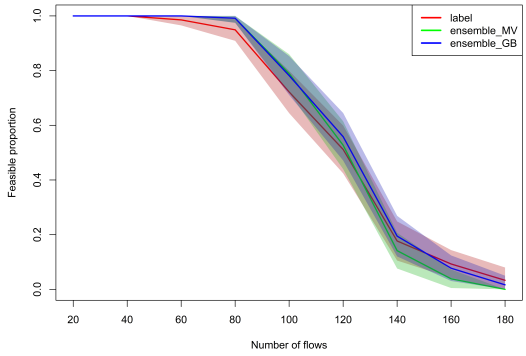
One of DSE functions for TSN network design is *Topology-Stress-Test* (TST), included in RTaW-Pegase software [18]. This function allows users to evaluate a communication architecture, whether it can successfully add more flows or services. Given a topology and traffic statistics resembling future services, the function iteratively increases the number of flows and generates corresponding configurations, configures them, then evaluation them

by schedulability analysis or simulation. The result is a graph representing the capability of the topology to support additional services. We note that evaluating the performance of TSN configurations is the bottleneck in DSE. Studies in [103, 104, 111] apply domain knowledge to maximally prune search space, however, the computation still takes hours to days, which definitely limits the R&D work. In the following, we discuss a new trade-off between prediction accuracy and computation time when applying the GNN model to DSE.

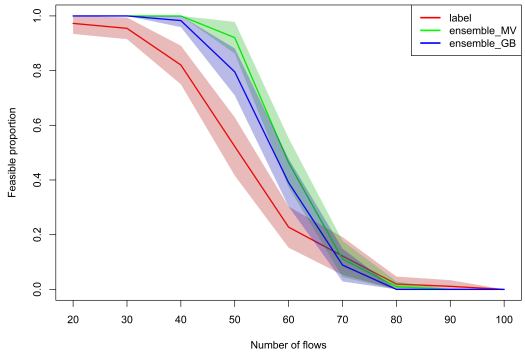
Accuracy obtained with GNNs

Figure 4.7 shows prediction accuracies from the ensemble of 32 individual GNN models in the context of DSE with the FACE, Renault, small and Volvo topology (Testing set 4, 6, 12 and 13, respectively). These four topologies have been introduced in several research studies as empirical applications of TSN in the automotive industry, for instance, the Daimler topology in [21]) is used in Testing set 12. The green curves are the performance of the ensemble with majority voting technique, and the blue curves correspond to GB technique. Data collected from TST runs on these topologies is considering as true labels and represented by red curves in the figure. The colored regions cover areas in the 95% confidence interval. Logically, the probability to be feasible (y axis), *i.e.* to meet all deadline constraints, decreases when the number of flows (x axis) increases.

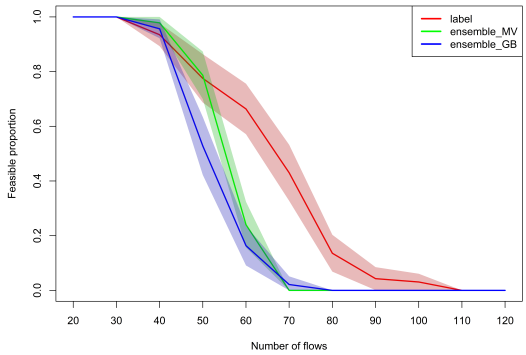
We observe from Figure 4.7(a) that both prediction curves nearly overlap the label curve. This is the result of the similarity between Testing set 4 and training set, with a slight difference that FACE topology in this testing set is much larger than the training topologies. In Figure 4.7(b) with Renault topology and Figure 4.7(d) with Volvo topology, prediction curves are still close to the label curves although large discrepancies between the training set and corresponding testing sets. The prediction accuracy is pessimistic with the small topology, as there is a gap between prediction curves and the label curve in Figure 4.7(c). However, this pessimistic does not actually affect DSE since practical designers usually consider the upper part of the curves, e.g. above 75% of configurations are feasible, to guarantee the extend of the architecture with new services. In summary, experimental results in Figure 4.7 and Table 4.4 suggest that the prediction accuracy of the ensemble of 32 GNN models is high enough for DSE, which can facilitate the early-stage



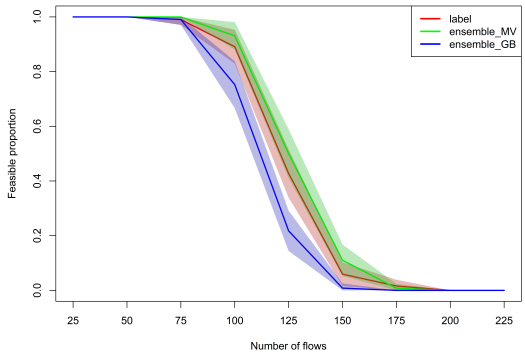
((a)) FACE topology (testing set 4)



((b)) Renault topology (testing set 6)



((c)) Small topology (testing set 12)



((d)) Volvo topology (testing set 13)

Figure 4.7: We compare the predictions of GNN to true feasibility values on 4 testing sets with distinct topologies. x-axis and y-axis are respectively the numbers of flows and the proportion of feasible configurations in the data sets. The color areas represent the 95% confidence interval.

design decisions concerning architectures and technologies.

Speedup factor with GNN

When encoding TSN configurations as graphs, we keep embedding matrice and connection matrice of all graphs in the same size, as described in § 4.4.2. This is required by the deep learning library TensorFlow when using mini-batches of samples during training. The result is that the GNN model predicts the feasibility of any TSN configuration in a constant time, *i.e.* it does not belong to the number of flows or the complexity of the topology. Details of the execution times of the GNN model are presented in the following.

- *Training time*: Each individual GNN model in the ensemble needs 10mn9s for

training with 500 iterations and a mini-batch size of 32. As a result, it takes 5h25mn to finish training 32 models on a single computer with the TensorFlow library. However, the training time is not a part of DSE since we need to train the GNN model only once and designers can use the pre-trained model as a convention in deep learning. In addition, individual models can easily be trained in parallel, like one per GPU.

- *Testing time*: The ensemble with majority voting takes 88s to evaluate 1000 configurations (in a testing set). The ensemble with GB needs more time (+1ms for each configuration), resulting in non-considerable +1s for each testing set. On the other hand, schedulability analysis takes much more time to provide the feasibility. The small topology in Testing set 12 is considered as the simplest topology, which requires schedulability analysis to spend 6788s (1h53mn) with 1000 configurations. The most complicated topology, FACE topology in Testing set 4, requires 41h55mn for schedulability analysis on the same number of configurations. Accordingly, the ensemble of 32 individual GNN models is faster than schedulability analysis from 77.13 times to 1714.77 times.

4.6 Related works

Graph neural network (GNN) has been evolved from multiple deep learning paradigms, such as adopting the concept of shared feature maps in Convolutional Neural Networks to be message-passing algorithm. GNN is a promising tool to model communication networks as they are structural data. One of the first applications of GNN in networking is to predict TCP connection throughput [15]. Other studies use GNN to generate routing for distributed nodes [67] or optimize routing in Software-Defined Networks [14]. Recent GNN applications focus on predicting outputs of particular systems. For instance, authors in [64] apply GNN to Multiprotocol Label Switching (MPLS), in order to speed up the verification of MPLS configurations. A study in [76] uses GNN to predict contention between flows and reduce the computation time of Network Calculus analysis, an important formal framework in real-time network analysis.

The first ML application in Ethernet TSN is to predict the feasibility of TSN configu-

rations. In our technical report and the study in the previous chapter [1,2], standard and simple ML algorithms are used: supervised classification with k-NN and unsupervised clustering with K-means, both present promising results with four TSN scheduling solutions. In [3], we introduce a hybrid model combining ML prediction and schedulability analysis to control the uncertainty of ML algorithms. However, these ML algorithms are only validated on testing sets that use identical network topology with the training data. In this chapter, we use GNN that is specialized to model structural data such as computer networks. Accordingly, the proposed GNN model has the ability to generalize over heterogeneous topology and traffic patterns.

The difference between our GNN model and the previous works that also use GNN in real-time systems should be emphasized. The GNN model in [76] is used to predict the best contentions models at each location in the network, which leads to efficient Network Calculus models and reduces execution time for a feasibility test. On the other hand, our GNN model predicts the outcome of a priority allocation algorithm, *i.e.* whether it can derive feasible priority assignment for a TSN configuration. During the search for optimal priority assignment, the algorithm will run many feasibility tests and each test derives the feasibility of all flows. The GNN model needs to have a chain of correct predictions in order to correctly predict the results of the algorithm. That means our GNN model actually solves a broader problem.

4.7 Conclusion and future work

The study in this chapter, to answer the research question **RQ3** in Section 1.2, is on the quest toward applying AI to automate DSE in real-time systems. The GNN-based deep learning model is used, for the first time, to predict the feasibility of Ethernet TSN networks, which further improves the efficiency of resource usage to design systems. The proposed GNN models provide two practical advantages for the designers: 1) it automates feature selection step and thus the dependence on domain expertise to choose the right features is eliminated, and 2) it can generalize well on various topology shapes, *i.e.*, we do not have to re-train the model for each new topology. We also apply the ensemble techniques as a fault-tolerant mechanism to mitigate the risk of unpredictability by neural networks

constituting the GNN model. The experiment results on 13 distinct test cases show that an ensemble of 32 individual GNN models achieves prediction accuracies from 79.3% to 90%, and faster than schedulability analysis 77 to 1715 times. Such positive results make GNN a viable verification method and open more opportunities for design-space exploration.

The experimental results also expose a risk that an ML model works well on many test cases but performs poorly on other cases, and therefore emphasizes the role of extensive evaluation. For instance, conventional ML algorithms do not have a consistent performance on unseen data. This is only unveiled after the evaluation of 13 testing sets with different network topologies. On the contrary, the prosed GNN model has low fluctuation on the performance over testing sets, which proves its generalized ability.

In our view, there are many possibilities to improve the performance of the GNN model. The training set can be enlarged and focuses on the networks that are hard to predict. We can also improve the structure of neural networks in the model, for instance, using more advanced activation functions, adding more layers, or fine-tuning hyperparameters. The prediction accuracy of the ensemble can be further improved with better ensemble techniques, stacking and XGBoost are two of the potential candidates. In the long term, the researchers can develop an ML framework that uses a core GNN and operates with several TSN mechanisms to have the greatest benefit from the flexibility of neural networks.

Chapter 5

Improvements to the Graph Neural Network Model

Abstract

Graph Neural Network (GNN) is an advanced deep learning model that can effectively predict the feasibility of TSN configurations. The GNN model accesses the raw data from a network and encodes it as a graph, which allows the model to exploit complex relations among physical and logical components in the network. However, moderate prediction accuracy, from 79.3% to 90% in the experiments of the previous chapter, is the main challenge for the adoption of GNN in practical domains. This study aims to improve the performance of the GNN model with two approaches. We first address the limitations of the existing training set, then introduce more diverse sets and focus on the networks that are hard to predict. We also enhance the neural networks that constitute the GNN model with modern techniques such as advanced neuron activation functions and a global pooling layer. The combination of these techniques leads to a significant increase in the accuracy of the ensemble GNN model, up to 11.9% on 13 test cases. These improvements only have a small negative influence on the execution time, *i.e.* the speedup factor is still from 50 to 1125 comparing to the conventional schedulability analysis. In addition to advanced ML techniques for the supervised GNN model, we also apply semi-supervised learning and self-supervised learning, which results in a modest gain in performance. These results contributes to the development of more reliable ML-based feasibility predictions

techniques for design space exploration.

The work presented in this chapter has been published in [5]. It aims to address research question **RQ4** in Section 1.2:

RQ4: Can we apply more advanced deep learning techniques to improve the performance of the ML models?

5.1 Introduction

In the previous chapter, Graph Neural Network (GNN) has been applied to predict the feasibility of TSN configurations. The proposed GNN model, based on the general GNN model in [44], encodes a TSN network as a graph, while physical and logical components in the networks (flows, links, queues) are corresponding nodes in the graph. Complex relations among components of the network are aggregated into embeddings vectors that represent nodes. Experimental results show that the GNN model is generalized to predict the feasibility of TSN networks with various topology shapes and traffic patterns. The application of ensemble techniques is required to control the risk of uncertainty caused by neural networks in GNN. An ensemble of 32 individual GNN models presents a good performance on 13 realistic testing sets: the prediction accuracies on all test cases are higher than 79%, while GNN is faster than schedulability analysis at least 70 times.

In this chapter, we aim to improve the performance of GNN with several enhancements. Experiments in this chapter are conducted progressively. That means an experimental evaluation is performed when a new technique is introduced. If the new technique leads to a better prediction accuracy overall, it is integrated into the GNN model for the next experiment. In the first phase, called the improvement, there are some changes in the training set and the GNN model, however, the size of the training set and the structure of the model remain unchanged. In more details, 1) the training set contains more diverse topology shapes instead of only star shapes as in the previous chapter. We also generate more TSN configurations that seem hard to predict, therefore GNN is actively required to improve its inducing ability. 2) The identity activation function in neural networks is replaced by more expressive functions to better approximate non-linear distributions. A global pooling layer is added to the GNN model, while multiple channel mechanism is

also explored. These techniques are embedded into the *improved GNN model*.

In the second phase, called the enhancement, both the GNN model and the training set benefit from the expansion in size and structure. The *expanded GNN model* bases on the improved model with some changes. We enlarge the size of the embedding vectors that represent nodes in the graphs, therefore, more synthetic features are generated to capture extra information. The model is also trained longer, which significantly increases the prediction accuracy. The training set is doubled in size and leads to the most drastic acceleration in the performance of GNN. The expanded GNN model has an increase of up to 11% in prediction accuracy compared to the base model in the previous chapter. The extra execution time required is relatively small, thus the speedup factors are still 50 to 1125 times compared to conventional schedulability analysis. These results guarantee that the GNN model serves to identify feasibility TSN networks in a compute-efficient manner.

Some sections in this chapter are devoted to discussing other learning paradigms beyond supervised learning. We apply semi-supervised learning to leverage the availability of massive unlabeled samples in the training set. In addition, self-supervised learning, another deep learning paradigm, is also introduced to better learn the embeddings in the graphs that encode TSN configurations. Although these two ML types cannot totally replace the supervised GNN model, it opens new directions to apply to different areas of real-time networking beyond feasibility prediction.

The rest of this chapter is organized as follows. Section 5.3 introduce an enhanced training set that is more diverse and focuses on hard-to-predict TSN configurations. Advanced deep learning techniques are also integrated into the improved GNN model. In Section 5.4, we expand neural networks constituting the GNN model, which provides a larger capability to obtain more patterns. We also double the size of the training set, thus let the GNN model expose more training samples. Section 5.5 is an application of semi-supervised learning as an effort to reduce the number of labeled samples. In Section 5.6, a self-supervised learning model is introduced, which uses GNN as a core block to learning the embeddings of samples without labels. Section 5.7 presents the related works of GNN and other deep learning techniques in networking. In Section 5.8, we summarize the results of this chapter and highlight possible directions to expand the study.

5.2 Preliminary

This chapter briefly introduces the system model of TSN configurations using in the experiments. In addition, a summary of GNN is mentioned. We also define the base GNN model to compare with the improved GNN model in the next chapter.

5.2.1 System model

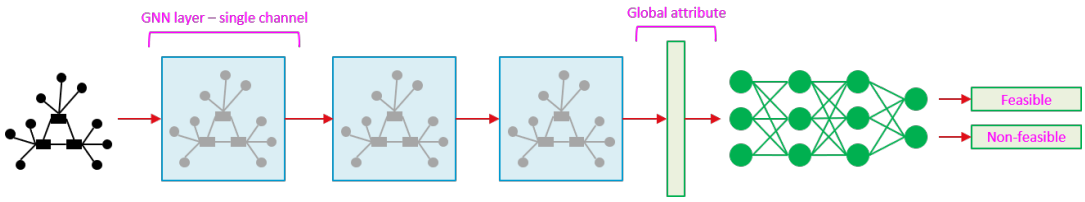
This chapter is a follow-up study of the previous chapter, thus the system model does not change. We still focus on priority assignment, *i.e.*, predict whether a priority allocation algorithm can provide a feasible solution for a TSN configuration. The priority allocation algorithm is Concise Priorities (in the RTaW-Pegase software [18]), which is a variant of the Optimal Priority Assignment (OPA) algorithm [83]. Network topology and traffic characteristics are the same as in the previous chapter. All TSN configurations in the training set are star-shaped. During the study, topologies in the training set can be changed and mentioned where appropriate. There are 10,000 random labeled configurations in the training set, and each testing set has 1000 unlabeled configurations over 13 test cases, as in the previous chapter.

5.2.2 Graph Neural Network

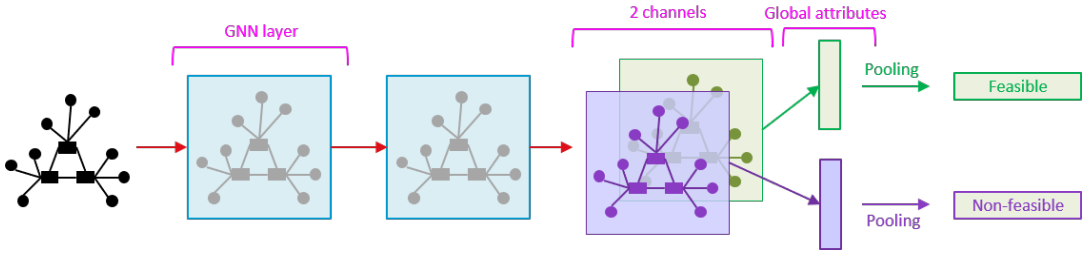
The GNN model proposed in the previous chapter is considered the base GNN model, depicted in Figure 5.1(a). It has a single channel with three layers, *i.e.* it aggregates embeddings of nodes with three loops. We note that in this context, the term *layer* means consecutive blocks in the GNN model instead of layers in the neural network. The base GNN model contains four neural networks: each block has one, together with a final neural network for feasibility prediction. All neural networks have three fully connected layers as in the previous chapter. In the experiments, the base GNN model is improved, however, the encoding method and learning algorithm are identical.

5.2.3 The base GNN model

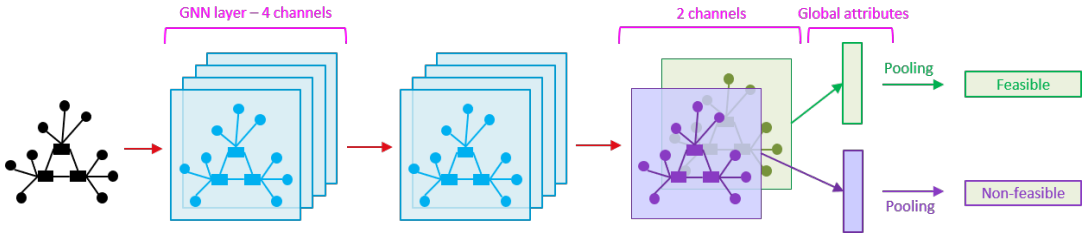
In this first experiment setup, Experiment 0, the base GNN model is similar to the original model in [112] and the previous chapter, except for two changes that are mentioned later



((a)) Base GNN model.



((b)) Global pooling.



((c)) Multiple channels

Figure 5.1: Some improvements applied to the base GNN model (a): global pooling layer (b), and multiple channels (c).

in this paragraph. Each neural network in the model has three layers and all layer has single-channel. The embedding vector has 16 features and the identity activation function is still used in this experiment. To prepare for the improvements of the GNN model in the next experiments, we introduce two changes to the original model. First, the maximum number of nodes in the graph is reduced from 500 to 350. The memory usage of GNN is significantly less due to this reduction, accordingly, it lets more room for the expansion of neural networks. Its effect on the overall performance is very weak since there is $<1\%$ of the training samples are discarded due to oversized. Second, the training process is increased to 20 epochs. Although the training process takes more time to finish, this change allows the GNN model with larger neural networks to achieve its full learning potential. During a training epoch, all training samples are used, which is different from the original model that chooses training samples randomly with replacement.

Table 5.1: Summary of all changes in the experiments. 1^{gl} and 4^{gl} denote the use of a global pooling layer with 1 and 4 channels, respectively. Exp. and Abbr. are respectively Experiment and Abbreviation. Imp. and Ext. resp. stand for the Improved model and the Expanded model. E31 and E32 denote Experiment 3, resp. at step 1 or step 2. In other experiments, we also use this naming convention.

Exp.	**Abbr.**	**GNN model**					**Training**		
		# nodes	**Acti-va-tion**	**# layers**	**# chan-nels**	**Emb. size**	**# iters**	**# sam-ples**	**Training set**
Exp. 0	**Base**	**350**	**identity**	**3**	**1**	**16**	**20**	**10,000**	**Star-shaped**
Exp. 1	E11	–	–	–	–	–	–	–	Random
Exp. 2	E21	–	–	–	–	–	–	–	Focused
Exp. 3	E31	–	ELU	–	–	–	–	–	Star-shaped
	E32	–	ELU	–	–	–	–	–	Focused
Exp. 4	E41	–	–	–	1^{gl}	–	–	–	Star-shaped
	E42	–	–	–	1^{gl}	–	–	–	Focused
Exp. 5	E51	–	–	–	4^{gl}	–	–	–	Star-shaped
	E52	–	–	–	4^{gl}	–	–	–	Focused
	Imp.	**350**	**ELU**	**3**	1^{gl}	**16**	**20**	**10,000**	**focused**
Exp. 6	E61	–	–	–	–	–	50	–	focused
	E62	–	–	5	–	–	20	–	–
	E63	–	–	3	–	32	–	–	–
	E64	–	–	3	–	32	50	–	–
	Ext.	**350**	**ELU**	**3**	1^{gl}	**32**	**50**	**10,000**	**Focused**
Exp. 7	E71	–	–	–	–	16	20	20,000	–
	E72	–	–	–	–	32	50	–	–

5.3 Improving the training set and the GNN model

During the experiments in this section, we enhance the performance of the GNN model by incrementally changing the training set and the neural networks in the model. The improved training set is based on the insights gained from the weakness of the model when training with the old samples, while the model is equipped with the recent advances in deep learning research.

5.3.1 Experiments 1 and 2: Improving the training set

The experimental results in the previous chapter [112] show that the GNN model has low prediction accuracies on some test sets that have network topologies different from the shape of the network topologies in the training set. In particular, all networks in the training set have star-shaped topologies while we make an unrealistic assumption that the GNN model can have the ability to generalize beyond this shape. Besides, the small range in the traffic statistics also limits the learning scope of GNN. Therefore, in this experiment, Experiment 1 (denoted as E11, means experiment 1 - step 1), we introduce more diversity into the training set to support the learning of the GNN model. In particular, shapes of network topologies in the training set are chosen randomly, while packet payload sizes and deadline constraints of flows in the configurations are more random, *i.e.*, they have a wider range in the traffic statistics (see the table regarding traffic patterns in Chapter 4).

Besides the limitation in the training set due to under diversity in both topology shapes and traffic statistics, we also discover a wrong prediction pattern of the GNN model, that it has a higher frequency on a definite range of traffic intensity. When a set of configurations is generated based on predefined traffic statistics, there is a certain proportion of the configurations that are overloaded, *i.e.*, the load of at least one link in the network reaches 100% or over. The range mentioned is when >40% of configurations generated are overloaded. In this range, most configurations are non-feasible and thus the GNN model can predict the feasibility correctly without any learning effort. On the other hand, predicting the feasibility is also an easy task when no generated configurations are overloaded, then most of the configurations are feasible. In both cases, the GNN model has fewer chances to enhance its ability, subsequently, reduces the prediction accuracy with configurations out of these ranges. Therefore, in this Experiment 21 (E21), a *focused training set* is used. The new training set is based on the non star-shaped network topologies in E11 and the configurations are generated randomly, however, it guarantees that two-third of the training set are in the region such that from 0 to 40% of the configurations generated are overloaded.

Both experiments E11 and E21 have the same hyper-parameters of the neural networks as in the base GNN model. The results of the experiments are shown in Figure 5.2 as well as the performance of the base model for the sack of comparison. The red curve with

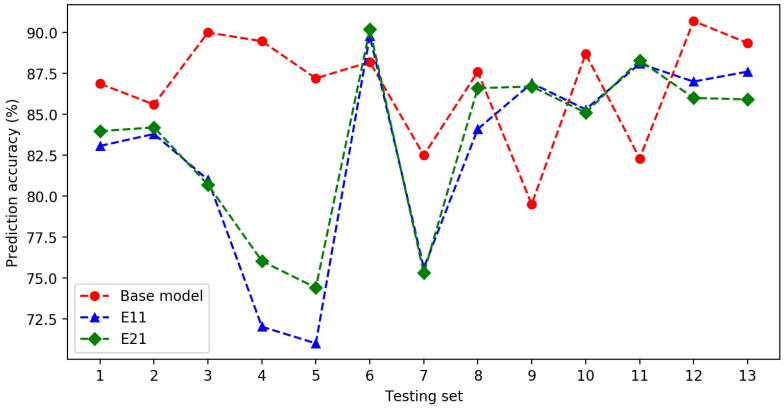


Figure 5.2: In Experiment 1 and Experiment 2, resp. denoted as E11 and E21, we use two training sets that do not only have star-shaped topologies as in the base GNN model. This change leads to better accuracies in some testing sets, however, the accuracies decrease in testing sets #1 to #5 that have star-shaped topologies.

dots in the figure represents the base model, while E11 is the blue curve with triangles and E21 is the green curve with diamonds. Being trained with a training set made up of high diversity in network shapes, the prediction accuracy of GNN model in E11 is higher than the base model in the testing sets #6, #9 and #11, which also have non star-shaped topologies. The model in E21 with the focused training set performs better than the E11 model in overall, however, the prediction accuracy of E21 is lower than the base model with some testing sets that have star-shaped networks. Due to the advantages and limitations in both, discarding any model would impose missing a chance to have a better GNN model when neural networks are improved. Therefore, the base GNN model and the model in E21 are all inherited in the next experiments.

5.3.2 Improving the GNN models

Experiment 3: Using a more expressive neuron activation function

There is no activation function for the neural networks in the base model, *i.e.*, the identity activation function is used. It makes the training of the GNN model easier to converge since the training of neural networks is based on backpropagating prediction error through the gradient, and the identity function does not condition the gradient vanishing phenomenon. However, due to the role of neural networks that learn to approximate or discover patterns, the downside of using identity activation function is that the neural networks become

weak approximators, in particular when learning the complex non-linear relationships in a graph. Experiment 3 comprising E31 and E32 aims to overcome this limitation by using more expressive non-linear activation functions. We have experimented with four functions, chosen by their features, in the development of the model. Sigmoid and tanh are two conventional activation functions that have the output in the range [0, 1] and [-1, 1] respectively, which can be used to indicate the probability of a configuration to be feasible. However, these two functions contribute to the gradient vanishing phenomenon, subsequently, the training of the GNN model cannot converge. The other two activations are Exponential Linear Unit (ELU) and Scaled Exponential Linear Unit (SELU), which have limitless output. Since ELU performs equally to SELU and has a simple formula, we only analyze the result of ELU and the other three activations are not mentioned further.

In Figure 5.3, the red curve with dots represents the accuracies of the base model, that uses identity activation trained with star-shaped networks. The orange curve with diamonds is E31, its GNN model corresponds to the base model but using ELU activation function. Two models in E21 (blue curve with triangles) and E32 (green curve with squares) are trained with the focused training set but respectively using identity function and ELU function. These results show the advantage of ELU activation function over the identity function used in the previous study, with the only exception on the testing set #9 and #11. In particular, compared to E31, the GNN model in E32 trained with the focused training set outperforms or matches on all testing sets. Accordingly, ELU activation function and the focused training set offer a clear boost in the prediction accuracy of the GNN model.

Experiment 4: Using neural network with global pooling

Global attribute plays an important role in the GNN model, shown in Figure 5.1(a), as it is the aggregation of the relationships between principal components in the graph through the message passing algorithm. This vector is used as the input of the final neural network in the GNN model to predict the feasibility of the corresponding configuration. However, the prediction neural network also adds an intermediate between the loss function and the main GNN block, subsequently, the gradient backpropagating from the loss can be vanishing before it helps to update the GNN block. In this Experiment 4, we challenge the existence of the final neural network by replacing it with a global pooling layer, a deep

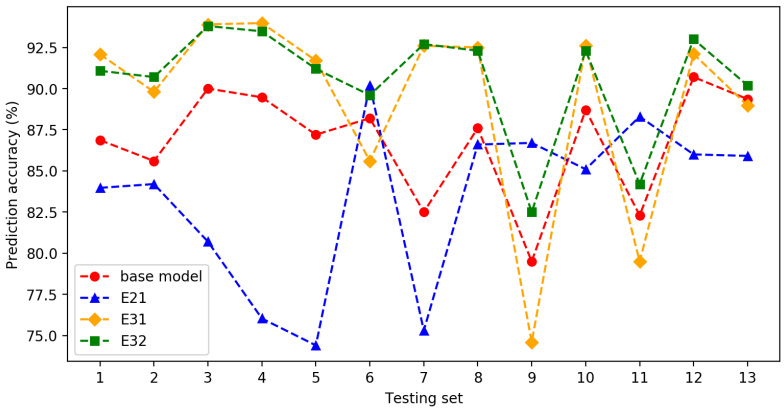


Figure 5.3: In Experiment 3, denoted as E31 and E32, we evaluate the influence of using ELU activation function. E31 is the base GNN model that uses ELU, while E32 has a similar setup as E31 but uses the focused training set. E21 uses the focused training set and not ELU, shown for comparison. E31 and E32 have higher prediction accuracies than the models that do not use ELU, except on testings sets #6, #9, and #11.

learning technique popular in image processing and shown in Figure 5.1(b). In order to apply global pooling, we double the number of channels in the final layer of the neural network f_v that derives global pooling. Therefore, two global pooling vectors are generated and each is averaged to be a value, *i.e.*, average global pooling. The values are used for feasibility prediction of the configuration in a direct manner. For instance, the first value represents feasible and the second value for non-feasible, then the GNN model predicts a configuration to be feasible if the first value is larger. In this experiment, we decide the order of the value to be [feasible, non-feasible] before the training process and it keeps unchanged.

In Figure 5.4, prediction accuracies of the GNN models in Experiment 4 that use global pooling layer are shown together with Experiment 3 for the sake of comparison. The models in E31 (red curve with dots) and E41 (orange curve with diamonds) are trained with the same star-shaped networks, however, the latter replaces the prediction neural network with the global pooling layer. In a similar order, the blue curve with triangles (E32) and the green curve with squares (E42) are trained with the focused training set but with and without the global pooling layer, respectively. The results from each pair of models almost overlap over 13 test cases, which suggests that the global pooling layer and thus removing the final neural network does not worsen the overall performance. In the next experiments, we will always use global pooling as it reduces complexity in the

structure, accordingly supports the development and maintenance of the GNN model. It is worth noting that the global pooling layer does not burden the computation since the heavy load by doubling the channel is compensated by the removal of the final neural network.

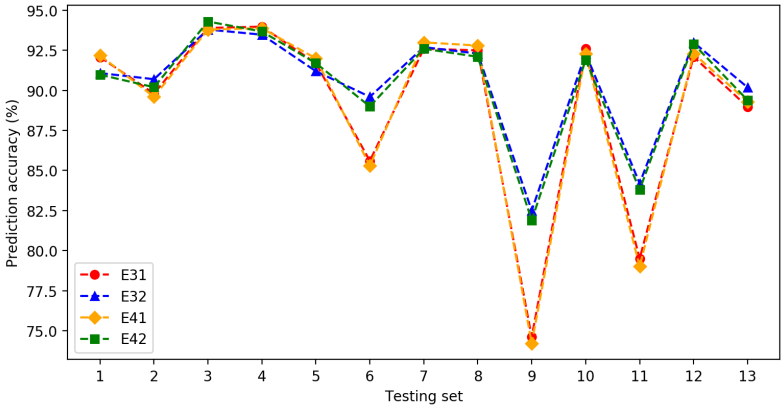


Figure 5.4: In Experiment 4, we evaluate the influence of using a global pooling layer, trained with star topologies (E41) and the focused training set (E42). The global pooling layer does not lead to improvement in the performance, however, it suppresses the final neural network and thus makes the GNN model simpler.

Experiment 5: Using multiple channels

Many deep learning models such as Convolutional Neural Network use multiple channels as a common technique to exploit different views from input data. In this Experiment 5, we also apply the same technique to the GNN model by increasing the number of channels from one to four, as shown in Figure 5.1(c), with all layers of two neural networks: f_v to update node embeddings and f_u to update the global attribute. There is an exception in the final layer of f_v, which has only two channels in order to be compatible with the next global pooling layer. The experiment will answer the question of whether more channels can provide more perspectives of the graph and thus improve the prediction accuracy.

Figure 5.5 shows the experiment results for two pairs of GNN models. The models in E41, the red curve with dots, and E51, the orange curve with diamonds, are trained with star-shaped networks but using single and multiple channels, respectively. The blue curve with triangles (E42, single channel) and the green curve with squares (E52, multiple channels) are the prediction accuracies of two models trained with the focused training set. In contrast to the hypothesis, increasing the number of channels in the neural networks does not lead to a considerable improvement in the overall performance of the GNN

model. On the other hand, for each layer, multiple channels impose an expansion in the first dimension of the weight matrix corresponding to that layer in the neural network. Consequently, this technique complicates the training process and incurs more memory as well as training time. The neural networks in the GNN model therefore only have single channel in the next experiments, except the final layer of f_v with two channels for global pooling.

In summary, this section explores the possibilities to increase the prediction accuracy of the GNN model through the improvement of the training set as well as the neural networks in the model. In Experiment 3, the ELU activation function replaces the identity function and provides a clear boost in the performance on all 13 test cases. The GNN model predicts the feasibility directly from the global pooling layer in Experiment 4, and the final neural network is removed to reduce the structural complexity. On the other hand, Experiment 5 shows that using multiple channels in the neural networks does not lead to a noticeable improvement, while this technique incurs a higher cost to train the model. We also observe that in all experiments, the GNN models trained with the focused training set can generalize better than the models trained with star-shaped networks. Based on these experimental results, an *improved model* that uses the ELU activation function, the global pooling layer, and trained with the focused training set, is recommended in next experiments. It is worth noting that the improved model also inherits from the base model a maximum of 350 nodes in the graph.

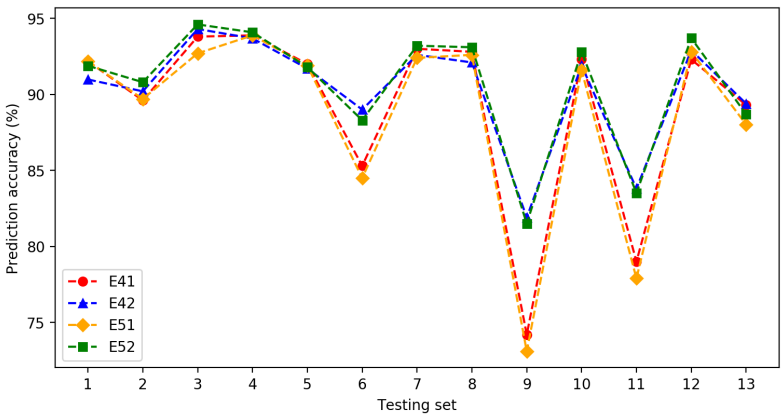


Figure 5.5: In Experiment 5, we evaluate the influence of using multiple channels at the first two layers of the GNN model, trained with star topologies (E51) and the focused training set (E52). This technique does not increase the prediction accuracies overall, while it requires more computation resources.

Table 5.2: A summary of prediction accuracies, expressed in %, of all GNN models in this study. The baseline is the ratio of feasible configurations in the data sets, derived by precise analysis.

Testing sets	Baseline	Base GNN model	Improved GNN model	Expanded GNN model	Expansion of training set	
					Improved model	Expanded model
Testing set 1	57.9	**86.9**	91.0	92.3	91.4	92.6
Testing set 2	53.9	**85.6**	90.2	91.5	90.7	90.0
Testing set 3	47.5	**90.0**	94.3	95.4	93.3	94.6
Testing set 4	62.2	**89.5**	93.7	94.5	93.0	94.4
Testing set 5	73.3	**87.2**	91.7	91.0	90.5	89.0
Testing set 6	36.0	**88.2**	89.0	88.7	90.5	91.8
Testing set 7	51.4	**82.5**	92.6	93.6	93.4	94.5
Testing set 8	41.6	**87.6**	92.1	95.3	93.3	94.0
Testing set 9	22.7	**79.5**	81.9	81.1	84.4	87.1
Testing set 10	39.1	**88.7**	91.9	92.7	91.7	92.0
Testing set 11	22.0	**82.3**	83.8	83.9	86.0	88.1
Testing set 12	48.0	**90.7**	92.9	93.9	92.7	93.7
Testing set 13	57.8	**89.4**	89.4	88.9	91.2	82.2

5.4 Expanding the GNN model and the training set

In Section 5.3, we have defined the improved model, which inherits from the base model and injects some changes. However, we do not increase the original model size, in contrast, it is suppressed since the removal of the final neural network. In this section, we enlarge the learning capability of the GNN model by expanding the neural networks and number of samples in the training set. The models in the next experiments are based on the improved model in the previous section.

5.4.1 Expanding the GNN model

In Experiment 6, we progressively enlarge the neural networks in the models and enhance the training process. These changes support the expansion in the learning capability of the GNN model.

- In E61, the number of training epochs is increased from 20 to 5, thus avoids the case that the training is stopped before the GNN model reaches the peak performance.

- In E62, we enlarge neural networks in the model by adding two more layers, *i.e.*, the number of layers is 5 instead of 3 as in the improved model.
- In E63, we double the embedding vector size from 16 to 32.
- In E64, a combination of E61 and E63 allows the GNN model to have a larger embedding size and being trained longer.

Prediction accuracies of GNN models in four experiments are shown in Figure 5.6, comparing to the improved model. The model in E61, trained longer with 50 iterations, has a clear boost in many test cases. In contrast, an increasing number of layers in the neural networks constituting the GNN model, as in E62, worsen the performance. Although more layers mean more weights for learning in the neural networks, it potentially makes the gradient vanishing phenomenon become more severe. As a result, the GNN model becomes harder to train [1]. On the other hand, a larger embedding size in E63 positively gains the generalization of the GNN model, as the prediction accuracies increase even with the cases that are hard to predict such as test case #9. Finally, in E64, the accumulation of a large embedding vector and more training iterations, defined as the *expanded model* in the following, has the best performance among the GNN models presented in the figure.

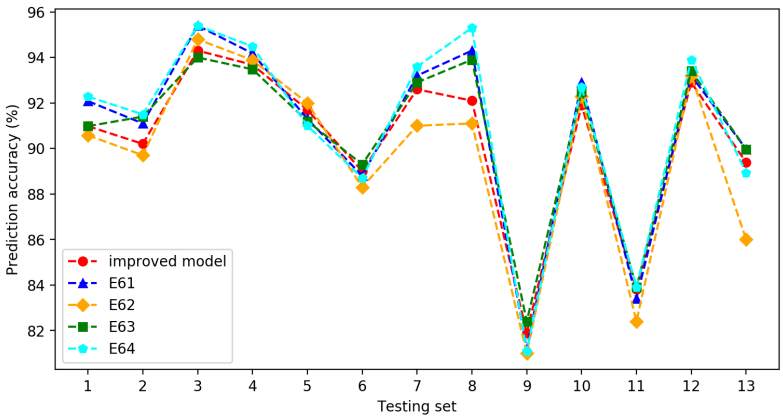


Figure 5.6: In Experiment 6, we evaluate the influence of scaling up the size GNN model: the number of training epochs is increased from 20 to 50 (E61), the number of layers is increased from 3 to 5 (E62), the size of embedding is expanded from 16 to 32 (E63), and finally, E64 combines the larger embedding and more training epochs.

[1] This problem can be mitigated by adopting the residual network (ResNet) architecture, however, it is beyond the scope of this chapter.

5.4.2 Expanding the training set

Previous experiments with the expanded model indicate the limitation of the approach to enlarge neural networks, as it complicates the structure and requires larger memory. In the following experiments, we enlarge the training set as a more traditional approach to improving the performance of the GNN model. The *expanded training set* contains random-shaped networks generated in a similar process as the focused training set, however, the number of configurations doubles from 10,000 to 20,000. The improved GNN model in Section 5.3 and the expanded GNN model in Section 5.4 are both trained with the expanded training set, respectively denoted as E71 and E72. The results of the experiments answer questions about the benefit of more training samples and furthermore, whether a larger model is needed when the training data is amble.

In Figure 5.7, prediction accuracies of the improved model training over the expanded training set are shown via the orange curve with diamonds. Comparing to the same model traning with the smaller training set, the larger training set enhances the generalization of the model: the prediction accuracies are higher than 84.4% in all test cases. The results agree with other deep learning studies, e.g. [41], which also indicates that a larger training set is one of the key factors for neural networks to gain better accuracies.

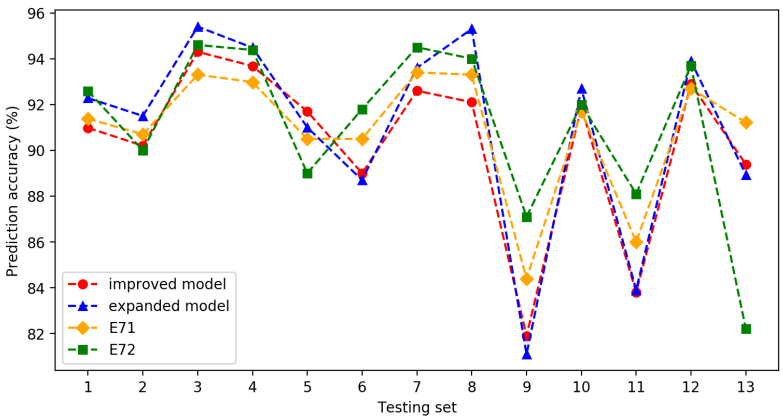


Figure 5.7: In Experiment 7, we evaluate the influence of larger training sets made up of 20000 samples instead of 10000 and use it to train the improved and expanded GNN models (E71 and E72, resp.). Both models can generalize better with the larger training set, except the test case #13 that is discussed in §5.4.3.

With the expanded GNN model, the larger training set is even more impactful comparing to the improved model, as 12 over 13 test cases has prediction accuracies >87.1%.

It is valuable in practical applications, which usually require the machine learning to be consistent and the prediction accuracies is not too low. There is an exception on test case #13 with Volvo topology. The reason is that, configurations in this testing set has AVB-shaped streams, which is very distictive traffic and not apprear in the training set. When the GNN model is exposed to more training samples, it does not only achieve higher prediction accuracy but also gain better ability to separate the true traffic pattern from the aliens. Combining with the larger model in E72, the accuracy with test case #13 drops as an exception, but it is not an abnormal phenomenon and can be tackled by introducing the new traffic pattern to the training set.

5.4.3 Analysis of the experimental results

In Table 5.2, we present the accuracies of all GNN model in the previous sections: the base model, the improved model and the expanded model with focused training set as well as expanded training set. A trivial model that always predicts configurations as feasible positions the baseline, however, it is different from a random guess and thus the accuracies can be less than a half. The prediction accuracies of the base model is in range [79.5%, 90.7%]. Ranges of accuracies [84.4%, 93.3%] and [82.2%, 94.6%] are respectively from the improved model and the expanded model.

For the sake of comparison, Figure 5.8 summarizes changes in the prediction accuracies of all models from the base model. The improvement in the performance is remarkable, since the GNN models are trained with the focused training set, containing random shaped networks, but still gain better accuracies on star-shaped networks in test cases #1 to #5. In particular, there is an increase of more than 10% accuracy for test case #7 with Renault topology, observed from all four models. In general, changes during experiments accelerate the learning capability of GNN models, which lead to considerable improvement in the prediction accuracies except test case #13.

Due to the limitation in the learning capacity of the GNN model, it can only learn a definite number of patterns that make configurations to be feasible. When training with a small number of networks, the model learns general patterns and thus the prediction accuracies are modest on all test cases. With the expanded training set, the number of samples increases to 20,000 and allows the GNN model to obtain specialized patterns.

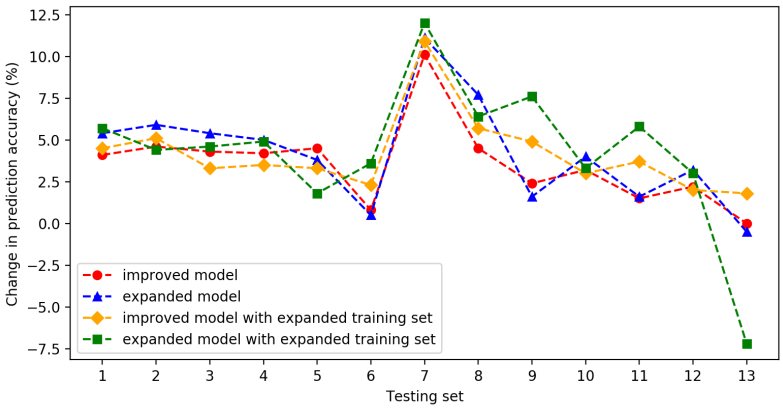
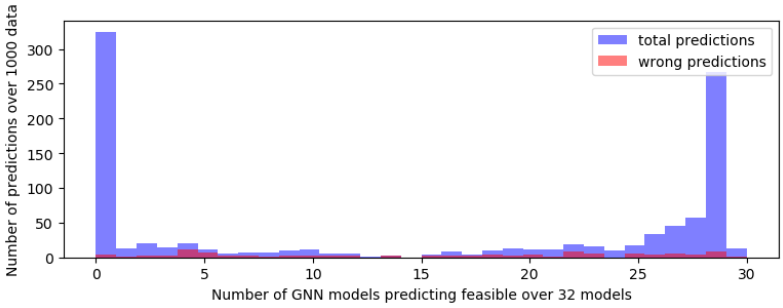


Figure 5.8: Four variant GNN models show variations in predictions accuracies, expressed in %, with the based GNN model.

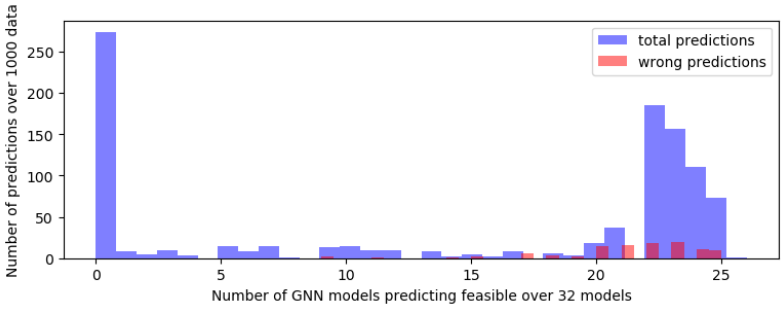
As a result, the accuracies are boosted for some testing sets but detrimental in case there are alien traffic statistics that have not been tested with the specialized feasible patterns discovered by the model, which is the test case #13 with AVB-shaped streams.

The exception of the test case #13 is further analyzed in Figure 5.9. Among 32 models in the ensemble, this figure shows how many models predict a configuration to be feasible, and for each configuration in the testing set. In the ideal case, all models in the ensemble should have a consensus on the feasibility prediction of a configuration, and the bin in the figure corresponding to the value 32 is added by one if predicting feasible, or value 0 if predicting non-feasible. However, the empirical distributions are more spread when the model and the training set become larger, as in the order of the figure. For instance the improved model, the highest bin on the right is not at position 32, which implies that some models predict incorrectly. Low prediction accuracy in the test case #13 is therefore the result of the decline in the consensus of the ensemble. On the one hand, the results affirm the importance of ensemble because there exists a risk that some individual models have the wrong prediction on certain configurations. On the other hand, there is an opportunity to enhance the prediction accuracies by increasing the number of models in the ensemble. The proportion of GNN models that predict feasible can also be used to measure the uncertainty of the prediction and trigger the schedulability analysis, as a hybrid model proposed in [3] and Chapter 3.

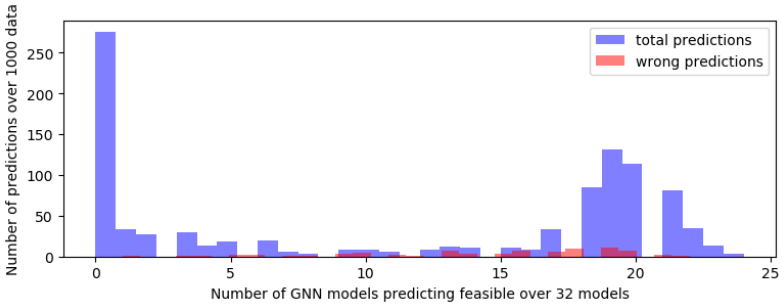
Besides, the performance with the test case #13 supports our suggestions when applying ML to a technical problem. First and foremost, we need to build a strong evaluation system



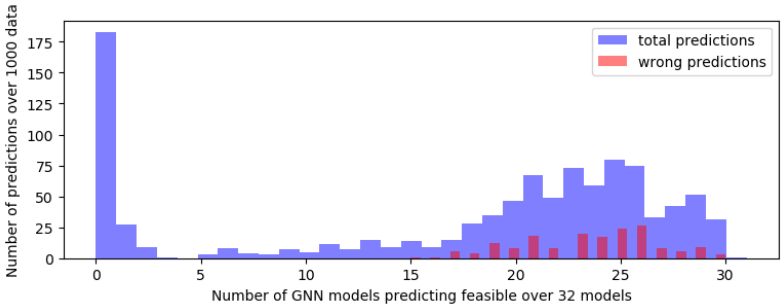
((a)) The improved model.



((b)) The expanded model.



((c)) The improved model with expanded training set.



((d)) The expanded model with expanded training set.

Figure 5.9: Testing set #13 with Volvo topology: frequencies in histograms of the number of individual GNN models that predict as feasible (in the ensemble). It is counted on each TSN configuration and summed up for 1000 configurations in the testing set. Bin 0 is the case that all individual GNN models in the ensemble predict a configuration as non-feasible, and bin 32 for feasible.

for the GNN model. In our opinion, the researchers should build multiple real-world testing sets at the beginning of an ML study to cover as many case studies as possible. Second, we should guarantee the heterogeneity of the training set, thus the training set is able to cover novel patterns in the testing sets. Third, an ensemble approach is essential to mitigate the wrong prediction of an individual GNN model. As shown in Figure 5.9, the ensemble of GNN models still has a relatively high prediction accuracy of 82.2% on the testing set 13 although many GNN models have wrong predictions.

Table 5.3: Execution times of the ensemble of 32 individual GNN models, compared to schedulability analysis. The first row is the absolute value of running times on a single GPU. While the running time of GNN is a constant on all configurations, it varies in schedulability analysis depending on the complexity of TSN networks. In the second row, the first column is the speedup factor of GNN, corresponding to the smallest and largest factor. The rest is the relative speedup factor compared to the base GNN model.

Testing time	**Schedula -bility analysis**	**Base GNN model**	**Improved GNN model**	**Expanded GNN model**	**Expanded training set**	
					Improved model	**Expanded model**
single GPU	–	0m21s	0m24s	0m32s	0m24s	0m32s
Relative time	77 / 1715	**1.0**	1.14	1.52	1.14	1.52

5.4.4 Execution times of training and testing

In this study, the experiments are performed on the High-Performance Computing (HPC) cluster [113] located at the University of Luxembourg. The 32 GNN models in the ensemble are trained in parallel, on 32 GPUs (NVIDIA V100), while the ensemble is tested on a single GPU and the models are executed in order.

The configurations in the training sets, evaluation set and testing sets are generated by the library of RTaW-Pegase 3.7.5 software [18] run on platform Java JDK-13. The labels of the configurations that indicate the feasibility are derived from the Concise Priorities algorithm available in Pegase. This is a version of OPA that implements schedulability based on network calculus, which takes 471ms on average the derive the feasibility for a TSN configuration with 500 streams (detailed measurements are provided in [1]). Therefore, it would take more time to know whether there exists a feasible priority assignment solution for a configuration since OPA requires multiple steps and each step need the verification of the feasibility.

Table 5.3 shows the execution time of four proposed GNN models as well as the base model with a testing set containing 1000 TSN configurations. The size of networks does not affect the execution time since the graphs, *i.e.* inputs of the models, always have the maximum number of nodes. It can easily be achieved by padding zero to empty nodes in the initial feature matrix and connection matrix, as a common technique in neural networks. Besides, we do not count the time demanded to derive graphs (by encoding TSN configurations), since it is a fraction of the execution time of the GNN models and the task can be done in parallel.

The base model takes 0m21s to predict the feasibility of 1000 TSN configurations. However, the larger and more advanced GNN models only have a slight effect on the execution time, with an increase of 11s at most. The expanded model requires 0m32s to predict the feasibility of 1000 configurations, or an increase of 52% compared to the base model. For the comparison with the schedulability analysis, the base model has a speed-up factor of 77 times with the most complex testing set and 1715 times with the simplest one. The expanded model still has the speed-up factor from 50 to 1125 times. Accordingly, we reaffirm the need to apply the improvement on the GNN model since the cost incurred is low but the increase in the prediction accuracy is significant, as discussed in §5.4.2 and §5.4.1.

5.5 Application of semi-supervised learning

5.5.1 Introduction to semi-supervised learning

Semi-supervised learning [114] is a training type in ML that combines labeled and unlabeled samples to achieve a better performance than solely labeled samples. In semi-supervised learning, a training set typically contains a small number of labeled samples and a large number of unlabeled samples. In the first step, a supervised ML model is trained with labeled samples. After finishing this step, we then apply the trained model to predict the labels of unlabeled samples. The predictions are then assigned to unlabeled samples to become their pseudo-labels. The ML model is trained again with the training sets mixing both types of samples. On the on hand, semi-supervised learning can achieve better performance than supervised learning since the ML model is exposed to more data,

which is crucial to discover more patterns. On the other hand, as the ML model is trained with the labels generated by itself, there is a risk that it does not learn new information and therefore the return of extra training time is not considerable. There is no theoretical foundation for semi-supervised learning as with supervised learning, thus the usefulness of semi-supervised learning can only be verified for each application. For instance, a study in [115] applies semi-supervised learning to gain better performance, up to 2% than supervised learning on the well-known ImageNet data set for object recognition.

5.5.2 Performance of GNN with semi-supervised learning

We use the improved GNN model introduced in previous sections to evaluate the influence of semi-supervised learning. There are 10,000 labeled configurations and 100,000 unlabeled configurations in the training set. The generation of 100,000 unlabeled configurations takes around one day on a single desktop computer. We note that this process can be faster if we loosen the criteria to choose the number of flows for generated configurations. Nonetheless, this running time can be eliminated as the generation of 10,000 labeled configurations takes around two weeks and requires more computation power.

In the first step, we train the GNN model with only labeled configurations. Then after each training iteration, we use the trained GNN to predict the feasibility of unlabeled configurations. The predictions are the pseudo-labels of these configurations. That also means semi-supervised learning incurs further costs in the training process. We note that with a large number of samples, neural networks can be over-trained, *i.e.* overfitting, and cannot optimize towards the goal. In order to avoid this phenomenon, we use the Dropout technique [116] with the dropout rate of 0.1. We also re-train an ensemble of 32 individual GNN models using only supervised-learning with 10,000 configurations, however, the dropout does not affect the overall performance.

The performance of semi-supervised learning is shown in Figure 5.10. With 1000 labeled samples, the prediction accuracies with semi-supervised learning (yellow curve with diamonds) are higher than the accuracies of supervised learning (red curve with dots) on 12/13 test cases. However, in some cases, the prediction accuracies of semi-supervised learning are lower than 80%, which is considered as a minimum accuracy to be practical. With 10,000 labeled samples, the performance of semi-supervised learning (green curve

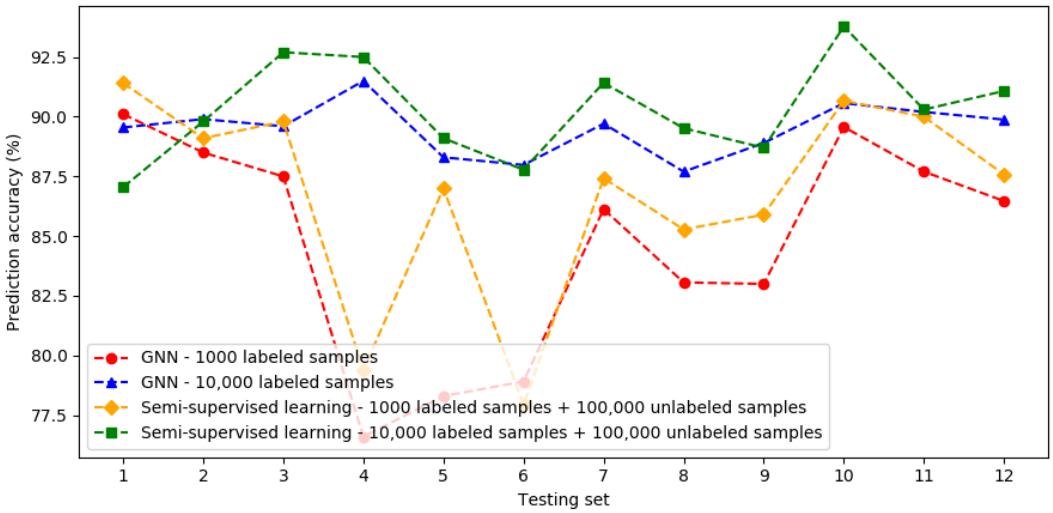


Figure 5.10: Performance of GNN with semi-supervised learning, compared to supervised learning. There are 10,000 labeled samples and 100,000 unlabeled samples in the training set. Although Semi-supervised learning improves the performance of GNN on many testing sets., it does not achieve its main goal of reducing the number of labeled samples in the training set.

with squares) is better than supervised learning (blue curve with triangles) on 10/13 test cases. In particular, in test case #1 that contains TSN configurations with the same patterns as in the training set, semi-supervised learning shows a decrease in the prediction accuracy of more than 4%. As the main purpose of semi-supervised learning is to reduce the number of labeled samples, it does not achieve the goal. We also note that training GNN with semi-supervised learning is much costly than supervised learning. Since the number of unlabeled samples is 10 times more than labeled samples, the training phase of semi-supervised learning is at least longer 10 times than supervised learning, in addition to the execution time to generate pseudo-labels after each training iteration. In conclusion, semi-supervised learning provides an extra trade-off between performance and training time, while we cannot guarantee that it will outperform supervised learning in all cases. Future studies may find a better method to apply semi-supervised learning, for instance, update pseudo-labels after several iterations instead of each iteration as in this experiment.

5.6 Self-supervised learning (SSL) for GNN model

In this section, we explore possibilities to further improve the performance of GNN with an advanced deep learning technique beyond supervised learning, namely self-supervised learning (SSL) that can learn to generate useful features from unlabeled TSN configurations. Although the results presented in this section are negative, *i.e.* the use of SSL leads to an overall performance that is worse than the baseline GNN model, we consider SSL as a technique that has the potential to further enhance the performance of GNN with the use of non-labeled training data .

5.6.1 Introduction to SSL

The concept of SSL

Recently, self-supervised learning (SSL) has become one of the mainstream research topics in ML due to its ability to leverage the massive amount of non-labeled data that is available [117]. SSL is a deep learning paradigm, which has two separate training phases:

- In the first phase, the SSL model learns to generate useful features of training data. The ultimate goal of this phase is to push apart the representative vectors of each data point in the feature space.
- In the second phase, a classifier is applied to predict labels of data (represented by synthetic feature vectors with large distances). Therefore, the classification task with SSL is simpler than without SSL.

The intuition of SSL is illustrated with an example in the following. The problem under study is object recognition: how to identify that an image contains either a dog or a cat? To train an ML model with supervised learning, a set of labeled samples needs to be available. In contrast, the SSL model is trained to obtain the distinct representative vector of each sample in the first phase, then use these synthetic feature vectors for classification in the second phase. Figure 5.11 illustrates the method to train an SSL model with an example (from [8]). Given two images of a cat (denoted by C0) and a dog (D0) as shown in Figure 5.11 (a), we capture two points of view C1 and C2 from C0, and a point of view D1 from D0. Since C1 and C2 are actually from the same object, their representative vectors

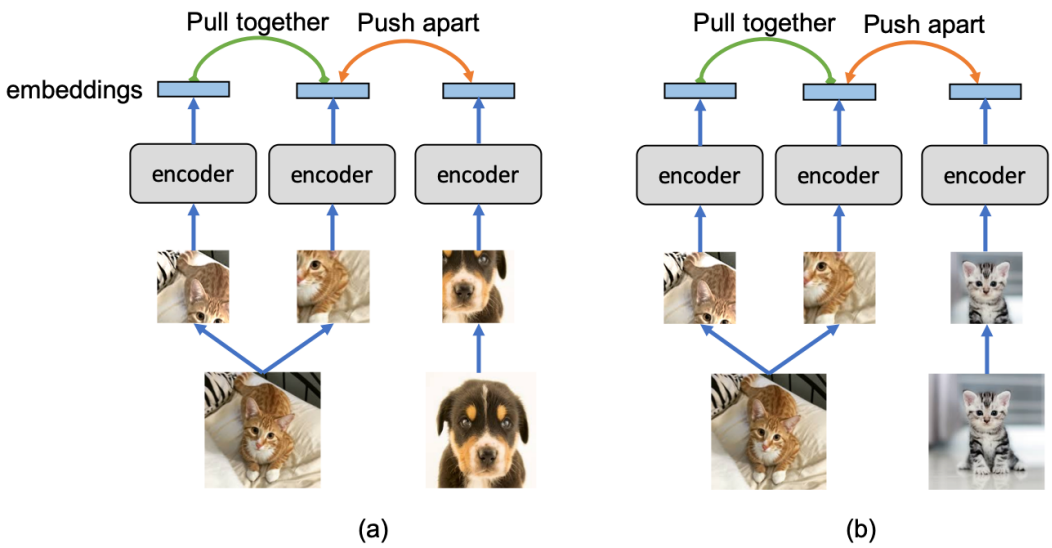


Figure 5.11: An example of SSL: On the left hand-side, there are two points of view of the same cat's image as positive samples, and a point of view of the dog's image as a negative sample. An encoder is used to generate three embedding vectors. The role of the encoder is to maximize the similarity between embeddings of positive samples while minimizing the similarity between positive and negative samples. The figure on the right hand-side works in the same manner. This figure is from [8].

in the feature space should be closer together, ideally identical. On the other hand, the representative vector of D1 should be away from both C1 and C2 as they are from different objects. We note that the encoder used to make representative vectors (*i.e.* synthetic feature vector or embedding) of all three points of view C1, C2, and D1 is the same deep learning structure, for instance, a convolutional neural network for images. The loss function needs to be well chosen in order to achieve the goal of this training phase.

Why do we need to push apart embeddings corresponding to the pictures of two cats, as shown in Figure 5.11 (b)? This is because if the distance between any pair of samples becomes larger, it will lead to a feature space of lower density. In other words, any data point will occupy a larger distinct area in the feature space. Accordingly, a classifier can further improve the accuracy due to the large boundary between data points and therefore larger boundary between groups of data in the feature space.

The concept of SSL is also popular in practice. For instance, it can easily recognize an object although only part of it is shown. The reason is that usually an object is a combination of some distinct patterns. An SSL model can only enhance the distance between objects if it learns these patterns, which will be embedded in the corresponding

feature vectors. Therefore, in the second phase of SSL, a classifier can utilize these embeddings to predict the classes of data.

SSL architectures

SSL has been already widely used in object recognition, for instance the Siamese network [118]. However, this learning paradigm has not been considered as a stand-alone research topic until a standard framework called SimCLR [117] was introduced. This framework introduces the concept of positive samples that belong to the same object and negative samples from different objects, similar to the example in Figure 5.11. The role of the framework is to maximize agreement between positive samples and minimize agreement between positive-negative samples, thus this learning method is called contrastive learning. It is important to note that in [117], the authors do not mention the need for different groups of data. This indicates that SSL does not require any information about the labels of samples in the training set. For instance, if the training set contains images of dogs and cats, these two groups are not identified and separated by SSL. Positive samples are thus many different points of view of the same image, while negative samples are points of view of different images. This framework showcases an important advantage of SSL, which is that it just requires unlabeled training samples, that are much easier to obtain in practice.

The limitation of SimCLR is its sensitivity to the way the different points of view of the same data are obtained. In object recognition, an image can be cropped or rotated to generate positive samples, however the performance of SSL could be heavily affected by the quality of these transformation methods. In [119], the authors introduce a new SSL approach called BYOL that does not use positive-negative samples during training. The authors instead use a pair of teacher-student encoders: the teacher is actually the old student. During the training, the student network is updated while the parameters of the teacher network are fixed. Given a training sample, the teacher and the student generate two different points of view of the same data, however from the temporal perspective instead of a spatial perspective. After several training iterations, the teacher network is updated by the parameters of the student network. While the BYOL approach eliminates the need of contrastive samples, it requires to carefully choose the time step to update the

teacher network, otherwise the student network becomes too advanced and the SSL model cannot reach the agreement between two representative vectors.

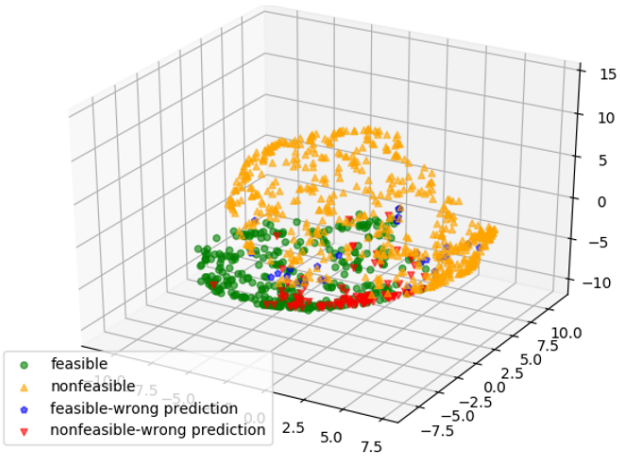
5.6.2 Enhancing the GNN model with SSL

Many applications of SSL to GNN have been proposed (see [120] for a survey of these applications). Most of them apply contrastive learning from [117], which requires the generation of positive-negative pairs of samples. Some popular methods to generate different points of view for a graph are:

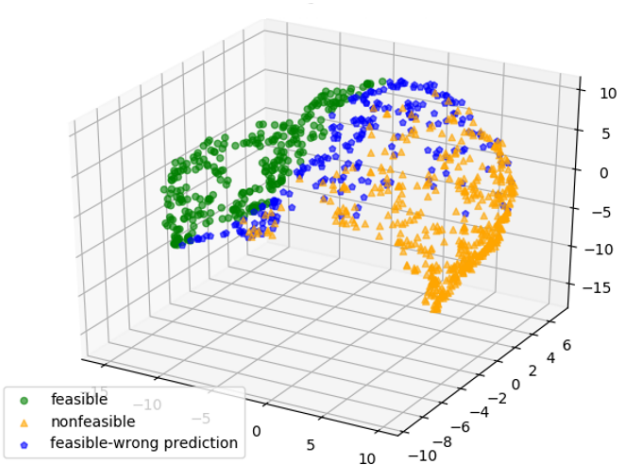
- masking one or more nodes in the graphs,
- masking some edges between nodes,
- extract a sub-network,

or a combination of these methods. We note that these methods are more appropriate for a large graph as changes in some nodes and edges only make a small effect on the overall status of the graph.

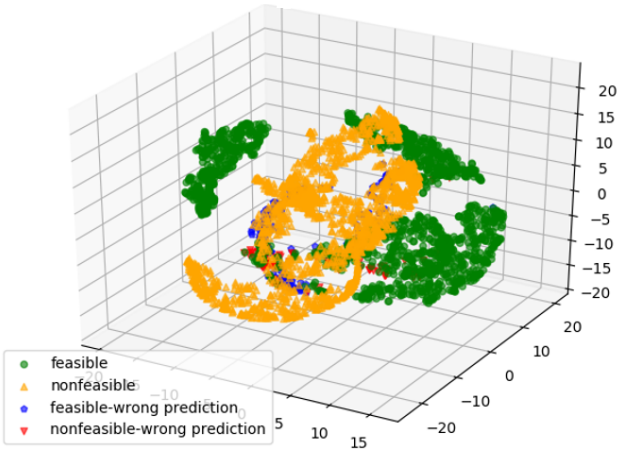
In our study, the SimCLR framework [117] is adopted for the sake of simplicity. The encoder in SSL is the improved GNN introduced in previous sections, however, the final neural network for feasibility prediction is removed. We note that some improvements of GNN in this chapter such as the global pooling layer are not used as the encoder needs to produce embedding vectors, *i.e.* the global attribute in our case. For each unlabeled TSN configuration in the training set, we generate a pair of positive samples by masking some nodes in the corresponding graph. That means for some chosen nodes, their initial embeddings are set to zeros. Instead we do not mask any edges in the graphs, due to the fact that the number of edges is usually small and thus masking edges can change the status of the graph in a non-trivial way. After being trained, a k-NN classifier will use the embeddings produced by SSL to predict the feasibility of TSN configurations. In this section, we use the same training set as the one used for semi-supervised learning in Section 5.5, which contains 100,000 unlabeled configurations for the first phase and 10,000 labeled configurations for the second phase of SSL.



((a)) Testing set 6 - Accuracy: 87.7%



((b)) Testing set 7 - Accuracy: 77.4%



((c)) Testing set 13 - Accuracy: 92.7%

Figure 5.12: Visualization of embeddings that represent TSN configurations on some testing sets. These embeddings are produced by the SSL model after being trained with 100,000 unlabeled configurations. The 3D plots are generated by t-SNE available in Python.

5.6.3 Experimental results and discussions

In Figure 5.12, we visualize the embeddings synthesized by the SSL model after being trained with 100,000 non-labeled samples, using the t-SNE visualization tool available in Python. t-SNE [121] applies a particular dimension reduction algorithm to show only important features in the 3D plots. Results for three test cases are shown in the figure, together with the prediction accuracies when applying the k-NN classifier. The green dots and yellow triangles represent true positives and true negatives, respectively. The blue pentagons are false negatives (feasible configurations being wrongly predicted as non-feasible), and the red triangles are false positives (non-feasible configurations being wrongly predicted as feasible). The embeddings of feasible configurations are clearly separated from non-feasible ones in Figure 5.12(c), which leads to a high accuracy. However, the separation is vague in Figure 5.12(a), in particular there is an area of embeddings that is hard to segment in Figure 5.12(b), consequently both testing sets have low prediction accuracies. A large variability in the results can also be observed in other testing sets. For some test cases, SSL helps to separate the feasible and non-feasible configurations, which contributes to an increase in accuracy. However, in other test cases, the separation between the two groups is not clear and thus some mixed areas still exist, similar to the gray areas in Chapter 3 that cause k-NN to have poor performance. We note that Figure 5.12 only shows the results of an individual SSL model. An ensemble of 32 individual SSL models has also been explored, which however provides inferior results compared to supervised learning and thus is not shown here.

While SSL actually improves the prediction accuracy in some test cases, it causes worse performance in most cases. We suspect that the transformation process to generate contrastive samples is the reason for inferior results. Since each graph encodes a real-time TSN configuration, masking a node in the graph means removing a component in the network, either flows, links or queues. This affects the timing feature of a real-time system significantly and thus the feasibility of the network can totally change. As a result, a pair of positive samples may have different feasibility, which violates the assumption of SSL that they are different views of the same object. To the best of our knowledge, there is no SSL method that can overcome this difficulty. Besides SimCLR, we also experiment with other SSL techniques but they cannot produce better results. Therefore, we think that SSL

is not the best approach for predicting the feasibility of real-time networks at this moment.

5.7 Related works

Since GNN has become a mainstream research topic in ML, many studies have focused on improving its performance. Graph Convolutional Nets [49] is a GNN model that imitates the well-known convolutional neural network in image recognition. In [122], the authors apply an attention layer on top of GNN, as this layer allows each node to "pay attention" to the important neighbor edges or nodes. In order to enhance the ability of GNN in classification, a study in [123] proposes to add a K-means clustering layer on top of GNN. These studies provide more powerful tools for the developer to build a better GNN model.

The application of semi-supervised learning to GNN can be found in many studies. For instance, in [124], the authors analyze the effectiveness of applying semi-supervised learning in the Graph Convolutional Nets proposed by [49]. Beyond the scope of GNN, semi-supervised learning is one of the most promising techniques in deep learning to overcome the bottleneck of labeled samples. Many advanced deep learning models require millions of labeled data to reach state-of-the-art performance. The labeling process is done manually, which incurs a huge cost and effort. In contrast, unlabeled data is usually available on the Internet or can be generated in only a fraction of the time needed to generate labeled data. Therefore, it is natural for the researchers to leverage the advantage of the massive amount of unlabeled data. For instance, in [115], hundred millions of unlabeled images are collected from the Internet and combined with a million labeled images in the ImageNet data set, which improve the prediction of the deep learning model by 2%.

In recent years, self-supervised learning has reached the performance of supervised learning in several areas. Many SSL techniques are applied to GNN, as surveyed in [120]. We note that SSL only learns the representatives of the input data, in particular, it enlarges the distance between any pair of data in the same manner. What should we do if we also want to enlarge the distance between two groups of data in order to facilitate the classification? In [125], the authors propose a supervised contrastive learning model that leverages the power of SSL in representative learning. A clustering algorithm is used on

top of SSL that "guides" it to construct the representative vectors towards classifying two or several data groups. In our perspective, SSL is a major technique for a deep learning model to achieve a higher level of intelligence, due to its ability to enhance the usefulness of synthetic features. We observe that in supervised learning, for each training data, there is only one bit of information transferred to the DL model through the label. In SSL, as we compare the input data, the information transferring to the DL model is richer, which gives DL more opportunities to learn the patterns and the complex relations between them.

5.8 Conclusion and future work

In this chapter, to answer research question **RQ4** in Section 1.2, we improve the prediction accuracy of the GNN model through progressive experiments. As a result, this deep learning approach becomes more practical to serve as a viable alternative to schedulability analysis in the verification of Ethernet TSN networks.

For the training set, random-shaped networks replace star-shaped networks, traffic statistics are more diverse, and we also generate more training configurations that are hard to predict. Advanced deep learning techniques are integrated into the GNN model, including a more expressive activation function, the global pooling layer and a large embedding vector as well. A combination of these changes into an expanded GNN model, trained with an expanded training set, shows a significant improvement in terms of prediction accuracy over the base model. Indeed, with these improvements, 9 out of 13 test cases have a prediction accuracy higher than 90% and no case less than 82%.

The main drawback of the improvements of the base GNN model is the increase in execution time and memory usage. In addition, one test case experiences a drop in performance, which suggests that traffic patterns should be even more diverse. The size and structure of neural networks may also have an important impact on the model's performance, these dimensions have still not been explored in this study. Besides, recent deep learning studies introduce novel methods such as the attention mechanism on GNN [126] and the focal loss function [127], which accelerates the learning capability of neural networks. These are the promising directions for our ongoing work to further improve the performance of the GNN model.

Chapter 6

Conclusion

Abstract

This chapter summarizes the works conducted during the thesis, which have addressed the research questions listed in the introduction. Although GNN has been introduced to speed up DSE of Ethernet TSN networks, we only apply it to predict the feasibility of TSN configurations with priority assignment. Future studies can extend the application of GNN to other TSN scheduling mechanisms, as well as to other problems arising in real-time systems. We also highlight the importance of an ML model that can accumulate knowledge from multiple tasks to achieve a high level of transfer learning. In our point of view, the ability to leverage skills of different tasks is the key to solving major problems in real-time systems effectively.

6.1 Summary of the thesis work

In the studies performed in the thesis, we have answered the four research questions presented in Section 1.2 with the objective to further automate the design of real-time TSN networks with the use of artificial intelligence in design-space exploration.

The first contribution, to answer **RQ1**, is the use of the simple k-NN algorithm to predict the feasibility of TSN networks when configured with four distinct TSN scheduling policies: FIFO, manual priority assignment, near-optimal priority assignment derived by the Concise Priorities algorithm (CP8), and Pre-shaping. The input of the ML algorithm are five manually-selected features relevant to the feasibility patterns of TSN configurations.

The k-NN algorithm shows its ability to predict feasibility with high accuracy on both simple TSN policy like FIFO (95.94% accuracy) as well as complex policies like priority assignment and Pre-shaping (91.84% accuracy).

However, the weakness of conventional ML algorithms like k-NN is also clear: they are too simple to capture complex relations in TSN networks, and therefore they can only work on the network topologies used during the training. In addition, the manual feature selection approach cannot capture all useful features. To overcome these limits, we propose a graph neural network (GNN) model that can take as inputs the raw data of a configuration and encode it as a graph, which provides an answer to **RQ3**. The encoding step is an important step to convert a TSN configuration to euclidean data that can be used by ML classification algorithms: flows, links, and queues in a TSN configuration are encoded by nodes in the graph and represented by embedding vectors, while topology and routing are represented by connections between nodes. A message-passing algorithm is applied to propagate information among nodes in the graph, exploiting thus complex relationships among components in the corresponding network. As a result, the main advantage of the proposed GNN model is that it can generalize to a wide range of topologies and traffic patterns. The model has been successfully tested on 13 realistic test cases consisting of TSN configurations with diverse topologies, such as center-based topology, backbone, ring, and so on. An ensemble of 32 GNN models gives a prediction accuracy ranging from 79.3% to 90% depending on the test cases, while the speedup factor over conventional schedulability analysis ranges from 77 to 1715 times.

More advanced deep learning techniques, such as the use of more expressive activation functions, the addition of a global pooling layer, as well as enlarging the neural networks making up the GNN model, have been explored to further improve the overall prediction accuracy, which addresses **RQ4**. With those improvements, the prediction accuracy obtained on the 13 test cases is significantly improved, now ranging from 81.1% to 95.4%. In addition, efforts have also been put into building a better training set that not only involves more randomness but also generates more training configurations whose feasibility are hard to predict, enhancing thus the prediction capability of the GNN model. Indeed, the prediction accuracy increases by up to 11.9% while the speed up factor over schedulability analysis still remains from 50 to 1125 times. These applications of GNN

suggest that AI can be, in some settings at least, a viable verification method for feasibility tests. Furthermore AI-based feasibility tests can be used either in alone or combined with conventional schedulability analyses in a hybrid approach (to answer **RQ2**). In conclusion, the studies in the thesis contribute to speeding up DSE with AI-assisted algorithms when designing Ethernet TSN networks, and may provide insights to tackle other difficult problems in real-time systems with AI approaches.

6.2 Future research directions

While the application of GNN to Ethernet TSN networks has shown significant results in Chapter 4 and Chapter 5, it should be noted that generating the training sets is a time-consuming task that requires domain knowledge. As a result, future studies should try to mitigate this problem by applying novel learning paradigms, which require fewer training samples. In addition, we can expand the application of GNN to TSN mechanisms beyond priority assignment, which was the main focus of this thesis.

6.2.1 Further enhancement of GNN model

In the future, we will consider new methods to reduce the efforts needed to generate the training sets for GNN. Our research group has spent 6 months only for generating labeled synthetic configurations. In the thesis, we also apply semi-supervised and self-supervised learning to reduce the number of labeled configurations needed, however, the results obtained are modest so far. With the acceleration of deep learning research this topic [128, 129], SSL may outperform supervised learning in the future and allow GNN to predict with more accuracy. In a recent study [130], the authors introduce a novel approach to train an ML model that works with multiple tasks, each task being a classification problem. In this approach, there are many individual ML models, each one is trained with a specialized task then generates pseudo-labels, a.k.a. the ML models predict the output of training samples [1]. The final ML model is trained with these pseudo-labels, and thus the knowledge from individual ML models is transferred to the final model. This promising approach can also be applied to real-time TSN networks. In all experiments, we generate

[1] Pseudo-labels are not true labels of the samples but labels generated, i.e. predicted, by the ML models.

TSN configurations for the training set randomly. However, we can leverage more expertise on real-time systems and TSN by considering the feasibility prediction with each topology as a specialized task and applying the approach in [130]. That means we do not have to generate a massive amount of topologies but only some typical topologies. An ML model is needed for each task that classifies feasible and non-feasible configurations with fixed topology, then generates pseudo-labels. A final ML is trained with all configurations using pseudo-labels. The advantage of this approach is that designer can control the training process, and it potentially reduces the number of labeled training samples needed. We are excited to see whether this approach can reach the same performance as training with standard training data.

6.2.2 Diversify application of GNN in TSN networks

Another research direction is to apply the GNN model to other TSN mechanisms and predict which mechanisms, or combination of mechanisms, can provide feasible scheduling for a TSN network. This is useful since the application of GNN in this thesis is still limited to priority assignment, while other TSN mechanisms may provide better performance, as shown with Pre-shaping in Chapter 3. The main challenge for this research problem is to train a distinct GNN model for each TSN mechanism, corresponding to a distinct training set. As it takes at least one month to generate a training set for a single TSN mechanism, we anticipate that this task will take several months only for the generation of training sets. In addition, keeping a GNN model for each TSN mechanism implies more memory and computing resource. In this scenario, the approach in [130] may be useful, as we can train distinct GNN models with specialized tasks and apply transfer learning to the final GNN model.

Beyond feasibility prediction, there are many areas in real-time networking that can be automated with AI. For instance, given a TSN configuration, the OPA or its variant algorithms are applied to derive the optimal priority allocation solutions. However, this process is compute-intensive, as it requires many feasibility tests. As GNN has been applied to predict the results of feasibility tests in this thesis, it is natural to expand its scope of application to priority allocation. Deep Reinforcement Learning is an ML paradigm which is suitable to address such problems, as it learns to perform the set of

actions that maximizes rewards in the future. This is equivalent to decide whether the priority level of a flow should be increased or lowered down in order to maximize the number of flows that satisfy deadline constraints. However, we note that deriving a near-optimal priority allocation solution takes on average 8 steps [2], which means 8 feasibility tests. Even in the case that GNN achieves a high prediction accuracy of about 90% for each feasibility test, as in Chapter 5, the overall prediction accuracy for priority assignment could drop to merely $(90\%)^8 = 43\%$ after 8 consecutive tests. In conclusion, we need to simultaneously improve the performance of GNN and find the best methods to apply GNN to other areas in real-time systems, which may involve re-designing some configuration algorithms like the example given suggests.

6.2.3 Final thoughts

Artificial Intelligence, re-vitalized since 2012 with deep learning, is progressing quickly. However, the main challenge in AI is that learning multiple tasks to reach general AI is still not solved effectively. Let's assume that an engineer learns how to determine the feasibility of TSN configurations using the priority assignment mechanism. This person then learns to determine the feasibility with CBS and Preshaping. The person can certainly leverage what she/he has learned in the previous task with priority assignment to the new tasks with CBS and Preshaping, and therefore reduce the learning time. In the other direction, the person can use what he has learned with CBS and Preshaping to enhance his knowledge of priority assignment. With more and more tasks mastered, the person will accumulate knowledge and accelerate the learning speed. Currently, we still need to train an ML model for separate tasks in real-time systems, which is very time-consuming. Future studies should pursue the direction of achieving a high level of transfer learning. The ability to accumulate knowledge and leverage skills across a large number of tasks is, in our opinion, the key to solving problems in real-time systems more effectively.

[2] Our empirical experiments show that the Concise Priorities algorithm, available in the Pegase software used in this thesis, takes on average 8 steps to derive the near-optimal priority assignment for a TSN configuration.

6.3 On the use of our works in the industry

Although AI is a new research field, an increasing body of good practices has been accumulated over years. This section discusses how to utilize the studies done during this thesis to the industry in an effective manner.

6.3.1 Large training set and Active learning

The size of the training set is the single most important factor that affects the performance of an AI model. This conclusion is very popular among AI practitioners, however, we argue that just collecting as much data as possible at the beginning of a project is not necessarily a good practice as evidenced in the following cases:

- During the improvements of the GNN model in Chapter 5, we have shown that changes in the training set can improve the prediction accuracy of the GNN model up to 11.2%. That means not only the training set size but also appropriate training data can strongly affect the results of the model.

- Generating a ”small” training set with 10.000 TSN configurations takes up to two months on a desktop PC (with full CPU usage). During the thesis, our research group only needed to change the training data from center-based topologies to random topologies. We expect that if more important changes are needed, a project may take more time just to generate appropriate training data.

We therefore suggest *active learning* for the application of our studies in industry. This learning paradigm can be done with the following steps.

- It starts with a collection of non-labeled TSN configurations, which can be done with very little efforts.

- Then one uses schedulability analysis to label a small number of training samples, for instance, 100 configurations.

- These labeled configurations are used to train the GNN model, then the model will predict the feasibility of non-labeled configurations. The results are not only

the feasibility predictions but also the confidence of the model about each of its predictions.

- We select some configurations with a low prediction confidence, *i.e.* configurations that the model found very difficult to predict, and label these configurations with schedulability analysis. They are then merged with the existing labeled samples to become the new training set.

- The process repeats until some predefined criteria are met: training time, performance of the GNN model, and so on.

With active learning, we can also discover which topologies or traffic patterns are hard to predict and fine-tune the training set accordingly. In doing so, we reduce the risk of using resources inefficiently, which reminds us of the Agile methodologies in software development.

6.3.2 Domain expertise

We note that domain knowledge has a crucial role during the development of the GNN model, in particular the selection of the samples that are appropriate for the training set. As mentioned above, the generation of the labeled TSN configurations is compute-intensive, thus we need to carefully choose the topologies and traffic patterns before generating the training samples. For instance, in Chapter 4, the low prediction accuracy for backbone-shaped networks suggests that more such types of networks need to be added into the training set. This is a use-case for active learning, if some configurations are hard to predict, then similar TSN networks need to be added to the training set.

Although GNN eliminates the manual selection approach, manual features chosen based on domain expertise can still be useful in practice. For instance, we can embed these features into the initialization of the global attribute vector to directly inform the GNN about important characteristics of the network. This step can be valuable in the context of a small training set with only e.g. 10.000 configurations, which may not be enough for GNN to exploit all useful information, and thus manual features could represent a worthy additional source of information.

6.3.3 Advanced GNN block

We note that the studies of GNN in this thesis do not use sophisticated ML models. Indeed, in chapter 4, and we have intentionally built the GNN model using only fully-connected neural networks and message passing learning algorithms. The performance of the GNN model is improved in Chapter 5 mostly by a larger training set and an increase of the size of the neural networks. Although the prediction accuracy of the GNN model improves up to 11.2%, the training time for an ensemble of 32 individual GNN models increases from 15 minutes to around 3 hours. Therefore, the application of GNN to the industry can consider more powerful recurrent neural networks to further improve performance and reduce the training time. Gated Recurrent Unit (GRU) and Long-Short Term Memory (LSTM) are two mature techniques that have been used for GNN in networking (see [14] and [15]). These recurrent neural networks can be applied to our GNN model with minimal effort.

Importantly, the development of a GNN model for TSN networks requires not only domain expertise but also background about ML. Such barriers hinder the application of AI to solve more problems in real-time systems. In this context, frameworks such as IGNNITION [131] have been introduced, which is built on top of current deep learning libraries to facilitate the programming of a GNN model. In IGNNITION, GRU and LSTM are included together with other GNN blocks. The message-passing algorithm and other state-of-the-art learning methods are also added to the framework, which further reduces the gap between GNN and networking, and real-time systems in general.

6.4 Discussion and perspective

This chapter summarizes the studies in the thesis towards speeding up the design space exploration of Ethernet TSN networks with artificial intelligence. The first study is the application of conventional machine learning algorithms to predict the feasibility of TSN networks with high prediction accuracy. The hallmark of the thesis is Graph Neural Network that can model TSN networks by encoding not only physical and logical components but also complex relationships in the networks, thus it can generalize the prediction ability to a wide range of topologies and traffic patterns. Future studies can focus on the performance improvement of the GNN model with better learning techniques

that require fewer training samples, which reduces the cost of generating large training sets. More importantly, the application of GNN can be expanded to more TSN mechanisms, which we consider as one of the most important research directions in the near future. These extensions imply the construction of multiple GNN models, one for each TSN mechanism, with the proposed learning paradigm. Last but not least, an even more powerful model that can predict the feasibility of a configuration for all TSN mechanisms may be developed. This would open the door to the possibility of predicting the best TSN scheduling solution for a given configuration.

Bibliography

[1] N. Navet, T. Mai, J. Migge, Using Machine Learning to Speed Up the Design Space Exploration of Ethernet TSN Networks, Tech. rep., University of Luxembourg (2019).
URL http://hdl.handle.net/10993/38604

[2] T. Mai, N. Navet, J. Migge, On the Use of Supervised Machine Learning for Assessing Schedulability: Application to Ethernet TSN, in: Proceedings of the 27th International Conference on Real-Time Networks and Systems, 2019, pp. 143–153.

[3] T. Mai, N. Navet, J. Migge, A Hybrid Machine Learning and Schedulability Analysis Method for the Verification of TSN Networks, in: 2019 15th IEEE International Workshop on Factory Communication Systems (WFCS), IEEE, 2019, pp. 1–8.

[4] T. Mai, N. Navet, Deep Learning to Predict the Feasibility of Priority-based Ethernet Network Configurations, ACM Transactions on Cyber-Physical Systems (2021).

[5] T. Mai, N. Navet, Improvements to Deep-Learning-based Feasibility Prediction of Switched Ethernet Network Configurations, in: The 29th International Conference on Real-Time Networks and Systems (RTNS2021), 2021.

[6] N. Navet, F. Simonot-Lion, Automotive Embedded Systems Handbook (2008).

[7] J. Migge, J. Villanueva, N. Navet, M. Boyer, Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks, in: Embedded Real-Time Software and Systems (ERTS 2018), Toulouse, France, 2018.
URL https://hal.archives-ouvertes.fr/hal-01746132

[8] L. Junnan, Prototypical Contrastive Learning: Pushing the Frontiers of Unsupervised Learning, retrieved 2021/09/08, https://blog.einstein.ai/prototypical-contrastive-learning-pushing-the-frontiers-of-unsupervised-learning/.

[9] A. Rivera, E. Bravo, M. Jimenez, R. Palomera, Design Review and Innovations in Low-voltage Differential Signaling Drivers, in: The 2004 47th Midwest Symposium on Circuits and Systems, 2004. MWSCAS'04., Vol. 3, IEEE, 2004, pp. iii–339.

[10] A. Nasrallah, A. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, H. El-Bakoury, Ultra-low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research, IEEE Communications Surveys & Tutorials 21 (1) (2018) 88–145.

[11] M. Ashjaei, L. Bello, M. Daneshtalab, G. Patti, S. Saponara, S. Mubeen, Time-Sensitive Networking in Automotive Embedded Systems: State of the Art and Research Opportunities, Journal of Systems Architecture (2021) 102137.

[12] C. Mauclair, M. Gutiérrez, J. Migge, N. Navet, Do We Really Need TSN in Next-Generation Helicopters? Insights From a Case-Study, in: 2021 AIAA/IEEE 40th Digital Avionics Systems Conference (DASC), IEEE, 2021.

[13] C. Braun, L. Havet, N. Navet, NETCARBENCH: a Benchmark for Techniques and Tools Used in the Design of Automotive Communication Systems, IFAC Proceedings Volumes 40 (22) (2007) 321–328.

[14] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, A. Cabellos-Aparicio, Unveiling the Potential of Graph Neural Networks for Network Modeling and Optimization in SDN, in: Proceedings of the 2019 ACM Symposium on SDN Research, 2019, pp. 140–151.

[15] F. Geyer, Performance Evaluation of Network Topologies Using Graph-based Deep Learning, in: Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools, 2017, pp. 20–27.

[16] Plexxi, Latency in Ethernet Switches, retrieved 2019/04/25, http://www.plexxi.com/wp-content/uploads/2016/01/Latency-in-Ethernet-Switches.pdf (2016).

[17] M. Houtan, B.and Ashjaei, M. Daneshtalab, M. Sjödin, S. Afshar, S. Mubeen, Schedulability Analysis of Best-Effort Traffic in TSN Networks, in: 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2021, pp. 1–8.

[18] RealTime-at-Work, RTaW-Pegase: Modeling, Simulation and Automated Configuration of Communication Networks, retrieved 2019/01/24, https://www.realtimeatwork.com/software/rtaw-pegase.

[19] L. Maile, K. Hielscher, R. German, Network Calculus Results for TSN: An Introduction, in: 2020 Information Communication Technologies Conference (ICTC), IEEE, 2020, pp. 131–140.

[20] M. Boyer, N. Navet, X. Olive, E. Thierry, The PEGASE Project: Precise and Scalable Temporal Analysis for Aerospace Communication Systems with Network Calculus, in: International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, Springer, 2010, pp. 122–136.

[21] N. Navet, J. Seyler, J. Migge, Timing Verification of Real-time Automotive Ethernet Networks: What Can We Expect from Simulation?, in: 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Toulouse, France, 2016. URL https://hal.archives-ouvertes.fr/hal-01292312

[22] IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks, IEEE Std 802.1Q-2018 Edition (2018).

[23] R. Davis, L. Cucu-Grosjean, M. Bertogna, A. Burns, A Review of Priority Assignment in Real-Time Systems, Journal of Systems Architecture 65 (C) (2016) 64–82. doi:10.1016/j.sysarc.2016.04.002.
URL https://doi.org/10.1016/j.sysarc.2016.04.002

[24] T. Hamza, J. Scharbarg, C. Fraboul, Priority Assignment on an Avionics Switched Ethernet Network (QoS AFDX), in: 2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014), IEEE, 2014, pp. 1–8.

[25] L. Zhao, Q. Li, Y. Xiong, Z. Zheng, H. Xiong, Using Multi-Link Grouping Technique to Achieve Tight Latency in Network Calculus, in: 2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC), IEEE, 2013, pp. 2E3–1.

[26] IEEE Standard for Local and Metropolitan Area Networks – Virtual Bridged Local Area Networks Amendment 12 Forwarding and Queuing Enhancements for Time-Sensitive Streams, IEEE Std 802.1Qav-2009 Edition (January 2009).

[27] N. Navet, J. Migge, J. Villanueva, M. Boyer, Pre-shaping Bursty Transmissions under IEEE802.1Q as a Simple and Efficient QoS Mechanism, SAE Int. J. Passeng. Cars – Electron. Electr. Syst. 11 (04 2018). doi:10.4271/2018-01-0756.

[28] IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic, IEEE Std 802.1Qbv-2015 Edition (March 2016).

[29] H. Kopetz, G. Bauer, The Time-Triggered Architecture, Proceedings of the IEEE 91 (1) (2003) 112–126.

[30] W. Steiner, An Evaluation of SMT-based Schedule Synthesis for Time-Triggered Multi-hop Networks, in: 2010 31st IEEE Real-Time Systems Symposium, IEEE, 2010, pp. 375–384.

[31] S. Craciunas, R. Oliver, M. Chmelík, W. Steiner, Scheduling Real-time Communication in IEEE 802.1 Qbv Time Sensitive Networks, in: Proceedings of the 24th International Conference on Real-Time Networks and Systems, 2016, pp. 183–192.

[32] IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption, IEEE Std 802.1Qbu-2016 Edition (August 2016).

[33] D. Thiele, R. Ernst, Formal Worst-Case Performance Analysis of Time-Sensitive Ethernet with Frame Preemption, in: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2016, pp. 1–9.

[34] S. J. Russell, P. Norvig, E. Davis, Artificial Intelligence: A Modern Approach, 3rd Edition, Prentice Hall, 2010.

[35] K. Friston, The free-energy principle: A unified brain theory?, Nature reviews neuroscience 11 (2) (2010) 127–138.

[36] J. Hawkins, S. Blakeslee, On intelligence, Macmillan, 2004.

[37] P. Domingos, The Master Algorithm: How the Quest for the Ultimate Learning Machine will Remake Our World, Basic Books, 2015.

[38] J. Clune, AI-GAs: AI-Generating Algorithms, an Alternate Paradigm for Producing General Artificial Intelligence, arXiv preprint arXiv:1905.10985 (2019).

[39] C. Bishop, Pattern Recognition, Machine Learning 128 (9) (2006).

[40] Nvidia, What is A Machine Learning Model?, retrieved 2022/05/01, https://blogs.nvidia.com/blog/2021/08/16/what-is-a-machine-learning-model/ (2021).

[41] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[42] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the Game of Go with Deep Neural Networks and Tree Search, Nature 529 (7587) (2016) 484–489.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research 12 (Oct) (2011) 2825–2830.

[44] P. Battaglia, J. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., Relational Induc-

tive Biases, Deep Learning, and Graph Networks, arXiv preprint arXiv:1806.01261 (2018).

[45] F. Scarselli, M. Gori, A. Tsoi, M. Hagenbuchner, G. Monfardini, The Graph Neural Network Model, IEEE Transactions on Neural Networks 20 (1) (2008) 61–80.

[46] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: A System for Large-scale Machine Learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.

[47] M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, Geometric Deep Learning: Going Beyond Euclidean Data, IEEE Signal Processing Magazine 34 (4) (2017) 18–42.

[48] A. Krizhevsky, I. Sutskever, G. Hinton, Imagenet Classification with Deep Convolutional Neural Networks, Advances in Neural Information Processing Systems 25 (2012) 1097–1105.

[49] T. Kipf, M. Welling, Semi-Supervised Classification with Graph Convolutional Networks, arXiv preprint arXiv:1609.02907.

[50] J. Gilmer, S. Schoenholz, P. Riley, O. Vinyals, G. Dahl, Neural Message Passing for Quantum Chemistry, in: International Conference on Machine Learning, PMLR, 2017, pp. 1263–1272.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, I. Polosukhin, Attention is All You Need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.

[52] X. Wang, R. Girshick, A. Gupta, K. He, Non-Local Neural Networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7794–7803.

[53] M. Allamanis, E. Barr, P. Devanbu, C. Sutton, A Survey of Machine Learning for Big Code and Naturalness, ACM Computing Surveys 51 (4) (2018) 81:1–81:37. URL http://doi.acm.org/10.1145/3212695

[54] A. Ashouri, W. Killian, J. Cavazos, G. Palermo, C. Silvano, A Survey on Compiler Autotuning Using Machine Learning, CoRR abs/1801.04405 (2018). arXiv:1801.04405.
URL http://arxiv.org/abs/1801.04405

[55] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. Reyes, M. Shyu, S. Chen, S. Iyengar, A Survey on Deep Learning: Algorithms, Techniques, and Applications, ACM Computing Surveys (CSUR) 51 (5) (2018) 1–36.

[56] M. Wang, Y. Cui, X. Wang, S. Xiao, J. Jiang, Machine Learning for Networking: Workflow, Advances and Opportunities, IEEE Network 32 (2) (2018) 92–99. doi:10.1109/MNET.2017.1700200.

[57] W. Jiang, Graph-based Deep Learning for Communication Networks: A Survey, arXiv preprint arXiv:2106.02533 (2021).

[58] J. Suárez-Varela, S. Carol-Bosch, K. Rusek, P. Almasan, M. Arias, P. Barlet-Ros, A. Cabellos-Aparicio, Challenging the Generalization Capabilities of Graph Neural Networks for Network Modeling, in: Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos, 2019, pp. 114–115.

[59] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, A. Cabellos-Aparicio, RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN, IEEE Journal on Selected Areas in Communications 38 (10) (2020) 2260–2270.

[60] A. Badia-Sampera, J. Suárez-Varela, P. Almasan, K. Rusek, P. Barlet-Ros, A. Cabellos-Aparicio, Towards More Realistic Network Models Based on Graph Neural Networks, in: Proceedings of the 15th International Conference on emerging Networking Experiments and Technologies, 2019, pp. 14–16.

[61] M. Ferriol-Galmés, J. Suárez-Varela, K. Rusek, P. Barlet-Ros, A. Cabellos-Aparicio, Scaling Graph-based Deep Learning Models to Larger Networks, arXiv preprint arXiv:2110.01261.

[62] D. Pujol-Perich, J. Suárez-Varela, M. Ferriol-Galmés, B. Wu, S. Xiao, X. Cheng, A. Cabellos-Aparicio, P. Barlet-Ros, IGNNITION: Fast Prototyping of Graph Neural Networks for Communication Networks, Association for Computing Machinery, New York, NY, USA, 2021, p. 71–73.
URL https://doi.org/10.1145/3472716.3472853

[63] F. Geyer, DeepComNet: Performance Evaluation of Network Topologies Using Graph-based Deep Learning, Performance Evaluation 130 (2019) 1–16.

[64] F. Geyer, S. Schmid, DeepMPLS: Fast Analysis of MPLS Configurations Using Deep Learning, in: 2019 IFIP Networking Conference (IFIP Networking), IEEE, 2019, pp. 1–9.

[65] D. Awduche, MPLS and Traffic Engineering in IP Networks, IEEE Communications Magazine 37 (12) (1999) 42–47.

[66] M. Ferriol-Galmés, J. Suárez-Varela, P. Barlet-Ros, A. Cabellos-Aparicio, Applying Graph-based Deep Learning to Realistic Network Scenarios, arXiv preprint arXiv:2010.06686.

[67] F. Geyer, G. Carle, Learning and Generating Distributed Routing Protocols Using Graph-based Deep Learning, in: Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks, 2018, pp. 40–45.

[68] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.

[69] J. Suárez-Varela, A. Mestres, J. Yu, L. Kuang, H. Feng, P. Barlet-Ros, A. Cabellos-Aparicio, Routing Based on Deep Reinforcement Learning in Optical Transport Networks, in: 2019 Optical Fiber Communications Conference and Exhibition (OFC), IEEE, 2019, pp. 1–3.

[70] T. Salimans, J. Ho, X. Chen, S. Sidor, I. Sutskever, Evolution Strategies as a Scalable Alternative to Reinforcement Learning, arXiv preprint arXiv:1703.03864 (2017).

[71] T. Mai, N. Dao, M. Park, Real-time Task Assignment Approach Leveraging Reinforcement Learning with Evolution Strategies for Long-term Latency Minimization in Fog Computing, Sensors 18 (9) (2018) 2830.

[72] P. Almasan, J. Suárez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, A. Cabellos-Aparicio, Deep Reinforcement Learning Meets Graph Neural Networks: Exploring a Routing Optimization Use Case, arXiv preprint arXiv:1910.07421 (2019).

[73] P. Almasan, S. Xiao, X. Cheng, X. Shi, P. Barlet-Ros, A. Cabellos-Aparicio, ENERO: Efficient Real-Time Routing Optimization, arXiv preprint arXiv:2109.10883 (2021).

[74] P. Almasan, J. Suárez-Varela, B. Wu, S. Xiao, P. Barlet-Ros, A. Cabellos-Aparicio, Towards Real-Time Routing Optimization with Deep Reinforcement Learning: Open Challenges, arXiv preprint arXiv:2106.09754 (2021).

[75] G. Bernárdez, J. Suárez-Varela, A. López, B. Wu, S. Xiao, X. Cheng, P. Barlet-Ros, A. Cabellos-Aparicio, Is Machine Learning Ready for Traffic Engineering Optimization?, arXiv preprint arXiv:2109.01445 (2021).

[76] F. Geyer, S. Bondorf, DeepTMA: Predicting Effective Contention Models for Network Calculus Using Graph Neural Networks, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 1009–1017.

[77] F. Geyer, S. Bondorf, On the Robustness of Deep Learning-Predicted Contention Models for Network Calculus, in: 2020 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2020, pp. 1–7.

[78] F. Geyer, S. Bondorf, Graph-based Deep Learning for Fast and Tight Network Calculus Analyses, IEEE Transactions on Network Science and Engineering 8 (1) (2020) 75–88.

[79] F. Geyer, A. Scheffler, S. Bondorf, Tightening Network Calculus Delay Bounds by Predicting Flow Prolongations in the FIFO Analysis, in: Proceedings of the 27th IEEE Real-Time and Embedded Technology and Applications Symposium, 2021.

[80] J. Le Boudec, P. Thiran, Network Calculus: A Theory of Deterministic Queuing Systems for the Internet, Vol. 2050, Springer Science & Business Media, 2001.

[81] J. Migge, J. Villanueva, N. Navet, M. Boyer, Performance Assessment of Configuration Strategies for Automotive Ethernet Quality-of-service Protocols, in: Automotive Ethernet Congress, Munich, 2018.

[82] N. Navet, J. Villanueva, J. Migge, Automating QoS Protocols Selection and Configuration for Automotive Ethernet Networks, in: SAE World Congress Experience (WCX018), session "Vehicle Networks and Communication (Part 2 of 2)", Detroit, USA, 2018.

[83] Audsley, N.C., On Priority Assignment in Fixed Priority Scheduling, Information Processing Letters.

[84] N. Navet, J. Villanueva, J. Migge, M. Boyer, Experimental assessment of QoS protocols for in-car Ethernet networks, in: 2017 IEEE Standards Association (IEEE-SA) Ethernet & IP @ Automotive Technology Day, San-Jose, Ca, 2017. URL https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/other/d1-06_navet_experimental_assessment_of_qos_protocols_v2.pdf

[85] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer Science & Business Media, 2009.

[86] E. Dougherty, J. Hua, C. Sima, Performance of Feature Selection Methods, Current Genomics 10 (6) (2009) 365–374. doi:10.2174/138920209789177629.

[87] J. Li, K. Cheng, S. Wang, F. Morstatter, R. Trevino, J. Tang, H. Liu, Feature Selection: A Data Perspective, ACM Computing Surveys (CSUR) 50 (6) (2017) 1–45.

[88] F. Farris, The Gini Index and Measures of Inequality, The American Mathematical Monthly 117 (10) (2010) 851–864.

[89] M. McHugh, Interrater Reliability: the Kappa Statistic, Biochemia Medica: Biochemia Medica 22 (3) (2012) 276–282.

[90] H. Bauer, J. Scharbarg, C. Fraboul, Applying Trajectory Approach with Static Priority Queuing for Improving the Use of Available AFDX Resources, Real-Time Systems 48 (1) (2012) 101–133.
URL https://doi.org/10.1007/s11241-011-9142-9

[91] M. Boyer, N. Navet, M. Fumey, Experimental Assessment of Timing Verification Techniques for AFDX, in: 6th European Congress on Embedded Real Time Software and Systems, Toulouse, France, 2012.
URL https://hal.archives-ouvertes.fr/hal-01345472

[92] N. Navet, J. Migge, Insights Into the Performance and Configuration of TCP in Automotive Ethernet Networks, in: 2018 IEEE Standards Association (IEEE-SA) Ethernet & IP @ Automotive Technology Day, London, 2018.

[93] N. Navet, L. Cucu, R. Schott, Probabilistic Estimation of Response Times Through Large Deviations, in: Work-in Progress of the 28th IEEE Real-Time Systems Symposium (RTSS'2007 WiP), Tucson, United States, 2007.

[94] I. Juutilainen, J. Röning, A Method for Measuring Distance From a Training Data Set, Communications in Statistics - Theory and Methods 36 (14) (2007) 2625–2639. doi:10.1080/03610920701271129.

[95] D. Kazakov, I. Bate, Towards New Methods for Developing Real-time Systems: Automatically Deriving Loop Bounds Using Machine Learning, in: 2006 IEEE Conference on Emerging Technologies and Factory Automation, IEEE, 2006, pp. 421–428.

[96] M. Bartlett, I. Bate, J. Cussens, Instruction Cache Prediction Using Bayesian Networks, in: Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence, IOS Press, Amsterdam, The Netherlands, The Netherlands, 2010, pp. 1099–1100.
URL http://dl.acm.org/citation.cfm?id=1860967.1861224

[97] D. Griffin, B. Lesage, I. Bate, F. Soboczenski, R. Davis, Forecast-based Interference: Modelling Multicore Interference from Observable Factors, in: 25th International Conference on Real-Time Networks and Systems (RTNS 2017), 2017.

[98] F. ul Islam, M. Lin, L. Yang, K. Choo, Task Aware Hybrid DVFS for Multi-core Real-time Systems Using Machine Learning, Information Sciences 433-434 (2018) 315 – 332.
URL https://doi.org/10.1016/j.ins.2017.08.042

[99] H. Mao, M. Alizadeh, I. Menache, S. Kandula, Resource Management with Deep Reinforcement Learning, in: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, HotNets '16, ACM, New York, NY, USA, 2016, pp. 50–56. doi:10.1145/3005745.3005750.

[100] J. Rivas, J. Gutiérrez, M. Harbour, A Supercomputing Framework for the Evaluation of Real-time Analysis and Optimization Techniques, Journal of Systems and Software 124 (2017) 120 – 136.
URL https://doi.org/10.1016/j.jss.2016.11.010

[101] T. Chen, C. Guestrin, XGBoost: A Scalable Tree Boosting System, in: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'16, ACM, New York, NY, USA, 2016, pp. 785–794. doi:10.1145/2939672.2939785.
URL http://doi.acm.org/10.1145/2939672.2939785

[102] R. Davis, N. Navet, Controller Area Network (CAN) Schedulability Analysis for Messages with Arbitrary Deadlines in FIFO and Work-Conserving Queues, in: 9th IEEE International Workshop on Factory Communication Systems, 2012, pp. 33–42.

[103] N. Navet, J. Villanueva, J. Migge, Early-stage Topological and Technological Choices for TSN-based Communication Architectures, in: 2019 IEEE Standards Association (IEEE-SA) Ethernet & IP @ Automotive Technology Day, Detroit, Mi, 2019.
URL https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/other/eipatd-presentations/2019/D1-03_NAVET_Earlystage-topological_and_Technical_choices_for_TSN.pdf

[104] N. Navet, H. Bengtsson, J. Migge, Early-stage Bottleneck Identification and Removal in TSN Networks, in: Automotive Ethernet Congress, Munich, 2020.
URL https://www.realtimeatwork.com/wp-content/uploads/AEC2020-UL-Volvo-RTaW-web.pdf

[105] J. Frankle, M. Carbin, The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, arXiv preprint arXiv:1803.03635 (2018).

[106] R. Olson, W. Cava, Z. Mustahsan, A. Varik, J. Moore, Data-driven Advice for Applying Machine Learning to Bioinformatics Problems, in: Proceedings of the Pacific Symposium on Biocomputing 2018, World Scientific, 2018, pp. 192–203.

[107] T. Salimans, D. Kingma, Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks, in: Advances in Neural Information Processing Systems, 2016, pp. 901–909.

[108] F. Such, A. Rawal, J. Lehman, K. Stanley, J. Clune, Generative Teaching Networks: Accelerating Neural Architecture Search by Learning to Generate Synthetic Training Data, arXiv preprint arXiv:1912.07768 (2019).

[109] A. Karpathy, A Recipe for Training Neural Networks, retrieved 2020/08/14.
URL http://karpathy.github.io/2019/04/25/recipe/

[110] O. Philipp, K. Stefan, S. Eric, Model-Based Resource Analysis and Synthesis of Service-Oriented Automotive Software Architectures, in: 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2019, Munich, Germany, September 15-20, 2019, IEEE, 2019, pp. 128–138. doi:10.1109/MODELS.2019.000-8.
URL https://doi.org/10.1109/MODELS.2019.000-8

[111] O. Creighton, N. Navet, P. Keller, J. Migge, Towards Computer-Aided, Iterative TSN and Ethernet-based E/E Architecture Design, in: to appear at the 2020 IEEE Standards Association (IEEE-SA) Ethernet & IP @ Automotive Technology Day, Munich, 2020.

[112] T. Mai, N. Navet, Deep Learning to Predict the Feasibility of Priority-Based Ethernet Network Configurations, Tech. rep., University of Luxembourg, submitted to ACM TECS (2020).

[113] S. Varrette, P. Bouvry, H. Cartiaux, F. Georgatos, Management of an Academic HPC Cluster: The UL Experience, in: 2014 International Conference on High Performance Computing & Simulation (HPCS), IEEE, 2014, pp. 959–967.

[114] X. Zhu, Semi-Supervised Learning Literature Survey (2005).

[115] Q. Xie, M. Luong, E. Hovy, Q. Le, Self-Training with Noisy Student Improves Imagenet Classification, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 10687–10698.

[116] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, The Journal of Machine Learning Research 15 (1) (2014) 1929–1958.

[117] T. Chen, S. Kornblith, M. Norouzi, G. Hinton, A Simple Framework for Contrastive Learning of Visual Representations, in: International Conference on Machine Learning, PMLR, 2020, pp. 1597–1607.

[118] L. Bertinetto, J. Valmadre, J. Henriques, A. Vedaldi, P. Torr, Fully-Convolutional Siamese Networks for Object Tracking, in: European Conference on Computer Vision, Springer, 2016, pp. 850–865.

[119] J. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Pires, Z. Guo, M. Azar, et al., Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning, arXiv preprint arXiv:2006.07733 (2020).

[120] Y. Xie, Z. Xu, J. Zhang, Z. Wang, S. Ji, Self-Supervised Learning of Graph Neural Networks: A Unified Review, arXiv preprint arXiv:2102.10757 (2021).

[121] L. Van der Maaten, G. Hinton, Visualizing Data Using t-SNE., Journal of Machine Learning Research 9 (11) (2008).

[122] K. Thekumparampil, C. Wang, S. Oh, L. Li, Attention-based Graph Neural Network for Semi-Supervised Learning, arXiv preprint arXiv:1803.03735 (2018).

[123] F. Tian, B. Gao, Q. Cui, E. Chen, T. Liu, Learning Deep Representations for Graph Clustering, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 28, 2014.

[124] Q. Li, Z. Han, X. Wu, Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[125] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, D. Krishnan, Supervised Contrastive Learning, arXiv preprint arXiv:2004.11362 (2020).

[126] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph Attention Networks, arXiv preprint arXiv:1710.10903 (2017).

[127] J. Mukhoti, V. Kulharia, A. Sanyal, S. Golodetz, P. Torr, P. Dokania, Calibrating Deep Neural Networks Using Focal Loss, arXiv preprint arXiv:2002.09437 (2020).

[128] J. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Pires, Z. Guo, M. Azar, et al., Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning, arXiv preprint arXiv:2006.07733 (2020).

[129] J. Zbontar, L. Jing, I. Misra, Y. LeCun, S. Deny, Barlow Twins: Self-Supervised Learning via Redundancy Reduction, arXiv preprint arXiv:2103.03230 (2021).

[130] G. Ghiasi, B. Zoph, E. Cubuk, Q. Le, T. Lin, Multi-Task Self-Training for Learning General Representations, arXiv preprint arXiv:2108.11353 (2021).

[131] D. Pujol-Perich, J. Suárez-Varela, M. Ferriol, S. Xiao, B. Wu, A. Cabellos-Aparicio, P. Barlet-Ros, IGNNITION: Bridging the Gap Between Graph Neural Networks and Networking Systems, arXiv preprint arXiv:2109.06715 (2021).