ALARM: Digital Twin-Assisted Reinforcement LeARning for Anomaly Mitigation in vehicular Edge Computing

Chandrajit Pal, Sangeet Saha, Xiaojun Zhai, and Klaus D. McDonald-Maier
School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, UK
Email: {chandrajit.pal, sangeet.saha, xzhai, kdm}@essex.ac.uk

***Abstract*—As smart vehicles connect to the Internet of Vehicles (IoV), their electronic systems face severe cyberattack risks. Missing a time-sensitive task during an attack can be disastrous. While current defences often rely on static rules to fix system failures, in this paper (ALARM), we introduce an adaptive, Digital Twin (DT)-assisted Reinforcement Learning (RL) framework to detect and mitigate varying threats. To overcome the risks of training and executing AI algorithms on cars in real time, a DT safely pre-trains the RL agent using simulated attacks. During an intrusion, the agent acts as a smart healing module, reallocating tasks to healthy processors in real-time to meet strict deadlines while satisfying resource constraints, ensuring safe vehicle operation. Experiments show that ALARM maintains a Quality of Service (QoS) between 27% and 68% as system utilisation varies from 40% to 90%. The system is highly efficient at detecting threats, reaching 98.49% accuracy even when the signs of an attack are minute. This makes ALARM a powerful tool for making vehicle networks more reliable and energy-efficient, essential for passenger safety in the IoV.**

***Index Terms*—Imprecise Computation (IC), Digital Twin, Precedence-constrained Task Graphs (PTGs), Graph Attention Networks (GAT), Internet of Vehicles (IoV), Electronic Control Units (ECUs), Reinforcement Learning (RL).**

I. Introduction

The growing integration of smart vehicles into the Internet of Vehicles (IoV) has brought a significant increase in the use of Electronic Control Units (ECUs) in automobiles. These ECUs act as small computers handling essential tasks like vehicle-to-vehicle communication, sensing, navigation etc. While this connectivity makes vehicles smarter, it also creates a larger target for cyber threats [1], [2], a scenario as illustrated in Figure 1. Because automotive systems deal with critical, hard real-time tasks, failing to meet a processing deadline due to a cyberattack or system delay can cause accidents.

To keep systems running under strict energy (due to battery powered) and timing limits, on-board vehicle computation often utilises the Imprecise Computation (IC) task model [3]. This model splits tasks into a 'necessary' part (required for basic safety) and an 'optional' part (which improves accuracy but can be skipped if time runs short). To achieve the minimum acceptable Quality of Service (QoS), all necessary tasks must finish before the deadline. If extra resources are available, optional tasks can also run either partially or fully, to make the initial results more accurate. Essentially, the more time spent on these optional parts, the higher the overall QoS can be achieved [4]. For example, in vehicle-to-vehicle (V2V) communication, it is better to receive a lower-quality video frame on time than to miss the frame entirely. Similarly, in target tracking, a rough estimate of a location delivered within the deadline is more useful than an accurate location that arrives after it is no longer needed.

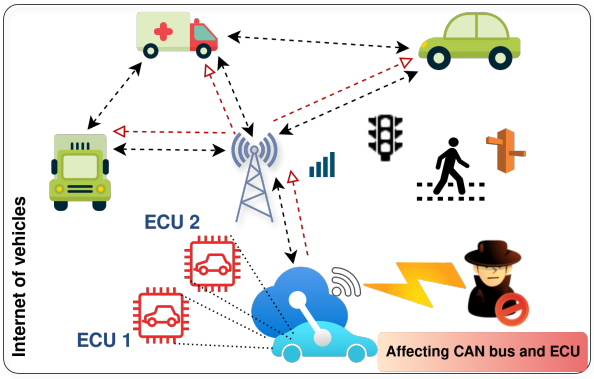

Fig. 1: A typical scenario illustrating the Internet of Vehicles

Earlier security methodologies managed to implement these tasks using static rules, which often use strict protocols to meet deadlines and also indiscriminately reduce processor voltage to save power. However, lowering the voltage can make the system less reliable. Furthermore, when modern vehicle networks face unpredictable attacks such as malware that deliberately drains battery power or slows down a processor, the rigid, pre-programmed rules struggle to adapt, often leading to system failures. To overcome the limitations of static rules, recent research has successfully turned to Reinforcement Learning (RL). They are highly effective in dynamic environments, since it learns from prior experiences. Recent studies show that Deep Reinforcement Learning is highly effective for dynamic task scheduling and resource allocation in vehicular edge computing, successfully minimising latency and energy consumption [5]. Similarly, RL has been applied to anomaly detection in IoV networks, proving its unique ability to continuously adapt to new, evolving attack patterns [2].

However, training/adaptation of these decision-making RL agents on a physical car on the fly is time-consuming and

expensive. Because RL agents learn through trial and error, the system would have to encounter numerous safety-critical failures (such as failing to apply a brake in time) to learn the correct defensive actions.

Hence, we introduce ALARM, a Digital Twin (DT)-assisted RL framework that ensures reliable QoS under severe faults. By utilising a DT for offline pre-training, ALARM provides a *warm start* that exposes the agent to diverse simulated faults (arising out of attacks) and unprecedented events. This bypasses the risks of trial-and-error in an operational vehicle, where an untrained agent's initial mistakes could lead to system instability. Upon deployment, the pre-trained agent functions as an active healing/error mitigation module, capable of reassigning tasks and meeting deadlines and energy limits.

The key contributions of ALARM are summarised below:

- ALARM introduces a custom-designed Digital Twin to safely pre-train the RL agent on fine-grained hardware data, eliminating the safety risks of training an AI algorithm on live operational vehicular platforms.
- We propose an intelligent healing module that uses RL to schedule tasks during active attacks, continuously balancing necessary and optional components of a task based on real-time task deadlines and energy limits.
- Experiments show that ALARM successfully maintains a QoS between 27% and 68% under varying system workloads (40% to 90%), achieving high detection and mitigation accuracy even against subtle cyber threats.

II. Proposed Methodology

Proposed ALARM features a reliability-aware scheduler that maintains high performance even during system faults. It uses a multi-ECU system to run safety-critical applications, which are modelled as a graph of dependent tasks (Fig. 3).

It assigns tasks to appropriate processing cores at specific times to meet the timing and power constraints. However, if the system detects unusual power spikes and/or if tasks start missing deadlines due to unwanted delays and/or QoS degrades, it hints about some abnormality and the anomaly detection module is enabled. This module identifies threats by carefully analysing the relationships among different Hardware Performance Counters (HPCs) collected from the car's electronic control units (ECUs). Once a problem is found, it labels the specific processing core as faulty. Immediately, ALARM creates a new schedule based on the remaining time, energy, and healthy processing cores (as depicted in Fig. 2). This backup plan ensures all essential tasks are finished, and as many optional tasks as possible are completed, maximising the system's overall performance. Its efficiency is proven by its high Normalised QoS (NQ), computed using Eqn. 3.

A. System and threat Modelling

Considering an FPGA SoC evaluation platform with a set of prototyped homogeneous n multiprocessing ECU cores denoted as $C = \{c_1, c_2, ..., c_n\}$ in its programmable logic (PL) (Ref. Fig. 4). Each core is defined by its specific settings of voltage, frequency and power levels. We modelled a real-time vehicular application (A) as a Precedence Task Graph (PTG), as shown in Fig. 3 given by $G = (T_k, E_d)$, which is both directed and acyclic. T_k and E_d are the tasks (nodes) and edges (connecting the tasks/nodes), respectively, representing the precedence relationships for the different task pairs.
T_k denotes a task set ($T_k = \{T_{ki} \mid 1 \leq i \leq |T_k|\}$) and E_d a list of directed edges ($E_d = \{\langle T_{ki}, T_{kj} \rangle \mid 1 \leq i, j \leq |T_k|; i \neq j\}$), illustrating the precedence relationships among different task pairs. It is to be noted that an edge precedence means task T_{kj} can only start execution after its predecessor T_{ki} completes. To satisfy the real-time constraints of application A, all task nodes must complete execution before the final deadline. We decomposed each task T_{ki} ($1 \leq i \leq n$) into a mandatory necessary component N and an enrichment-based optional component O_P. While N_i must be fully executed to ensure functional safety, O_P is triggered only upon the completion of N and is executed partially or entirely to maximise the Quality of Service (QoS) within the remaining resource budget.

The execution length for every task T, is represented as:

$$Ln_i = N_i + \lambda \times O_P i \tag{1}$$

where λ is a set of n discrete values and denotes the executed optional component portion.

The parameter $\lambda = 1$ signifies the full execution of the optional component $O_p i$, maximising result accuracy. The system-level QoS is determined by the total O_P cycles completed across all tasks. We assume n_i distinct versions for each task T_{ki}, such that $T_{ki} = \{T^1_{ki}, T^2_{ki}, \ldots, T^{n_i}_{ki}\}$. As illustrated in Fig. 5b, a task may have 14 optional units. Under normal operation, λ ranges from 0 to 1 across these units, while an anomaly scenario may force a reduced execution range to mitigate attack effects. The energy consumption E_i for a task T_{ki} with length Ln_i is given as [6]:

$$E_i = Ln_i \times POW_i \tag{2}$$

where POW_i is the power consumption of the assigned processing core. While increasing O_p execution improves accuracy and QoS, it also extends Ln_i and raises energy consumption. Consequently, the scheduling objective is to maximise QoS by selecting higher task versions while minimising total energy.

Due to a fault, firstly, a processing core might completely stop working due to a runtime error or hidden malware causing a fault. Secondly, malware or an ageing hardware can cause unexpected delays, preventing the system from finishing tasks on time. Third, attackers might plant malware that secretly drains the system's power by running extra, hidden circuits. Even if this does not slow things down right away, it can use up the energy budget too early, leaving later tasks without enough power to finish before their deadlines. Finally, if the measured quality of service drops below a set safe level, it acts as a warning sign that an anomaly is happening (Ref. Fig. 5b). Algorithm 1 describes the procedure of anomaly monitoring and the detection process. While this 'Threshold QoS' (Alg. 1) is a static, safety-critical bound ensuring mandatory tasks

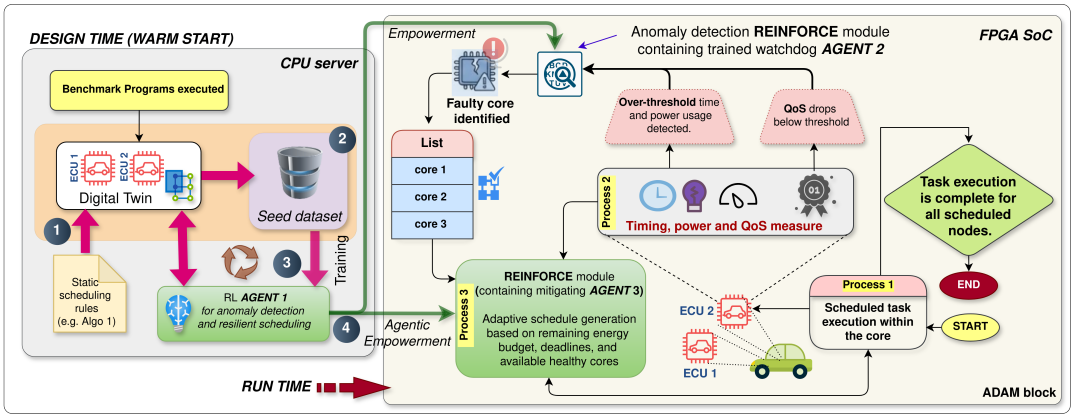

Fig. 2: ALARM flow chart

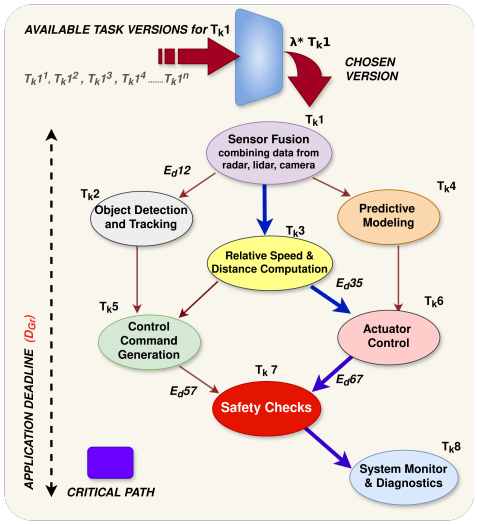

Fig. 3: Task Graph of in-vehicular control architecture [7]

Algorithm 1: Implementing Security: Anomaly Detection And Monitoring (ADAM)

Input: (i) Sensor derived timing and Power information, (ii) Acceptable QoS, (iii) Predefined Timing and Power values, (iv) Derived QoS.

Output: Processor core designated as normal/compromised

```
for every Task T_ki running on Processing core PC_i do
    if (Predefined Power value < Derived Power consumption) ∨
    (Predefined Time < Observed Time) ∨
    (Derived QoS < Threshold QoS) then
        Activate the anomaly detection REINFORCE module Agent 2; /* Refer Figure 2 */
        if FAULTY==1 then
            Processoring core C_i is tagged compromised;
            Call REINFORCE module Agent 3 to schedule the remaining tasks on the healthy cores (Replacing the compromised processor with a backup core operating at a high frequency);
        else
            Go to step 12;
    else
        Under normal conditions, the system adheres to its standard execution schedule.
```

finish, ALARM's core detection remains highly dynamic, tracking continuous distribution shifts via the VAE and GRU to catch threats before this static limit is breached.

B. Proposed REINFORCE module implementation

Our security implementation focuses on two key stages: detecting anomalies and mitigating their effects using our proposed *REINFORCE* module. As shown in Fig. 4, we developed a fine-grained hardware analyser that processes time-series data from Hardware Performance Counters (HPCs) to identify threats. The *REINFORCE* module is implemented in the processing system (PS) executed by the ARM core of the FPGA SoC (Ref. Fig. 4).

Raw HPC data are normalised and passed through a 1D convolution layer (using a kernel size of 5) to extract high-level features. Two *Graph Attention Networks* (GATs) operate simultaneously. One extracts the spatial relationships among different processor signals, while the other captures temporal features over time. The outputs from the GATs are combined and fed into a 24-unit *Gated Recurrent Unit* (GRU). This GRU captures long-term data dependencies and has an exceptionally small footprint of only 492 KB. The GRU output splits into two parallel networks. The first is a Variational Autoencoder (VAE) reconstruction model (size 1.2 MB) that learns the normal distribution of the data. The second is a forecasting model (size 2.4 MB) utilising two fully connected layers to predict future processor behaviour. These models generate a reconstruction loss and a forecasting loss, which highlight deviations from expected normal behaviour. The output is fed to a watchdog RL *Agent 2*. Since the Watchdog's task is a rapid, binary decision based on pre-processed data from the forecasting and reconstruction models, we developed a lightweight Deep Q-Network (DQN). This network is ideal for discrete action spaces and requires minimal computation.

The input layer (i.e. the state space) consists of 2 percep-

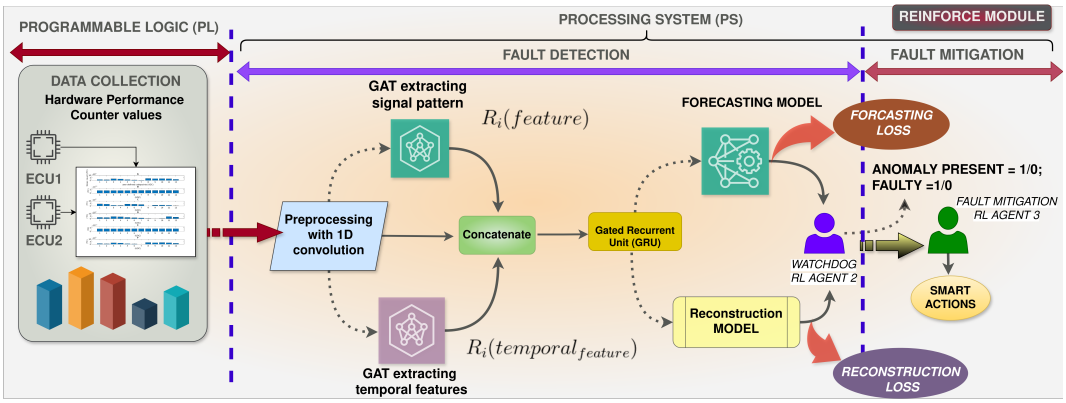

Fig. 4: Anomaly detection and mitigation module (Ref. ADAM and REINFORCE blocks in Fig. 2) on FPGA SoC

trons that receive the continuous values of the forecasting error and the VAE reconstruction probability at the current time step. The network utilises a shallow MLP architecture with two hidden layers of 64 and 32 neurons, respectively, using ReLU activation functions to introduce non-linearity. With this shallow architecture, the Watchdog agent contains approximately $2,300$ parameters. Using 32-bit floating-point precision, the entire compiled model requires less than 10 KB of memory. The final output layer (i.e. the action space) consists of 2 perceptrons representing the Q-values for the discrete actions: *Normal* (0) and *Faulty/compromised* (1). We assigned $+1$ for correct classification (catching an attack), -1 for flagging normal behaviour as an attack and -10 for failing to detect a real threat.

The second RL *Agent 3* (mitigating the detection of faults) handles the complex scheduling environment involving precedence constraints. We utilise Proximal Policy Optimisation (PPO), an actor-critic algorithm known for its stability. To exactly accommodate the target system (an 8-node task graph (T_k1 to T_k8) running on a 2-core ECU cluster), the dimensions are structured accordingly. The input layer requires a dimension of 12. This flattened vector consists of 1 continuous value for the remaining energy budget, 1 for the remaining deadline time, 2 binary values indicating the health status of the 2 processing cores, and 8 binary values representing the execution status (completed/pending) of the 8 tasks. Both the Actor (policy) and Critic (value) networks consist of three hidden layers with 128, 64, and 32 perceptrons, utilising ReLU activations to capture the interdependencies of the task graph. The Actor utilises a multi-discrete output layer with a dimension of 20 (split into $8 + 2 + 10$). The agent outputs logits to select a task ID (from 8 options), a target processor core (from 2 options), and a QoS scalar (λ) (from 10 discrete levels representing the percentage of the optional component to execute). A dynamic *Action Mask* is applied directly to the 8 task logits before the softmax activation, thereby reducing the probability of selecting incomplete parent tasks to zero. The Actor network contains exactly $12,660$ parameters, and the Critic network contains $12,033$ parameters. Using 32-bit precision, the combined memory footprint of the entire Healer agent is approximately 98.8 KB. Because both lightweight agents execute natively as co-located processes on the same ARM PS, the handoff latency from Watchdog detection to Agent 3's mitigation is negligible. (microsecond scale).

For reward computation, we assigned $+10$ for successful completion of all tasks within the deadline and energy budget, A scaled bonus in between $(0-1)$ based on the NQ and -20 for missing a hard deadline or running out of energy.

C. Digital Twin assisted warmstart

(Ref. Fig. 2) Training RL algorithms directly on operational vehicle ECUs is unsafe, as trial-and-error exploration could miss hard real-time deadlines and cause catastrophic physical failures. To solve this, ALARM safely *warm starts* the agents within the Digital Twin DT (which we designed (software library) by matching the exact hardware specifications of an FPGA-prototyped ECU). The benchmark programs are executed in the prototyped ECU cores within the DT environment and utilising the protocols in the Algo. 1 described in [8] operating on the DT, the seed dataset (step 2) is generated. This dataset is used to train the DQN-based watchdog detecting *Agent 1* and PPQ-based mitigating *Agent 1'* in the server offline(i.e. design time).

The agents are pre-trained using the Adam optimiser in two distinct phases. First, in the static baseline phase, the agents learn basic precedence constraints and normal system behaviour [8]. Next, during the dynamic randomisation phase, the interactive environment introduces simulated real-world system noise and distinct fault profiles, such as sudden mathematical injections to mimic cyberattacks, and progressive drift to simulate hardware ageing. Using this safe environment, the Watchdog *Agent 1* trains over $10K$ episodes with heavy penalties for false negatives, while the mitigating/healing *Agent 1'* trains for $50K$ episodes offline in a server (step 3). This comprehensive process allows the algorithms to fully converge on optimal policies. Following this step, they are deployed on the operational FPGA as watchdog *Agent 2* and mitigating *Agent 3* through agentic empowerment (step 4). The mitigating *Agent 1'* trains on the basic protocols as given below (with

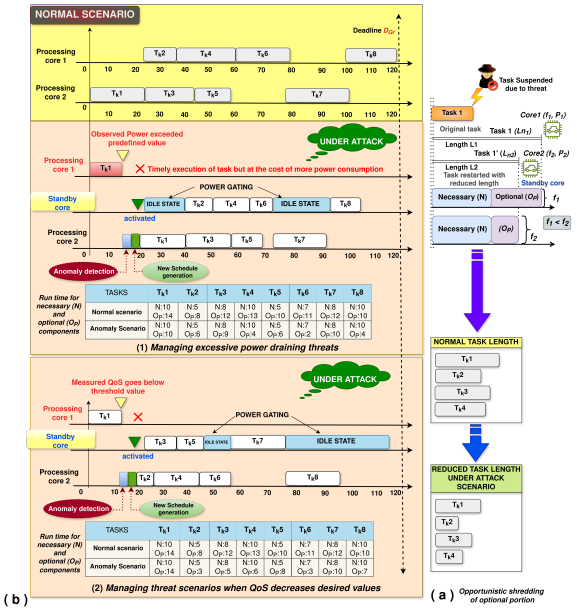

Fig. 5: (a) A threat *suspends* compromised tasks and restarts them on standby cores. Trims optional cycles to save energy during recovery (b) Threat mitigation: Uses standby high-frequency cores and clock gating to meet deadlines, while shedding optional task units to maintain QoS.

example actions in Fig. 5), i.e., following Algo. 2 in [8] as summarised below:

The scheduling process operates as follows: (1) *Initialisation:* Identify available processing cores and initialise the task list. (2) *Selection & allocation:* Assign ready tasks to free cores based on predecessor completion. (3) *Task version selection:* Choose the highest task version (more the Op optional portion, the higher the version is) compatible with the energy budget and deadline constraint. (4) *Execution & monitoring:* Run tasks while tracking time and energy consumption. (5) *Update & repeat:* Release cores upon completion and iterate until all tasks are finished or constraints, such as deadline and energy, are met. When a core gets compromised, backup cores are employed to take over. (6) *Finalisation:* Compute the achieved Normalised QoS (NQ). Example output shown in Fig. 5. The NQ is computed as the *ratio of the executed optional component to the total number of available optional components of all the tasks of an application, as:*

$$NQ_A = \frac{\sum_{i=1}^{n} O_P ei}{\sum_{j=1}^{n} O_P Lj} \tag{3}$$

where $O_P e$ is the executed optional component and the denominator is the total number of available optional components of all n tasks of application A. The ranges of necessary N_i and optional $O_P i$ components of a task are obtained from [9].

III. Experimentation and Results

Our simulation modelled an AMD-Xilinx Zynq-7000 XC7Z045 SoC FPGA. The *REINFORCE* module, consisting of the GAT model, forecasting and reconstruction models, and the dual Agent RL algorithms are running their inference on the ARM processor residing on the processing system (PS) of the FPGA SoC. The two processing cores are configured in the programmable logic (PL) section, prototyping the ECU (Ref. Fig. 4). The Watchdog agent's detection module was initially trained on the EEMBC Autobench suite [10]. As shown in Table I, the module achieved a general classification accuracy of 98.49%, proving its ability to successfully isolate unknown, malicious sub-routines from normal task execution.

TABLE I: Performance of EEMBC Autobench benchmark

Benchmarks	Accuracy	Precision	Recall	F1 SCORE
bitmnp	97.2	99	98.5	99.19
a2time	96.8	98.2	98.41	98.14
puwmod	97.6	99.16	97.14	98.43
rspeed	98.2	99.15	98.14	98.62
tblook	96.2	98.42	97.64	97.98
idctrn	99.3	0	0	0
ttsprk	98.63	0	0	0
General performance	98.49	98.92	98.2	98.52

A. Threat Detection Analysis

To simulate realistic vehicular threats, four anomaly types (instantaneous, constant, progressive drift, and bias) were injected into the high-frequency SPMD dataset [11], which contains real vehicle velocity and acceleration data. Table II compares the detection of instant anomalies against state-of-the-art models, including MSALSTM-CNN [12], WKN-OC [13] and RENOWNED [8]. ALARM notably excels at detecting extremely subtle threats, even at a minute anomaly magnitude of base value $+10 \times N(0, 0.01)$, it achieved 97.6% accuracy. Table III further validates this robustness under mixed anomaly conditions, demonstrating peak F1 scores of 91.85% for constant anomalies and 92.14% for bias-based threats [14].

B. System Mitigation and QoS Recovery

Once an anomaly is detected, the *Agent 3* executes its mitigation policy. Figure 6 maps the system's Normalised Quality of Service (NQ) against core utilisation. Under normal conditions, NQ decreases as utilisation rises. During an attack, NQ drops steeply. However, post-mitigation, the *Agent 3*'s dynamic rescheduling successfully recovers system performance. By cleverly shedding optional task components, the system successfully maintains an NQ ranging from 27% (under heavy 90% utilisation) to 68% (at 40% utilisation). This proves the RL *Agent 3* can neutralise threats while strictly adhering to real-time deadlines and energy budgets. It is to be noted that *system utilisation* is simply defined as the ratio of the total execution time needed for all tasks to the maximum allowed deadline. If utilisation goes up, it means the system is trying to pack more tasks into the same amount of time, forcing it to drop *optional* task components and lower the QoS.

IV. Conclusion

This paper introduces ALARM, a dynamic, dual-agent Reinforcement Learning framework designed to secure edge-

TABLE II: Instantaneous anomaly detection performance: ALARM vs. state-of-the-art models (SPMD dataset [11])

	MSALSTM-CNN(%)[12]			WKN-OC(%)[13]			RENOWNED (%) [8]			**ALARM (%)(Ours)**		
Anomaly Magnitude	Acc	Sens	Prec	Acc	Sens	Prec	Acc	Sens	Prec	Acc	Sens	Prec
base value+10*N(0,0.01)	76.4	51	97.2	94.7	60.2	94.8	96.2	62.4	97.6	**97.6**	**65.1**	**98.2**
base value+25*N(0,0.01)	84.1	54.6	98.1	96.5	65.8	96.7	96.8	65	**97.2**	**97.8**	**68.4**	97
base value+50*N(0,0.01)	90.2	75.9	97.9	98	73.5	98.4	98.4	75.7	98.7	**98.9**	**78.2**	**99.1**
base value+100*N(0,0.01)	95.8	89.6	98.4	99.0	90.7	98.8	99.0	91.8	99.3	**99.5**	**94.1**	**99.7**
base value+200*N(0,0.01)	96.02	93.5	98.3	**99.4**	94.6	99.2	99.3	96.3	99.6	99.35	**97.5**	**99.8**

TABLE III: Comparing ALARM detection performance with *mixed anomaly types* on SPMD dataset [11].

Anomaly Magnitude	Sensors	MSALSTM-CNN [12]		WKN-OC [13]		RENOWNED [8]		**ALARM (Ours)**	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1
Instant: $25 * \mathcal{N}(0, 0.01)$	1	85	72.3	92.81	74.03	96.79	81.62	**97.85**	**83.10**
	2	91.4	79.65	95.64	81.37	95.87	83.76	**97.12**	**85.24**
	3	87.49	76.91	**96.5**	72.9	96.4	76.21	96.45	**78.45**
Constant: $U(0, 1), d = 3$	1	92.54	81.67	91.57	82.01	90.46	84.93	**93.65**	**86.72**
	2	93.84	78.97	93.87	90.39	95.82	88.36	**96.94**	**91.85**
	3	89.21	80.61	90.92	82.57	94.32	84.09	**95.61**	**86.14**
GD: linespace(0, 1), $d = 1$	1	92.69	83.12	94.5	85.24	96.37	87.43	**97.62**	**89.15**
	2	93.21	80.78	92.65	83.17	94.32	**86.77**	**95.80**	85.62
	3	89.08	80.37	93.14	79.97	97.39	82.53	**98.51**	**84.38**
Bias: $U(0, 2), d = 5$	1	95.32	84.21	92.34	86.32	96.93	88.12	**98.24**	**89.65**
	2	96.12	82.36	95.12	89.81	97.65	90.37	**98.81**	**92.14**
	3	95.36	90.47	94.35	86.57	**97.56**	89.21	97.3	**91.80**

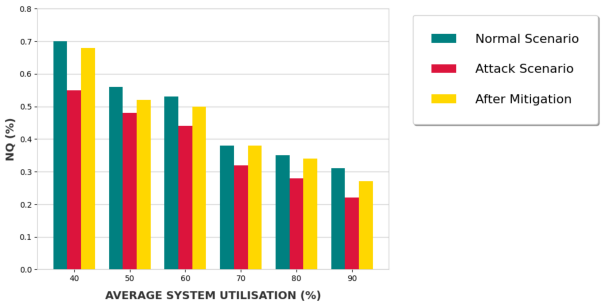

Fig. 6: Normalised QoS (NQ) variation in normal, attack and after mitigation scenarios.

enabled IoV systems against runtime cyber-physical threats. By safely pre-training a lightweight DQN Watchdog and a PPO mitigating agent within a Digital Twin emulator, ALARM successfully eliminates the reliance on rigid, human-defined thresholds. Experimental evaluations on an ARM hardware prototype demonstrate that ALARM can rapidly detect subtle anomalies and dynamically reschedule precedence-constrained tasks, successfully maintaining a desired QoS without violating strict energy budgets or deadlines. Currently, scaling the PPO agent beyond the evaluated 8-node/2-core architecture requires complete offline retraining. Our next plan is to expand the mitigating *Agent 3* action space dynamically to handle heterogeneous multicore architectures. Furthermore, to address the lack of online learning and prevent failures against unprecedented novel attacks, we plan to integrate federated learning into the ALARM framework, which will allow multiple connected vehicles to collaboratively share anomaly experiences and continuously update the watchdog agent's detection policy across active IoV environments.

ACKNOWLEDGMENT

This work is supported by the UK Engineering and Physical Sciences Research Council through grants, EP/Z533749/1. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any arising Author Accepted Manuscript version.

REFERENCES

[1] C. Chen *et al.*, "Traffic flow prediction based on deep learning in internet of vehicles," *IEEE transactions on intelligent transportation systems*, vol. 22, no. 6, pp. 3776–3789, 2020.

[2] S. Jamshidi *et al.*, "Application of deep reinforcement learning for intrusion detection in internet of things: A systematic review," *Internet of Things*, vol. 31, p. 101531, 2025.

[3] N. Irtija *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.

[4] 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.

[5] Z. Wu *et al.*, "Deep reinforcement learning-based task offloading and load balancing for vehicular edge computing," *Electronics*, vol. 13, no. 8, p. 1511, 2024.

[6] B. Salami *et al.*, "Fairness-aware energy efficient scheduling on heterogeneous multi-core processors," *IEEE Tran. on Comp.*, vol. 70, no. 1, pp. 72–82, 2020.

[7] D. Senapati *et al.*, "Hmds: A makespan minimizing dag scheduler for heterogeneous distributed systems," *ACM Tran. on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–26, 2021.

[8] C. Pal *et al.*, "Renowned: a real-time anomaly detection and mitigation framework in edge-enabled iov," *IEEE Internet of Things Journal*, 2025.

[9] L. Mo *et al.*, "Approximation-aware task deployment on asymmetric multicore processors," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1513–1518.

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

[11] F. Van Wyk *et al.*, "Real-time sensor anomaly detection and identification in automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1264–1276, 2019.

[12] A. R. Javed *et al.*, "Anomaly detection in automated vehicles using multistage attention-based convolutional neural network," *IEEE Trans. on Int. Transp. Sys.*, vol. 22, no. 7, pp. 4291–4300, 2020.

[13] Z. He *et al.*, "Wkn-oc: a new deep learning method for anomaly detection in intelligent vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 3, pp. 2162–2172, 2023.

[14] C. Pal *et al.*, "Apparent: Ai-powered platform anomaly detection in edge computing," *IEEE Transactions on Sustainable Computing*, 2025.