Learning-Assisted Schedulability Analysis: Opportunities and Limitations

Sanjoy Baruah[1*], Pontus Ekberg[2] and Marion Sudvarg[3]

[1*]Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, Missouri, United States.
[2]Department of Information Technology, Uppsala University, Uppsala, Sweden.
[3]Department of Physics, Washington University in St. Louis, St. Louis, Missouri, United States.

*Corresponding author(s). E-mail(s): baruah@wustl.edu;
Contributing authors: pontus.ekberg@it.uu.se; msudvarg@wustl.edu;

Abstract

We present the first (to our knowledge) Deep-Learning based framework for real-time schedulability-analysis that guarantees to never incorrectly mis-classify an unschedulable system as being schedulable, and is hence suitable for use in safety-critical scenarios. We relate applicability of this framework to well-understood concepts in computational complexity theory: membership in the complexity class NP. We apply the framework upon the widely-studied schedulability analysis problems of determining whether a given constrained-deadline sporadic task system is schedulable on a preemptive uniprocessor under both Deadline-Monotonic and EDF scheduling. As a proof-of-concept, we implement our framework for Deadline-Monotonic scheduling, and demonstrate that it has a predictive accuracy exceeding **70%** for systems of as many as 20 tasks *without making any unsafe predictions*. Furthermore, the implementation has very small (**<1** ms on two widely-used embedded platforms; **<4** $\boldsymbol{\mu}$s on an embedded FPGA) and highly predictable running times.

Keywords: Schedulability analysis; Computational complexity: NP-completeness; Learning-Enabled Components (LECs); Deep Learning

1 Introduction

With Deep Learning (DL) already widely used in autonomous Cyber-Physical Systems (CPS's) for purposes of perception, research efforts are underway to also use it to *speed up computation* – this is particularly meaningful for autonomous CPS's that are not tethered to the power grid and hence must make do with relatively simple computing platforms on board. In this work we investigate the use of DL to speed up a form of computation that is commonly and repeatedly performed in real-time CPS's: *schedulability analysis*, which is the process of validating the correctness of timing properties. Many basic and fundamental forms of schedulability analysis are known to be computationally intractable and hence applying DL to speed it up seems a reasonable goal. However, schedulability is frequently a safety-critical property: incorrectly mis-classifying an unschedulable system as being schedulable could have potentially catastrophic consequences. There is, to our knowledge, no prior DL-based schedulability analysis that guarantees to never return 'false positives' — to incorrectly declare some unschedulable system to be schedulable. In this paper, *we are proposing the first conceptual framework for using Deep Learning for schedulability analysis that guarantees to return no false positives*, and is hence suitable for use in safety-critical systems.

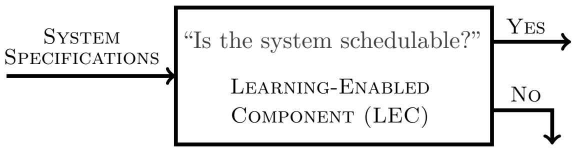


Fig. 1 LEC-based schedulability analysis

Envisioned use-cases.

Safety-critical systems were traditionally relatively simple and closed, and were intended to operate under tightly controlled conditions. This is rapidly changing: modern CPS's can be enormously complex and are required to operate safely and effectively in open environments that are characterized by a good deal of uncertainty. With such systems becoming increasingly more dynamic as a means of being adaptive to changing conditions in their operating environments, schedulability analysis algorithms need frequent re-execution during run-time (often as part of *admission control* procedures) as the workload and/ or platform changes in ways that were not anticipated during pre-runtime analysis. Pseudo-polynomial running times are often far too large for such algorithms to be suitable for runtime use. This directly leads to a need for extremely efficient schedulability-analysis algorithms, often upon computationally very limited platforms, which motivates the question that is explored in this manuscript: can we train *Learning-Enabled Components* (LECs) to classify system specifications as either satisfying a given schedulability property, or failing to do so? – see Figure 1. Doing so enables the safety-critical computing community to leverage off the tremendous advances in DL and related AI technologies that have occurred over the past two decades or so. However, although DL has proved very effective in solving a wide range of problems, it has also been observed [1] that DL does not necessarily perform very well upon *all* problems: given the increasing need for rapid schedulability analysis, we believe it merits investigation whether the approach of Figure 1 is (or can be rendered) effective for schedulability analysis.

This work.

In this manuscript we report on our findings from a conceptual and experimental evaluation of DL-based schedulability analysis, that we have conducted with the goal of understanding its scope and limitations. The ***main conclusion*** that we are able to draw is this:

> Deep Learning (DL) is applicable for solving some, but not all, schedulability-analysis problems of interest. There is a systematic approach for determining whether DL is applicable for solving a given schedulability-analysis problem. A framework can be defined for applying DL upon those schedulability-analysis problems for which it is determined to be applicable.

This conclusion suggests a two-step approach to applying DL for schedulability analysis: *(i)* **identifying** schedulability-analysis problems that can be delegated to DL and determining how such delegation is to be done; and *(ii)* actually **developing** DL systems for solving these problems. *This paper primarily focuses on the first step*: figuring out how to identify schedulability-analysis problems that are amenable to solution using DL-based techniques, and defining a DL-based framework for solving these problems. We believe that developing the 'best' DL systems for those problems that are identified as being suitable requires close collaboration with experts in Machine Learning with the requisite knowledge and skills to choose and train the appropriate NN architectures. That is in itself an entire research project, which, while critically important in order to make best use of the results we derive here, does not fall within the scope of the ideas that we seek to present in this paper. We therefore defer detailed investigation on this second step to future work; here, we focus on the first step, and use simple proof-of-concept implementations for well-studied schedulability-analysis problems to demonstrate the relevance and applicability of our proposed approach and the accompanying framework.[1]

Contributions.

The main contribution of this paper is **the development of a conceptual framework** for using Deep Learning for schedulability analysis that guarantees to never incorrectly classify an unschedulable system as being schedulable; this is, to our knowledge, the *first* work on DL-based schedulability analysis that can make such a guarantee. In greater detail:

- We derive an exact (necessary and sufficient) condition for our framework to be applicable. That is, we *identify a precise condition* (stated as Proposition (1) in

[1] In other words, we are not claiming that our DL implementations are the best possible: while we realize that they may perhaps be improved by making use of more advanced results from Deep Learning, we consider doing so to be beyond the scope of this paper.

Section 3) for determining whether any particular schedulability-analysis problem is suitable for solving via our framework.
- We *illustrate the applicability* of Proposition (1) by identifying schedulability-analysis problems that are amenable to DL-based solution, as well as ones that are not. We develop simple proof-of-principle implementations of DNN-based schedulability tests for some of the schedulability-analysis problems that are shown to be amenable to DL-based solution, and *experimentally* evaluate these DNNs along various dimensions (their effectiveness; run-time overheads; FPGA implementation) upon synthetically generated workloads.

Organization. The remainder of this manuscript is organized in the following manner. In Section 2 we formally describe the specific schedulability-analysis problems that we will be studying from a DL perspective. We present our proposed framework for DL-based schedulability analysis in Section 3. We have implemented and evaluated this framework on the problems that are described in Section 2; our evaluation experiments are detailed in Section 4. We conclude in Section 5 by discussing some related work and placing our results within the larger context of real-time scheduling theory.

2 Background: Schedulability Analysis

In this section we briefly describe (and provide the needed background information on) the schedulability-analysis problems that we will, in the following sections, examine from the perspective of developing DL-based solutions. Since our emphasis in this paper is primarily on Deep Learning, we have chosen to focus upon very simple and particularly well-studied schedulability-analysis problems with which most members of the real-time computing community are already familiar. In Section 5 (paragraph titled "**Other schedulability-analysis problems**") we will briefly discuss how the ideas contained in this paper may be generalized and extended to additional schedulability-analysis problems, and list some such problems.

The sporadic tasks model [2].

The scheduling of collections of independent sporadic tasks $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ upon a shared preemptive processor is one of the most widely-studied problems in real-time scheduling theory. Each sporadic task $\tau_i = (C_i, D_i, T_i)$ is characterized by three non-negative integer parameters: its worst-case execution time (or WCET) C_i, its relative deadline D_i, and its inter-arrival separation parameter (or period) T_i. Sporadic task systems with $D_i \leq T_i$ for all tasks are called constrained-deadline systems. We consider the following two schedulability-analysis problems: is a given constrained-deadline sporadic task system guaranteed to always meet all deadlines upon a preemptive uniprocessor platform, when scheduled using the *(i)* Fixed-Priority (FP) and *(ii)* Earliest-Deadline First (EDF) scheduling algorithms?

Fixed-Priority (FP).

In FP scheduling, each task is statically assigned a priority prior to run-time and at each instant during run-time the currently active job that has been generated by the highest-priority task is scheduled for execution.[2] Determining whether a given task system is FP-schedulable is known to be NP-complete [4, 5]; hence, it makes sense to explore the use of deep learning to speed up FP-schedulability analysis.

It has been shown [6–8] that a necessary and sufficient FP-schedulability condition for task system Γ is that for each $\tau_i \in \Gamma$, the recurrence:

$$R_i \geq C_i + \sum_{\tau_j \in \mathrm{hp}(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \tag{1}$$

should have a positive solution for R_i that is no larger than τ_i's relative deadline D_i (here, $\mathrm{hp}(\tau_i)$ denotes the tasks with greater priority than τ_i). *Response-Time Analysis (RTA)* deploys straightforward techniques for solving such recurrences to determine the smallest value of R_i satisfying this recurrence for each τ_i, and declares the system to be FP-schedulable if and only if $R_i \leq D_i$ holds for all $\tau_i \in \Gamma$.

Earliest-Deadline First (EDF).

In EDF scheduling, jobs are prioritized according to their deadlines: at each instant during run-time the currently active job whose deadline (arrival time + relative-deadline parameter of the task that generated it) is the closest in the future is scheduled for execution. EDF-schedulability analysis is known to be coNP-complete [9], and it is therefore again meaningful to explore whether deep learning can help speed things up. *Processor Demand Analysis (PDA)* is an exact technique for schedulability analysis of constrained-deadline sporadic task systems that are scheduled by EDF upon a preemptive uniprocessor. This technique is centered upon the concept of the *demand bound function* (DBF): for any sporadic task $\tau_i = (C_i, D_i, T_i)$ and any interval-duration $t \geq 0$, $\mathrm{DBF}_i(t)$ denotes the maximum possible cumulative execution requirement by jobs of task τ_i that both arrive in, and have their deadlines within, any contiguous interval of duration t. The following formula for computing $\mathrm{DBF}_i(t)$ was derived in [2]:

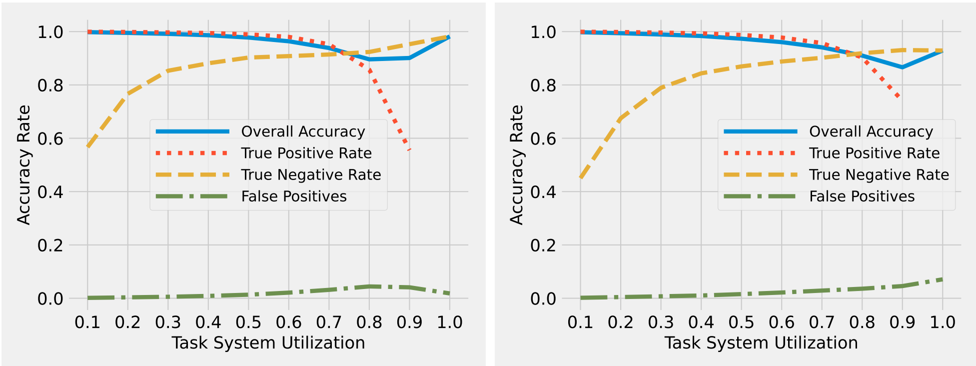
$$\mathrm{DBF}_i(t) = \max\left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1, 0\right) \cdot C_i \tag{2}$$

and it was shown that a necessary and sufficient condition for $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be EDF-schedulable upon a preemptive unit-speed processor is that the following condition hold for all $t \geq 0$:

$$\sum_{\tau_i \in \Gamma} \mathrm{DBF}_i(t) \leq t \tag{3}$$

It was also proved in [2] that Condition (3) need only be checked for values of t that are of the form $t \equiv (k \times T_i + D_i)$ for some non-negative integer k and some $i, 1 \leq i \leq n$; furthermore, only such values that are no larger than the least common multiple of the T_i parameters of all the tasks need be tested. The set of all such values of t for

[2] It is known [3, Thm 2.4] that the deadline monotonic (DM) priority assignment, in which tasks with smaller D_i parameters are assigned greater priority, is optimal for constrained-deadline sporadic task systems. Hence, we focus our attention in this paper on FP-schedulability analysis of systems for which priorities are assigned in DM-order.



(a) FP Schedulability

(b) EDF Schedulability

Fig. 2 Performance of DNN schedulability classifiers for systems of 4 tasks, plotted as a function of system utilization – see Section 3.1. The 'Overall Accuracy' curve denotes the fraction of generated task systems that are correctly classified by the DNN as being schedulable or not. The 'True Positive Rate' ('True Negative Rate,' respectively) curve denotes the fraction of schedulable (not schedulable, resp.) task systems that are correctly identified as such. The 'False Positives' curve denotes the fraction of generated task systems that are incorrectly classified by the DNN as being schedulable.

which it needs to be checked that Condition (3) is satisfied in order to verify EDF-schedulability is called the *testing set* for task system Γ and often denoted $\mathcal{T}(\Gamma)$. It is known [2] that the cardinality $|\mathcal{T}(\Gamma)|$ of the testing set $\mathcal{T}(\Gamma)$ may in general be exponential in the representation of Γ; however, it has been shown [10, Theorem (3.1)] that a smaller testing set, of cardinality pseudo-polynomial in the representation of Γ, can be identified for *bounded-utilization* task systems —systems Γ satisfying the additional condition that $\sum_{\tau_i \in \Gamma} U_i \leq c$ for some constant c strictly smaller than 1.

3 A Framework for Learning-Enabled Schedulability Analysis

In this section we motivate and describe our proposed framework for enabling the safe and effective use of DL for doing schedulability analysis. We start out (Section 3.1) briefly describing DL-based implementations that we have built, according to the framework provided in Figure 1, for our two schedulability-analysis problems of interest (preemptive uniprocessor FP- and EDF-schedulability analysis of constrained-deadline sporadic task systems). In Section 3.2 we point out some problems that arise in such implementations. We propose a solution to these problems in Section 3.3 by defining an enhancement, in Figure 3, to the earlier framework of Figure 1, and derive, in Section 3.4, a precise condition for determining which schedulability-analysis problems are amenable to solution using this enhanced framework.

3.1 LECs for schedulability analysis

As stated in Section 1, the goal of this research is to develop LECs based on deep learning for doing schedulability analysis. As a first step towards achieving this goal, we trained simple *multilayer perceptrons* (MLPs) to perform FP and EDF schedulability-analysis for small task systems in accordance with the framework of Figure 1. In particular, we trained a pair of networks, each with two 15-node fully-connected hidden layers, to perform binary classification for predicting FP and EDF schedulability respectively for sporadic task systems of 4 tasks[3] – the observed performance of these networks are presented in Figure 2. Two important observations emerged:

1. DL appears to be very effective in classifying systems as schedulable or not: we see from Figure 2 that for 4-task systems, predictive accuracy exceeds 95% for both FP and EDF schedulability analysis. (Additional experiments, reported in Section 4, indicate that prediction accuracy does not degrade too steeply with system size: it still exceeds 92% for FP schedulability of 20-task systems.)
2. DL makes occasional mistakes: classification accuracy is not 100% for either FP or EDF schedulability analysis.

The first of these observations is grounds for optimism: it shows the promise of DL for identifying schedulable systems. The second observation, however, gives us pause since it emphasizes the well-known fact that Deep Learning will occasionally make mistakes: erroneously classify a schedulable system as unschedulable, or vice versa. We must understand the consequences of such errors, and take mitigative steps to ensure they do not compromise system safety, before we can use LEC-based schedulability analysis in safety-critical systems. We point out that classification errors are of two kinds:

1. A FALSE NEGATIVE, with a schedulable system incorrectly classified as being unschedulable; or
2. A FALSE POSITIVE, whereby an unschedulable system is classified as being schedulable.

Below we discuss the implications of each kind of error.

3.2 The problem with False Positives

We saw above that LECs for schedulability analysis are, while effective, liable to making occasional mis-classifications – both false negatives and false positives. A false negative may result in a schedulable system being needlessly rejected as being unschedulable, but this is a necessary consequence of using Deep Learning: DL, by its very nature, solves problems approximately rather than exactly. However, *false positives present a safety hazard* since a potentially unschedulable system is misidentified as being schedulable. Though the number of false-positives for our binary classifiers were low (of the systems of 4 tasks that we generated, 1.8% were incorrectly deemed DM schedulable and 2.1% EDF schedulable), *the only acceptable rate for safety-critical systems is zero* and so we must be able to eliminate *all* false positives if we are to use DL for schedulability-analysis for safety-critical systems.

To eliminate the possibility of false positives, we propose that when DL-based components are used for schedulability-analysis and declare a system to be schedulable, they be additionally required to generate a *justification* for this decision in the form of a ***certificate***. Note that the certificate itself may serve as both a declaration, and a justification, of schedulability — it should not be necessary to execute separate networks to produce a classification and a certificate. We require that this certificate must be

[3] A detailed description of the training process and experiments conducted is provided in Section 4.

efficiently verifiable by a (different) algorithm that is based on 'traditional' algorithmic techniques in that it does not make use of Deep Learning and related AI techniques; it is only if this verification algorithm agrees that the certificate validates schedulability do we deem the system specifications to have passed the schedulability-analysis test.

This proposed enhanced framework for DL-based schedulability analysis is depicted in Figure 3.

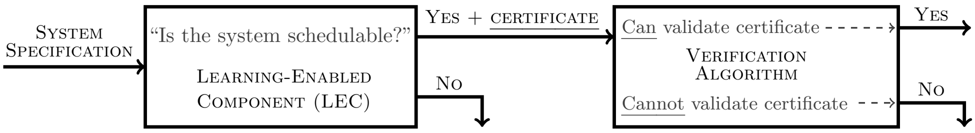


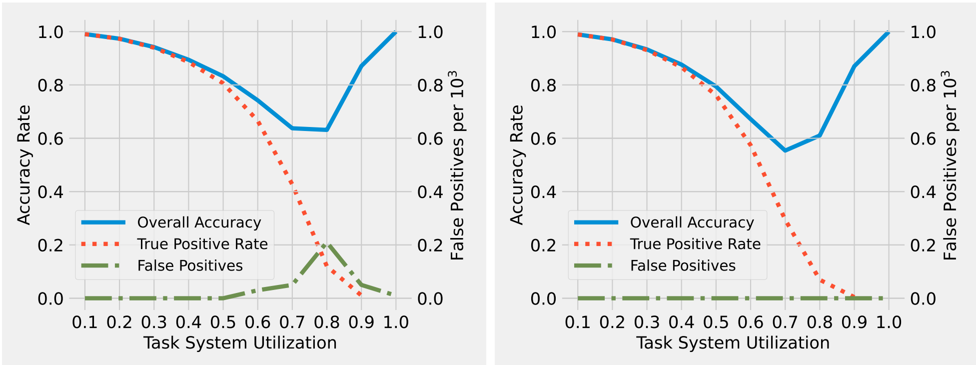
Fig. 3 A Framework for LEC-based Safety Verification. The LEC must additionally generate a *certificate* for any system determined to be schedulable; this certificate should be efficiently verifiable by the verification algorithm.

3.3 Choosing Suitable Certificates

Our proposed framework for DL-based schedulability analysis (depicted in Figure 3) requires that the LEC generate a certificate for systems it classifies as schedulable. But what should this certificate look like? To understand this, let us separately consider each of the two schedulability-analysis problems for which we have developed LECs as discussed in Section 3.1.

FP schedulability. Recall, from Section 2, that task system Γ is FP-schedulable if and only if there is a value of R_i no larger than D_i satisfying Recurrence (1) for each $\tau_i \in \Gamma$. A certificate for the FP-schedulability of task system Γ could simply be such values for R_i, one per task in Γ; given such a certificate, the module labeled VERIFICATION ALGORITHM in Figure 3 can clearly efficiently verify that for each $\tau_i \in \Gamma$, the provided value of R_i does indeed satisfy Recurrence (1) and is $\leq D_i$.

To investigate whether we could get LECs to generate such certificates, we trained an alternative set of MLPs to predict the R_i values via regression, rather than simply (as in our initial strawman approach) providing a binary classification. The network for doing so contains 4 fully-connected hidden layers, each with 30 neurons (more details are provided in Section 4). A task system is deemed to be FP-schedulable if these predicted R_i values are each $\leq$ the corresponding D_i values; we again plot the predictive accuracy in Figure 4 (a). Note that the predictive accuracy in this plot is generally lower than in the corresponding plot for the binary (schedulable/ unschedulable) classifier (Figure 2 (a)); it is, however, not unacceptably low in light of the fact (also stated earlier) that we are reconciled to approximate, rather than exact, solutions from DL. Furthermore in this case, we can *validate* claims of schedulability by having a verification algorithm check that the certificates generated by the MLP do indeed satisfy the corresponding response-time equations (Recurrence (1)) – we plot the accuracy post-validation in Figure 4 (b). Note that, although accuracy overall decreases slightly with verification (from 85.1% to 82.7%), unsafe false positives are eliminated entirely.



(a) Unverified schedulability (overall acc.: 85.1%)   (b) Verified schedulability (overall acc.: 82.7%)

Fig. 4 FP schedulability with certificates for sets of 4 tasks. Note the different scale of the right-side y-axes for false positives. Overall (i.e., summing across all utilizations), 74.1% of schedulable systems were verifiably identified as being such.

EDF schedulability. Let us now turn our attention to EDF schedulability: what should the certificates to be generated by the LEC be? An examination of the EDF schedulability-analysis condition reveals that Expression (3) $\left(\sum_{\tau_i \in \Gamma} \text{DBF}_i(t) \leq t\right)$ is required to hold for *all* values of t in the testing set $\mathcal{T}(\Gamma)$. And since $\mathcal{T}(\Gamma)$ may contain exponentially many distinct values of t, a certificate enumerating all elements of $\mathcal{T}(\Gamma)$ would require that the module labeled VERIFICATION ALGORITHM in Figure 3 take exponential time to verify the veracity of this certificate, thereby negating the very purpose of using LEC's to speed up schedulability-analysis. Thus the idea that worked above for FP-schedulability, of having the LEC generate a certificate that can be used by the verification algorithm for validating the associated schedulability condition (Recurrence (1)) appears to not be applicable for EDF-schedulability. Indeed, *we were unable to instantiate the framework of Figure 3 to become applicable for EDF-schedulability*; in Section 3.4 below we show that it follows from computational complexity theory [11, 12] that we are unlikely to be able to do so.

3.4 The applicability of the proposed framework

Let us examine the framework of Figure 3 a bit more closely. Recall that our goal in using DL for schedulability analysis is to obtain greater run-time efficiency: we want to be able to make schedulability-analysis decisions faster than could be done using traditional schedulability-analysis algorithms. Now, there is a lot of excellent research on how one should implement LECs (particularly DNN-based ones) to have efficient (and predictable) running times (see, e.g., [13–15] – this list is by no means exhaustive); we expect that one can use the results of this research to obtain very efficient implementations of the LEC in Figure 3 (indeed, we demonstrate examples of this in Section 4). That leaves the verifier of Figure 3: we want this, too, to be implemented in an efficient manner. We argue that it is reasonable to require that this verifier should have running time no worse than a (low-order) polynomial in the size of the task system whose schedulability is being determined. This requirement immediately relates the applicability of the framework of Figure 3 to well-studied concepts in computational complexity theory [11, 12], in particular, the complexity class NP – "NP *is the class of [problems] that can be verified by a polynomial-time algorithm.*" [16, p. 1058]. Hence the requirement that the certificate be verifiable in polynomial time implies that the framework is applicable to schedulability-analysis problems that are in NP; this is formally stated in the following proposition:

Proposition 1. Restricting that the module labeled "VERIFICATION ALGORITHM" in Figure 3 have no worse than polynomial running time, it is necessary and sufficient for a schedulability condition to belong to the complexity class NP in order for it to be checkable using the framework of Figure 3. □

Hence, in order to determine whether a schedulability-analysis problem can be verified using DL through the framework presented in Figure 3 or not, it is necessary to demonstrate its membership (or non-membership, respectively) in the complexity class NP. To prove that a schedulability-analysis problem belongs to NP, one must furnish a polynomial-time verification algorithm for the problem. However, how can one demonstrate its *non*-membership in NP? In this case, established results from computational complexity theory come into play. There exist various complexity classes (a few are depicted in Figure 5) that are very widely believed to be distinct from NP, meaning they contain problems $\notin$ NP. Recall from computational complexity theory that a problem is considered *hard* for a complexity class if it is, in an intuitive sense, at least as computationally difficult to solve as every other problem within that class (or more precisely, every problem in the complexity class can be polynomial-time reduced to this hard problem). Thus, *showing a schedulability-analysis problem to be hard (or complete) for any complexity class believed to be distinct from* NP *(such as* coNP*) provides substantial evidence that it is not a member of* NP.

The conclusions we had drawn from first principles in Section 3.3, that FP-schedulability analysis fits the framework of Figure 3 whereas EDF-schedulability analysis does not, follow directly from Proposition 1: as stated in Section 2, FP-schedulability analysis is NP-complete [5] and therefore in NP; EDF-schedulability analysis, however, is coNP-complete [9] and therefore not in NP (assuming the widely-believed conjecture that NP $\neq$ coNP – see Figure 5).

4 Evaluation

In this section we describe and discuss the experiments that we have conducted for evaluating, from various perspectives (including predictive accuracy and run-time implementation overhead, as well as the possibility of FPGA implementation), the effectiveness of DL-based solutions for preemptive uniprocessor FP-schedulability analysis. Our choice of uniprocessor FP schedulability-analysis as the problem upon which to illustrate our approach merits some explanation: despite the inherent intractability (NP-hardness) of the problem, superbly engineered implementations of RTA do exist that are very efficient in practice upon most problem instances and hence this is perhaps not the problem that first comes to mind as needing faster algorithms. We have nevertheless chosen FP-schedulability analysis as the problem upon which to illustrate our approach for primarily pedantic reasons – this is a problem that

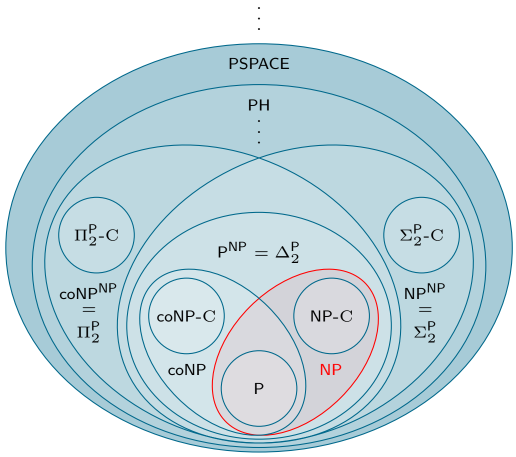


Fig. 5 Some common complexity classes, with NP marked in red. It is widely believed that no region in this diagram is empty – each is populated with problems.

is very well known by most of the real-time computing community and hence our target reader can focus on the conceptual framework without needing to constantly remind themselves of minutiae about the problem being solved. Additionally, focusing on FP-schedulability allows us to draw a contrast with EDF-schedulability, another commonly-studied schedulability-analysis problem that is often compared and contrasted with FP-schedulability analysis — see, e.g., [17], and which, by Proposition (1), cannot be solved using our DL-based framework (since it is coNP-hard).

4.1 Generating Synthetic Workloads

We build individual DNN models for FP-schedulability analysis of systems of 2 to 20 tasks. As *training data*, we generate one million synthetic task sets for each system size considered, as follows. We consider utilizations from 0.1 to 1.0 in steps of 0.1; for each utilization, we generate 10^5 sets of tasks. For each set, the utilization U_i of each task τ_i is assigned according to the UUniSort algorithm [18]. Task periods T_i are then assigned uniformly[4] in the range 1–1000, and workloads C_i are characterized according to $C_i = U_i \cdot T_i$. As we are considering schedulability of constrained-deadline tasks, we assign deadlines uniformly in the range $[C_i, T_i]$; tasks are then sorted in ascending order of deadline to reflect DM prioritization.

For each task system, we use RTA [6–8] to find the smallest value of R_i that satisfies Recurrence (1) for each task. This response time is then checked against the deadline; if $R_i \leq D_i$ for every task, the task set is deemed FP schedulable.

To support a proof of concept for EDF schedulability, we also perform processor demand analysis for sets of 4 tasks. Those for which Condition (3) is satisfied for all points in the testing set are deemed EDF schedulable.

[4]Although Emberson et al. [19] recommend a log-uniform distribution to reflect realistic task sets, we have opted for a uniform distribution to provide even coverage of the input space for training purposes.

To test how well our models generalize to similar synthetic tasksets, we generate as *test data* an additional million synthetic task sets using the same methodology (but a different random seed) for each task system size considered (2 to 20 tasks).

4.2 Evaluating Binary Classification

We begin with an evaluation of LEC-based schedulability analysis according to the framework in Figure 1.[5] To do so, we train a collection of simple multilayer perceptron (MLP) models to classify task systems as FP-schedulable or unschedulable. Each model accepts as its input the parameters of a constant number of tasks; we train models for systems of 2–20 tasks.

Training Methodology.

For each task set size considered, we construct an MLP using PyTorch [20] with the architectural template depicted in Figure 6. As inputs, the model takes the execution time C_i, period T_i, and deadline D_i of each task τ_i, with tasks sorted in ascending priority order. We observe that the demand bound function used in processor demand analysis (Equation (2)), as well as the recurrence expression used for response-time analysis (Equation (1)), *both have the task period in the denominator of a term.* We therefore also include $1/T_i$ as an input to the model. The network consists of 2 fully-connected hidden layers of 15 neurons that use rectified linear (ReLU) activation functions. The output layer has a single node using a sigmoid activation function. If the output value is >0.5, the set of tasks is classified as Schedulable; otherwise it is Unschedulable.

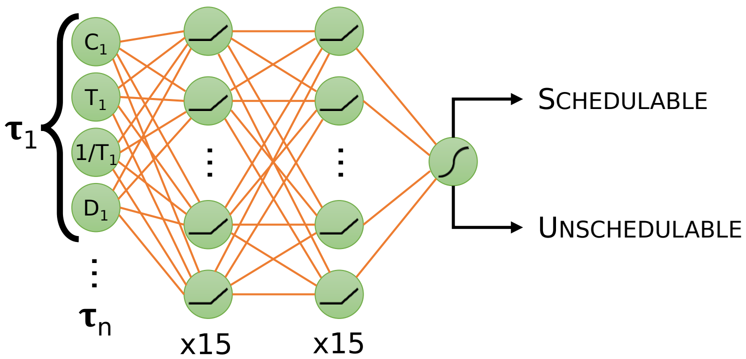


Fig. 6 MLP for binary classification of schedulability.

Each model is trained using the corresponding million sets of tasks generated as training data, using an 80%/20% training/validation split. Input data is shuffled, then fed in batches of size 1000. Training is performed over 100 epochs, stopping early if

[5] Recall that this framework does *not* guarantee an absence of false positives, and is therefore not recommended for use for safety-critical purposes. We evaluated this framework initially primarily to investigate whether it is even possible to use DL to recognize schedulable systems.

no improvement in the validation data is observed for 10 epochs. We use the Adam optimizer [21] with a learning rate of 0.001 and a weight decay of 0.0001.

Observations.

We have previously presented the results for 4-task systems (Figure 2 (a)); results for other system sizes are summarized in Figure 7 in the form of a plot of the overall accuracy as a function of system size. We observe that, while accuracy degrades slightly as the number of tasks increases, it remains above 92% even for 20-task systems. A 95% confidence interval obtained via nonparametric bootstrapping by resampling 1000 times remains within 0.06% of the accuracy, and is therefore too narrow to visualize in the plot.

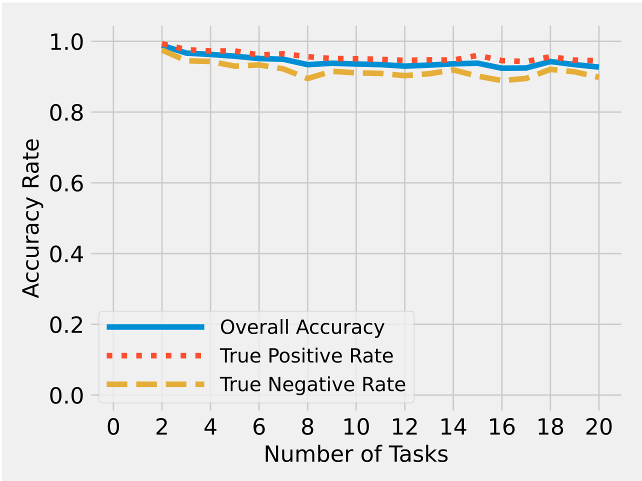


Fig. 7 Accuracy of binary classification for FP-schedulability.

4.3 Evaluating the Framework of Figure 3

We now describe our exploration of *verifiable* LEC-based schedulability analysis according to the framework in Figure 3.

Training Methodology.

For each taskset size considered, we construct an MLP with the model architecture shown in Figure 8. This model differs from the binary classifier (Figure 6) in some crucial ways. The model for predicting schedulability of n tasks (again sorted in priority order) outputs a set of predicted response times R'_i for $2 \le i \le n$ (R_1 is not predicted by the model, as it can be trivially computed as $R_1 = C_1$). The task system is then classified SCHEDULABLE if for each task τ_i, $R'_i \le D_i$; the result is then *verified* by checking whether every predicted value R'_i satisfies Recurrence (1). Four key insights guide the training methodology:

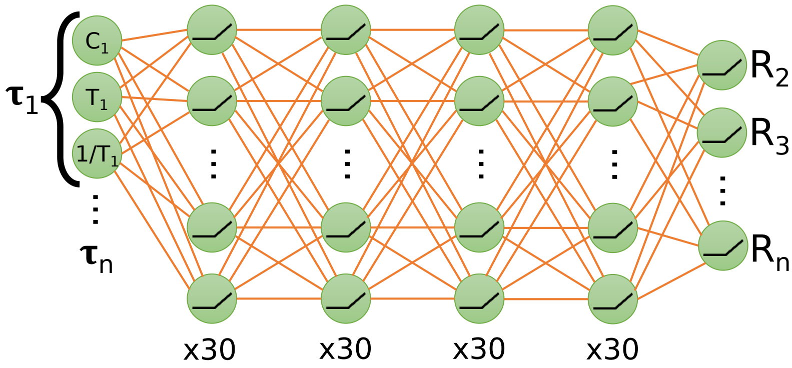


Fig. 8 MLP for computing R_i's (response times)

1. *This model extracts more information.* Because we are asking our model to estimate response times, rather than simply perform a binary classification, the network needs to be more complex. In this case, we use 4 fully-connected hidden layers of 30 neurons each (each hidden neuron, as well as the outputs, use a ReLU activation function).
2. *Response times are independent of deadlines.* The recurrence relation used to calculate the response time of a task does not depend on the deadline of that task. Therefore, deadlines D_i are *not* provided as inputs to the model.
3. *Predicted response times should not be too large.* This is obvious; a prediction that is too large might exceed the deadline for an otherwise schedulable task. We want the response times to be as small as possible, *but*
4. *Predicted response times should not be less than the true value.* A predicted response time that is too large might still satisfy the recurrence, and might still be less than the constrained deadline of the task. However, a prediction that is too small will never satisfy the recurrence.

With these last two insights in mind, we devise a training strategy using a custom loss function:

$$\mathcal{L} = \begin{cases} \left(\frac{R'_i - R_i}{R_i}\right)^2 & \text{if } R'_i \geq R_i \\ \left(w \cdot \frac{R'_i - R_i}{R_i}\right)^2 & \text{if } R'_i < R_i \end{cases} \tag{4}$$

This function computes the normalized mean squared error, but applies an additional weighting term w to negative error values (where a weight w=1 makes this equivalent to the normalized mean squared error). This has the desired effect of rewarding predictions that are close to the true value, while more heavily penalizing predictions that undershoot the true value. Training batch loss is computed as the mean over individual input losses.

Our training methodology is the same as that of the binary classifier described in Section 4.2. To decide what value to assign to our penalty term w, we first train 10 networks, each for sets of 3 tasks, using values of w distributed in log-uniform fashion from 1–1000.

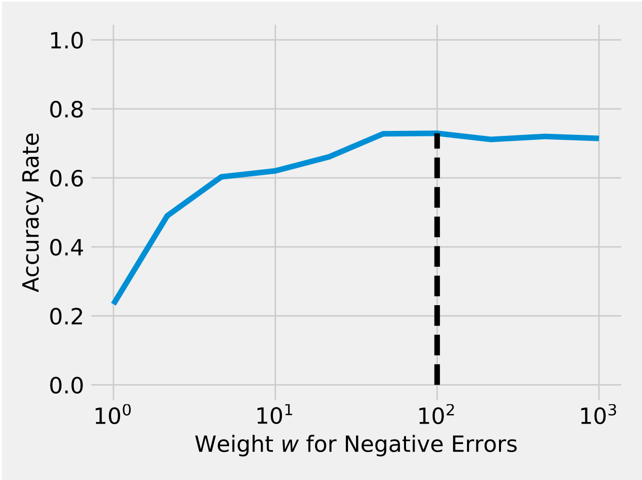


Fig. 9 Determining the appropriate value of w (see Section 4.3)

Once trained, we evaluate the accuracy of each model — a prediction is considered accurate if *(i)* each predicted value of R'_i satisfies Recurrence (1), and *(ii)* the model correctly classifies the task set as SCHEDULABLE or UNSCHEDULABLE. We plot the accuracy of each model over the 10^6 task sets that comprise our test data in Figure 9, observing that w=100 performs the best. We then scale this approach, training models for systems comprising 2–20 tasks with w fixed at 100.

Metrics for evaluation.

We evaluate our framework according to three different metrics:

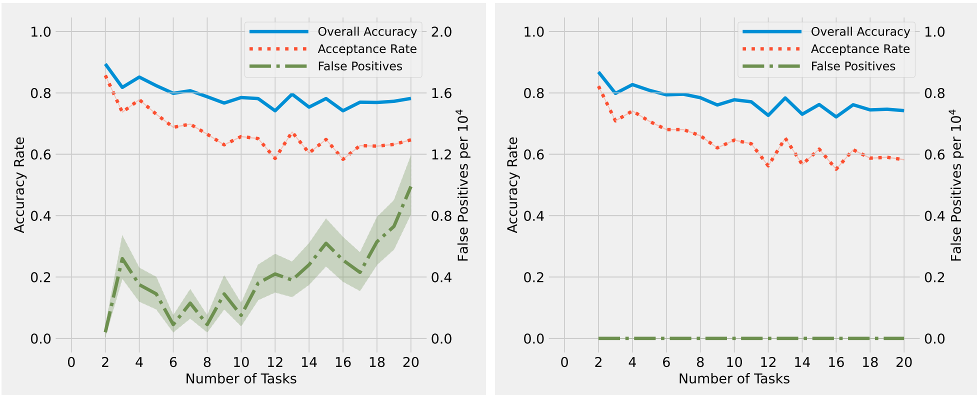
1. Predictive accuracy, i.e., the rate at which classification of a set of tasks as SCHEDULABLE or UNSCHEDULABLE is both *correct* and *verifiable* (i.e., the predicted values R'_i satisfy the recurrence); or
2. Acceptance rate, i.e., the percentage of SCHEDULABLE tasks that are classified as such. This is equivalent to the *sensitivity* of the test, or its true positive rate.
3. False positives, i.e., the number of task systems that are incorrectly classified as SCHEDULABLE.

While predictive accuracy is the metric by which many Machine Learning models are judged, real-time systems developers are likely more interested in finding schedulable systems as often as possible — correct identification of UNSCHEDULABLE task sets may not be as meaningful. However, as we have stressed, incorrectly identifying unschedulable task sets as SCHEDULABLE presents a safety hazard.

Observations.

We evaluate the models that were trained using a fixed penalty weight w=100. For each, we compare the above-listed evaluation metrics (predictive accuracy, acceptance rate, and number of false positives) when the predicted values R'_i are used to classify schedulability, and when these predictions are additionally verified. We have previously plotted unverified and verified schedulability as a function of system utilization for

4-task systems (Figure 4); these metrics for task systems of 2–20 tasks are summarized in Figure 10. Figures 10 (a) and (b) plot unverified and verified schedulability as a function of system size. As expected, predictive accuracy degrades with verification (though it remains above 72.1% for systems of up to 20 tasks); however false positives that may compromise safety are eliminated. Moreover, although accuracy degrades slightly as new tasks are added,[6] this approach nonetheless identifies and verifies well over half of the SCHEDULABLE task systems even for systems of as many as 20 tasks. As before, we obtain 95% confidence intervals via nonparametric bootstrapping by resampling 1000 times; these are shown as a shaded region around each series, although they are too narrow to easily visualize for overall accuracy and acceptance rate.



(a) Unverified schedulability.

(b) Verified schedulability.

Fig. 10 Evaluation metrics, plotted as a function of system size, of MLPs for computing response times. Note the different scale of the right-side y-axes for false positives.

4.4 Generalizing to Different Task Parameters

We have shown so far that our MLP (Figure 8) performs well at correctly and verifiably identifying schedulable task sets *when provided with test data generated using the same parameters as the training data*. However, growing evidence suggests that many Machine Learning models do not generalize well to real-world scenarios that differ from their training [22]. Generality is of particular concern for our framework, especially because sets of tasks in real-world applications do not often display the uniform properties displayed in our training data [19, 23].

To evaluate our model's ability to generalize when transferred to new scenarios, we generated alternative sets of tasks using different parameters. This time, to avoid having each task set's total utilization reflected in our training data, we used utilizations from 0.05 to 0.95 in steps of 0.1, generating 10^5 task sets for each value. For added realism,

[6]This makes sense, as the number of input features and values predicted increases, despite the number and size of the hidden layers remaining constant. We defer to future work the question of how much to grow the network, either by adding layers or adding nodes to existing layers, to maintain accuracy as tasks are added.

we selected periods from a log-uniform distribution per [19], instead of the uniform distribution in the training data.

We evaluated our LEC on sets of 4 tasks thus generated; results are illustrated in Figure 11. Overall accuracy after verification was 66.1%. This is 0.80× the verified accuracy when applied to test data generated with the same parameters as the training data, demonstrating that our model generalizes reasonably well.

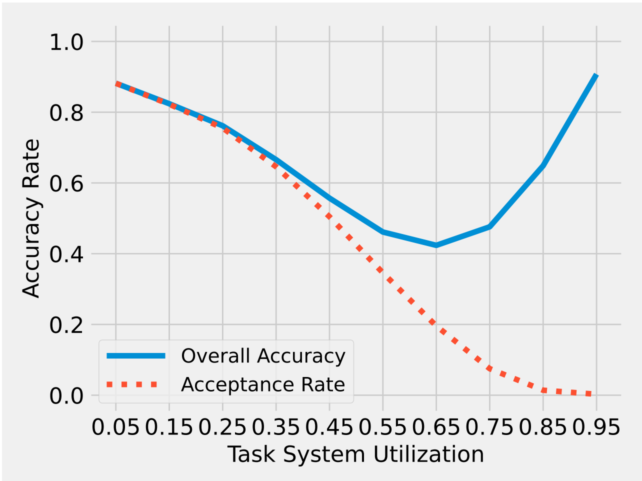


Fig. 11 FP schedulability with certificates, when generalizing to sets of 4 tasks generated per Section 4.4.

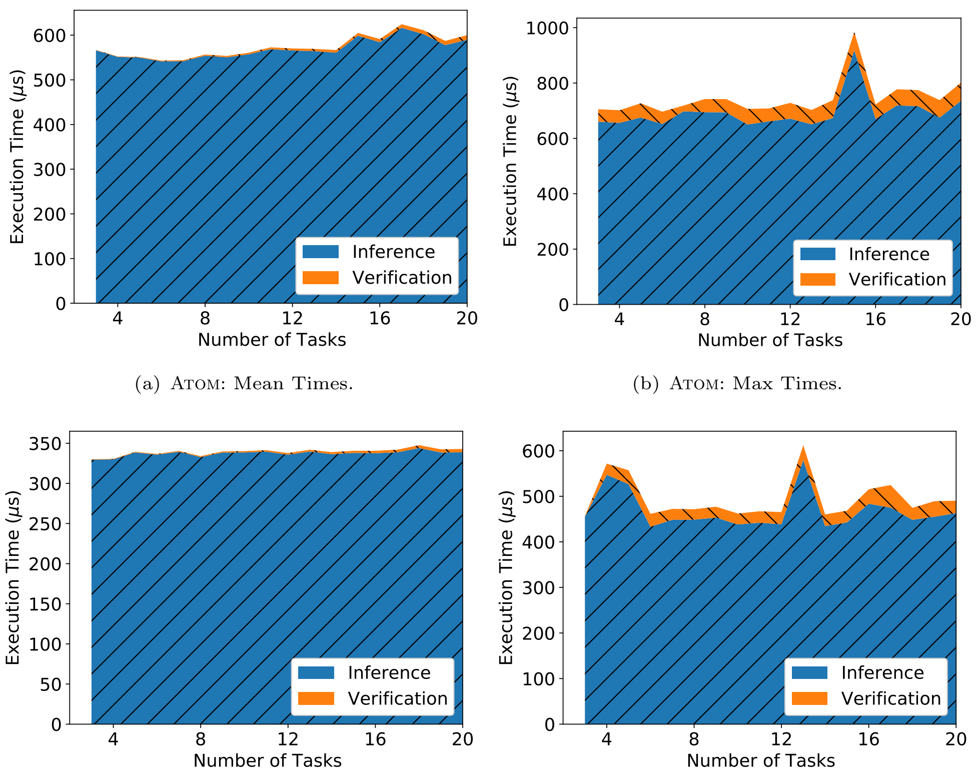
4.5 Execution Time Performance

Since many of our target applications are embedded systems, we have implemented our framework on select commonly-used embedded computing platforms and measured the execution duration to check whether these are acceptable for on-line use; we now report on these experiments.

Experimental Setup.

We generate task systems using the parameters described in Section 3.1, but this time we produce 1000 sets of tasks at each utilization for each number of tasks considered (3–20, for a total of 180 000 task sets).

We serialize our trained NN models to load them into a C++ program that is linked against PyTorch's compiled `libtorch` library module. Our program performs inference on a *single* set of tasks at a time, after which the predicted response times are verified and checked against task deadlines to determine schedulability. Prior to running inference over each group of 1000 task sets, we allow the corresponding model 20 "warm-up" iterations. To compare our LEC framework against an exact analysis, in the same program we also also implement the algorithm of Audsley et al. [24] to solve



(a) Atom: Mean Times. (b) Atom: Max Times.

(c) RPi4: Mean Times. (d) RPi4: Max Times.

Fig. 12 Execution time statistics for LEC framework.

the recurrence expression for response-time analysis in Equation (2). Our program is compiled with GCC using optimization level `-O3`.

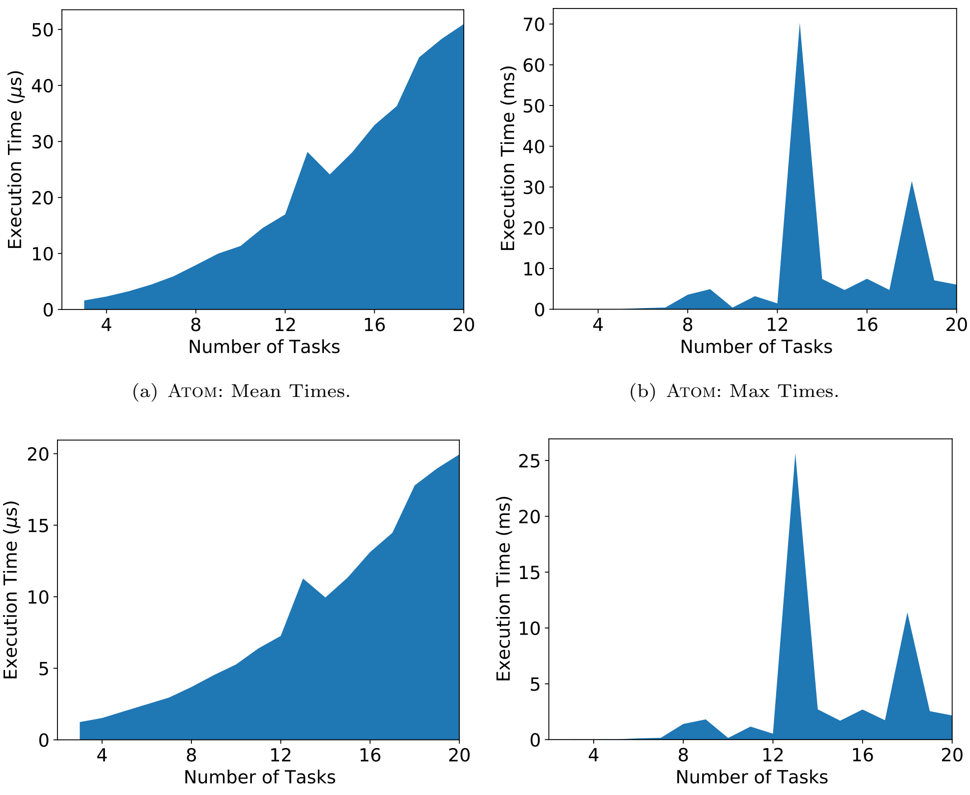
We measure execution times on two platforms (both with CPU throttling disabled):

1. Atom is a WinSystems EBC-C413 industrial single-board computer with an Intel Atom E3845 (x86_64) 4-core CPU and 8GB of RAM, running at 1.92GHz with Linux 5.15.0;
2. RPi4 is a Raspberry Pi 4 Model B, which has a Broadcom BCM2711 64-bit SoC with a Cortex-A72 (ARM v8) 4-core CPU and 4GB of RAM, running at 1.80GHz with Linux 5.15.16.

Results and Discussion.

We calculate the mean and maximum execution times across the 10 000 sets of tasks tested for each taskset size. Results for the LEC framework are plotted in Figure 12, and for exact response time analysis are plotted in Figure 13, from which several observations about our DL-based approach arise:

1. *It is efficient.* On the Atom, inference runs in under 620 μs and verification in under 11 μs, on average. The Rpi4 is even more efficient, running inference and verification respectively in under 345 μs and 4.2 μs on average.



(a) Atom: Mean Times.

(b) Atom: Max Times.

(c) RPi4: Mean Times.

(d) RPi4: Max Times.

Fig. 13 Execution time statistics for response time analysis.

2. *It is predictable.* The maximum observed execution times for the LEC framework remained under 986 μs on the Atom and under 629 μs on the RPi4. For each number of tasks considered, the maximum across the 10 000 tested task sets did not exceed 1.8× the mean on either platform. In contrast, exact response-time analysis was observed to take nearly 70 ms on the Atom and 25 ms on the RPi4 in the worst-case, which is over 1000× slower than the mean. This predictability makes a verifiable DL-based approach more suitable for online task admission, where overheads must remain bounded to maintain timeliness.
3. *It scales well with system size.* As the number of tasks increases, the execution time trends upwards only slightly. As Figure 8 illustrates, the number of inputs to and outputs from each model increase with the number of tasks, but these extra calculations are dominated by the number of neurons (120 total) in the fully-connected hidden layers.

While PyTorch provides an elegant framework for training models, and `libtorch` is a convenient way to wrap model inference into efficient C++ programs, it incurs significant overhead [25]. We therefore investigate whether we can achieve faster performance when deploying our MLP to an FPGA hardware accelerator.

4.6 FPGA Implementation

The rapid recent increase in size and complexity of NNs has spurred interest in performing DNN inference on specially-deployed FPGA kernels [26], often achieving highly-predictable execution times [14, 27]. This motivates us to evaluate the performance of our verifiable MLP for predicting response times when synthesized for execution on an FPGA.

Experimental Setup.

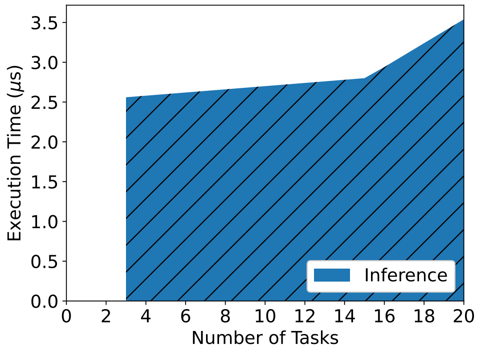
In this work, we select the AMD Xilinx XC7K325T FPGA which is deployed in real-world embedded applications, such as high-altitude balloon instruments for gamma ray detection [28, 29]. Its low power requirements make it suitable for the sorts of embedded environments where predictable schedulability analysis is likely to be most useful.

We implement our MLP illustrated in Figure 8 using high-level synthesis (HLS) in Vitis version 2024.1. We use hand-written and optimized matrix-multiply functions to implement the multiply-accumulate logic representing the linear layers, and a function to synthesize the comparators that represent each ReLU. Weights and biases are expressed as 32-bit floating-point values. The HLS code is written in C++ and uses preprocessor directives to provide a template for different model sizes based on the number of tasks. Dataflow pipelining enables multiple circuits to execute portions of the computation in parallel, reducing end-to-end latency.

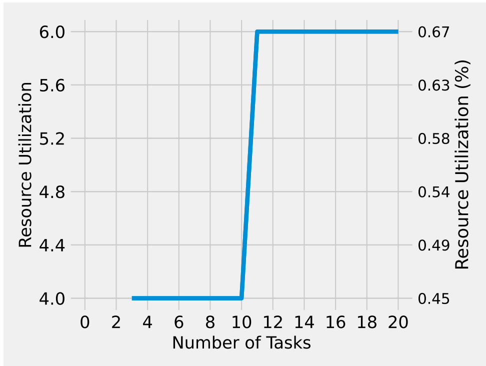
Results and Discussion

We synthesize the kernel for task sets of size 3–20 and use the Vitis HLS emulation tools to profile its latency and area usage. Results are plotted in Figure 14, from which several observations arise:

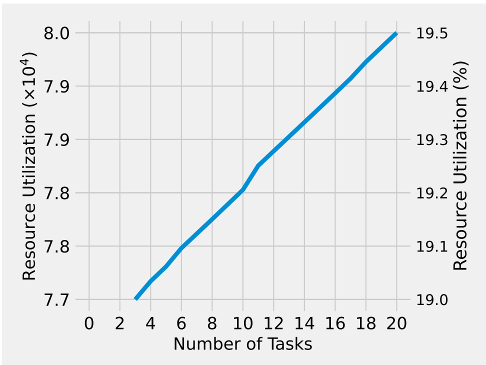
1. *It is efficient.* In Figure 14 (a), we plot the execution times associated with each number of tasks. The total inference time, including transferring data from the host to the FPGA (task parameters) and back to the host (response times), remains below 4 μs for up to 20 tasks, two orders of magnitude faster than for the Atom and RPi4. It is also more than 5$\times$ faster than even the average-case execution time of exact response time analysis on the RPi4, and nearly three orders of magnitude faster than the worst-case.
2. *Execution times scale linearly.* As Figure 8 illustrates, the size of the MLP's input and output layers scale linearly with the number of tasks; the execution times of associated matrix-vector multiplies therefore scale quadratically. However, as shown in Figure 14 (a), the parallelism achieved by our synthesized FPGA logic enables roughly linear scaling of execution times. The piecewise linear trend exhibited by the relationship between inference latency and problem size is explained by the pipelined nature of the FPGA logic. Inference can begin as data is still transferring onto the chip, meaning that growth in different parts of the circuit dominate the change in latency as the number of tasks increases.
3. *Area scales linearly.* An FPGA provides a set amount of utilizable resources, which defines the area over which logic can be synthesized. To implement the parallelism necessary to achieve execution times linear in the number of tasks, we have to also



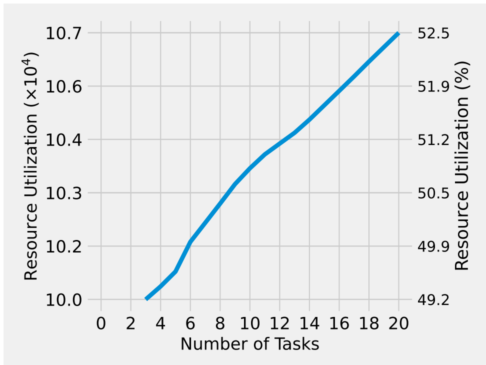
(a) Execution Times.



(b) BRAM Utilization.



(c) FF Utilization.



(d) LUT Utilization.

Fig. 14 FPGA speed and area statistics.

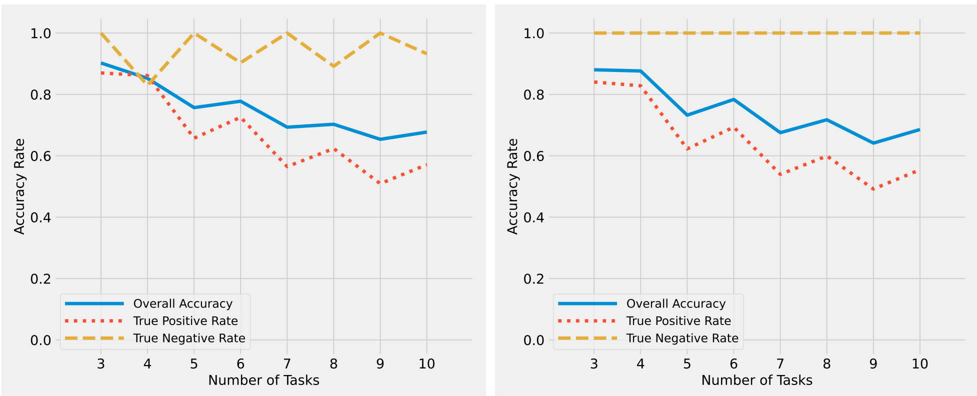
increase the area of the synthesized logic as the number of tasks grow. Figures 14 (b)–(d) show counts and overall percentage of block RAM (BRAM), flip flop (FF), and lookup table (LUT) resources used. Note that although BRAM cells utilized are expected to scale roughly linearly with the number of tasks, the synthesis tools group these into blocks which are often allocated in sets of 2; hence, the jump from 4–6 BRAM blocks. Not shown is the percentage of multiply-accumulate digital signal processor (DSP) slices used, which remained a constant 750 (89%).

These results indicate that the straightforward and predictable logic of our MLP model makes it amenable to deployment on an embedded FPGA. Utilization of BRAM and FF resources remains low, though LUT utilization exceeds 50% for sets of 20 tasks, and DSP utilization is a constant 89%. To allow simultaneous deployment of other logic — an embedded platform that includes an FPGA accelerator might need it for other applications as well — might therefore require reducing the LUT and DSP area required. Techniques exist to tune and optimize based on speed and area tradeoffs [29–31], but these are outside the scope of our proof-of-concept.

5 Context and Conclusions

Schedulability analysis is often computationally very expensive; in this manuscript, we have reported on our efforts at using deep learning to speed it up. We have found that it seems feasible to train even simple DL network architectures such as multilayer perceptrons (MLPs) to accurately classify system specifications as being either schedulable or unschedulable: despite not being experts in DL and without inordinate effort, we were able to train MLPs to do preemptive uniprocessor EDF and FP schedulability classification at accuracy rates above 92% for task systems with as many as 20 tasks.

Since misclassifying an unschedulable system as schedulable represents a safety hazard, we have proposed a framework (Figure 3) for DL-based schedulability analysis that detects all such classification errors. We have formally established that this framework is applicable for speeding up exactly those schedulability analysis problems that lie within the complexity class NP; we have demonstrated this applicability for the NP-complete FP-schedulability analysis problem and have concluded that the framework cannot be instantiated directly for EDF since EDF schedulability analysis is coNP-complete [9] and therefore likely $\notin \mathsf{NP}$. We have extensively evaluated our FP-schedulability analysis implementations on synthetically generated workloads; the results are very encouraging and point to the potential and promise of using DL for doing schedulability analysis.



(a) Unverified schedulability. (b) Verified schedulability.

Fig. 15 Preliminary results for multiprocessor partitioned DM schedulability on sets of 3–10 tasks. We train an MLP with three hidden layers of 50 nodes that takes the task parameters as inputs and partitions the tasks amongst two processors, using our MLPs for uniprocessor FP-schedulability analysis (that are described in Section 4.3) to verify the FP-schedulability of each partition. Verified accuracy remains above 64% while the acceptance rate (i.e., the proportion of schedulable task sets verifiably identified) remains above 49%.

Other schedulability-analysis problems.

As mentioned at the start of Section 4, our choice to use the relatively simple problem of uniprocessor FP schedulability-analysis as our running example is driven by our intent to make it easier for our target audience to follow along with minimal effort. The computational complexity of very many other schedulability-analysis problems are known;[7] those that are in NP can be implemented in our framework, whereas those that are hard for classes unlikely to be contained in NP cannot. For instance, we see from [32, Fig. 2] that *partitioned FP schedulability-analysis of constrained-deadline sporadic task systems* is in NP and hence implementable within our framework, whereas partitioned FP schedulability-analysis of constrained-deadline *periodic* task systems is unlikely to fit our framework since it lies at or above the second level of the Polynomial Hierarchy [33] (and hence unlikely to be in NP under the widely-held assumption that the Polynomial Hierarchy has > 2 levels). It is similarly known that many multiprocessor DAG-scheduling problems are in NP, and hence implementable within our framework (the associated certificates of schedulability could be processor assignments and/ or preemption instants).

Incorporating improved DL techniques.

In closing, we reiterate a point we had made in Section 1 and reëmphasize the *proof-of-principle* nature of our study: we seek to establish a framework for applying DL to solve schedulability-analysis problems. Accordingly, we have devoted much of our efforts at formulating, and rigorously characterizing the applicability of, this framework. Although prior work has applied DL to such problems —a survey of such work is available in [34]— ours is the first, to our knowledge, that uses complexity theory to formalize the set of problems that can be solved by DL while guaranteeing efficient elimination of unsafe false positives. We believe that developing the 'best' DL systems for any particular schedulability analysis problem for which our framework is applicable requires collaboration with experts in DL and does not fall within the scope of the ideas that we are presenting in this paper, and leave as future work a detailed incorporation of the latest findings in DL into our framework. As an illustration of such possible incorporation in the future, we point out that we have also instantiated our framework for partitioned FP scheduling of constrained-deadline sporadic task systems upon multiprocessor platforms (as mentioned above, shown [32, Fig. 2] to be NP-complete) – some preliminary results are plotted in Figure 15. A very recent work [35] reported success in training Graph Attention Networks to partition *implicit-deadline* sporadic task systems (task systems in which $D_i = T_i$ for all tasks τ_i) for FP-scheduling upon multiprocessors. We plan to explore the feasibility of extending [35] to the partitioning of constrained-deadline task systems; if successful we could, in principle, easily replace our multilayer perceptron (MLP) with such a Graph Attention Network and thereby seamlessly incorporate this advance in Deep Learning into our framework, and thereby obtain a partitioned FP-schedulability analysis algorithm that

[7] E.g., [32, p. 366] provides, in tabular form, a comprehensive summary of the computational complexity of schedulability-analysis for partitioned EDF and FP scheduling of various variants of periodic and sporadic task systems upon multiprocessor platforms of different kinds.

offers superior performance to what is depicted in Figure 15, whilst continuing to guarantee the absence of false positives.

References

[1] Kawaguchi, K.: Deep learning without poor local minima. In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 29. Curran Associates, Inc., ??? (2016). https://proceedings.neurips.cc/paper_files/paper/2016/file/f2fc990265c712c49d51a18a32b39f0c-Paper.pdf

[2] Baruah, S., Mok, A., Rosier, L.: Preemptively scheduling hard-real-time sporadic tasks on one processor. In: Proceedings of the 11th Real-Time Systems Symposium, pp. 182–190. IEEE Computer Society Press, Orlando, Florida (1990)

[3] Leung, J.Y.-T., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation **2**, 237–250 (1982)

[4] Eisenbrand, F., Rothvoss, T.: Static-priority real-time scheduling: Response time computation is NP-hard. In: Proceedings of the Real-Time Systems Symposium. IEEE Computer Society Press, Barcelona (2008)

[5] Ekberg, P., Yi, W.: Fixed-priority schedulability of sporadic tasks on uniprocessors is NP-hard. In: 2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017, pp. 139–146. IEEE Computer Society, ??? (2017). https://doi.org/10.1109/RTSS.2017.00020 . https://doi.org/10.1109/RTSS.2017.00020

[6] Joseph, M., Pandya, P.: Finding response times in a real-time system. The Computer Journal **29**(5), 390–395 (1986)

[7] Lehoczky, J., Sha, L., Ding, Y.: The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: Proceedings of the Real-Time Systems Symposium - 1989, pp. 166–171. IEEE Computer Society Press, Santa Monica, California, USA (1989)

[8] Wellings, A., Richardson, M., Burns, A., Audsley, N., Tindell, K.: Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal **8**, 284–292 (1993)

[9] Eisenbrand, F., Rothvoß, T.: EDF-schedulability of synchronous periodic task systems is coNP-hard. In: Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (2010)

[10] Baruah, S., Howell, R., Rosier, L.: Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. Real-Time Systems: The International Journal of Time-Critical Computing **2**, 301–324 (1990)

[11] Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, ??? (1994)

[12] Arora, S., Barak, B.: Computational Complexity - A Modern Approach. Cambridge University Press, ??? (2009). http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264

[13] Kang, W., Chung, J.: DeepRT: predictable deep learning inference for cyber-physical systems. Real-Time Systems **55**(1), 106–135 (2019) https://doi.org/10.1007/s11241-018-9314-y

[14] Huang, S., Pearson, C., Nagi, R., Xiong, J., Chen, D., Hwu, W.-m.: Accelerating sparse deep neural networks on FPGAs. In: 2019 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–7 (2019). https://doi.org/10.1109/HPEC.2019.8916419

[15] Sun, Y., Zheng, L., Wang, Q., Ye, X., Huang, Y., Yao, P., Liao, X., Jin, H.: Accelerating sparse deep neural network inference using GPU tensor cores. In: 2022 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–7 (2022). https://doi.org/10.1109/HPEC55821.2022.9926300

[16] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms., 4th edn. MIT Press, ??? (2022)

[17] Buttazzo, G.C.: Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, 2nd edn. Springer, ??? (2005)

[18] Bini, E., Buttazzo, G.C.: Measuring the performance of schedulability tests. Real-Time Syst. **30**(1–2), 129–154 (2005) https://doi.org/10.1007/s11241-005-0507-9

[19] Emberson, P., Stafford, R., Davis, R.I.: Techniques for the synthesis of multiprocessor tasksets. In: WATERS Workshop at the Euromicro Conference on Real-Time Systems, pp. 6–11 (2010). 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems ; Conference date: 06-07-2010

[20] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol. 32, pp. 8024–8035 (2019)

[21] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[22] Risi, S., Togelius, J.: Increasing generality in machine learning through procedural content generation. Nature Machine Intelligence **2**(8), 428–436 (2020)

[23] Kramer, S., Ziegenbein, D., Hamann, A.: Real world automotive benchmarks for

free. In: 6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), vol. 130 (2015)

[24] Audsley, N.C., Burns, A., Richardson, M.F., Wellings, A.J.: Hard real-time scheduling: The deadline-monotonic approach. IFAC Proceedings Volumes **24**(2), 127–132 (1991)

[25] Georgiou, S., Kechagia, M., Sharma, T., Sarro, F., Zou, Y.: Green AI: Do deep learning frameworks have different costs? In: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), pp. 1082–1094 (2022). https://doi.org/10.1145/3510003.3510221

[26] Guo, K., Zeng, S., Yu, J., Wang, Y., Yang, H.: [dl] a survey of fpga-based neural network inference accelerators. ACM Trans. Reconfigurable Technol. Syst. **12**(1) (2019) https://doi.org/10.1145/3289185

[27] Khoda, E.E., Rankin, D., Lima, R.T., Harris, P., Hauck, S., Hsu, S.-C., Kagan, M., Loncar, V., Paikara, C., Rao, R., Summers, S., Vernieri, C., Wang, A.: Ultra-low latency recurrent neural network inference on FPGAs for physics applications with hls4ml. Machine Learning: Science and Technology **4**(2), 025004 (2023) https://doi.org/10.1088/2632-2153/acc0d7

[28] Sudvarg, M., *et al.*: Front-End Computational Modeling and Design for the Antarctic Demonstrator for the Advanced Particle-astrophysics Telescope. In: Proc. of 38th International Cosmic Ray Conference, vol. 444, pp. 764–17649. Sissa Medialab, ??? (2023). https://doi.org/10.22323/1.444.0764

[29] Sudvarg, M., Zhao, C., Htet, Y., Konst, M., Lang, T., Song, N., Chamberlain, R.D., Buhler, J., Buckley, J.H.: Hls taking flight: Toward using high-level synthesis techniques in a space-borne instrument. In: Proc. of 21st International Conference on Computing Frontiers. ACM, ??? (2024). https://doi.org/10.1145/3649153.3649209

[30] Makrani, H.M., Farahmand, F., Sayadi, H., Bondi, S., Dinakarrao, S.P., Homayoun, H., Rafatirad, S.: Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design. In: 2019 29th International Conference on Field Programmable Logic and Applications (FPL), pp. 397–403. IEEE Computer Society, Los Alamitos, CA, USA (2019). https://doi.org/10.1109/FPL.2019.00069 . https://doi.ieeecomputersociety.org/10.1109/FPL.2019.00069

[31] Zhao, C., Dong, Z., Chen, Y., Zhang, X., Chamberlain, R.D.: Gnnhls: Evaluating graph neural network inference via high-level synthesis. In: 2023 IEEE 41st International Conference on Computer Design (ICCD), pp. 574–577 (2023). IEEE

[32] Ekberg, P., Baruah, S.: Partitioned scheduling of recurrent real-time tasks. In: 2021 IEEE Real-Time Systems Symposium (RTSS), pp. 356–367 (2021). https:

//doi.org/10.1109/RTSS52674.2021.00040

[33] Stockmeyer, L.: The polynomial-time hierarchy. Theoretical Computer Science **3**, 1–22 (1976)

[34] Bian, J., Arafat, A.A., Xiong, H., Li, J., Li, L., Chen, H., Wang, J., Dou, D., Guo, Z.: Machine learning in real-time internet of things (iot) systems: A survey. IEEE Internet of Things Journal **9**(11), 8364–8386 (2022) https://doi.org/10.1109/JIOT.2022.3161050

[35] Lee, S., Lee, J.: A graph attention network approach to partitioned scheduling in real-time systems. IEEE Embedded Systems Letters (2024) https://doi.org/10.1109/LES.2024.3376801