Evaluation of AI-based Meta-scheduling Approaches for Adaptive Time-triggered System

Daniel Onwuchekwa, Moumita Dasandhi, Samer Alshaer, Roman Obermaisser
Department of Embedded Systems
University of Siegen
Siegen, Germany

***Abstract*—Adaptation in time-triggered systems is motivated by the need to attain higher energy efficiency, carry out fault recovery functions, and adapt to changing environmental conditions. Adaptation can also be attained by modifying the allocation of services to resources, substituting failed resources or transitioning into a degraded service mode. Time-triggered systems can deploy meta-scheduling to enhance adaptation. Nevertheless, state-of-art meta-schedulers suffer state explosion, and pose storage constraints for cyber-physical systems. This work proposes an architecture that uses machine learning to infer new schedules at run-time, thus eliminating the need to store the large schedule sets generated by a meta-scheduler. It investigates the performance of three machine learning models, Random Forest Classifier(RFC), Artificial Neural Network (ANN), and Encoder/Decoder Neural Network (E/D NN). It is observed that the performance of models are dependent on the complexity of the scheduling problem. The E/D NN model performed better than the ANN and RFC for all job sizes. The resulting accuracies for 15 jobs are 98.39%, 95.33%, 86.86% for E/D NN, RFC, and ANN, respectively. The resulting accuracies for 60 jobs are 87.67%, 60.94%, 81.05% for E/D NN, RFC, and ANN, respectively. The implication of this work is that the proposed approach provides a means to compensate for the trade-off between storage capacity of a CPS and the number of schedules for each adaptation scenario. The proposed machine learning based architecture is able to capture more scenarios without the need to store schedules.**

***Keywords*—Adaptation, Artificial Intelligence, Artificial Neural Networks, Meta scheduling, Random Forest, Time-Triggered Systems.**

I. Introduction

Adaptation in a Cyber-Physical System (CPS) is motivated by the need to manage energy, failure and changing environmental conditions adequately. Energy management is required for most CPS, especially battery-operated devices used in safety-critical applications. Common energy management CPS techniques include Dynamic Voltage and Frequency Scaling (DVFS) and clock gating. These techniques have limitations when used for safety-critical systems. Their application can lead to unpredictable timing and fault propagation due to shared resources, especially when exploited for dynamic slack [1]. When certifying safety-critical systems, two types of energy sensitivity distinctions are made, including energy efficiency without detrimental effect on safety and energy efficiency as a part of a safety argument.

This work has received funding from the ECSEL Joint Undertaking(JU) under grant agreement No 877056. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Italy, Austria, Germany, France, Finland, Switzerland.

Adapting to failures in CPS can be achieved by fault recovery, thereby enabling a system to switch to configurations that exclude failed resources. Fault recovery also reduces the redundancy degree in safety-critical CPS, lowering the cost of fault tolerance. Approaches to fault recovery can include adapting by modifying the allocation of services to resources, substituting services that have failed, or switching the system to a degraded mode.

Adaptation can be triggered by changing environmental conditions such as temperature, pressure, or user-defined scenarios to maintain the desired system state.

Safety-critical systems can exploit the properties of time-triggered architectures to guarantee stringent safety and timing requirements, such as meeting tight deadlines. The properties of time-triggered architectures include the avoidance of resource contention without dynamic resource arbitration, implicit synchronisation, timing guarantees, implicit flow control, and fault containment.

An Adaptive Time-Triggered Multicore Architecture (ATMA) was introduced in prior work to enhance adaptation in CPS systems [1]. ATMA described and modelled a time-triggered infrastructure which can support adaptation by eliminating failed resources using online system reconfiguration. The ATMA also supports adapting to dynamic slack or changing environmental conditions. The adaptation is based on a meta-scheduler infrastructure, where multiple schedules for different scenarios are generated offline. In response to the occurrence of the scenario which is recognised during runtime, the system switches schedules to one that eliminates failed resources, exploits dynamic slack, or adapts to a change in environmental conditions.

However, the proposed architecture in [1] suffers a challenge with the meta-scheduler, which exposes a state explosion during the generation of schedules. Although two algorithms were proposed to handle the state explosion, the implementation of these algorithms in [2] shows a limitation in the number of adaptable scenarios.The first algorithm is termed "re-convergence horizon" where schedules are only computed within a predefined bound. The second is the "path re-convergence" where scenarios that result in schedules that are already in the graph of computed schedules are linked instead of adding a new schedule node to the graph. including schedules for scenarios that are not covered at design time. In order to cover more scenarios, more schedules have to be generated; therefore, the horizon is increased. The large number of schedules generated

for adaptation also poses a challenge for the storage in CPS.

This paper proposes a machine learning-based approach to handle the large number of schedules generated by a meta-scheduler. Based on the observed context event, the machine learning model serves for inference on an updated schedule. The machine learning approach is considered since it eliminates the need to deploy many schedules on the CPS. In addition, the machine learning model will be able to predict schedules that are required for adaptation, including schedules for scenarios that are not covered at design time. Also, the recent technology trend where AI-accelerators are integrated into embedded hardware, such as the Xilinx versal Adaptive Compute Acceleration Platform (ACAP) devices [3], provide the enabling hardware platforms that supports the approach. The proposed work presents an approach to exploring adaptation in a statically configured time-triggered application using AI-based scheduling schemes. It offers an adaptation solution for safety-critical applications such as aerospace, automotive, and health systems that utilize a time-triggered paradigm.

The contributions of this work are as follows:

- A machine learning-based model for determination of schedule using inference is developed time-triggered schedule prediction is developed.
- We investigate the performance of three machine learning models; Random Forest Classifier (RFC), Artificial Neural Network (ANN), and Encoder/Decoder Neural Network (E/D NN). The performance evaluation is based on comparing the complexity and accuracy of the models.

The remainder of this paper is structured as follows. Section II discusses the related works. A background on meta-scheduling is provided in section III. The proposed machine-learning based scheduler is presented in section IV. An evaluation technique for the proposed approach is setup in section V. We discuss the results in section VI, and conclude in section VII.

II. Related Work

A. Related works on metascheduling

In prior work we proposed a meta-scheduling algorithm where a scheduler is invoked repeatedly to generate schedules for different scenarios. The proposed algorithm is not limited to a specific scheduler. For example, it can utilise schedulers that provide solutions based on Genetic Algorithm (GA), Mixed Integer Linear Programming (MILP), and Ant Colony Optimisation (ACO).

Babak in [4] proposed an optimisation method which establishes the minimum energy consumption for slack events by mixed-integer quadratic programming (MIQP) equations. The work used the meta-scheduling algorithm proposed in [1]. Issues regarding handling the large schedule size generated due to state explosion were not covered.

Muoka et al. in [2] attempted to solve the state explosion problem of the meta scheduler algorithm by implementing two algorithms. These algorithms include path reconvergence and a reconvergence horizon. The path reconvergence horizon dealt with removing Deja Vu schedules from the list of computed schedules. The reconvergence algorithm established bounds for the meta-scheduler to add schedules to the list of schedules required for adaptation. In contrast, we propose a solution that uses machine learning to eliminate the need for tightly bounding the limits of adaptation when implementing the reconvergence horizon.

B. AI-based Scheduling

There are several applications of machine learning to solve different scheduling problems, such as job-shop and workflow scheduling. A hybrid deep neural network scheduler (HDNNS) was introduced by Zang et al. [5] to solve the job-shop scheduling problem. A convolution two-dimensional transformation (CTDT) was used to convert JSSP's irregular scheduling information into regular data. The HDNNS architecture is designed by combining a deep convolution layer, a fully connected layer and a flattening layer. In contrast, we explored the applicability of neural networks where a scheduler is trained with data from a meta-scheduler, thus more focused training and potentially higher accuracy.

Deep reinforcement learning is applied to the scheduling of time-triggered Ethernet in [6]. A scheduling agent is first trained offline and later deployed for the online scheduling of time-triggered flows. Similarly, Stacked Autoencoders-based Deep Reinforcement Learning was implemented by Jiang et al. in [7] for online resource scheduling in large-scale mobile-edge computing networks. The authors proposed a scheduling framework with components to minimise the sum of weighted task latency for Internet-of-Things users. According to the authors, the deep reinforcement learning approach is proposed for optimising offloading decisions, power transmission, and resource allocation in the large-scale Mobile Edge Computing (MEC) system. However, we applied the Encoder/Decoder Neural network for adaptive scheduling using data generated from a MSG.

Melnik et al. in [8] proposed a scheduling approach using Artificial Neural Networks (ANN) and Reinforcement Learning (RL) for a workflow system. The authors focused on analysing the scheduling problem using RL, which helps form the input states and considers the structure of the workflow. The Neural Networks Scheduling (NNS) algorithm is based on RL to learn how to provide qualitative schedules in the workflow makespan. In contrast, we have used the ANN algorithm for the adaptive scheduling, where we predicted the priorities before subsequently computing new schedules. The priority prediction approach ensures that only correct schedules are computed while guaranteeing safety.

Li et al. in [9] proposed a production rescheduling framework using machine learning techniques and industry optimisation algorithms. The authors first focused on modelling the Flexible Job-Shop Scheduling Problem (FJSP), after which they solved FJSP using a hybrid meta-heuristic approach. Finally, they proposed a rescheduling framework which uses machine learning and optimisation algorithms such as tabu search to make rescheduling decisions. The machine learning algorithms used include Multi-layer Perceptron (MLP), Support Vector Machine (SVM) and Random Forest Classifier (RFC). In contrast, we

focused on the adaptive scheduling with an architecture that evaluates the applicability of ANN, RFC, and E/D NN.

A machine learning-based approach is used by Shiue et al. in [10] for real-time scheduling systems (RTS). The authors used an ensemble based on the wrapper feature selection approach. The base classifiers of RTS were created using GA along with a Neural Network, Decision tree and SVM Classifiers. Each base classifier was trained using bagging data in the proposed ensemble RTS. Finally, the base classifiers were ensembled by a majority voting strategy. According to the authors, the ensemble technique could enhance the generalisation ability of the RTS with respect to the classic ML-based classifier approaches. The experimental results showed that the proposed ensemble RTS classifier could generalise better than the individual base classifiers. Similarly, we evaluated the knowledge of ensemble learning (i.e. Random Forest) in our implementation. In contrast, our work applies RFC for adaptive scheduling.

Scheduling in the cloud computing domain is of interest to many researchers and often applies machine learning techniques. Gondhi et al. [11] reviewed papers regarding scheduling in cloud computing. The authors mentioned various advanced intelligent scheduling algorithms (with a swarm-based approach) that are used in the cloud computing domain. The target of scheduling in cloud-based computing is to provide load balancing, enable a scalable framework, and reallocate resources.

III. Background on Meta-scheduling

The meta-scheduler is a tool for computing time-triggered schedules considering the system's conceptual, spatial and temporal dimensions [12]. At any point in time, for each context, the respective schedules will be responsible for determining the sequence of the resource usages within the system's time frame. The context of the meta-scheduler is presented in [1] where it was utilised in conjunction with an *Adaptive Time-triggered Multi-core Architecture (ATMA)* and an agreement protocol to enable scenario-based adaptation.

The meta-scheduler computes a **Multi-schedule Graph** (MSG) using an **application model**, **platform model** and a **context model** as its input. The MSG is generated at design time for time-triggered systems and it is a Directed Acyclic Graph (DAG), $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where each vertex, $s_i \in \mathcal{V}$ represents a schedule *i*, and the edge $c_{ik} \in \mathcal{E}$ represents context events that enable the schedule transitions $s_i \rightarrow s_k$. During runtime, the time-triggered system utilises one schedule (node) at a given time instance from the MSG. The system traverses from one node to another one when a context event occurs.

The *application model* provides the computational task details, including their Worst-Case Execution Time (WCET), deadline and resources (i.e. memory and I/O). The *platform model* is used to describe the available hardware resources such as cores, memories, input-outputs, routers and the physical links where applicable. Finally, the *context model* is used to describe all the considered context events in the meta-scheduler. These context events can include faults, dynamic slack, and environmental changes intended for adaptation.

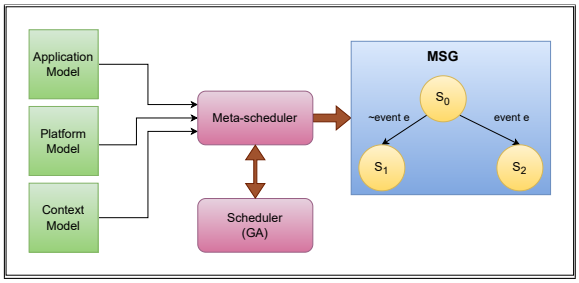

Figure 1: Generation of MSG

The generation of the MSG is illustrated using Figure 1. The meta-scheduler generates the MSG at design time. The meta-scheduler takes in the application, platform, and context models as inputs. Afterwards, it invokes a scheduler multiple times to generate new schedules that conform to a given context event. The scheduler component can generate solutions using different algorithms such as Genetic Algorithm (GA), Mixed Integer Linear Programming (MILP), and Ant Colony Optimisation (ACO). However, this work utilised the GA as the scheduler component, and the meta-scheduler computes offline the MSG for all potential context events.

Algorithm 1 describes the meta-scheduler component in Figure 1. It consists of a procedure which takes as input the application model (AM), platform model (PM), context model (CM), a set of fixed decision variables (FIX), and the previous schedule (*prev*). Before calling the meta-scheduler procedure, the variable FIX and the MSG indicated as SG in the algorithm are initialised.

The procedure begins by invoking the scheduler component to compute a base schedule (initial schedule S_0). New schedules that are subsequently computed are compared with the schedules in the MSG for identity. The existence of the identity is referred to herein as Deja Vu, indicating that the schedule already exists. A new schedule will not be computed if there is a Deja Vu node. Instead, the current schedule with its corresponding edge context will be connected to the Deja Vu node. If there is no Deja vu node, the algorithm computes a new schedule for every event in the context model by including the event's impact in the computed schedule. The meta scheduler considers mutually exclusive events. For instance, a dynamic slack value for given task makes it impossible for another slack event for the same task. These context events in the CM are applied to the AM or PM, and a set of decision variables is fixed. The fixing of the decision variables arises from the need to pin down events that have already occurred while computing a new schedule. For instance, as we step through the schedule-timeline, jobs or tasks that are in the past are fixed. The meta-scheduler procedure is then recursively invoked to cover all context events.

A significant challenge for meta-schedulers is state explosion mentioned in [1], which handled the issue with a reconvergence of paths algorithm and a reconvergence horizon. The reconvergence of path concept is integrated into the Algorithm 1 using

the already explained Deja Vu concept. The reconvergence horizon was integrated and studied in [2].

Algorithm 1 Algorithm of Meta-Scheduler

Require: *initial application model: (AM)*, *initial platform model: (PM)*, *initial context model: (CM)*, *initial multi-scheduled graph:* $(SG = \{\})$ and *initial fixed decision variables:* $(FIX = \{\})$

procedure META-SCHEDULER($AM, PM, CM, FIX, prev$)
  invoke scheduler(AM,PM,FIX) to obtain schedule S
  $S = \{< d, t(d) >\}$ ▷ decision variable d with action time $t(d)$
  $n =< S, CM >$ ▷ new node for schedule graph
  if $(n \in SG)$ **then** ▷ Deja Vu
    connect previous node $prev$ to existing node n in SG
  else
    add n to SG
    if $(prev \neq NULL)$ **then**
      connect node $prev$ to new node n in SG
    end if
    while $(CM \neq \{\})$ **do**
      e = earliest context event from CM with event time $t(e)$
      EX = context events that are mutually exclusive with e
      $CM' = CM \setminus (EX \cup \{e\})$
      AM' = result of applying e to AM
      PM' = result of applying e to PM
      $FIX = \{< d, t(d) >\in S \mid t(d) \leq t(e) \vee t(d) \geq t(e) + HORIZON\}$
      recursively invoke META-SCHEDULER (AM', PM', CM', FIX, n)
    end while
  end if
end procedure
meta-scheduling: invoke META-SCHEDULER (AM, PM, CM, FIX, NULL)

IV. AI-BASED SCHEDULING ARCHITECTURE

The architecture of our proposed AI-based adaptive scheduler is depicted in Figure 2. The proposed model is subdivided into two parts, namely, *design time* and *run time*. The *design time* refers to the offline activities prior to the deployment of schedules in the system, while the *run time* depicts the activities that occur during the operation of the time-triggered system.

A. Design Time

The design time begins with the execution of Algorithm 1, to generate the MSG. As shown in Figure 2, the meta-scheduler takes in application, platform, and context model and repeatedly invokes a scheduler for each context to create the MSG.

The proposed architecture presents a technique to handle the short comings from the methods proposed in [1] and [2] to handle state explosion of the MSG. The short comings are that the use of the reconvergence horizon algorithm only limits adaptation within certain reconfigurable bounds in the time-triggered scheduling timeline. This previous solution enforces a trade-off between the size of the MSG and the number of context events that can be adapted to. This trade-off is due to the limitations of the adaptable context events within a pre-defined horizon.

Name	Description
job_id	ID of each Job
runs_on	Machine ID of each job execution
start_time	Starting time of a job execution
wcet	Worst Case Execution Time of each job
msg_id	Message ID used for communication between different jobs
inj_time	Message injection time
msg_size	Size of the message
ridx	Router index (i.e start index and its further hops)
context	32-bits context event information
centrality	Number of in-degree and out-degree connections to the current node

TABLE I: Input Features

Our approach does not directly deploy the MSG during run-time and the complete MSG does not have to precomputed at design time. Therefore, the issue of state explosion does not appear in the meta-scheduling. Nevertheless, more data generated in the MSG is beneficial to the machine learning approach proposed for run time deployment.

1) Dataset: After the MSG is generated using Algorithm 1, a dataset is created from it. Each node (i.e schedule) contained in the MSG is stored in a JSON file format. A python script is used to extract features required for the machine learning algorithms.These input features are described in Table I.

As shown in Table I, the edge features are designed as a 32-bit context event vector. The vector is mapped to contain information regarding *context type*, *context value*, *job ID*, *context time* and *device ID*.

The output features are the job priorities extracted from each MSG node. The choice of predicting the priorities is motivated by safety considerations. Machine learning models are generalised approximation models and cannot be relied on to always produce a correct schedule without a verification mechanism. We adopt the approach of predicting only job priorities, which are then used to perform the online computation of schedules. The advantage of this technique over other fast state-of-art dynamic schedulers such as list scheduling is that we can benefit at run time, the optimal schedules computed by high-performance optimisation algorithms such as GA at design time. The machine learning model for scheduling is attained by learning the schedules generated by the GA and then deploying it for online priority prediction.

2) Machine Learning models: The proposed architecture is not limited to the use of a fixed machine learning model. Nevertheless, this work explores three machine learning models, RFC, ANN, and E/D NN. Since the scope of this work is not intended to deal with specific machine learning algorithms, we only briefly describe them below with corresponding references that point to more details regarding each algorithm.

- Random Forest Classifier is an ensemble learning method and a Supervised Machine Learning Algorithm that is used widely in classification and regression problems. The algorithm is based on the creation of multiple decision trees from bootstrap samples of the entire dataset and the splitting in each decision tree can be performed using minimum *Gini Index* or minimum *Information Gain* [13]. The Gini Index/Information Gain measures the number of times a random variable is incorrectly computed. The final result of the Random Forest is determined based on the majority vote predictions from each decision tree (in the case of classification) or by computing the average of the predictions from each decision tree (in the case of regression).
- Artificial Neural Network trains a model to perform prediction by reducing the error in the network. The network digitally mimics the human brain and consists of interconnected nodes (neurons) following the forward and backward propagation. The nodes in a neural network are arranged sequentially in layers (input layer, hidden layer(s) and output layer). These neurons in subsequent layers are densely connected but not within the same layer (i.e. output from one layer behaves as an input to the next layer). During *forward propagation*, the weighted sum of inputs are sent through the network. During *back-propagation*, the error computed by the output layer will be traversed back to the network using *Gradient Optimiser Procedure* to minimise the error [14].
- Encoder/Decoder Neural Network trains a model to reconstruct data from a condensed encoded representation and ignores the noise in the data. The architecture of E/D NN is almost similar to the ANN architecture with a slight variation. The algorithm has two main components, such as *the encoder* which encodes the input vector into a reduced vector space and *the decoder* which is responsible for reconstructing the information from the encoded vector space. This algorithm is useful for optimally compressing the data in a simple linear function and complex non-linear function [15]. Therefore, it is utilized in various applications such as dimensionality reduction, and anomaly detection.

B. Run time

During run time, a time-triggered system switches schedules in response to the occurrence of a context event. The new schedule is the one which factors in the adaptation, such that the system responds to the changes presented as context events.

When a context event occurs, the trained model is used to predict the new schedule. The model makes prediction by taking as input the current active schedule and the context event which occurred. The current schedule is passed through a feature extraction process before being loaded into the model. The trained model predicts the priority of each jobs which is then used to reconstruct the schedule. The schedule is also verified before a final output is presented for utilisation.

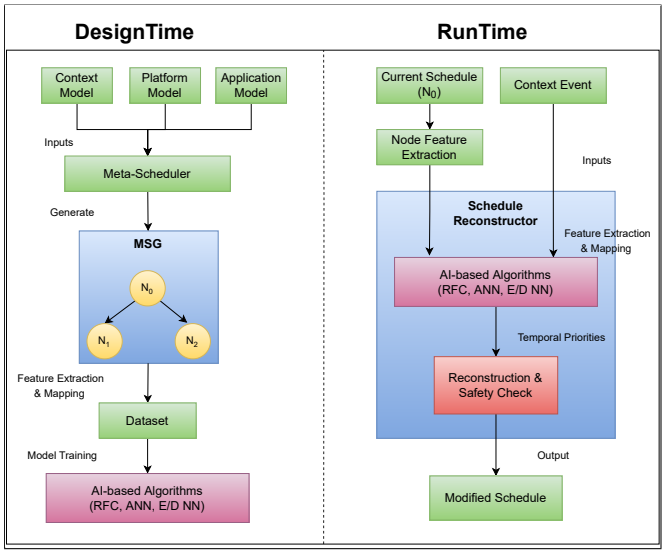

Figure 2: Proposed Model[16]

V. Evaluation Setup

We evaluate our contributions using three machine learning algorithms; RFC, ANN, and E/D NN. The experimental setup is carried out in three steps.

- Firstly, data generation and analysis are carried out.
- Secondly, the machine learning model is set up, and hyper-parameter tuning is performed.
- Thirdly, the prediction of the target values and evaluation of the machine learning models are performed.

A. Data generation and analysis

Data generation is carried out using Algorithm 1. We generated the MSG with an example platform model shown in Figure 3, and it consists of 3 example routers (*R1 - R3*) and 6 processing elements (*PE1 - PE6*). An application model is created for five different job sizes; 15-jobs, 20-jobs, 30-jobs, 40-jobs, and 60-jobs. The meta-scheduler algorithm is deployed for the different workloads using one platform model. The different workloads result in diverse input feature sizes. For example, a job size of 60 requires 1233 input feature variables, from which 1200 are schedule features, 32 are the edge features, and 1 is the node centrality; where each job has an identification number, worst case execution time, parents, children, deadlines, message size and route included as features. However, a job size of 20 requires 513 input feature variables; 480 schedule features, 32 edge features, and one node centrality. These features are shown and explained in Table I. The size of the dataset is 16,384 samples for each of the job-size.

For each of the generated schedules in the MSG, we extracted the temporal priority of the jobs to get the output features.

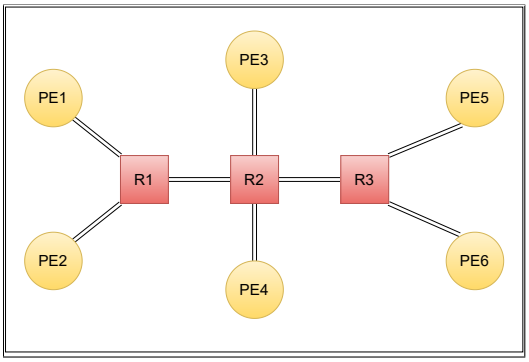

Figure 3: Experiment Platform Model

Afterwards, the dataset is analysed for missing values. There was no further action taken since there were no missing values found. A univariate analysis is performed on the data set to understand the data distribution and range variation and check for outliers. Consequently, the results for some of the features with zero variance were dropped. These features include the *job_id*, *msg_id*, *msg_size* and *router indexes*. Finally, one-hot encoding is performed for the target variables, and the dataset is split into 80% training sets and 20% test sets.

B. Modelling and Hyper-parameter Tuning

The dataset independently uses the machine learning algorithms (Artificial Neural Network, Random Forest Classifier and Encoder/Decoder Neural Network) to create robust models and train the model to predict the output features. Apart from the input and output feature size, which corresponds to different job sizes, the structure/architecture of the machine learning models are fixed for different job sizes. We evaluate the prediction performance of the model across different job sizes.

The Scikit-Learn (Sklearn) python library [17] is used for the Random Forest Classifier model, with a maximum depth of 10, a minimum sample leaf of 5, a max feature of 150, and 50 trees. These parameters were attained using the "GridSearchCV" class.

The TensorFlow library is used to implement the ANN and the E/D NN. We used the *Sequential* class to create a fully connected (dense) ANN. Three hidden layers are configured with 35, 18, and 10 neurons for the first, second, and third layers, respectively. These neuron sizes and the number of layers are selected based on the results from multiple trials. The input and output sizes are based on the number of job-size selected, as explained above. A learning rate of 0.001 is configured, and the batch size is set to 128. For the E/D NN, an encoding dimension of 64 is selected as optimal after multiple trials.

C. Prediction and Performance

Each model predicts the output feature variables for the test dataset (i.e. the temporal priorities of each job). The predicted values are compared with the actual values of the test dataset to measure each model's performance. The implementation is evaluated using two performance metrics; overall accuracy of the model prediction and the model complexity.

VI. Results and Discussions

The hyper-parameters explained in section V are tuned such that neither models have overfitting or underfitting. The loss function is computed to find the best fit model for the ANN and E/D NN models. It is the distance between the actual and predicted output of the algorithms. After 200 epochs, a minimum loss value of 0.05 is attained for the ANN model. Furthermore, after 100 epochs, a minimum loss value of 0.01 is attained for the E/D NN model.

The result of the experiment for the different models (RFC, ANN, E/D NN) is presented in Table II. The columns of the table show the number of jobs, the input and output feature size, complexity, and corresponding accuracy of the models. The corresponding values for the input and output features of these job sizes are common across all the models.

The model complexity is used to identify how challenging it is to learn from the data. The model complexity is evaluated using the number of parameters used to train the model. The more the number of parameters in an algorithm, the more chances of model overfitting. Therefore, if two models fit the data equally well, a model with lower complexity will be given higher precedence. Figure 4 compares the complexity values of RFC, ANN, and the E/D NN against different job sizes.

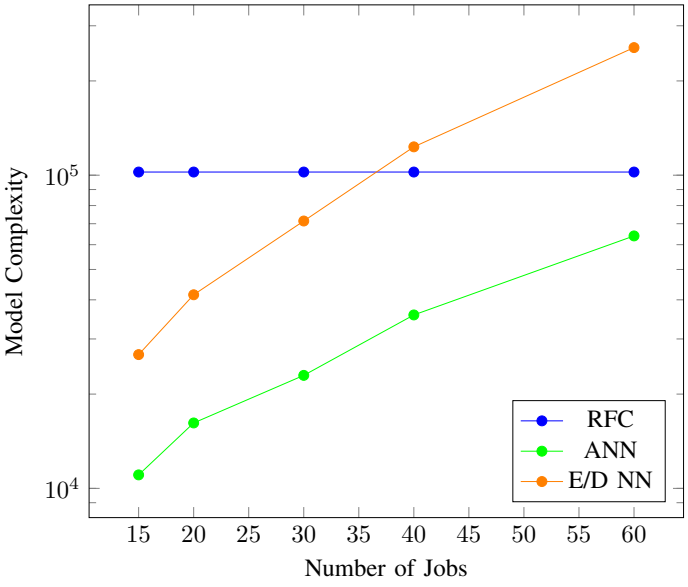

Figure 4: Complexity comparison of RFC, ANN, and E/D NN

We define the complexity of the RFC as the number of splits performed to build the entire random forest.

$$complexity_{RFC} = ((2^{d+1}) - 1) * m \quad (1)$$

Where m is the number of trees in the random forest, and d is the maximum depth of each tree

Model	No. of Jobs	Input Feature Size	Output Feature Size	Complexity	Accuracy
Random Forest Classifier	15	339	180	102,350	95.33%
	20	513	300		80.39%
	30	609	690		77.25%
	40	873	1320		82.34%
	60	1233	3120		60.94%
Artificial Neural Network	15	339	180	11,043	86.86%
	20	513	300	16,178	85.94%
	30	609	690	22,953	83.60%
	40	873	1320	35,798	81.23%
	60	1233	3120	63,998	81.05%
Encoder /Decoder Neural Network	15	339	180	26,740	98.39%
	20	513	300	41,516	96.36%
	30	609	690	71,410	94.84%
	40	873	1320	123,176	92.47%
	60	1233	3120	255,536	87.67%

TABLE II: Detailed performance evaluation of the models

The complexity of the RFC model remains constant at a value of 102350, which is because the input/ output vector size does not affect the complexity computation. In RFC, it is only the number and depth of the trees that have an impact on the complexity computation. However, the model complexity increases for the ANN and E/D NN model as the number of jobs increases, even though there is no change in the model structure. The increment in complexity is because the input and output of the neural networks are adapted to accommodate different job sizes.

The accuracy context is accomplished by comparing the actual and predicted test data in a loop. The accuracy metric is considered because the number of samples is evenly distributed among classes. The accuracy in this work is defined using Equation VI.

$$accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} * 100 \qquad (2)$$

Where,
TP = Number of 1's that are correctly predicted,
TN = Number of 0's that are correctly predicted,
FP = Number of 1's that are misclassified,
FN = Number of 0's that are misclassified

Figure 6 shows a plot of the model accuracy against the number of jobs. It is observed that the E/D NN outperforms both RFC and ANN. The highest accuracy was attained across all job sizes. Nevertheless, there is a decrease in accuracy as the number of jobs increases. The decrease in accuracy reflects the increasing model complexity and the fixed model structure.

Similarly, the accuracy of the ANN model also reduces with the increasing number of jobs. The decrease in accuracy is not as much as both E/D NN and RFC. The decreasing accuracy for ANN ranges from 86.86% to 81.05%. In contrast, the decreasing accuracy for E/D NN ranges from 98.39% to 87.67% and RFC from 95.33% to 60.94%.

The RFC performed more than the ANN for small job sizes but is the most unstable model with varying job sizes. For this reason, it can be drawn that the RFC does not provide a desirable generalized model for the meta-scheduling application when the model structure is fixed.

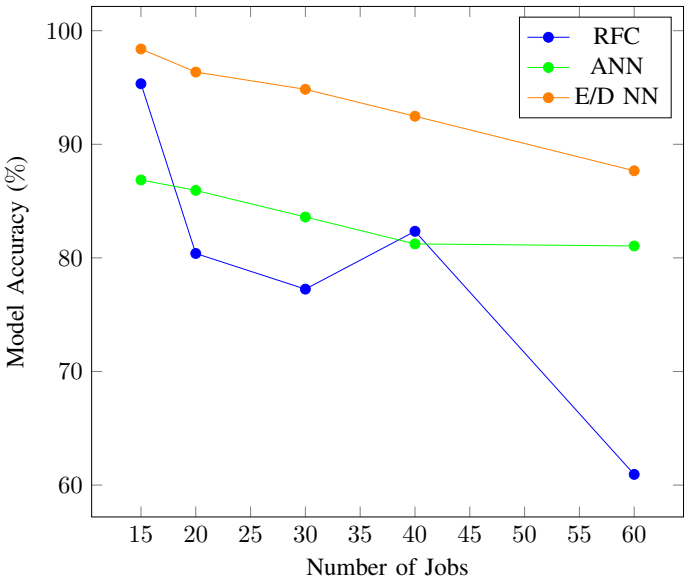

Figure 5: Accuracy comparison of RFC, ANN, and E/D NN

The outputs of the machine learning algorithms are used as inputs and inserted into the reconstruction model as previously mentioned. The reconstruction model outputs full schedules based on the predicted priorities. Figure 6 below shows an example run of 16484 schedules, each schedule has it's slack context event run in random time throughout the schedule, causing a parameter refered to as reconstruction percentage; to further illustrate, assume context event occurrence in the middle of a schedule run. The result would be 50 % of the schedule is reconstructable, the other 50 % is fixed since it already occured and the past cannot be change. Each algorithm is tested over the samples to generate output predicted priorities and then run through the reconstructor to generate complete schedules.

VII. Conclusion

This work proposes a machine learning-based architecture solution to solve the issue of operating meta-scheduler in run-time and solve the problem of hugging resources and state space explosion. Machine learning is exploited to predict the temporal job priorities, after which schedules are reconstructed. This approach provides a way for meta-schedulers to predict

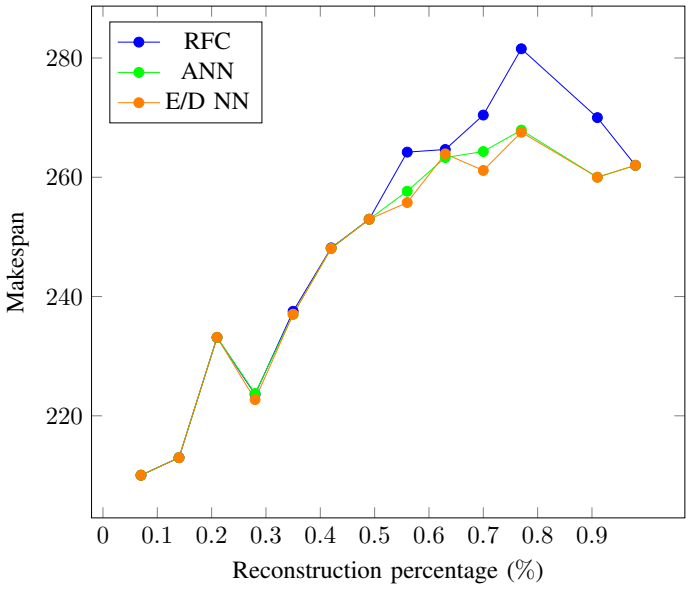

Figure 6: Makespan comparison between reconstructed schedules generated from each inference

schedules that are required for adaptation, but not covered in the schedules generated using a pre-defined horizon. Furthermore, we investigate the performance of Random Forest Classifier (RFC), Artificial Neural Network, and Encoder/Decoder Neural Networks (E/D NN) in the meta-scheduling framework. The investigations are based on different job sizes to study how our proposed models perform with the different workloads. The complexity of RFC for a given model remains fixed even as the scheduling problem increases. ANN performs better than E/D NN in terms of complexity. However, poor accuracy is observed in RFC when predicting the temporal priorities for different workloads in the multi-schedule graph. Better accuracy results are observed for E/D NN than a typical ANN.

Acknowledgement

One of the authors; S.Alshaer would like to thank the German Academic Exchange Service (DAAD) for funding his Phd studies allowing his contribution to this paper. In addition, the authors would like to acknowledge the contributions of Uni siegen's Omni Cluster for the allowing the data generation process.

References

[1] R. Obermaisser, H. Ahmadian, A. Maleki, Y. Bebawy, A. Lenz, and B. Sorkhpour, "Adaptive time-triggered multi-core architecture," *Designs*, vol. 3, no. 1, p. 7, 2019.

[2] P. Muoka, D. Onwuchekwa, and R. Obermaisser, "Adaptive scheduling for time-triggered network-on-chip-based multi-core architecture using genetic algorithm," *Electronics*, vol. 11, no. 1, p. 49, 2021.

[3] K. Vissers, "Versal: The xilinx adaptive compute acceleration platform (acap)," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 83–83.

[4] B. Sorkhpour, "Scenario-based meta-scheduling for energy-efficient, robust and adaptive time-triggered multi-core architectures," Ph.D. dissertation, Universität Siegen, 2019. [Online]. Available: https://dspace.ub.uni-siegen.de/handle/ubsi/1471

[5] Z. Zang, W. Wang, Y. Song, L. Lu, W. Li, Y. Wang, and Y. Zhao, "Hybrid deep neural network scheduler for job-shop problem based on convolution two-dimensional transformation," *Computational intelligence and neuroscience*, vol. 2019, 2019.

[6] C. Zhong, H. Jia, H. Wan, and X. Zhao, "Drls: A deep reinforcement learning based scheduler for time-triggered ethernet," in *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021, pp. 1–11.

[7] F. Jiang, K. Wang, L. Dong, C. Pan, and K. Yang, "Stacked autoencoder-based deep reinforcement learning for online resource scheduling in large-scale mec networks," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9278–9290, 2020.

[8] M. Melnik and D. Nasonov, "Workflow scheduling using neural networks and reinforcement learning," *Procedia Computer Science*, vol. 156, pp. 29–36, 2019.

[9] Y. Li, S. Carabelli, E. Fadda, D. Manerba, R. Tadei, and O. Terzo, "Machine learning and optimization for production rescheduling in industry 4.0," *The International Journal of Advanced Manufacturing Technology*, vol. 110, no. 9, pp. 2445–2463, 2020.

[10] Y.-R. Shiue, R.-S. Guh, and K.-C. Lee, "Development of machine learning-based real time scheduling systems: using ensemble based on wrapper feature selection approach," *International journal of production research*, vol. 50, no. 20, pp. 5887–5905, 2012.

[11] N. K. Gondhi and A. Gupta, "Survey on machine learning based scheduling in cloud computing," in *Proceedings of the 2017 International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence*, 2017, pp. 57–61.

[12] B. Sorkhpour, A. Murshed, and R. Obermaisser, "Meta-scheduling techniques for energy-efficient robust and adaptive time-triggered systems," in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. IEEE, 2017, pp. 0143–0150.

[13] G. Louppe, "Understanding random forests: From theory to practice," *arXiv preprint arXiv:1407.7502*, 2014.

[14] D. Kriesel, "A brief introduction on neural networks," 2007.

[15] J. Zhai, S. Zhang, J. Chen, and Q. He, "Autoencoder and its various variants," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2018, pp. 415–419.

[16] S. Alshaer, C. Lua, P. Muoka, D. Onwuchekwa, and R. Obermaisser, "Graph neural network based metascheduling in adaptive time-triggered architectures," *International conference on Emerging Technologies and Factory Automation*, vol. 22, no. 1, 2022.

[17] "Scikit-Library random forest," https://scikit-learn.org/stable/modules/grid_search.html, accessed: 2022-09-30.