REALITY: RL-PowEred AnomaLy Detection with Imprecise Computing in MulTi-core sYstems

Chandrajit Pal, Sangeet Saha, Xiaojun Zhai and Klaus McDonald-Maier

***Abstract*—The Approximate Computing (AC) model is used by contemporary real-time systems to balance accuracy and system resource constraints better while completing a set of time-sensitive tasks within a deadline. These AC assignments include both necessary and optional components. To achieve satisfactory results, the optional components can be carried out entirely or partially depending on the available resources. Considering a real-time multiprocessor system, for a set of interconnected AC tasks with deadlines and an energy budget, creates a workable schedule to execute the necessary and optional components of tasks to meet an anticipated quality of service (QoS) level. However, during execution, if a malware attack or bug affects one or more processor cores, the system may stop working altogether after deployment, causing unanticipated power outages or processing delays that prevent the system from finishing its task by the deadline. The goal of our proposed methodology REALITY is to investigate the possibility of intelligently rescheduling a set of dependant AC tasks upon detection of any anomalous situation, running in multi-core systems under system-wide constraints to maintain acceptable levels of QoS. During execution, REALITY monitors the operational behaviour of the processing cores by collecting Hardware Performance Counters (HPCs) and identifying any irregularities through an ML-powered anomaly detection mechanism. Upon detection, it applies remedial actions by intelligently rescheduling the tasks leveraging the Proximal Policy Optimisation (PPO)–based Reinforcement Learning (RL) algorithm while maintaining 70% QoS.**

***Keywords*—Approximate computation (AC), energy-aware scheduling, quality of service (QoS), Precedence-constrained Task Graphs (PTGs), Normalised QoS (NQ), Hardware Performance Counters (HPCs), Graph Attention Networks (GAT), Proximal Policy Optimisation (PPO).**

I. Introduction

In real-time systems, it is better to have partially accurate results within a deadline rather than fully accurate results obtained beyond it. For example, lower-quality frames acquired before a deadline are better than missing any frames altogether in video streaming applications [1]. The applications that are currently operating on these platforms are typically represented as Precedence-constrained Task Graphs (PTGs) [2], in which each task is represented as a node, and tasks' relationships are indicated by edges. Even though real-time embedded systems typically have limited resources, operate on battery power, and have restricted energy budgets, they still need to deliver superior performance and excellent service. [3]. If the system is unable to produce an accurate result on time, the same can be accomplished by producing an acceptable approximated result within a deadline leveraging Approximate Computing (AC) task model [4]. This method divides a task into *optional* and *necessary* components [1]. All necessary component execution must be finished by the deadline to obtain the minimum acceptable quality of service (QoS) level. The optional component can then be executed entirely or in part, based on available resources, to improve the accuracy of the initial output produced by the deadline. The QoS increases as more execution cycles are used on the optional component.

The challenge of energy-efficient real-time AC task scheduling is to improve performance while respecting underlying system constraints, which has been the subject of recent research. Authors in [5] proposed a scheduling mechanism that, in situations where the energy budget is constrained, accepts imprecise results and executes additional computations when there is more energy available. Their research is restricted to autonomous tasks, though. The authors of [6] and [7] examined dependent AC tasks and recommended using dynamic voltage frequency scheduling (DVFS) to create energy-efficient scheduling strategies. However, when DVFS reduces the supply voltage and frequency to conserve power, the system's soft error and transient fault rates increase sharply [8], affecting the system's reliability. A workaround was proposed by Wang et al. [9] to find a *optimal energy frequency* in multi-core processor platforms. Run-time errors frequently cause faults in modern multiprocessor systems that hinder their normal operation. Hardware trojan attacks and other intrusions tend to slow down processor speed, which causes real-time applications to miss deadlines [10]. In scenarios where the energy budget is fixed, malware can also lead to intentional power dissipation, which depletes the system's energy budget and makes it unusable [11]. To ensure that system execution proceeds as intended, it is imperative to incorporate error-resilient security modules that recognise these anomalies and take appropriate action in real time to mitigate them. Hence, *Given an AC task graph and a multi-core processor system, it is a challenging problem to successfully execute all associated tasks within the specified deadline while minimising runtime errors and meeting precedence- and energy-related constraints.*

In this paper, we propose REALITY, consisting of a reliability-aware intelligent scheduler that ensures the system maintains desired QoS in the presence of anomalies through intelligent detection and mitigation procedures through optimal scheduling. The choice of processing cores to schedule the task execution efficiently while still maintaining the QoS in the presence of faults varies widely on diverse target platforms and intent. Intelligent schedulers based on supervised learning

C.Pal, S. Saha, X. Zhai and Klaus McDonald-Maier are with the School of Computer Science and Electronics Engineering, University of Essex, CO4 3SQ Colchester, UK.

necessitate a reasonable quantity of training data, whereas another branch of ML namely Reinforcement Learning (RL) distributes tasks to cores under diverse platforms and constraints and is independent of data and any apriori pre-trained model, which makes it popular in scenarios with limited data including dynamic scenarios which necessitates adaptation on the fly. Specifically, RL creates an optimal policy through continual interaction with the environment.

Few RL-based studies on task allocation for optimal power and thermal management have shown promising results. They [12], [13], [14], [15] leverage the well-known Q and deep Q-learning algorithms for policy optimisation which is a better fit for their application-specific parameters with discrete state space. We have chosen a state-of-the-art PPO-based RL algorithm [16] that matches our use case dealing with QoS, time, and power measures as continuous parameters for better performance with continuous state space. We considered multi-core processor systems and dependent real-time AC task-graph shown in Fig. 2. REALITY schedules the tasks by allocating them to appropriate processing cores at specific time instants. When extra power dissipation during task execution is detected during runtime, or if a task cannot be completed in the allotted time, causing a decrease in the desired QoS, the anomaly detection module is activated. This enables it to detect anomalies in the system (using a thorough analysis based on the correlation among multiple observed HPCs), if any, and marks the associated processing core as faulty. Subsequently, as illustrated in Fig. 1, a new schedule is generated based on the available energy budget, remaining execution time, remaining idle processing cores and task list.

The following highlights our primary contributions:

1) REALITY schedules AC tasks by implementing the necessary components and appropriate optional components according to the available energy budget, deadline, task and available processing cores.
2) Our intelligent graph attention network-based runtime security mechanism correlates multiple observed Hardware Performance Counters (HPCs) of a processing core to identify abnormalities in processor performance.
3) For an AC task graph in a multiprocessor system, our proposed RL framework utilises PPO to generate an optimised task schedule which ensures successful execution of all associated tasks within a specified time constraint, minimising any runtime error arising out of an anomaly while fulfilling energy and precedence-related constraint achieving a high NQ of 70%.

II. System Modelling and Assumptions

Task modelling: Considering an embedded system platform with n homogeneous processing cores $C = \{c_1, c_2, ..., c_n\}$. Taking a look at a real-time application (A) represented as a directed and acyclic PTG (Fig. 2), which is expressed as $G_r = (T_k, E_d)$. The list of directed edges, $E_d = \{\langle T_k i, T_k j\rangle \mid 1 \leq i, j \leq |T_k|; i \neq j\}$, represents the list of tasks, and T_k denotes a task set ($T_k = \{T_k i \mid 1 \leq i \leq n\}$), illustrating the precedence relationships among various task pairs. Task $T_k j$ cannot begin

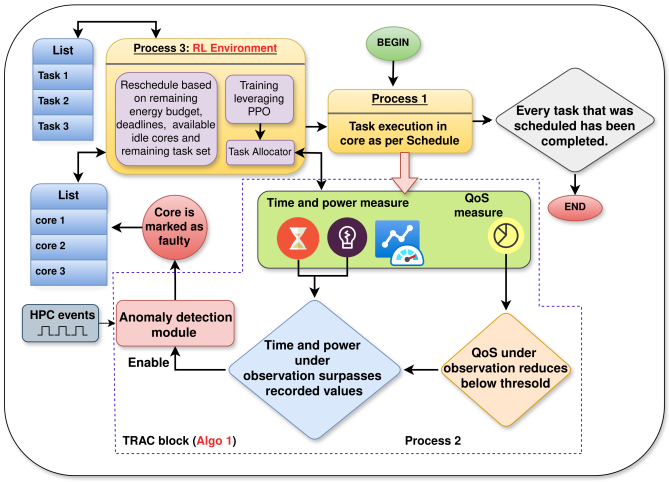


Fig. 1: REALITY Workflow

execution until $T_k i$ has finished, as indicated by an edge $\langle T_k i, T_k j\rangle$, which indicates precedence.

Since this is a real-time application, for the entire application (A) to meet its deadline, all associated task nodes must be executed within the interval. Every task $T_k i$ ($1 \leq i \leq n$) was broken down into two parts: an optional component O_i and a necessary component N_i. To get a satisfactory outcome, the required steps must be carried out. The optional component O_i is only partially or fully executed after the required component N_i has been finished.

The execution length for each task $T_k i$ can be expressed as:

$$L_i = N_i + \lambda \times O_i \tag{1}$$

where λ, which denotes the percentage of the implemented optional component, is a number between 0 and 1. As a result, $\lambda = 1$ denotes the completion of all O_i units to yield the most precise outcome. The quality of service (QoS) or accuracy of results at the system level is determined by the total number of O_i component cycles completed for each task. Additionally, n_i distinct task versions $T_k i$ are assumed; that is, $T_k i = \{T_k i^1, T_k i^2, \ldots, T_k i^{v_i}\}$.

On the other hand, additional completion of the optional part (which makes the task longer) will increase task accuracy, which will raise the quality of service (QoS) of the system. The scheduling objective is to run an enhanced version of each task to maximise QoS.

III. Threat Modelling

Usually, threats originate from manufactured or natural sources. Intruders are the ones who carry out intentional manual threats; adversaries in the foundry could be an ideal instance as they could introduce malware in firmware or introduce hardware-based faults during the fabrication of the multi-processor system, which could lead to a number of issues like power draining, execution termination, and delays. The list of dormant threats is as follows:

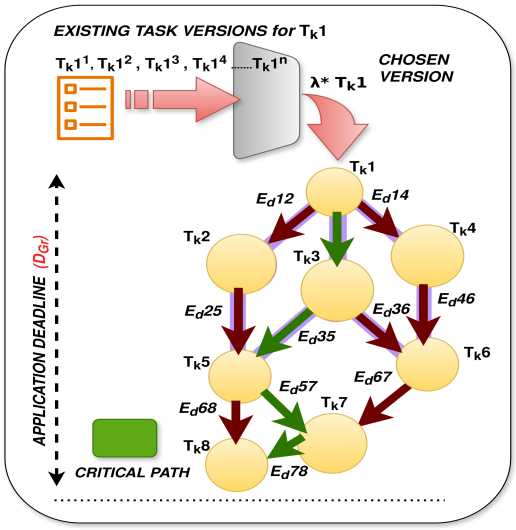


Fig. 2: The Task Graph

A. processing core termination

This might occur due to a runtime error or the introduction of hardware trojan-like malicious software that stops it from functioning as illustrated in Fig. 3b with the typical scenario depicted in Fig. 3a.

B. Unexpected delay in task execution

The two potential causes are malware or ageing processor hardware. This makes it impossible for the real-time system to finish by the deadline, as Fig. 3c illustrates.

C. Intentional draining of power

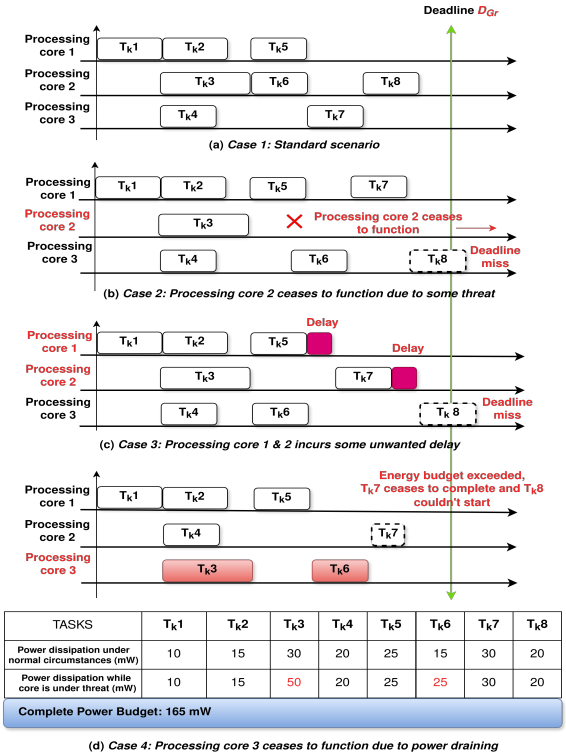
Malware that adversaries have implanted typically causes this. Deliberate malware has power-dissipating circuitry that operates concurrently with the original circuit, in addition to unintentional delays that cause extra power draining. Although they might not have an impact on timing, if they are ignored, they will cause excessive power consumption and eventually deplete the energy budget of the system. If the energy budget runs out in the early or middle phases, the PTG's lowest-level tasks won't have enough energy to finish their tasks. Fig.3d illustrates the system's inability to complete the tasks before the deadline D_{G_r}.

D. Reduced QoS

An anomalous situation is indicated when the measured QoS falls below a predetermined threshold (which is determined empirically by conducting experiments for a specific percentage of utilisation).

IV. Proposed Methodology

REALITY module is composed of two major design blocks namely the moniToRing and Anomaly deteCtion (TRAC) block and the remedial RL-based scheduling block (Fig. 1). Based on a predetermined regular schedule, REALITY begins operating (process 1).



TASKS	T_k1	T_k2	T_k3	T_k4	T_k5	T_k6	T_k7	T_k8
Power dissipation under normal circumstances (mW)	10	15	30	20	25	15	30	20
Power dissipation while core is under threat (mW)	10	15	50	20	25	25	30	20

Fig. 3: Illustrating the threat Model

TRAC block (process 2) detects the threats by observing the execution *time*, *power* consumption of each processing core and measured QoS of the complete platform. The system triggers the anomaly detection module in the TRAC block if this exceeds the ground-truth documented values (of time and power) and the *observed QoS* falls below a predetermined threshold. If the anomaly is confirmed, the corresponding processing core is marked as faulty, and the intelligent scheduler is called upon to create a new fresh schedule based on the available healthy processing cores at that moment and the remaining energy budget (process 3).

A. Anomaly detection and Mitigation Implementation

Anomaly detection and its effects mitigation through appropriate scheduling are the two facets of our proposed security implementation.

1) Alarm for Anomaly: Section III discusses the anomalies arising from various threats. Three variables are used to identify abnormal conditions: power, timing, and variations in QoS.

Timing based alarm:

Intentional delays, ageing, and defects affect the timing parameter. If the observed execution time exceeds the predefined time, an alarm is raised to activate the anomaly detection module *(Algorithm 1) TRAC*.

Alarm based on Power and QoS: Intentional malware consumes the system's energy budget by dissipating excessive power through separate, concurrently operating power-dissipating processes that run alongside the original circuit.

Algorithm 1: safety precaution: moniToRing and Anomaly deteCtion (TRAC)

Input: i. Observed execution Time and Power values obtained from sensors,
ii. A defined QoS.
iii. Predefined Timing and Power information,
iv. Observed QoS.
Output: Labelling the processor core as healthy/faulty

```
for every Task T_ki running on Processing core C_i do
    if (Observed Power value > Predefined Power value) ∨
    (Observed Timer value > Predefined Timer value) ∨
    (Observed QoS < Defined QoS) then
        The anomaly detection module is enabled;
        if found FAULTY then
            Processoring core C_i is marked faulty;
            The proposed scheduler is called to schedule the
              remaining tasks on the non-faulty healthy
              processing cores;
        else
            Go to step 12;
    else
        continues to adhere to the established execution
          schedule;
```

When the observed power consumption surpasses the predefined power consumption, the security module detects it and enables the anomaly detection module, which looks for faults and marks the related processing core as faulty. The remaining tasks on the unaffected processing cores are then scheduled by the scheduler module ensuring execution within the allocated energy budget and deadline. While scenarios mentioned in subsection III-D arise, similar steps are repeated to create a new schedule *(Algorithm 1) TRAC*. We trained our graph neural net-based attention model using the virtual and physical timer values generated at four different privileges in the processing core during the execution of benchmark programs. The specifics of this are outside the scope of this paper.

2) Security Preservation: The scheduler module is rerun after a processing core is identified as faulty to schedule the outstanding tasks on the remaining unaffected processing cores before the deadline and the allocated energy budget. Figure 4 illustrates an instance of handling the reduction in power-related threat handling procedure. As demonstrated excessive power draining triggered an alarm of suspicious events, activating the anomaly detection block. This block then labels core 1 as faulty upon detecting an anomaly, which causes Task 1 to fail to finish by the deadline. Subsequently, REALITY requests a fresh schedule, which calls for finishing all necessary tasks and easing the completion of optional ones. This schedule is intended to schedule the completion of the remaining tasks on the remaining, healthy processing cores within the available energy budget and deadline as described in the next subsection.

B. Proposed Timing and Energy constraint-based intelligent task scheduling

In this subsection, we will discuss the RL-based scheduling algorithm leveraged for generating fresh schedules at runtime for allocating tasks (represented as nodes in the PTG Fig. 2) at suitable processing cores. The PPO algorithm [16] is then used to train the RL-based model with appropriate parameters, which are covered later. These parameters include reward functions, learning rates, and the number of time steps to train. The training produces a task execution policy using a pre-established deterministic emergency action sequence specific to a domain. Subsequently, the task execution policy is responsible for assigning the tasks represented as nodes in the PTG (Fig. 2). While the parent nodes representing tasks are done with their execution, its child node is allotted to an idle core depending on the remaining energy budget and the application deadline at that instance, else the task version with the lesser optional component O (equation 1) is chosen.

1) RL-training parameters: To train the task set in an RL environment, we require multiple parameters and functions as described.

Reward function: A reward function can be defined in a way that rewards good action selection with a positive reward and penalises actions that lead to suboptimal states. Below is a description of the reward function's various component equations. Each complete task is characterised by an arrival time, start time, execution length and the instruction count given by $T_k i = (A_i, S_i, L_i, I_i)$. The execution length L_i for task i with version v_i is expressed as:

$$L_i = S_i + (\frac{I_i}{IPC})_{v_i} \tag{2}$$

where IPC is the instructions per cycle. Therefore the deadline penalty T for the task i is expressed as:

$$p(T) = min(0, T_i - (S_i + L_i)) \tag{3}$$

and the power penalty is computed as:

$$p(P) = -(S_{vi,vj} + L_i(\frac{\mathrm{d}}{\mathrm{d}t}P)_{v_j}) \tag{4}$$

where the power changing rate of task i with version v_j is given by $\frac{\mathrm{d}}{\mathrm{d}t}P$ and the energy switching overhead from task version v_i to v_j is S_{v_i,v_j}. The deadline penalty for the complete application A is given as:

$$p(T_A) = min(0, D_{G_r} - RT_{k^i} \times L_i)) \tag{5}$$

where application deadline is D_{G_r} and RT_{k^i} are the remaining tasks each with length L_i.

These penalties are used to define the reward functions where the model tries to minimise these penalties. The maximum energy is measured while a processing core is executing the highest version of a task along with maximum switching overheads S among versions if any and can be expressed as:

$$max(Eneg) = L_i(\frac{\mathrm{d}}{\mathrm{d}x}P)_{v_{max}} + max(S) \tag{6}$$

We again need to consider three different use cases to formulate the reward functions. *Case 1:* where the task list is empty (i.e. nothing to allocate), *Case 2:* Chosen core for task allocation is invalid since it is already busy, and *Case 3:* Core is available for chosen task allocation meant for scheduling. The corresponding reward functions based on these case studies need to be computed which consist of power and deadline components of tasks and are expressed as:

$$Pow = 10(\frac{max(Eneg) + p(P)}{max(Eneg)}) \tag{7}$$

$$D(T) = 10(\frac{p(T)}{T_i - A_i}) \tag{8}$$

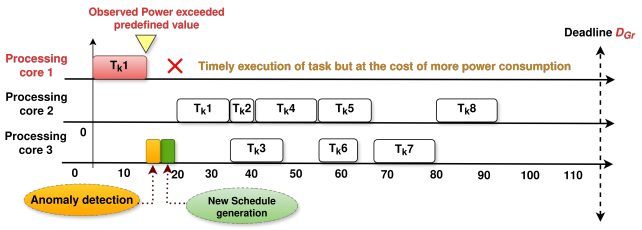
$$D(T_A) = 10(\frac{p(T_A)}{D_{(}G_r) - RT_{k^i} \times L_i}) \tag{9}$$

The computed reward functions for the 3 cases are:

$$Reward = \begin{cases} 0, & case1 \\ 10(\frac{A_i - t}{T_i - A_i}), 10(\frac{RT_{k^i} - t}{D_{G_r} - RT_{k^i}}) & case2 \\ Pow + D(T) + D(T_A), & case3 \end{cases} \tag{10}$$

Learning rate: A hyper-parameter that affects the algorithm's convergence escalation is the learning rate. Most of the time, the values are established empirically.

2) Proposed PPO algorithm-based training procedure: (PPO) is a member of a family of policy optimisation techniques that performs each policy update over several stochastic gradient ascent epochs. When the number of states and potential actions rises, the memory for storing state-action pairs in traditional reinforcement learning (RL) methods based on dynamic programming, such as Q-learning, rapidly grows to enormous values. These algorithms cannot handle the enormous state-action pair possibilities for instances where the state space is continuous. Nonetheless, PPO handles these situations effortlessly since trust-region optimisation is part of its goal. Moreover, PPO allows for the processing of gathered data across several epochs, thereby showing a faster convergence and an increased robustness which motivated us to utilise it for training.

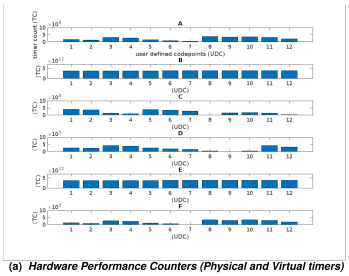


TASKS	T_k1	T_k2	T_k3	T_k4	T_k5	T_k6	T_k7	T_k8
Usual scenario	N:10 OP:14	N:5 OP:8	N:8 OP:12	N:10 OP:13	N:5 OP:10	N:7 OP:11	N:8 OP:12	N:10 OP:20
Anomaly Scenario	N:10 OP:3	N:5 OP:1	N:8 OP:4	N:10 OP:4	N:5 OP:6	N:7 OP:2	N:8 OP:5	N:10 OP:4

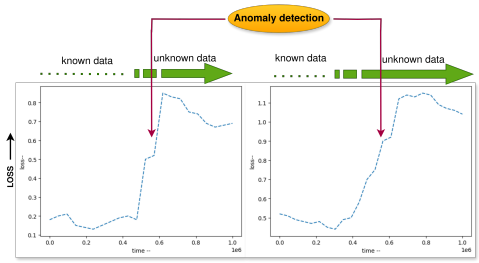
Run time for necessary (N) and optional (OP) components

Handling excessive power draining threats

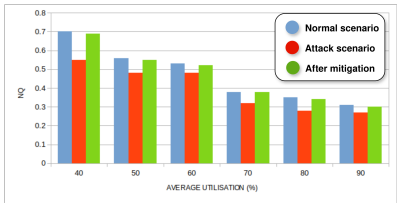
Fig. 4: Threat handling procedure



(a) *Hardware Performance Counters (Physical and Virtual timers)*



(b) *sudden change in the loss function when encountering anomaly*



(c) *NQ variation with average utilisation in normal scenario, Attack scenario and after mitigation*

Fig. 5: (a) Snapshot of HPC values of a processing core (b) sudden increase in the loss curve for the reconstruction and forecasting models upon detecting an anomaly (c) Normalised QoS (NQ) variation in normal, attack and after mitigation scenarios against utilisation percentages.

V. Experimentation and Results

We used the autobench application in our study. A number of benchmark algorithms are included in the popular automotive application package of the EEMBC benchmark suite [17]. The applications that make up this application are programmes, and each programme (referred to as PTG) is made up of several dependent and independent functions that we refer to as PTG tasks. Using the task graphs for free (TGFF) tool, we generated the task graphs [18]. The ARM MORELLO SoC prototype architecture's new, experimental, out-of-order CPU was developed using a homogeneous dual-core processing cluster environment akin to Neoverse-N1 processors, used for our experiments [19].

A. Proposed RL components implementation

Our RL components consist of the environment, actions, and reward functions and observations. We utilised the well-known OpenAI *gym* framework [20] for generating our custom *gym* **environment** providing all the necessary functionalities for executing a learning agent. The agents **action** generate the fresh schedules based on the number of idle cores, remaining tasks, deadline and energy budget. The **observation** is recorded as a (4xn) matrix consisting of the deadline, energy budget, execution length and arrival time of each task, with n

the number of tasks. Following the training a task execution policy is obtained which schedules the tasks on the fly.

B. Results

1) Related to anomaly detection: Figure 5(a) displays the snapshot of HPC values that were obtained on a processing core during the execution of a set of tasks from a benchmark program. To create undesirable delays and miss the deadline, we purposefully introduced a few unwanted programs. Since it did not come across such a pattern of data from the HPCs (physical and virtual timer counts) while training, the loss function of the AI models shows a sudden increase, indicating that REALITY assessed it as an anomalous instance and enabled the anomaly detection block, as shown in Fig. 5(b).

2) Related to NQ and utilisation: We have used task graphs for free (TGFF) [18] to generate task graphs.

Figure 5 illustrates the variation of normalised QoS with average utilisation. The ratio of the actual QoS achieved for the PTG to the maximum QoS obtained by executing the highest versions of every task is known as Normalised QoS (NQ). The PTG's entire execution time is expressed as T_{PTG}, which is the sum of the execution times of all of its subtasks. The deadline of a particular PTG is indicated by D_{PTG}. We compute the ratio of the T_{PTG} and D_{PTG} to get the average utilisation.For example, $U_t = T_{PTG}/D_{PTG}$. For our experimentation, assuming a fixed given deadline D_{PTG}, increasing the utilisation U_t increases the number of tasks and to accommodate it the optional portion of every task needs to be reduced, which in turn decreases the QoS (and the NQ) and vice versa. An attack reduces the system QoS which REALITY mitigates by generating a fresh schedule as illustrated in Fig. 5 (c). When the x-axis's utilisation is varied between 40 and 90%, the NQ also varies between 70%, to 30% (Fig. 5 (c)).

VI. Conclusion

Modern real-time applications, including those in the automotive and industrial sectors, prioritize deadline completion over precision using imprecise computing. A desired QoS can be ensured if time and resources permit. When tasks from an imprecise real-time computing application are transferred to a multiprocessor machine, vulnerabilities may arise. This study highlights several threats that impact the power, timing, and QoS of a computing system. Our proposed real-time scheduler maximizes Quality of Service (QoS) at 70% tested on benchmark programs with varying task utilization on a multiprocessor platform. Furthermore, our anomaly detection module instantly recognizes vulnerabilities at runtime and utilises an intelligent RL-based scheduler to guarantee that an application program is finished before the deadline.

Acknowledgment

This work is supported by the UK Engineering and Physical Sciences Research Council through grants, EP/X015955/1, EP/X019160/1 and EP/V000462/1, EP/V034111/1. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

References

[1] S. Saha et al., "Delicious: Deadline-aware approximate computing in cache-conscious multicore," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 2, pp. 718–733, 2022.

[2] M. Qamhieh, "Scheduling of parallel real-time dag tasks on multiprocessor systems," Ph.D. dissertation, Université Paris-Est, 2015.

[3] B. Hu, Z. Cao, and M. Zhou, "Energy-minimized scheduling of real-time parallel workflows on heterogeneous distributed computing systems," *IEEE Trans. on Services Comp.*, vol. 15, no. 5, pp. 2766–2779, 2021.

[4] I. et al., "Energy efficient edge computing enabled by satisfaction games and approximate computing," *IEEE Trans. on Green Commun. and Net.*, vol. 6, no. 1, pp. 281–294, 2021.

[5] K. Cao et al., "Qos-adaptive approximate real-time computation for mobility-aware iot lifetime optimization," *IEEE Tran. on CAD of Int. Cir. and Sys.*, vol. 38, no. 10, pp. 1799–1810, 2018.

[6] X. Li, L. Mo, A. Kritikakou, and O. Sentieys, "Approximation-aware task deployment on heterogeneous multi-core platforms with dvfs," *IEEE Tran. on CAD of Integrated Circuits and Systems*, 2022.

[7] L. Mo et al., "Approximation-aware task deployment on asymmetric multicore processors," in *2019 (DATE)*. IEEE, 2019, pp. 1513–1518.

[8] M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE Tran. on Parallel and Dist. Sys.*, vol. 28, no. 3, pp. 813–825, 2016.

[9] Y. Wang, M. Karimi, Y. Xiang, and H. Kim, "Balancing energy efficiency and real-time performance in gpu scheduling," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 110–122.

[10] T. Zhang, M. Tehranipoor, and F. Farahmandi, "Trustguard: Standalone fpga-based security monitoring through power side-channel," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[11] M. Abdelrehim, A. Patooghy, A. Malekmohammadi, and A.-H. A. Badawy, "Bic: Blind identification countermeasure for malicious thermal sensor attacks in mobile socs," in *2022 23rd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2022, pp. 1–6.

[12] M. HajiKhodaverdian et al., "Reinforcement learning-based task scheduling using dvfs techniques in mobile devices," in *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE, 2023, pp. 1–6.

[13] R. Vinay et al., "Power and memory efficient high-speed rl based run time power manager for edge computation," in *2023 IEEE 66th (MWSCAS)*. IEEE, 2023, pp. 546–550.

[14] J. Zhang, G. Yu, Z. He, L. Ai, and P. Chen, "Deeppower: Deep reinforcement learning based power management for latency critical applications in multi-core systems," in *Proceedings of the 52nd International Conference on Parallel Processing*, 2023, pp. 327–336.

[15] E. Russo et al., "Towards fair and firm real-time scheduling in dnn multi-tenant multi-accelerator systems via reinforcement learning," *arXiv preprint arXiv:2403.00766*, 2024.

[16] J. Schulman et al., "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[17] E. M. B. Consortium *et al.*, "The embedded microprocessor benchmark consortum," 2008.

[18] K. Vallerio, "Task graphs for free (tgff v3. 0)," *Official version released in April*, vol. 15, 2008.

[19] B. et al., "Verified security for the morello capability-enhanced prototype arm architecture," in *European Symposium on Programming*. Springer International Publishing Cham, 2022.

[20] G. Brockman et al., "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.