# Deep Learning to Predict the Feasibility of Priority-Based Ethernet Network Configurations

TIEU LONG MAI, University of Luxembourg, Luxembourg
NICOLAS NAVET, University of Luxembourg, Luxembourg

Machine learning has been recently applied in real-time systems to predict whether Ethernet network configurations are feasible in terms of meeting deadline constraints without executing conventional schedulability analysis. However, the existing prediction techniques require domain expertise to choose the relevant input features and do not perform consistently when topologies or traffic patterns differ significantly from the ones in the training data. To overcome these problems, we propose a Graph Neural Network (GNN) prediction model that synthesizes relevant features directly from the raw data. This deep learning model possesses the ability to exploit relations among flows, links, and queues in switched Ethernet networks, and generalizes to unseen topologies and traffic patterns. We also explore the use of ensembles of GNNs and show that it enhances the robustness of the predictions. An evaluation on heterogeneous testing sets comprising realistic automotive networks, shows that ensembles of 32 GNN models features a prediction accuracy ranging from 79.3% to 90% for Ethernet networks using priorities as the Quality-of-Service mechanism. The use of ensemble models provides a speedup factor ranging from 77 to 1715 compared to schedulability analysis, which allows a far more extensive design space exploration.

Additional Key Words and Phrases: Machine learning, Graph Neural Network, Schedulability analysis, Design Space Exploration, Time-Sensitive Networking

**ACM Reference Format:**
Tieu Long Mai and Nicolas Navet. 2021. Deep Learning to Predict the Feasibility of Priority-Based Ethernet Network Configurations. *ACM Transactions on Cyber-Physical Systems* 1, 1, Article 1 (January 2021), 26 pages. https://doi.org/10.1145/3468890

Authors' addresses: Tieu Long Mai, University of Luxembourg, 2, avenue de l'Université, Esch-sur-Alzette, Luxembourg, long.mai@uni.lu; Nicolas Navet, University of Luxembourg, 2, avenue de l'Université, Esch-sur-Alzette, Luxembourg, nicolas.navet@uni.lu.

XXXX-XXXX/2021/1-ART1
https://doi.org/10.1145/3468890

## 1 INTRODUCTION

### 1.1 Context of the study

*Ethernet is becoming the prominent wired high-speed network technology in real-time systems.* Real-time communication with Quality-of-Service (QoS) guarantees is essential in most critical systems, be it in the automotive, aerospace, telecommunication or industrial domains. In such systems, Ethernet is progressively becoming the prominent technology for wired high-speed communications. One of the reasons is that Ethernet has been constantly evolving and adapting to successfully address the needs of the new systems being developed. In particular, the IEEE 802.1 TSN TG (Time Sensitive Networking Technical Group), started in 2012, develops the technologies to address QoS requirements pertaining to timing, reliability and security. This work focuses on the basic timing requirement, which is to ensure that communication latencies are below the deadlines in any possible circumstances.

*The use of priorities as the main QoS mechanisms.* Ethernet TSN, see https://1.ieee802.org/tsn/, is a toolbox of standards whose main standard is IEEE802.1Q [20]. TSN defines mechanisms and protocols implementing various QoS strategies such the use of priorities, traffic shaping, time-triggered communications and frame preemption. In this work, we focus on the use of priorities, the fundamental QoS mechanism, that is supported by any commercially available hardware components. In TSN networks, streams can be allocated to at most 8 distinct traffic classes, each class with its own priority level, a mechanism developed within the IEEE802.1p working group eventually integrated into 802.1Q. The reader can consult [26] for a survey of TSN standards and ongoing works within TSN TG.

*Future-proof communication architectures with design-space exploration.* Ethernet TSN is becoming the prominent technologies for the core high-speed networks in next generation industrial plants and automotive Electrical/Electronic (E/E) architectures. In these two application domains, there has been a pivotal change which is that it cannot be assumed anymore that the functions, and thus the communication requirements, are known in advance and fixed over time. It has become crucial for OEMs to be able to add further functions/services through software update during the lifetime of the systems: what is needed is to design communication architectures that are future-proof. In that regard, Design-Space Exploration (DSE) approach that explores the possible evolutions of the software distributed over the communication architecture can help designers make sound early-stage design choices. Actually, DSE approaches that fulfill this need are already in use, or currently explored at several car manufacturers (e.g., BMW in [5, 37], Renault in [32] and Volvo in [27]), and have been available for a few years in COTS network design tools like RTaW-Pegase [2].

### 1.2 Definition of the problem

*Network performance evaluation is a bottleneck for DSE algorithms.* DSE algorithms typically involve creating, in an iterative manner, a large number of candidate solutions which are kept at later iterations based on their relative performance. In the field of real-time networks, depending on the type of performance requirements, performance evaluation is either done by simulation (e.g., for throughput constraints) or worst-case schedulability analysis (e.g., for deadline constraints). Performance evaluation is very compute intensive, which limits the size of the search space that can be explored. For instance, assessing whether a set of $10^6$ Ethernet TSN configurations with 350 flows each meet their deadline constraints under priority scheduling is estimated in [28], based on measurements, to take about 224 hours of CPU time. It should be additionally considered that the schedulability analysis of priority scheduling is fast compared to other QoS mechanisms like traffic shapers. The problem is even more acute as TSN allows the combination of several QoS mechanisms, each with user-chosen parameters sometimes on a per-egress port basis, resulting in an exponential increase of the number of candidate scheduling solutions.

*Machine learning to speed up performance evaluation.* A drastic speed-up in feasibility testing thanks to machine learning (ML) would unlock all the potential of DSE algorithms in the design of communication architectures. Therefore prior works [23, 24] have studied whether ML algorithms can be a faster alternative to conventional schedulability analysis to determine whether a real-time TSN network meets a set of timing constraints. Figure 1 shows the typical use of ML to speed up DSE by replacing mathematical analysis by predictions. Such ML-based feasibility verification will lead to "false positives", *i.e.* configurations deemed feasible while they are not. However, this does not raise a problem in DSE as long as the retained solutions, the ones presented to the designer, are verified by an analysis that is not prone to false positives, such as conventional schedulability analysis. Typically, the combinatorial problem of allocating the software components to the end-nodes is a use-case that could benefit from ML techniques like the one proposed in this work. Other DSE activities involving the comparison of different candidate design options (e.g. in terms of extensibility as in [5, 27]) may leverage on ML too. For such use-cases, it may not be needed to perform conventional schedulability analysis as DSE is used to make structural design choices and not to make configuration choices used at run-time.

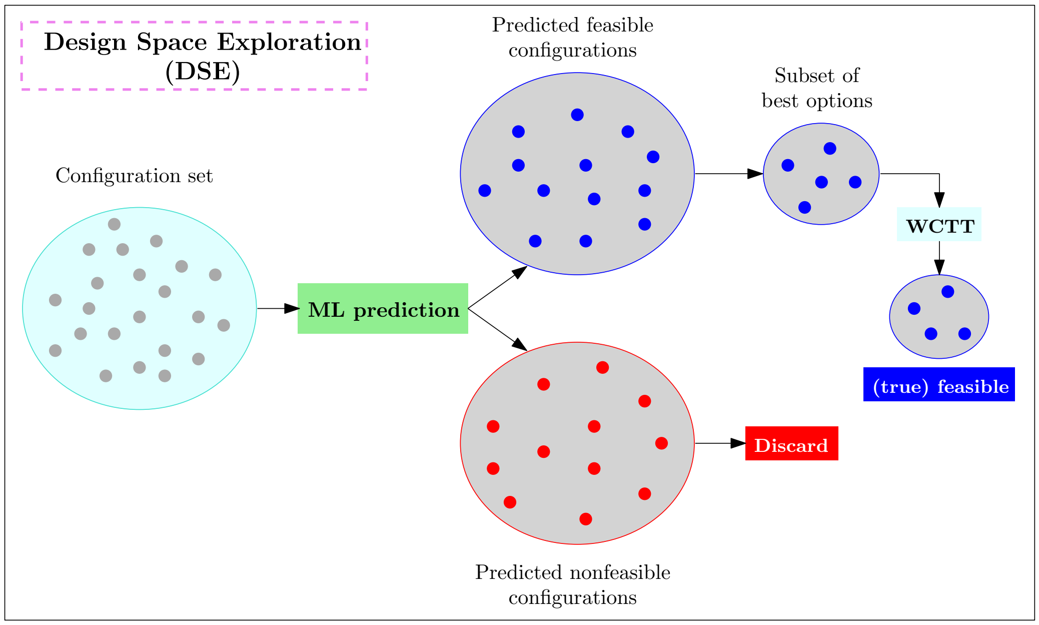

Design Space Exploration (DSE)
Configuration set
ML prediction
Predicted feasible configurations
Subset of best options
WCTT
(true) feasible
Discard
Predicted nonfeasible configurations

Fig. 1. Machine learning (ML) to speed up design space exploration by predicting whether candidate configurations are feasible instead of performing compute-intensive Worst-Case Traversal Time (WCTT) analyses. Since ML will lead to "false positives", the few retained solutions presented to the designer are verified by an analysis that is not prone to false positives.

Beyond the field of real-time systems, the use of machine learning to reduce the turnaround time of heavy computations in design activities, like done in this work, is explored in several application domains to improve productivity, ideally reaching "near interactive" design tools [36]. It is indeed now well documented that "creators need an immediate connection to what they are creating" [43] to work best.

*Research questions.* We address two main research questions in this work:

**RQ1:** We first ask whether deep learning with Graph Neural Networks (GNN) is more powerful than state-of-the-art conventional [1] ML techniques, such as gradient boosting, in the sense that a GNN-based model can be successfully applied to predict the feasibility of unseen network topologies? This question is of practical importance since a positive answer would mean that the prediction model does not need to be retrained for each new network topology, hence speeding up and easing the DSE process importantly.

**RQ2:** The second question investigated is how to maximize the accuracy of GNN-based models, for instance using ensemble techniques, and whether, ultimately, the achieved accuracy is sufficient to make sound early-stage design choices? If GNN-based prediction proves to be a viable alternative to mathematical analysis, this would mean that the designer can typically obtain answers from DSE in hours instead of weeks.

## 1.3 Limitations of existing solutions

Although recently deep learning has been successfully applied to the performance evaluation of communication networks (see [10, 11] and Section 6 for a review of related works), it has, to the best of our knowledge, not been yet applied to the verification of hard deadline constraints (*i.e.*, referred to as *feasibility* or *schedulability* analysis in the following). Although it still belongs to the realm of schedulability analysis, the problem addressed in this work is actually broader that verifying a network configuration once all its parameters have been set, the problem is to predict whether priority allocation algorithm will return an allocation leading to a feasible schedule. The priority allocation algorithm considered (see Section 2.2) is known to be very effective, even optimal in several contexts, but it involves performing many successive schedulability analyses and hence, constitutes a bottleneck in a DSE process.

Conventional ML techniques such as classification with k-NN and clustering with K-means [16] have already been utilized in [24, 28] as fast approximators to schedulability analysis. These studies conclude that prediction with conventional ML techniques is much faster than schedulability analysis with a speedup factor of 190 on typical automotive network topologies [28]. The prediction accuracy, better than 85%, would allow in our view their actual use for several of the use-cases of DSE. These approaches however suffer two limitations:

(1) The developers of the algorithm must select and possibly "engineer" the features that will be used by the ML algorithm. This process, which is time-consuming and error-prone, requires both know-how in ML and an expertise of the problem domain. A main advantage of deep learning is its ability to automate the feature engineering process during the learning phase.
(2) The works in [24, 28] only consider a single fixed network topology, the variability in the network configurations comes from the set of streams. This means that the ML algorithm must be retrained for each new candidate topology. In addition, in these works, the transmission patterns of the streams in the unseen configurations is assumed to not change considerably with respect to the training set.

This raises the question of how accurate conventional ML is when the topology or the traffic patterns change with respect to the training set, and whether there would be an ML model efficient at automating the feature engineering step for this specific problem. These open questions, as well as the practical limitations highlighted above, impede the use of ML for feasibility prediction.

## 1.4 Contributions of the paper

The first contribution of this study is to unveil, through extensive experiments on realistic automotive TSN networks, the main limitation of conventional ML algorithms: they do not perform consistently well on unseen network topologies, otherwise said, they do not generalize well outside the training sets.

[1] "Conventional" means here not based on deep neural networks.

Therefore, we propose a novel ML model based on Graph Neural Network (GNN) to predict the feasibility of TSN configurations. This is, to the best of our knowledge, the first graph-based deep learning model specialized for schedulability analysis, a major problem in the verification of real-time critical systems. The proposed GNN model encodes a complete network configuration as a graph where flows, links, and waiting queues are nodes in the graph. A global vector, synthesized in a process involving two distinct neural networks, contains all the information about the graph and relevant relationships between its constituents [2]. This global vector is used as input to a final standard multi-layer fully connected neural network that predicts the actual feasibility of the configuration. Experiments show that the proposed model can generalize to topologies and traffic patterns that are dissimilar from the ones making up the training set. The model also automates the feature selection process since it directly uses as input the raw characteristics of the TSN configurations.

Another contribution of this study is to introduce the use of ensembles of GNNs to mitigate the intrinsic unpredictability of neural network performance with respect to their training parameters, a major risk in using neural networks in the industry. Indeed, each trained GNN model will possess a different prediction accuracy, hence possibly leading to different design choices. An ensemble of GNNs can reduce the risk that a prediction is wrong, as a minority of models that are wrong will be overridden by the majority of correct models. The experiments in this work show the interest of ensemble learning for our specific GNN model to achieve a more consistent prediction accuracy. We explore the use of the basic majority voting and the much more sophisticated Gradient Tree Boosting, two techniques at the opposite ends of the complexity spectrum.

### 1.5 Organisation of the paper

The remainder of this paper is organized as follows. Section 2 introduces the system model and assumptions made. Section 3 presents the general GNN model and its specialization for predicting the feasibility of TSN networks. Section 4 describes the experimental setup while Section 5 analyses the results obtained with ensembles GNNs, by comparison with conventional ML algorithms. In Section 6, we recap on the related works about GNN and ML in networking. Finally, Section 7 concludes and identifies possible improvements and research directions.

## 2 SYSTEM MODEL

We consider a standard switched Ethernet comprising a set of switches, full-duplex links and end-nodes, each equipped with a network interface. The network supports unicast and multicast communications between a set of software components distributed over a number of end-nodes. In the following, both “traffic flow” or “traffic stream” refer to a sequence of frames sent to one or several receivers (*i.e.*, a multicast connection with *n* receivers generates *n* distinct traffic flows).

In the following, the term *traffic flow* and *traffic stream* refer to a sequence of frames sent from one sender

### 2.1 Assumptions on the network

In this study, a number of assumptions about the networks considered are made:

- All packets, also called frames, of the same traffic flow are delivered over the same path: the routing is static as it is today the norm in critical systems.
- It is assumed that there are no transmission errors and no buffer overflows leading to packet losses. The latter property can be checked with schedulability analysis as it returns both upper bounds on stream latencies as well as maximum memory usage at switch ports.
- Streams are either periodic, sporadic (*i.e.*, two successive frames become ready for transmission at least $x$ ms apart) or sporadic with bursts (*i.e.*, two successive bursts of *n* frames become ready for transmission

[2]The GNN terminology used in this work is from [4], which formalizes the general GNN model.

at least $x$ ms apart). The latter type of traffic corresponds for instance to video streams from cameras, as a camera frame cannot fit into a single Ethernet frame and must be segmented into several frames.
- The maximum size of the successive frames belonging to a stream is known or assumed to be of maximum size (*i.e.*, 1522 bytes), as required by schedulability analysis.
- The packet switching delay is assumed to be $1.3\mu$s at most. This value, which of course varies from one switch model to another, is in line with the typical latencies of modern Ethernet switches [38].

In this work, we assume that the network topology (layout, link data rates, etc) has been set before the communication needs are known. This is realistic with respect to industrial contexts, like the automotive and aeronautical domains, where most design choices pertaining to the topology of the networks and the technologies are made early in the design process, at a time when the software functions, and thus the communication needs are not entirely known. Indeed, many functions become available later in the development cycle or are added at later evolutions of the system.

## 2.2 Configuring the stream priorities

In the following, a *configuration* refers to a TSN network whose parameters have been all set. A configuration is said to be *feasible* if the timing constraints on the flows are all met. The configuration of TSN networks involves two main sub-problems:

- Grouping traffic streams into traffic classes and deciding the relative priorities of the traffic classes: TSN allows the use of up to 8 priority levels. This is referred to as *the stream priority assignment* problem.
- Beyond the priorities of the streams, optionally selecting and configuring for each traffic class an additional QoS protocol: shaping using the Credit-Based Shaper (CBS [17]), time-triggered transmission with the Time-Aware Shaper (TAS [18]), Frame Preemption ([19]), etc.

This work focuses on the fundamental QoS mechanism, which is the use of priorities to manage the interferences between streams. Therefore, finding a feasible configuration means finding a feasible priority assignment. The model developed in this paper predicts whether a given priority allocation algorithm, relying a certain schedulability analysis, will return a “feasible” priority allocation. The proposed predictive model can be used for any priority allocation algorithm and any schedulability analysis, provided that the labels of the samples in the training set are derived with these algorithms.

In the experiments of this study we have used the Optimal Priority Assignment (OPA) algorithm initially proposed for mono-processor systems [3]. OPA is the reference priority allocation algorithm in the field of real-time systems because of its low algorithmic complexity and its optimality for a variety of schedulability analyses [6, 7]. In particular, OPA has been shown optimal for a schedulability analysis of periodic/sporadic streams on AFDX network [15]. OPA is however not optimal with the state-of-the-art network calculus based Worst-Case Traversal Time (WCTT) analysis used in study, which implements an optimization called "traffic grouping" [46] leading the analysis to occasionally violate one of the assumptions required for OPA to be optimal [3]. If not optimal, preliminary experiments suggest that OPA is very efficient at finding feasible priority allocations in our context [4]. In the experiments of this work, we are relying on a variant of OPA called "Concise Priorities" (CP), that is available in the RTaW-Pegase software. CP and OPA differ only in how unfeasible configurations are handled, which is not relevant in the context of this work.

[3]Enhancing traffic grouping to make it "OPA compatible" is to the best of our knowledge an open problem.
[4]Developing priority allocation algorithms is out the scope of this work. Preliminary experiments we did with deep reinforcement learning on the same system model were unsuccessful at outperforming OPA.

### 2.3 Topology and traffic characteristics

In order to create the training set for the GNN prediction models, a number of "artificial" TSN configurations are generated. These configurations have the following characteristics in terms of their topology:

- Each network possesses a central switch, with two to five switches connected to that central switch (*i.e.* star topology).
- There are two to five end-nodes connected to each switch.
- The speed of each link is set at random to 100 or 1000Mbit/s. This corresponds to the most widely used link speeds in embedded systems, e.g. physical layers in the automotive domain comply either to 100BASE-T1 or 1000BASE-T1 today.

The characteristics of the traffic for the training networks are shown in Table 1. This corresponds to typical automotive traffic as in [27, 29, 32]. The number of receivers per stream ranges from one to three. When testing the GNN model, topology and traffic patterns of the TSN configurations are either similar or different to the training set. The details of the testing sets are shown in Section 4.1.

Table 1. Characteristics of the four types of streams used in the creation of the TSN networks for the training set. Frame sizes indicated here are data payload only. These characteristics can be changed for the testing sets, as indicated in the experiments. The performance requirement is to meet the stream deadline constraints.

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

## 3 PREDICTING NETWORK FEASIBILITY WITH GRAPH NEURAL NETWORK

The trend in ML is to shift from traditional ML algorithms [16] to deep learning with neural networks [39]. A main reason for that is deep learning's ability to extract features directly from high dimensional raw data. According to [4], there are four standard deep learning paradigms: *fully-connected neural networks* for general purposes, *convolutional neural networks* for image processing, *recurrent neural networks* for data sequences, and *graph neural network* (GNN) for modeling graph-based data. In the first three paradigms, the inputs are either ordered features, spatial relations (e.g., pixels in an image) or series (e.g., words in a sentence).

As a computer network is made up of network devices, nodes and switches, that are connected through links, fundamentally, it is a graph. Hence, GNN, which is meant to capture complex relationships in graph based data, is a promising tool to model communication networks. In our context, we apply GNN to synthesize, directly from the primary components (flows, links, queues), features that will generalize over different topologies and traffic patterns to predict the feasibility of TSN configurations.

It is worth noting that since GNN is more a paradigm than a concrete deep learning model, the exact structure, implementation and notations for GNN models vary in the literature. In this paper, we adopt the general model from [4], summarized in Section 3.1, and tailor it to our specific problem in Section 3.3.

## 3.1 GNN model structure

A TSN configuration is stored in a data structure capturing its physical and logical components and their characteristics: the flows (characterized by max. payload, transmission pattern and deadline), the links (characterized by their speed) and the egress queues in the switches [5]. These components are connected as defined by the topology and routing. The characteristics of a configuration are captured by a graph model that is made up of nodes and connections between them. An overview of the GNN model is depicted in Figure 2. A novelty of our model with respect to prior works [10, 13] is that it considers a single type of nodes whatever the components, which, in addition to being simpler, gave better results during the development of the model.

The graph model, at the center of Figure 2, contains in each node a vector, called the *embedding*. The embeddings are first initialized to the raw data that are the inputs of the model, then the embeddings will evolve throughout the learning phase as explained in the computational steps (see § 3.2). Once the training is completed, the global attribute is then used for prediction.

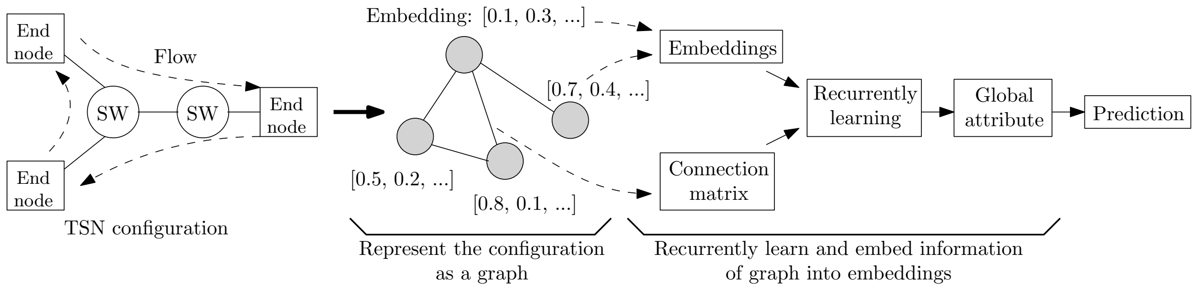


Fig. 2. Overview of the proposed Graph Neural Network (GNN) model. Flows, links and queues are all three captured by nodes in the graph (filled in gray). Each node has an initial embedding that captures information about the node, for instance, the data rate of a link scaled in [0,1]. The model recurrently learns the relation between physical and logical components and updates the initial embeddings. A global attribute, *i.e.*, the embedding of the graph, is derived from the embeddings of the individual nodes. This global attribute is then used to predict the feasibility of the configuration.

The GNN predicts whether there is a feasible priority assignment without manipulating and encoding priorities, but based on what has been learned in the training set. This work could be extended to other QoS mechanisms that could be applied on top of a pre-defined priority allocation (based on the criticality of the frames, their functional domains, etc). For instance, a similar GNN model could predict whether the Time-Aware-Shaper, the

[5]Experiments conducted with a more fine-grained model, considering the waiting queues in end-nodes, did not lead to better results. In our understanding this is because, in our context, most of the interferences between flows, and thus delays, occur at the egress ports of the switches which form the backbone of the network. The links from the nodes to the first switch, the access points to the backbone, are far less loaded. Other QoS mechanisms such as end-to-end shaping with CBS [17] could benefit from a more detailed model integrating the waiting queues in the end-nodes.

Credit-Based-Shaper, Frame Preemption, etc, would lead to a feasible solution. In that case, the priority, which is an attribute of a flow, would have to be encoded into the embeddings of the nodes corresponding to the flows (and set e.g. to -1 or 8 for the nodes corresponding to the links and the queues). Another necessary extension would be to model the multiple queues, one per priority level, in the graph of the GNN.

## 3.2 Computational steps in a general GNN model

In the general GNN model, a graph $G$ is a collection of embedding vectors $(u, v_i, e_k)$, where $u$ is the global attribute of the graph, $v_i$ is the embedding of node $i$ (or vertex $i$), and $e_k$ is the embedding of directed edge $k$. The update of the embeddings is done iteratively, over successive loops formalized by Equation 1.

(1) *Update edges*: update each edge $k$ by applying function $f_e$ taking as arguments $e_k$, the embedding of the destination node $v_{r_k}$, the embedding of the source node $v_{s_k}$ and the global attribute $u$.
(2) *Update nodes*: update each node $i$ by applying function $f_v$ taking as arguments the average of the embeddings of all incoming edges, $v_i$ and the global attribute $u$.
(3) *Update global attribute*: update $u$ by applying function $f_u$ taking as arguments the average of the embeddings of all edges, the sum of the embeddings of all nodes, and its previous value.

Functions $f_e$, $f_v$ and $f_u$ are learnable functions, usually neural networks. $f_e$ and $f_v$ are identical for all nodes and edges in the GNN. This process is iterated until all embeddings converge.

$$
\begin{aligned}
&\text{Update edges:} && e'_k = f_e(e_k, v_{r_k}, v_{s_k}, u)\\
&\text{Update nodes:} && e'_i = \sum_{k:r_i=i} e'_k\\
& && v'_i = f_v(\bar{e'_i}, v_i, u)\\
&\text{Update global attribute:} && e' = \sum_k e'_k\\
& && v' = \sum_i v'_i\\
& && u' = f_u(\bar{e'}, \bar{v'}, u)\\
& && e_k = e'_k; \quad v_j = v'_j; \quad u = u'
\end{aligned}
\tag{1}
$$

where the symbol $\bar{x}$ means the average of the embeddings over $x$. In the following, this general GNN model will be specialized for the feasibility evaluation of TSN configurations.

## 3.3 Encoding of TSN configurations as a graph

Our GNN model for TSN network feasibility prediction shown in Figure 3 is based on the general model from [4], introduced in 3.2, and the one proposed in [10] for the performance evaluation of TCP communications. At the core of all GNN models there is a graph, which captures information about the system according to modelling choices. In our context a TSN configuration is comprised of three main types of components, each modelled as a node in the graph with a corresponding embedding vector:

- *waiting queues*: each switch can be seen as a sub-network with connections between the ingress and egress ports. As in practice the switching delays are typically much smaller than the end-to-end communication delays (switching delays [38] are 1.3$\mu$s maximum in our experiments while end-to-end delays are in the ms range), we do not model the waiting times on the ingress ports. We assume that the queues at the egress ports are large enough for the traffic, as required in most critical systems, and no packet losses occur. We

model the waiting queue at an egress port as one node in the graph and its initial embedding is set 0 (*i.e.* all elements of the vector) to indicate that there is no message pending transmission at the startup.
- *link*: a link in today's switched Ethernet networks is bi-directional ("full-duplex"), therefore we model it as two nodes connected to the corresponding queues. The initial value of the first element of the embeddings vector is set to the link speed while all others are set to zero.
- *flow*: a flow is modeled as a node in the graph, be the flow unicast or multicast. The initial embedding of a flow encodes information about its payload size, burstiness, transmission period and deadline.

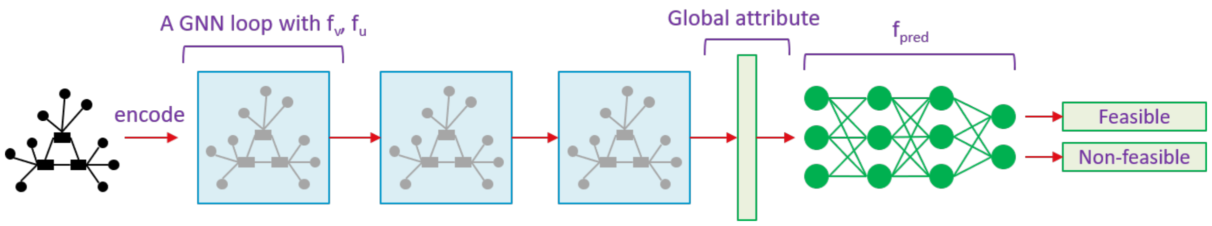


Fig. 3. Refinement of Figure 2 that gives more details of the GNN model. A TSN configuration is encoded as a set of embeddings that are iteratively updated with neural networks $f_v$ and $f_u$. After several loops, three in the experiments of the paper, the global attribute is used to predict the feasibility of the network configuration with a dedicated neural network $f_{pred}$.

Besides the embeddings of the nodes, the embedding of the graph, *i.e.* its global attribute, is initialized to 0. A *connection matrix* is used to capture the edges between nodes in the graph. As the topology of the network does not change, this matrix remains identical over the successive computational steps. The matrix is initialized to 0 and some of its elements are set to non-zero values as follows:

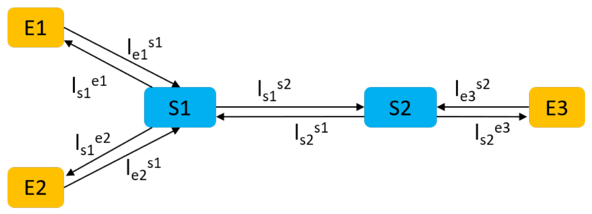
- *queue-link* edge: value 1 indicates that the link is physically connected to the corresponding egress port.
- *queue-flow* edge: value 1 indicates that the flow goes through the corresponding egress port.
- *link-flow* edge: a non-zero value indicates that the flow goes through the corresponding link: value 2 if the link is connected to the source node, 1 in the other cases.

Figure 4a shows the topology of an example TSN configuration. The TSN configuration is modeled as a graph, see Figure 4b, where the edges between queues and links are directional. At step 3, Figure 4c, the directions of the edges are removed. The reason to do that is the mutual dependencies between an egress port and the link it is connected to. On the one hand, if the link is slow, then the port queue will tend to fill up. On the other hand, the more packets in the queue, the more loaded the link. In Figures 4d, 4e and 4f, the three flows $f1, f2, f3$ are modelled by the corresponding nodes in the graphs as well as their connections to the links' and queues' nodes with respect to the routing. It should be noted that even though the graph is non directed, the routing of the flows can be derived as the source nodes are identified in the connection matrix.

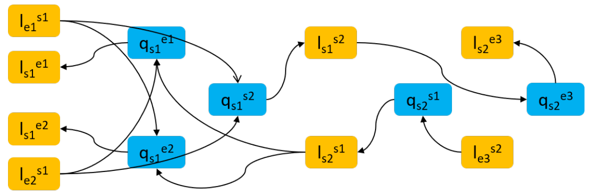
## 3.4 The GNN model for TSN and associated learning algorithm

The GNN model for TSN configurations is a specialization of the general GNN model presented in Section 3.2 [6]. In the general GNN model, there are three distinct functions to update the embeddings of the edges, nodes, and the global attribute. In our specialized GNN model, since all flows, links and queues are encoded as similar nodes in the graph, the model only uses two functions, $f_n$ and $f_u$, respectively to update the nodes' embeddings and the

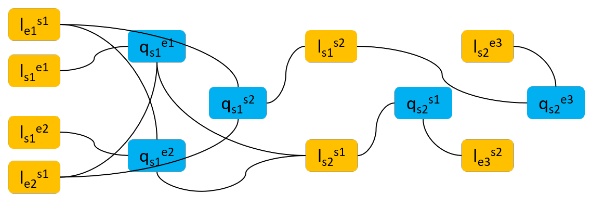
[6]Except otherwise said, the term *GNN model* will refer to our specialized GNN model in the rest of the paper.



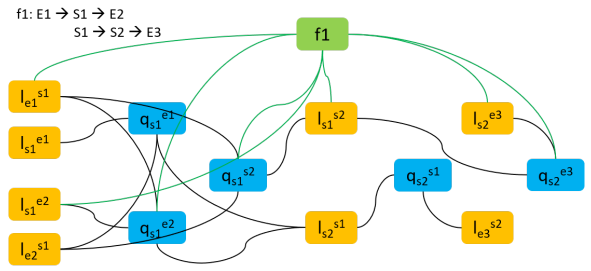
(a) Sample topology with 3 end-nodes and 2 switches. $l_{s1}^{e1}$ denotes the link from S1 to E1.



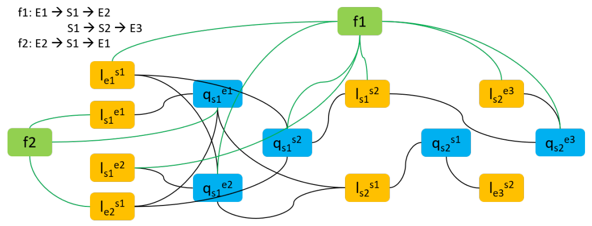
(b) Modelling of links and queues. $q_{s1}^{e1}$ denotes the waiting queue of S1 on the egress port connected to end-node E1.



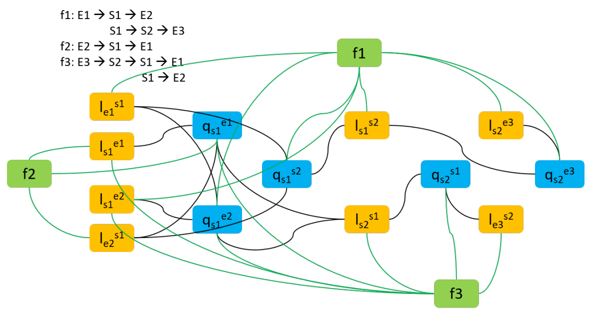
(c) Unidirectional edges.



(d) Adding multicast flow $f1$.



(e) Adding unicast flow $f2$.



(f) Adding multicast flow $f3$.

Fig. 4. Example of how to model a TSN configuration as the core graph of our GNN model. Yellow denotes end-nodes ((a) only) and links, blue denotes switches ((a) only) and queues, while green is for flows. For the sake of explanation the process is divided into steps but the GNN model only uses the final graph with all flows.

global attribute. Another function $f_{pred}$ is used to predict the feasibility of the TSN configuration based on the last update of the global attribute. Each function is a deep neural network (*i.e.*, a three-layer neural network).

The GNN model takes a TSN configuration as input and encodes it as a graph. The values of the nodes' embedding vector, the global attribute, and the connection matrix of the graph are initialized as explained in § 3.3. The computational steps of the GNN model are summarized in Algorithm 1 and explained next. Without loss of generality, we assume that all embeddings have size $l$. Given a graph, the GNN model, based on the connection matrix, knows the neighbors of any nodes in the graph. For each node $i$, the model aggregates the initial embeddings of all neighboring nodes into a vector $v_i'$ of size $l$. This aggregated vector[7] $v_i'$ is concatenated together with the initial embedding $v_i$ of node $i$ and with the initial global attribute $u$, to form a new vector $v_i'$ of size $3l$. This vector is then updated with neural network $f_v$ (line 7 in Algorithm 1). This operation is applied to all nodes in the graph. The model then concatenates the average of all node embeddings, denoted as $\bar{v}'$, with the initial global attribute $u$ to become a vector of size $2l$. This vector is updated by neural network $f_u$ to become $u'$ (line 9 and 10 in Algorithm 1). At this point, $v_i'$ becomes the new embedding of node $i$ and $u'$ the new global attribute (line 11 in Algorithm 1). The GNN model has updated all the embeddings and the first loop has terminated.

**Algorithm 1** Graph neural network (GNN)

```
1:  **Initialize:** $f_v, f_u, f_{pred}$, label $y$
2:  **for** each graph **do**
3:      **Input:** Initialize all $v_i$, $u$, connection matrix
4:      **for** each loop **do**
5:          **for** each node **do**
6:              $v_i' = \sum_{k:neighbor} v_k$                ▷ any nodes connected to node $i$
7:              $v_i' = f_v(v_i, v_i', u)$
8:          **end for**
9:          $v' = \sum_{\text{all nodes}} v_i'$
10:         $u' = f_u(\bar{v}', u)$
11:         Update $v_i = v_i', u = u'$
12:     **end for**
13:     Feasibility prediction: $\hat{y} = f_{pred}(u)$
14:     If training, at the end of each mini-batch, update $f_v, f_u, f_{pred}$ based on $loss(\hat{y}, y)$
15: **end for**
16: **return** $f_v, f_u, f_{pred}$
```

In our case, the GNN model consists of multiple loops: the updated embeddings are used for the next loop in the same manner as the initial embeddings for the first loop. After the final loop, the updated global attributed of size $l$ is taken as input by the neural network $f_{pred}$ whose result is a vector of size 2 corresponding to the predicted value for feasible and non-feasible (line 13 of Algorithm 1). If the first value is larger than the second value, the GNN model predicts the corresponding TSN configuration as feasible, while it will return non-feasible when the second value is larger.

During the training stage, at the end of each mini-batch [8] of training samples, the GNN model compares its prediction with the true feasibility of the TSN configuration, represented by a one-hot vector of size 2: [1, 0] for feasible and [0, 1] for non-feasible. The errors between the two predicted values and the true values observed over the last mini-batch, aka the *loss*, are back-propagated to update all three neural networks and reduce the prediction error for the next mini-batch (line 14 of Algorithm 1).

We note that the GNN model can only successfully predict the feasibility of a flow if the final global attribute captures enough supporting information, which includes information on the interfering flows, expected delays on links and queues, and interactions between those components. When the embedding of a node $i$ in the graph is updated by aggregating the embeddings of neighboring nodes, information is passed on to node $i$. This

[7]In our GNN model, we do not use element-wise average aggregation, as done in the general GNN model, but element-wise sum (see line 7 of Algorithm 1). This change proved to significantly improve the prediction accuracy.
[8]A mini-batch is a number of training samples to process, usually in parallel, before updating the model's parameters.

*message-passing* approach allows the GNN model to gradually aggregate and propagate information over nodes throughout the entire graph, and eventually to the global attribute. Loops in the GNN model also enhances this effect, with the caveat that more loops will make the training of the model more difficult because of the well-documented *vanishing gradient* problem [14] that occurs with backpropagation.

### 3.5 Ensembles of graph neural networks

Some variance in the results, here predictions, of neural networks identical in their structure but trained with different initializations is unavoidable [9]. Since our GNN model involves three neural networks, this variance can add up and lead a given GNN model to perform poorly on certain types of networks and fail silently, *i.e.* without the users possibly knowing about it. To mitigate this risk, we propose to use *ensembles* of GNN models and consider two ensemble techniques:

- Ensembles with majority voting: the final prediction is decided according to a majority vote. This is the simplest ensemble technique.
- Ensembles with Gradient Tree Boosting (GTB, see [16]): the prediction of each model is used as a feature in a gradient boosting algorithm. GTB's accuracy has been shown excellent on a variety of classification problems (e.g., GTB is the best classification algorithm out of 13 on 165 data sets in [34]).

To reduce the variance of each individual GNN model, we also apply the weight normalization technique introduced in [41] and successfully applied, for instance, in [42]. In ensemble techniques, there is a trade-off to be found between the number of models, which reduce the risk of wrong prediction, and the computation time. In the experiments of this work, we use 32 individual models, as done in [42], as it offers a good trade-off between run-time overhead (*i.e.*, parallel execution of the models is still possible on a single CPU) and prediction robustness.

## 4 EXPERIMENTAL SETUP

This paper aims to provide experimental answers to the research questions identified in Section 1. Experiments are conducted on training, validation and testing data sets as follows.

The training set is comprised of center-based topologies, also called star-based topology, commonly used in the industry. The evaluation set, that contains samples similar to the training set, is used to evaluate the models trained, and select the best ones for *out-of-sample evaluation* on the testing set. The samples in the testing set range from very similar to very dissimilar to the training samples in terms of network topologies and traffic patterns. The purpose of this heterogeneity in the testing sets is to assess the relative predictive ability of GNN and traditional ML algorithms on unseen configurations. In the following, we describe the characteristics of the data sets in §4.1, then the parameters of the GNN model in §4.2.

### 4.1 Data sets

*Training and evaluation sets.* TSN configurations in the training set are randomly generated based on the topology and traffic patterns described in Section 2.3. All topologies are center-based but their number of switches, links and ECUs vary in the ranges given in §2.3.

For each topology, we determine the maximum number of flows the topology can handle based on the percentage of overloaded configurations, *i.e.*, configurations with at least one link above 100% load and hence, that are know to be unschedulable. The number of flows leading to a percentage below 90% is considered to be the upper limit of the network load. This 90% threshold has been chosen based on the empirical observation that above 90% threshold practically all non-overloaded configurations are non feasible. This method possesses several advantages: generating the random configurations is simple and fast, the training set is heterogeneous in terms of the number of flows and it is not too unbalanced in terms of their schedulability. This is crucial to not create biases during the learning.

Training sets comprise 1000 topologies. There are 10 configurations generated for each topology, with a number of flows randomly chosen from 20 to the upper limit. All overloaded configurations are discarded. Consequently, the training set contains 10000 unique non-overloaded configurations. The evaluation set is generated with the same procedure, but with 100 topologies, leading to 1000 test configurations in total.

Table 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 |

*Testing sets.* A testing set is defined in terms of its topology and traffic patterns. Some of the topologies follow standard network architecture: star, ring, backbone and "dumbell" topologies (as used in a early TSN network at Daimler [30]). Other topologies come from recent works with automotive OEMs: Renault topology in [31], FACE topology from Renault group as well in [32] and Volvo topology in [27]. Finally, some topologies are modified versions of the OEMs' topologies that remain realistic and provide additional test cases to assess whether the GNN model generalizes outside the training set.

The description of the testing sets in terms of their topologies and traffic patterns is given in Table 2. *Random star-shaped topologies* indicates that the topologies in a testing set were randomly generated exactly as done for the training set configurations. *Training traffic statistics* refers to the traffic patterns of the training set, see Table 1. In Table 2, the testing sets are listed in the order of how different from the training set we judge they are. For instance, testing set 13 possesses a topology and traffic patterns which both are not in the training set. Each testing sets contains 1000 distinct configurations. The noteworthy changes in the testing sets with respect to the training set are listed below:

- *Testing set 2*: Training traffic statistics is modified as follows: the deadlines of best effort (BE) streams change from 20ms to 10% of the periods and the proportion of critical-audio-video-BE streams change from 20-20-30-30 to 10-10-50-50. This modification increases the proportion of video and best effort streams, which have much longer deadlines compared to critical and audio streams and thus are easier to schedule.

- *Testing set 3*: The speed of all links is set to 100Mbps, thus timing constraints will be violated with less flows.
- *Testing set 4*: FACE topology is much larger than the random star-shaped topologies in the training set, in terms of the number of switches and ECUs.
- *Testing set 5*: The deadlines of BE streams is increased to 10% of the periods instead of a fixed 20ms value.
- *Testing set 7*: The Renault topology is modified by increasing the speed of the inter-switch links SW1-SW2 and SW2-SW3 from 100Mbps to 1000Mbps, allowing more configurations to be feasible.
- *Testing set 8*: The modified Renault topology of Testing set 7 is re-used but the proportion of critical-audio-video-BE streams is changed from 20-20-30-30 to 10-10-50-10, *i.e.*, an increase in the proportion of video streams which are bandwidth demanding.
- *Testing set 9*: The Renault topology is extended with one switch connected to three new end-nodes. The extended topology leans towards a backbone topology rather than a center-based topology.
- *Testing set 13*: The Volvo topology is used, which includes redundant paths. The traffic statistic used is completely different from the training traffic statistics.

### 4.2 Parameters of the GNN model

The parameters of the GNN model, summarized in Table 3, have been set considering the good practices of deep learning [14, 21] and extensive trial-and-error experiments. The inputs of the GNN are the initial embeddings and the connection matrix. Each embedding vector is of size 16, and the number of nodes in the graph is 500. If the number of nodes is less than 500, the embedding matrix and the connection matrix is padded with 0 as classically done to obtain a fixed-size input, which is a requirement for neural networks.

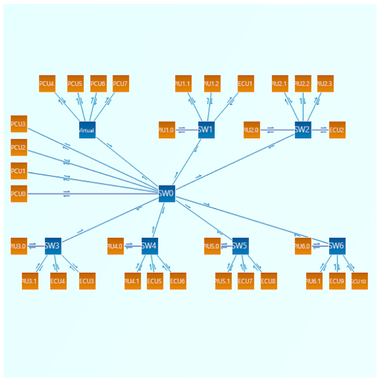
The GNN model involves three neural networks defining functions $f_v$, $f_u$, $f_{pred}$ that are used during the learning 3.4. Their structure is identical: an input layer, two hidden layers, and an output layer. The size of the hidden layers is 256, while the size of the input and output layers, given in Table 3, is specific to the function's purpose. The training of a GNN is performed over 500 iterations with mini-batch of size 32. The GNN model processes in parallel each graph in a mini-batch: it updates the embeddings of the nodes of a graph over 3 loops before deriving the final global attribute used for prediction. The three neural networks involved in our model are updated with the standard Adam optimizer available in TensorFlow [1] with a fixed learning rate of 0.00001 $\times$ mini-batch size. Each 50 training iterations, the best model so far, evaluated based on the prediction accuracy on the evaluation set, is saved for being included in an ensemble of GNNs.

It should be pointed that the neural networks in our model uses the identity activation function and not more expressive activation functions like sigmoid, ReLU or tanh functions. This is because in our experiments the identify function helped mitigate the vanishing gradient problem and speed up the learning.

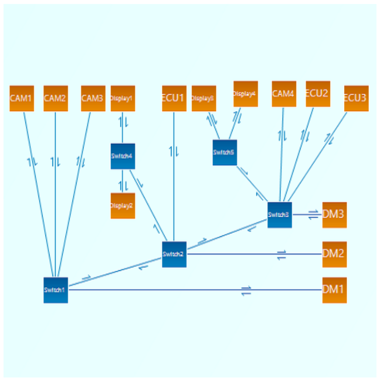
## 5 EXPERIMENTAL RESULTS

The training, evaluation and test configurations have been generated with a Java program using the library of the RTaW-Pegase 3.7.5 [2] software, denoted *Pegase library* in the following, running on Java JDK-13. The labels (*i.e.*, the feasibility of the configurations) have been derived with the Concise Priorities algorithm (see §2.2) available in the Pegase library. All experiments have been conducted on an Intel Core i9-9900k CPU (8 cores) with 64GB of RAM.

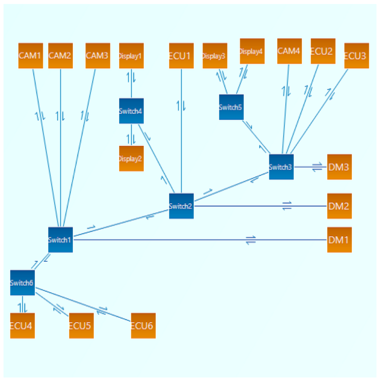
The rest of the programs for the experiments have been developed in Python 3.7. The standard Scikit-learn 0.22 [35] library provides the traditional ML classification algorithms. The graph neural networks are built with the TensorFlow 2.0 [1] deep learning framework. In TensorFlow, we purposefully do not activate the use of the GPU to keep a fair comparison with Concise Priorities in terms of running times. Two Python libraries, Numpy 1.17.4 and Pandas 0.25.3, are also used for data processing in the experiments.



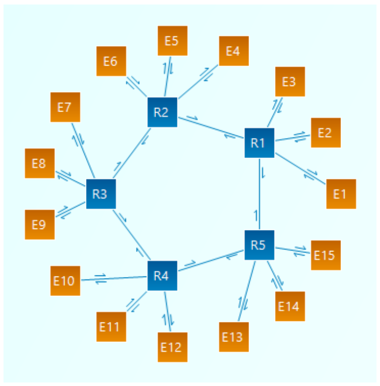
(a) FACE topology (testing sets 4, 5)



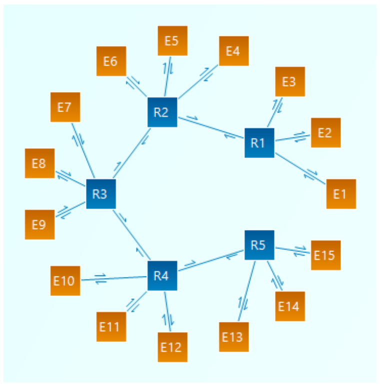
(b) Renault topology (testing sets 6, 7, 8)



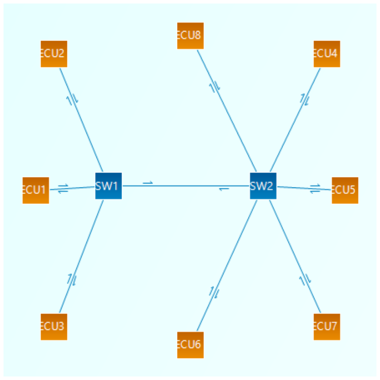
(c) Extended Renault topology (testing set 9)



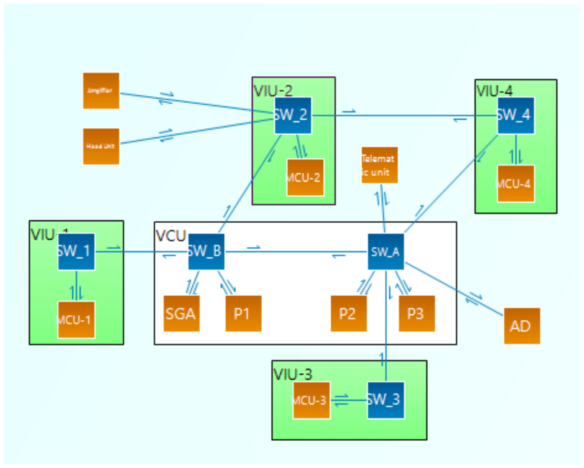
(d) Ring topology (testing set 10)



(e) Backbone topology (testing set 11)



(f) Small topology (testing set 12)



(g) Volvo topology (testing set 13)

Fig. 5. Topologies used in the testing sets (captures from RTaW-Pegase).

Table 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 × 3, 256, 256, 16] |
| Size of NN to update global attribute $f_u$ | [16 × 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 × mini-batch size |
| Optimization technique | Adam optimization |
| # of evaluations | 20 (one after each 50 training iterations) |
| Activation function | identity |

## 5.1 RQ1: Insight into when ML approaches based on manual feature selection fail

We first briefly summarize our previous works exploring the use of ML to automate the verification of TSN networks [23, 24, 28]. In these studies, the input features are manually chosen based on domain knowledge. Precisely, five features are selected: number of critical flows, audio flows, video flows, maximum load among the links, and the Gini index [8] to measure the unbalancedness of the link loads. The main advantage of this approach lies in its simplicity: these features can be extracted directly from the statistics of a configuration. These studies report positive results when ML algorithms are trained and tested on the same fixed topology, and when the traffic in the testing sets does not deviate importantly from the training set.

In this work, we extend these previous works by applying conventional ML algorithms on the new data sets presented in Section 4.1. We choose two algorithms that correspond to the simplest and the most advanced classification algorithms available in Scikit-learn [35]:

(1) k-Nearest Neighbors (k-NN), which predicts the feasibility of an unseen sample based on its Euclidian distance to training data in the feature space,
(2) Gradient Tree Boosting (GTB), or Gradient Boosting (GB) for short, an ensemble of decision trees whose components are feature values.

*Analysis of the prediction accuracy of traditional ML algorithms.* Table 4 shows the experimental results of the four competing techniques on the 13 testing sets. The baseline performance is the accuracy of a naive approach which predicts all configurations as feasible. Columns k-NN and GB respectively provides the prediction accuracy with k-NN and Gradient boosting. The values in red highlight discrepancies with other results on the same data set (*i.e.* testing sets 6, 9, 11 and 13). Testing sets 9 and 11, backbone-like topologies, as well as testing set 13, the specific Volvo topology, are very different from the star topologies of the training set configurations. Testing sets 6 and 7 have identical traffic patterns and very similar topologies (only the speed of two links has been changed to 1Gbit/s in testing set 7). The prediction accuracy is however significantly lower for testing set 6. This suggests that the performance of k-NN and GB is fragile to changes in the topology, i.e., they are not able to generalize.

There are several reasons why manual feature selection used with the traditional ML algorithms provides limited prediction accuracy. A first reason is that the selected features may not capture the right information to reason about the feasibility. For instance, the Gini index measures the unbalancedness of the link loads, its distribution however differs between center-based topology, which usually possesses well balanced loads, and backbone-like topology with high unbalancedness. As a result, the Gini index may be a misleading feature in many testing sets whose samples differ considerably from the training set. More importantly, the selected features, such as maximum load and Gini index, cannot effectively encode the complex interactions between flows, links and queues. The features cannot either capture basic information specific to each flow or link, like the speed of each link, the data payload of the flow's packets, etc. Finally, the learning algorithms are based on simple computational methods such as Euclidean distance or tree splitting and do not provide any mechanism to discover the underlying conditions required for a TSN configuration to be feasible. These limitations suggest that more consistent results could be obtained with other approaches like GNN that 1) use the entire raw information available as input, and 2) rely on more powerful learning paradigms.

Table 4. Prediction accuracy (%) of k-Nearest Neighbors (k-NN), Gradient Boosting (GB) and GNN on the 13 testing sets. The baseline is the proportion of feasible configurations in the data sets, En-MV and En-GB are the results of ensembles with majority voting and Gradient Boosting, respectively. We observe that conventional ML algorithms perform poorly on some testing sets (highlighted in red).

| **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 |

## 5.2 RQ2: Accuracy of GNN models

The aim here is to evaluate the accuracy of ensembles of GNN models by comparison with conventional ML algorithms. Ensembles are comprising 32 GNN models trained with the same parameters listed in Table 3.

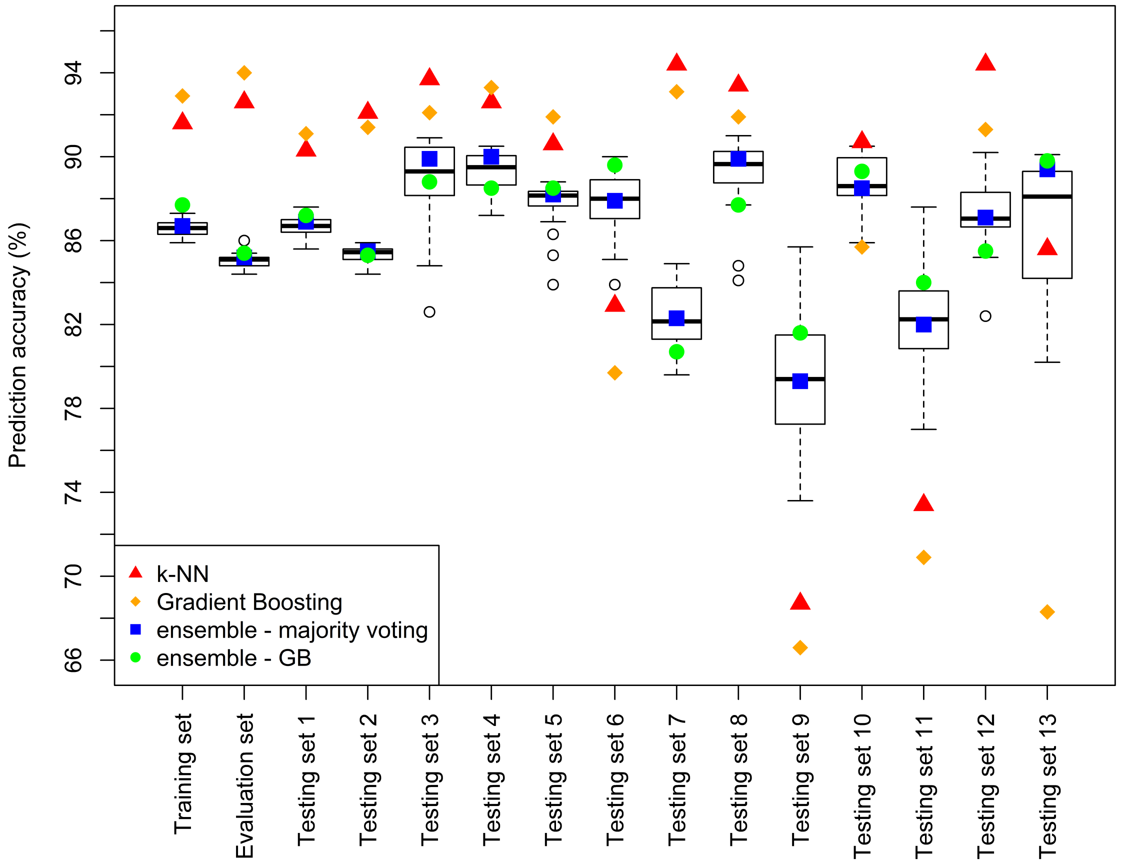


Fig. 6. Prediction accuracy (%) of ML models on all data sets. Some outliers have been removed. The boxplots show the variance within the 32 GNN models used in the two ensemble techniques. This variance increases on the right-hand side of the figure when the testing sets are more dissimilar from the training set.

The results on the 13 testing sets are represented graphically in Figure 6. The box plots indicate the variance within the GNN models, that is the *robustness* of GNNs, while the blue squares and green dots are the prediction accuracy of ensembles using majoring voting and GB, respectively. The results of k-NN and GB are denoted by red triangles and yellow diamonds. The details of the performance of all models are given in Table 4. It is worth noting that both k-NN and GB are deterministic algorithms, therefore ensembles cannot be used to boost their performance.

The box plots in Figure 6 show that the performance of the GNN models is robust on the first 3 testing sets, which are all center-based topologies. The prediction accuracy on testing sets 2 and 3 (either changing in their traffic or topology patterns with respect to the training set) does not considerably differ from testing set 1. This suggests what the GNN models learn from the training set generalizes beyond the training set.

When moving to the right of Figure 6, the variance in the performance of the GNN models tends to grow, as the differences between the testing and the training sets increase. Overall we observe that the performance

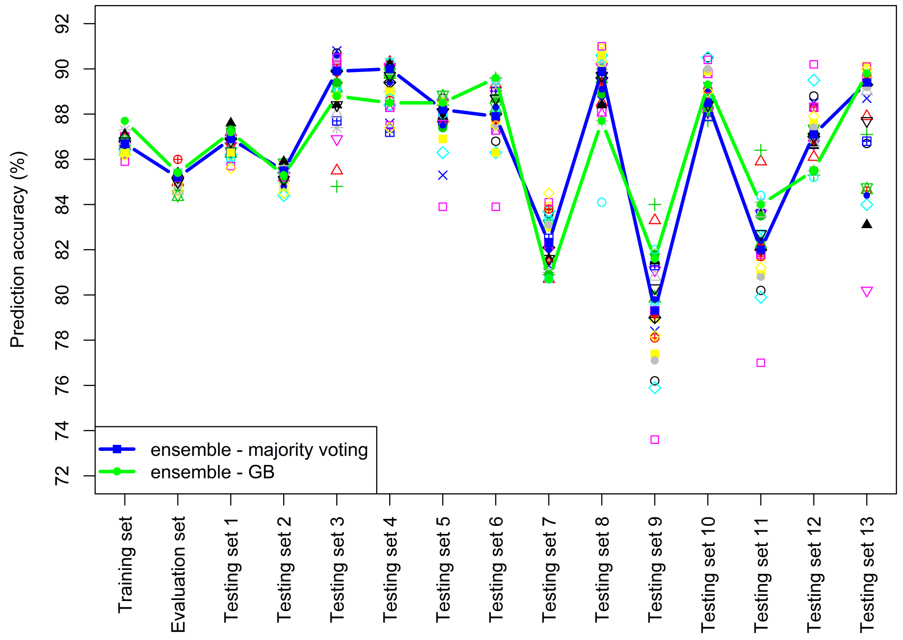


Fig. 7. Variance in the prediction accuracy of 32 GNN models (individual points with different shapes and colors). The blue and green lines are the accuracies obtained with ensembles, respectively majority voting and GB. We note that a GNN model performing well on a given testing set does not imply that the same model will perform consistently well among the other testing sets.

of ensembles of GNNs is less sensitive to changes in the topology than changes in the traffic patterns. For instance, the prediction accuracy of GNNs with testing set 5 is reduced by 1.4% compared to testing set 4 due to modifications in the traffic patterns. Importantly, the performance of ensembles of GNNs does not drop significantly with any testing sets containing unseen topologies or unseen traffic patterns, in contrast to the traditional ML algorithms.

On the other hand, Figure 7 also highlights a drawback of GNN, which is that some models may perform unacceptably poorly on some testing sets. For instance, a small proportion of the models have low accuracy with testing set 13. Such variabilities in the performance are an intrinsic characteristic of neural networks, which calls for the use of ensembles.

*Ensembles of GNNs as a fault-tolerant mechanism.* In general, the accuracy of the ensembles is close to the average of the 32 individual GNN models (see Table 4). The accuracy of the ensembles with majority voting and GB lies in the range [79.3, 90.0] and [81.6, 89.8] percents, respectively. It means that over all testing sets, ensembles predict feasibility correctly in at least 79% of the cases, and up to 90%. As can be seen in Figure 7, using ensembles is an effective technique to mitigate the problem of the large variability in the performance of GNN models, as such, ensemble techniques can be considered as a fault-tolerant mechanism for GNNs. We also observe that ensembles with GB are better than ensemble with majority voting in 9 out of the 13 testing sets (see Table 4). The difference is however not significant: 1.7% of improvement at most. This suggests that an ensemble with majority voting can be preferred due to its simplicity.

*False positive and false negative prediction rates.* A false positive is a configuration which is wrongly deemed feasible. Over the 13 testing sets, the ratio of false positives lies in the range [0.8, 17.6] percents (6.6% on average). A high false positive rate may lead to sub-optimal design choices, however it is not going to jeopardize the safety of the system as conventional techniques must be used anyway for the final configuration that is used at run-time.

The GNN model may also lead to false negatives: configurations which are wrongly predicted as non-feasible, and thus discarded. In the 13 testing sets, the ratio of false positives is in the range [0.1, 19.9] percents (6.7% on average). If the rate of false predictions cannot be precisely known in advance, it can be reduced using a hybrid model combining both ML and conventional schedulability analysis as explored in [23].

## 5.3 GNN models in the design space exploration of in-vehicle networks

DSE on virtual platforms is being increasingly experimented in the design of E/E architectures [5, 27, 32, 37]. Complexity of the technologies and architectures, time and cost effectiveness and, importantly, new evolutivity requirements through software update are key drivers for DSE. Other industrial domains, like industry 4.0 which has similar evolutivity requirements, should too benefit from such techniques.

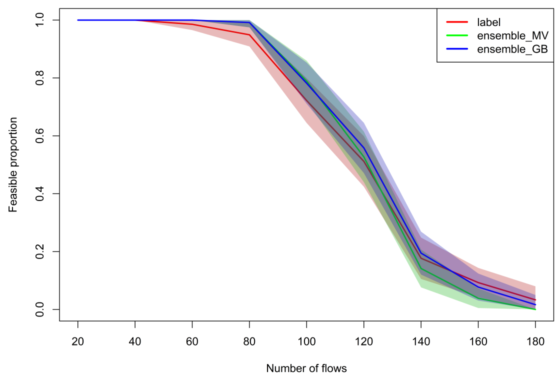
A typical DSE function is *Topology-Stress-Test* (TST) available in RTaW-Pegase which evaluates the probability that a communication architecture will be able to successfully support an additional number of flows, services or functions. It works by iteratively creating extended configurations of increasing size, configuring them, and then evaluating by schedulability analysis and simulation whether the application's requirements are met. As discussed in §1.2, the performance evaluation part is the bottleneck in that DSE algorithm. For instance, the experiments in [5, 27, 32], with a search space pruned as much as possible using domain knowledge, took hours to a few days of computation, which, in an industrial setting, hinders the R&D work. In this section, we discuss the new accuracy/turnaround time trade-off that can be obtained with the GNN model proposed in this work.

*Accuracy obtained with GNNs.* Figure 8 shows data obtained from a TST run on the FACE (testing set 4), Renault (testing set 6), small (testing set 12, Daimler topology in [30]) and Volvo topologies (testing set 13). Logically, we observe that when the number of flows increase ($x$ axis), the probability to meet all timing constraints decreases ($y$ axis). The curves are the predictions of the feasibility probabilities obtained with ensembles of 32 GNNs (green and blue curves) and the true probabilities (red curves) obtained by schedulability analysis. The colored region shows the areas within the 95% confidence interval.

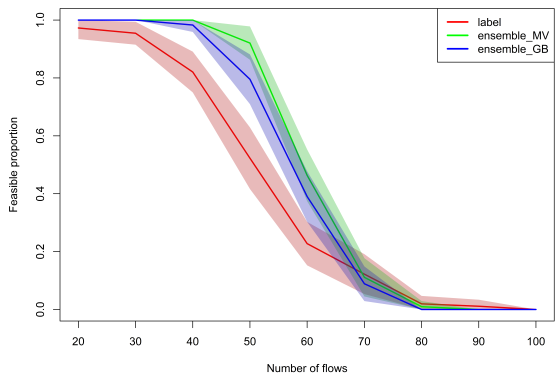
In Figure 8a, the prediction curves almost overlap the red curve, which can be explained by the fact that testing set 4 is similar to the training data. The curves in Figure 8b and Figure 8d, respectively for the Renault and Volvo topology are still close to the true labels, while these configurations are dissimilar from the training set. On the small topology in Figure 8c, there is however a gap between the predictions and the true values in the bottom part of the curve, indicating that the prediction is pessimistic. However, in practice designers focus on the upper part of the graphs, e.g. the part above 75% of success probability, as below the assurance that the architecture can be extended is too low. This experiment suggests that the accuracy of GNNs is high enough to be utilized in DSE to make early-stage design decisions in terms of architectures and technologies.

*Speedup factor with GNN.* Since the size of the embedding matrix and the connection matrix is fixed, the time it takes for a GNN model to predict the feasibility of any TSN configurations is almost constant (modulo caches states, instructions that do not execute in constant time, etc). In addition, all GNNs making up an ensemble, trained with the same parameters, behave identically from a timing point of view, allowing for an efficient parallelization. We summarize below the execution times of GNN models measured during the experiments:

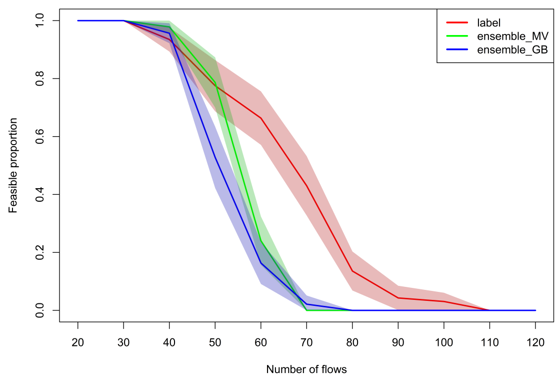
- *Training time*: each GNN model takes 10mn9s to be trained over 500 iterations with mini-batch of size 32, resulting in about 5h25mn to train the 32 models of an ensemble in a sequential manner on a standard



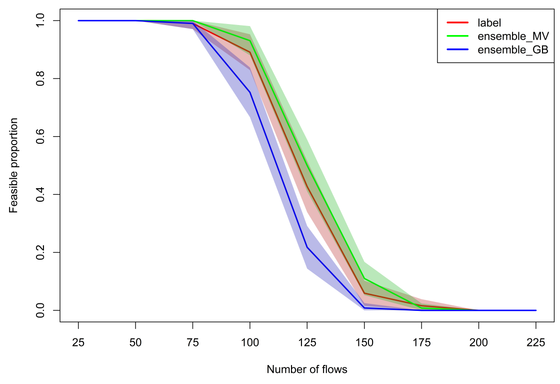
(a) FACE topology (testing set 4)



(b) Renault topology (testing set 6)



(c) Small topology (testing set 12)



(d) Volvo topology (testing set 13)

Fig. 8. Feasibility predictions with ensemble GNN models compared for 4 topologies to the true values. $x$ axis is the number of flows, $y$ axis the probability to meet all timing constraints. The areas within the 95% confidence interval are in color.

CPU. However, training time should not be counted in DSE since the models need to be trained only once, not for every new candidate architecture, and an industrial tool could ship with pre-trained models.

- *Testing time*: an ensemble of 32 GNN models with majority voting needs 88s to evaluate a testing set of 1000 configurations. An ensemble with GB takes slightly more time (+1ms per configuration), which can be ignored in our context. Considering the simplest topology, *i.e.*, the small topology in testing set 12, schedulability analysis takes 6788s, that is 1h53mn, to verify the same number of TSN configurations. For the largest topology, *i.e*, FACE topology of testing set 4, schedulability analysis requires 41h55mn. In our experiments, ensembles of 32 GNN models provides a speedup factor ranging from 77.13 up to 1714.77 compared to schedulability analysis.

## 6 RELATED WORKS: MACHINE LEARNING FOR NETWORKING

ML techniques [14, 16] have been used in a very wide range of applications, including object recognition, natural language processing, advertising and autonomous driving, to name a few. ML has also been applied in networking to solve specific tasks like intrusion detection system (IDS), dynamic routing and performance improvement at large [44]. For instance, [33] proposes an ML framework to predict round-trip time (RTT) of TCP connections at run time, whereas in [45] ML is used to generate better TCP congestion control protocols. Many studies have also applied advanced ML techniques, deep learning is for instance used in [25] to compute dynamic packet routing, and evolutionary reinforcement learning is used in [22] to guarantee balanced task assignment in fog computing networks.

Recently, a new model of deep learning called graph neural network (GNN) has been emerging as a promising tool to solve problems in networking. A study in [10] shows that GNN can predict the throughput of TCP connections with good accuracy. The same authors use GNN to learn and generate routing protocols among distributed networked nodes [12], while Rusek et al. demonstrates the potential of GNN to optimization routing in Software-Defined Networks [40]. Recently, GNN has been used to speed up the verification of Multiprotocol Label Switching (MPLS) configuration [13]. Another impressive application by the same authors is [11] where GNN is used to predict contention occuring between flows and use that information in Network Calculus analysis, a powerful formal framework for deterministic network analysis.

In the field of Ethernet TSN, the first application of ML, briefly introduced in Section 1, is predicting whether a TSN configuration is feasible. Both supervised and unsupervised ML algorithms are introduced [24, 28] and show promising results with all the TSN QoS mechanisms considered. A follow-up work in [23] introduces a hybrid approach, blending ML and mathematical analysis, to control the false prediction rate of ML models while still achieving a considerable speedup factor. It is worth noting that these approaches have only been validated with topologies used both in the training and testing sets. In this work, we make use of GNN that is tailored to model and learn on structured data, and thus possesses the potential to generalize over more diverse topologies and traffic patterns.

We emphasize the difference between our study and the existing work using GNN in real-time systems. The authors in [11] use a GNN model to learn and predict efficient Network Calculus models to avoid an exhaustive search during a feasibility test, while our GNN model predicts whether the optimal priority exists given a TSN configuration. Since the optimal priority assignment algorithm requires multiple feasibility tests and each test implies correct predictions of the feasibility of all flows, the problem our model aims to solve in this paper is actually broader.

## 7 CONCLUSION AND FUTURE WORK

This study is a contribution towards leveraging deep learning to further automate the design of industrial systems and, ultimately, design systems that are more efficient in terms of resource usage. Two well identified use-cases of deep-learning, and AI at large, in the design of critical systems are 1) fast prediction techniques that can replace, at some stages of the design, exact approaches, and 2) technology-agnostic configuration algorithms, *i.e.* algorithms not relying on extensive domain knowledge.

This paper contributes to the first use-case and presents what is, to the best of our knowledge, the first deep learning model for feasibility prediction of real-time Ethernet networks. A practical advantage of our model is that it automates the feature engineering process, and does not require domain expertise. In that regard, the model could potentially be efficient in other areas of real-time computing. We also introduce the use of ensembles of GNN models to control the uncertainty of individual models and improve the average prediction accuracy. In our experiments on realistic electronic architectures, ensembles of 32 GNN models provide a prediction accuracy ranging from 79.3% to 90%, with a speedup factor from 77 to 1715 compared to schedulability analysis. Such speed-up factors unlock new possibilities for design-space exploration and the development of near-interactive design tools.

We would like to emphasize the importance of an extensive evaluation before deploying ML models in production in the industry. For instance, experiments in this paper, unveil a main limitation of conventional ML algorithms: they do not perform consistently well on unseen data that are dissimilar from the training sets. On the contrary, over the 13 testing sets used in this work, the GNN model has proven an ability to generalize beyond the training data that is superior to traditional ML algorithms. In addition, while the techniques to improve the accuracy of k-NN or GB are limited, there are in our view possibilities to improve the GNN model by using larger training data sets, increasing the number of layers or number of neurons in neural networks, by fine tuning their parameters, or applying more advanced techniques such as Gate Recurrent Unit (GRU) or LSTM, to name a few. The model could be further improved with a dedicated study on the optimal size for the ensembles of GNNs and the best technique to use. This study should include stacking and XGBoost as candidate approaches.

A natural follow-up work is to extend the GNN model to predict the feasibility of networks implementing other TSN QoS policies than priorities, for instance traffic shaping (CBS [17]) and frame preemption ([19]). Ultimately, an ML-based prediction and configuration framework could encompass all TSN standards required for a given application domain.

## ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers whose comments and suggestions helped improve and clarify this manuscript.

## REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] RealTime at Work. [n.d.]. RTaW-Pegase: Modeling, Simulation and automated Configuration of communication networks. Retrieved 2019/01/24, https://www.realtimeatwork.com/software/rtaw-pegase.

[3] N.C. Audsley. 2001. On priority assignment in fixed priority scheduling. *Inform. Process. Lett.* 79, 1 (2001), 39–44.

[4] P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[5] O. Creighton, N. Navet, P. Keller, and J. Migge. 2020. Towards Computer-Aided, Iterative TSN-and Ethernet-based E/E Architecture Design. In *2020 IEEE Standards Association (IEEE-SA) Ethernet & IP @ Automotive Technology Day*. Munich. http://hdl.handle.net/10993/44490

[6] R.I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns. 2016. A Review of Priority Assignment in Real-Time Systems. *J. Syst. Archit.* 65, C (April 2016), 64–82. https://doi.org/10.1016/j.sysarc.2016.04.002

[7] R.I. Davis and N. Navet. 2012. 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*. 33–42.

[8] F.A. Farris. 2010. The Gini index and measures of inequality. *The American Mathematical Monthly* 117, 10 (2010), 851–864.

[9] J. Frankle and M. Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).

[10] F. Geyer. 2017. Performance evaluation of network topologies using graph-based deep learning. In *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools*. 20–27.

[11] F. Geyer and S. Bondorf. 2019. DeepTMA: predicting effective contention models for network calculus using graph neural networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1009–1017.

[12] F. Geyer and G. Carle. 2018. 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*. 40–45.

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

[14] I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep learning*. MIT press.

[15] T. Hamza, J.L. Scharbarg, and C. Fraboul. 2014. Priority assignment on an avionics switched Ethernet Network (QoS AFDX). In *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*. IEEE, 1–8.

[16] T. Hastie, R. Tibshirani, and J. Friedman. 2009. *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media.

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

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

[19] IEEE. 2016. *IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption* (IEEE Std 802.1Qbu-2016 ed.).

[20] IEEE. 2018. *IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks* (IEEE Std 802.1Q-2018 ed.).

[21] A. Karpathy. [n.d.]. A Recipe for Training Neural Networks. http://karpathy.github.io/2019/04/25/recipe/ Retrieved 2020/08/14.

[22] T.L. Mai, N.N. Dao, and M. Park. 2018. Real-time task assignment approach leveraging reinforcement learning with evolution strategies for long-term latency minimization in fog computing. *Sensors* 18, 9 (2018), 2830.

[23] T.L. Mai, N. Navet, and J. Migge. 2019. 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, 1–8.

[24] T.L. Mai, N. Navet, and J. Migge. 2019. 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.* 143–153.

[25] B. Mao, Z.M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani. 2017. Routing or computing? The paradigm shift towards intelligent computer network packet transmission based on deep learning. *IEEE Trans. Comput.* 66, 11 (2017), 1946–1960.

[26] A. Nasrallah, A.S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury. 2018. 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.

[27] N. Navet, H.H. Bengtsson, and J. Migge. 2020. Early-stage Bottleneck Identification and Removal in TSN Networks. In *Automotive Ethernet Congress.* Munich. http://hdl.handle.net/10993/46282

[28] N. Navet, T.L. Mai, and J. Migge. 2019. *Using machine learning to speed up the design space exploration of Ethernet TSN networks.* Technical Report. University of Luxembourg. http://hdl.handle.net/10993/38604

[29] N. Navet, J. Migge, J. Villanueva, and M. Boyer. 2018. 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). https://doi.org/10.4271/2018-01-0756

[30] N. Navet, J. Seyler, and J. Migge. 2016. 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. https://hal.archives-ouvertes.fr/hal-01292312

[31] N. Navet, J. Villanueva, and J. Migge. 2018. 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.

[32] N. Navet, J. Villanueva, and J. Migge. 2019. 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. http://hdl.handle.net/10993/40623

[33] B.A.A. Nunes, K. Veenstra, W. Ballenthin, S. Lukin, and K. Obraczka. 2014. A machine learning framework for TCP round-trip time estimation. *EURASIP Journal on Wireless Communications and Networking* 2014, 1 (2014), 47.

[34] R.S. Olson, W. La Cava, Z. Mustahsan, A. Varik, and J.H. Moore. [n.d.]. *Data-driven advice for applying machine learning to bioinformatics problems.* 192–203. https://doi.org/10.1142/9789813235533_0018

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

[36] R. Pell. 2020. Scientific ML promises "near interactive" design optimization speeds, Smart2Zero, European Business Press. https://www.smart2zero.com/news/scientific-ml-promises-near-interactive-design-optimization-speeds?news_id=128523

[37] O. Philipp, K. Stefan, and S. Eric. 2019. 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, 128–138. https://doi.org/10.1109/MODELS.2019.000-8

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

[39] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M.P. Reyes, M.L. Shyu, S.C. Chen, and S.S. Iyengar. 2018. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)* 51, 5 (2018), 1–36.

[40] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio. 2019. Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN. In *Proceedings of the 2019 ACM Symposium on SDN Research.* 140–151.

[41] T. Salimans and D.P. Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems.* 901–909.

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

[43] B. Victor. 2012. Inventing on Principle. https://www.youtube.com/watch?v=PUv66718DII

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

[45] K. Winstein and H. Balakrishnan. 2013. TCP Ex Machina: Computer-generated congestion control. *ACM SIGCOMM Computer Communication Review* 43, 4, 123–134.

[46] L. Zhao, Q. Li, Y. Xiong, Z. Zheng, and H. Xiong. 2013. Using multi-link grouping technique to achieve tight latency in network calculus. In *2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC)*. IEEE, 2E3–1.