2022 IEEE 40th International Conference on Computer Design (ICCD) | 978-1-6654-6186-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCD56317.2022.00042

Breakthrough to Adaptive and Cost-Aware Hardware-Assisted Zero-Day Malware Detection: A Reinforcement Learning-Based Approach

Zhangying He[1], Hosein Mohammadi Makrani[2], Setareh Rafatirad[3], Houman Homayoun[2], and Hossein Sayadi[1]

[1]Department of Computer Engineering and Computer Science, California State University, Long Beach, CA, USA
[2]Department of Electrical and Computer Engineering, University of California, Davis, CA, USA
[3]Department of Computer Science, University of California, Davis, CA, USA

***Abstract*—In this paper, we have identified and addressed pressing challenges associated with online and cost-effective malware detection based on Hardware Performance Counters (HPCs) information. Existing Hardware-Assisted Malware Detection (HMD) methods guided by standard Machine Learning (ML) algorithms have limited their study on detecting known signatures of malicious patterns; thus, neglecting to address unknown (zero-day) malware detection at run-time which is a more challenging problem since the malware HPC data does not match any known attack applications' signatures in the existing database. In addition, prior works have not presented a flexible and balanced solution that considers the trade-off between detection rate and implementation cost for adaptive selection of the best performing ML algorithms for online malware detection. In this paper, we first propose a unified feature selection method based on a heterogeneous feature fusion technique to effectively determine the most important HPC events for low-cost yet accurate malware detection. Next, we present *Reinforced-HMD*, a novel reinforcement learning-based framework for adaptive and cost-aware hardware-assisted zero-day malware detection based on desired performance metric and available hardware resources. To this aim, six classical and two reinforcement learning algorithms are implemented and their efficiency is thoroughly analyzed for detecting unknown malware using HPC events. Experimental results demonstrate that our *Reinforced-HMD* framework based on Upper Confidence Bound (UCB) learning approach achieves an accurate and robust detection rate with a 96% in both F1-score and AUC metrics for flexible and efficient zero-day malware detection while utilizing an optimal set of built-in HPC events.**

***Keywords*—Hardware Performance Counters, Machine Learning, Reinforcement Learning, Zero-Day Malware Detection.**

I. Introduction

The last decade has witnessed a vast growth in the complexity of cutting-edge digital systems. This has resulted in the emergence of new security vulnerabilities making the systems accessible targets for an increasing number of complicated cyber attacks [1], [2], [3]. With malicious software (a.k.a. malware) utilization continuing to rise across different application domains, the development of efficient malware detection techniques have grown to be more crucial as they feature as an early protection mechanism to guard the integrity and confidentiality of the authenticated users' data [4], [5].

Hardware-Assisted Malware Detection (HMD) methods [3], [6], [7], [8], [9], [10] have emerged to address the inefficiency of software-based detection solutions, including static analysis, incompetence in detecting obfuscated attacks, and excessive computational overheads on resource-limited systems. Figure 1 illustrates the general process of employing low-level hardware events for distinguishing malware from benign applications. Such methods are based on classical Machine Learning (ML) algorithms utilized on microarchitectural events collected from Hardware Performance Counter (HPCs) registers'[1] [11], [12], [13], [14]. Recent developments in the field of machine learning have been sparked by a huge increase in the volume of data in modern computer systems, leading to applications of these intelligent methodologies in a range of computing fields, including security [5], [15], [16], [17]. With ML-based protection countermeasures, the hardware systems could analyze patterns of applications to proactively respond to altering behavior at run-time and prevent conceivable attacks. In this work, as highlighted below we address important challenges of adaptive and cost-effective hardware-assisted malware detection that have been neglected in prior studies.

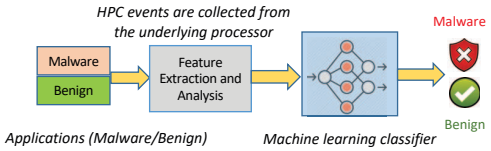

Fig. 1: General process of detecting malware using hardware events.

Challenge ❶: Unified Feature Selection for Online HMD. Despite their high performance, most modern microprocessors are equipped with a limited number of performance counter registers (from 2 up to 8) since the design complexity and cost of concurrent monitoring of HPC events are high [7], [14]. Nonetheless, there exist numerous hardware events with each of them representing a unique functionality in which monitoring all of them leads to data with high dimensionality. Our analysis shows that there have been different feature selection methods used in prior HMD solutions with no apparent justification [10], [14], [18], [19] ending up in different sets of top HPC events. More importantly, no unified feature selection solution exists that combines the functionality of different selection methods used across various existing works to determine the top HPC events for online and cost-effective malware detection.

Challenge ❷: Recognizing Zero-Day Malware. Malicious software has continued to evolve in sophistication and quantity over the past decade. Zero-day attack is a type of serious cybersecurity threat that exploits software security vulnerabilities that are undocumented (unknown) in the training database

[1]HPCs are specialized registers embedded in modern microprocessors to monitor the applications' hardware events (e.g., number of cycles and instructions that a program executed, its associated cache misses, etc.)

2576-6996/22/$31.00 ©2022 IEEE
DOI 10.1109/ICCD56317.2022.00042

of the detection mechanism [20]. Lack of signature history and clear remediation strategy has made detection of zero-day malware a long-standing challenge using the traditional off-the-shelf detection mechanisms such as static signature-based approaches. In addition, existing machine learning-based hardware-assisted malware detection methods have ignored the important problem of zero-day malware detection. Therefore, they are inherently unscalable and inflexible, as the inclusion of any new malware types would require training of new models which makes the solution inefficient.

Challenge ❸: Limitation of Standard ML Models for Adaptive On-device HMD. The existing ML-based HMD strategies lack a structured method to account for the performance vs. cost trade-offs within their target function. In addition, while standard ML-based approaches have proven to be more effective than their signature-based counterparts in recognizing formerly unknown malware, they typically suffer from a noticeably high false positive rate, which makes the application of a single standard ML-based detector for adaptive hardware-assisted malware detection inefficient. To successfully tackle the performance vs. cost-efficiency challenge of HMD methods, each HPC data needs to be analyzed by the most cost and computationally efficient ML model needed to correctly classify the HPC sample at run-time. However, such an adaptive on-device solution is not aligned with the application of standard ML-based malware detection methods with limited efficacy.

Challenge ❹: Inefficiency of Ensemble-based Detectors. Inspired by ensemble learning-based methods, using ML-based malware detectors together (e.g., voting, stacking, and boosting) have shown to enhance the detection rate and reduce the false positive rate of detection process. Despite the potential security enhancement, ensemble-based detection methods often incur significant overheads in terms of design cost and computational latency. In particular, the processing time required to analyze each sample is equal to or higher than that of the most time consuming ML model and the large number of detectors requires expensive computing resources (new hardware or cloud server) which makes the solution impractical for fast and low-cost on-device malware detection.

In response to the aforementioned challenges, in this work we propose *Reinforced-HMD*, an adaptive and cost-aware framework for online hardware-assisted zero-day malware detection. *Reinforced-HMD* is equipped with a novel heterogeneous ensemble feature selection method followed by an effective reinforcement learning-guided decision-maker that adaptively selects the most accurate and cost-efficient ML model for detecting unknown malware signatures. An effective feature selection based on a heterogeneous fusion method is presented by exploiting the correlation between HPCs from different selection methods to specify the most prominent HPC events for online malware detection without hampering the detection accuracy. Next, *Reinforced-HMD* formulates the hardware malware detection as a Reinforcement Learning (RL) problem [21], [22] by examining the ability of an autonomous agent in learning to take optimal actions/decisions for online malware detection to maximize a reward function while interacting with a stochastic environment.

As we will demonstrate in our work, a cost-sensitive selection model is required that takes into account both detection rate and implementation costs (e.g, latency, area overhead, etc.) of the base ML models and determines the best malware detector at run-time. For the purpose of thorough analysis, six classical and two well-known reinforcement learning algorithms are implemented and their efficiencies are comprehensively analyzed across different evaluation metrics for detecting unknown malware. To the best of our knowledge, this is the first work that addresses major challenges of adaptive, cost-efficient, and on-device unknown malware detection using a limited number of hardware features and introduces a unified and intelligent RL-guided solution to address them all. This research highlights the importance of adaptive scheduling of learning algorithms for flexible and efficient on-device malware detection. Furthermore, its outcome will aid the security researchers and computer architects to implement accurate and low-cost intelligent countermeasures for securing modern computer systems based on users' preferences and available hardware resources.

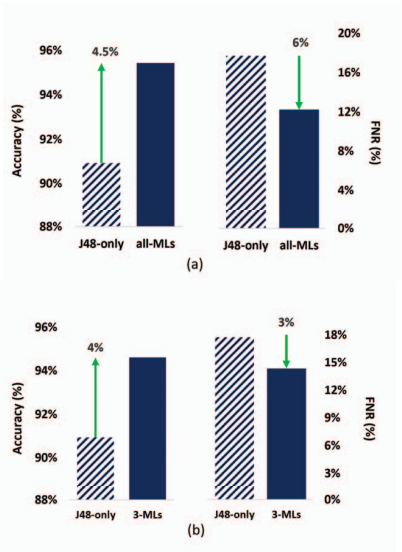

Fig. 2: Performance comparison of single best ML model (e.g, J48) with all tested ML models (case a), and three weak classifiers (case b) for online malware detection using hardware events.

II. Proposed Methodology

In this section, we describe the proposed cost-aware, adaptive RL-guided framework for detecting zero-day malware using optimal hardware events.

A. Motivational Case Studies

Question 1: *Is using the best ML model for on-device HMD enough?* Figure 2-(a) compares using just one of the best ML classifiers (e.g., J48) for unknown malware detection versus the case where multiple ML models are analyzed for adaptive HMD. We observed that the detection accuracy is increased by 4.5% (from 90.9% to 95.4%) when all MLs' efforts are examined. Moreover, as shown, the False Negative Rate (FNR) is decreased by 6% when multiple ML classifiers are examined for online HMD.

Question 2: *Can including weak models be useful in improving the overall accuracy?* Clearly, when each individual ML model is evaluated separately, including weak ML models into analysis seems to be useless due to their lower performance and/or larger overhead. However, our experiments indicate that in adaptive HMD by selecting the most accurate and cost-efficient ML model during run-time, including weak classifiers to our examination could lead into a higher overall

detection rate. To this end, we traced the effect of using one of the best ML models (e.g., J48) versus using three weak ML models including Logistic Regression (LR), Multi-layer Perceptron (MLP), and OneR. As seen in Figure 2-(b), the accuracy is increased by 4% and the FNR is decreased by 3% when including the extra three weak ML models' outcomes compared with only the best ML model.

B. Hardware Features Analysis

Figure 3 shows an overview of *Reinforced-HMD* that is mainly comprised of four components: data acquisition, feature fusion, training of RL system, and inference of RL system for defending against zero-day attacks during run-time. Benign and malware programs are profiled on an Intel Xeon X5550 machine. In order to effectively address the non-determinism and overcounting issues of HPC registers in hardware-based security analysis discussed in recent works [10], [23], we have extracted low-level CPU events available under *Perf* tool using a static performance monitoring approach where we can profile applications several times measuring different events each time. HPC events are monitored using the *Perf* tool with a sampling time of 10ms by running applications in an Linux Containers (LXC) as an isolated profiling environment [24]. LXC is an operating-system-level virtualization technique that allows developers to package and isolate applications with their entire runtime environment and unlike common virtual platforms such as VMWare, provides access to actual performance counters events. We executed more than 5,000 benign and malware applications. Benign applications include real-world applications comprising MiBench [25] and SPEC2006 [26], Linux system programs, browsers, and text editors. Malware applications, collected from and categorized by VirusShare and VirusTotal online repositories which comprise nine types of malware including Worm, Virus, Botnet, Ransomware, Spyware, Adware, Trojan, Rootkit, and Backdoor. After data acquisition, the HPC events are thoroughly analyzed using effective feature analysis methods.

1) Features Selection: As mentioned earlier, feature selection is a critical step of developing effective ML-based malware detectors based on hardware events. Also, counting all possible features would result in high-dimensional data, which increases computational complexity and induces delay. Moreover, including irrelevant features could reduce the performance of classifiers. To address this challenge, as shown in Figure 3 we introduce a unified two-step feature analysis process based on a heterogeneous ensemble feature selection technique that includes different feature selection methods followed by an effective feature fusion method to determine the most prominent HPC events. To explore the effect of different feature selection methods on the disparity of top HPC events, we first implement three feature selection methods in Scikit Learn [27] and record the top four features as a result of each method. The tested feature selection methods in this work are described below:

- *Recursive Feature Elimination (RFE):* RFE is a feature analysis method that fits an ML model and removes the weakest feature(s) till it reaches a specified features number [27]. RFE begins by building a Decision Tree classifier on the entire dataset and computing the weights of each and all features. It then gradually eliminates the less important features till the desired set is found.

- *Mutual Information (MI):* MI is a feature selection technique that calculates the mutual information between the HPC features X and labels Y. It primarily evaluates the amount of information for the target application's class (either malware or benign) gained from each HPC feature in X. Mutual information between features X and label Y are calculated based on the entropy estimation from each point's k-nearest neighbors [28].

- *Sequential Feature Selection (SFS):* Lastly, we employ a sequential feature selection with a backward reduction in Scikit Learn [27], [29]. Backward-SFS is a greedy procedure that starts with all the dataset and greedily removes the less important features till meeting the desired number of features.

Algorithm 1 Feature Fusion Method for Online HMD

Input: Top 8 HPC features's feature importance selected by three Feature Selection (FS) methods
Output: Unified 4 HPCs
Preparation:
- Normalize the feature importance for each FS method to [0,1]
- Merge them to one normalized p_matrix tabular data
- Assign a initial weight matrix, $w_j = 1/8$ for all 8 HPCs

while *run VIKOR* **do**
1. Determine the best and the worst values of all criteria function $max(f_{i,j})$ and $min(f_{i,j})$, i=1,2,...,length of p_matrix, j=1,2,...,8;
2. Compute the Values S_i and R_i, where $S_i = \sum_{j=1}^{n} w_j(f_j^* - f_{ij})/(f^* - f_j^-)$, and $R_i = max_j[w_j(f_j^* - f_{ij})/(f^* - f_j^-)]$;
3. Compute the Values Q_i, where $Q_i = v(S_i - S^*)/(S^- - S^*) + (1-v)(R_i - R^*)/(R^- - R^*)$, $i = 1, 2, ...,$ length of p_matrix, $v = (n+1)/2n, n = 8$;
4. Rank the Alternatives, sorting by the values S, R, and Q in ascending order;
5. Produce a Compromised Solution $A^{(1)}$, which is the best ranked by the measure Q (minimum) if the following two conditions are satisfied: (1) $Q(A^{(2)}) - Q(A^{(1)}) \geq 1/(m-1)$ where $A^{(2)}$ is the second ranked alternative by the measure Q;
(2) The alternative $A^{(1)}$ must also be the best ranked by S and/or R.
end

2) Feature Fusion (FF): Table I reports the top 4 HPC features chosen from the three tested feature selection methods. The result clearly shows the disparity of different selection methods' decisions. As observed, the top four most significant HPCs appear across ten features making the selection of the top four features uncertain. This highlights the importance of presenting a unified feature selection solution that accounts for functionality of different methods to determine the top HPC events for online and cost-efficient HMD.

To this end, *Reinforced-HMD* adopts a novel fusion technique called Multi-Criteria Decision-Making (MCDM) [30] that evaluates the conflicting criteria from several feature selection methods to obtain the final feature subset. MCDM is a powerful tool for making optimal decisions based on multiple criteria, including conflicting ones. In our experiments, given that three feature selection methods chose inconsistent hardware features, MCDM method assists to structure the goal and select a small set of essential features. MCDM has broad applications in many engineering problems with a rich set of algorithms. In our work, we employ a suitable one called VIKOR algorithm as our feature fusion method [31]. Algorithm 1 describes our implementation of the VIKOR feature fusion algorithm that contains five distinct steps and the pipeline of running the aggregated HPCs. The VIKOR method determines a compromise solution that the Q measure

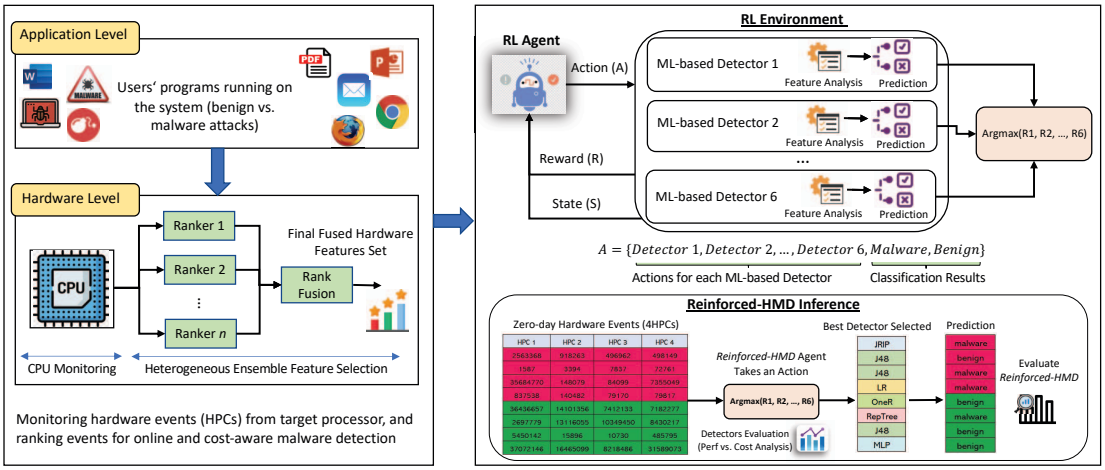

Fig. 3: Overview of *Reinforced-HMD*, a reinforcement learning-guided framework equipped with a novel heterogeneous ensemble feature selection method for adaptive and cost-aware hardware-assisted malware detection.

TABLE I: Disparity of four most significant HPC events across different feature selection methods (MI, RFE, and SFS)

HPCs	bus-cycles	cache-references	node-loads	branch-loads	L1-dcache-loads	LLC-loads	L1-dcache-stores	L1-dcache load misses	iTLB-load-misses	dTLB-store-misses
MI		✓	✓			✓			✓	
RFE	✓		✓		✓			✓		
SFS				✓		✓	✓			✓

provides a minimum of individual regrets for not selecting other features. We sort Q measure for all HPC features in ascending order and select the top four smallest Q measure corresponding HPC features. The final top four features are L1-dcache-loads, node-stores, node-loads, and L1-dcache-stores.

In summary, our feature fusion algorithm includes the following five steps:

Step 1. Determine the best (f_j^*) and the worst (f_j^-) values of all criteria functions. The best value represents the benefit while the worst value represents the cost of selecting the criterion.

Step 2. Compute the Values S_i and R_i. S_i is the summation of the difference between the benefits and the costs among eight features' data points. R_i is the maximum benefits over costs along eight features' data points.

Step 3. Compute the Values Q_i by the relation of S_i and R_i by calculating the majority of the criteria, where $Q_i = v(S_i - S^*)/(S^- - S^*) + (1-v)(R_i - R^*)/(R^- - R^*)$, $i = 1, 2, ...,$ length of p_matrix. S^* is the minimum of the sum of the benefits, S^- is the maximum of the sum of the costs, R^* is the minimum benefits for each criteria, R^- is the maximum cost for each criteria. v is a weight for the strategy of the "majority of criteria", 1-v is the weight of the individual regret.

Step 4. Rank the Alternatives, sorting by the values S (summation of benefits), R (maximum benefits), and Q (the regrets) in ascending order. The results are three ranking lists.

Step 5. Propose a Compromise Solution $A^{(1)}$, which is the best ranked by the measure Q (minimum regrets) if the following two conditions are satisfied:

(1) $Q(A^{(2)}) - Q(A^{(1)}) \geq 1(m-1)$ where $A^{(2)}$ is the second ranked alternative by the measure Q (minimum regrets). The difference between the alternative solution $A^{(2)}$ and proposed solution $A^{(1)}$ is larger than a threshold meaning the alternative solution has a solid higher regrets than the proposed solution;

(2) The alternative $A^{(1)}$ must also be the best ranked by S and/or R so that the suggested solution $A^{(1)}$ shows stability regarding summation of the benefits (S) and the maximum benefits (R) in step 4.

C. Threat Model

Recently developed ML-based malware detection methods have typically examined the effectiveness of their models using two major validation methods, including cross-validation and percentage splits. The cross-validation method splits the dataset into $K(1, ..., n)$ folds and selects one of them as a target testing dataset while the rest folds are used for the training dataset. And in the percentage split method, the dataset is divided into two sections based on the percentage setting allocated to training and the other to the testing set. However, the major issue with these validation techniques is that the testing data is split from the large dataset and is part of the same data type used in the training dataset. Hence, such validation methods could not imitate the zero-day or unknown testing results in real-world applications in which the trained ML classifiers should have never seen the testing dataset.

We use the top four HPCs to generate sub-datasets for training, validating, and testing baseline ML detectors, ensemble methods, and the RL agents. To model the zero-day malware threat type in our experiments, among all nine malware types, we randomly considered four types of malware from rootkit, backdoor, virus, and ransomware as the target zero-day test data. Also, we held 30% of all benign data as a zero-day test benign dataset. We kept both malware and benign aside to imitate the zero-day testing data in real-world applications in which the trained machine learning classifiers should have no knowledge about the new malware types and have never seen the testing dataset. The rest of the five types of malware and benign samples are considered for training and known test purposes, and we randomly split them into 70% for training and 30% for known testing. Notably, our *Reinforced-HMD* framework adopts the same settings for training and testing.

D. Reinforced-HMD: Training and Inference

As depicted in the top right part of Figure 3, the training process of the *Reinforced-HMD* includes the autonomous RL agents and RL environment. The goal of the RL system is to find the best-parameterized reward function such that the RL agent can select the most accurate and cost-effective malware detectors (according to users' preferences and available resources) that lead to high rewards (e.g., high detection rate, low hardware overhead). For base detectors, we examined the suitability of various standard machine learning classifiers including JRIP, J48, OneR, MultiLayer Perceptron (MLP), Logistic Regression (LR), and RepTree classifiers. These models are selected from different branches of machine learning and their detection models can be a binary classification model aligned with the zero-day malware detection task. In addition, the figure shows the inference and evaluation process of the RL system by presenting a case study on feeding tabular HPCs data into the *Reinforced-HMD* for adaptive scheduling of detectors. Algorithm 2 describes the *Reinforced-HMD* training and inference procedures.

Algorithm 2 Process of RL-guided *Reinforced-HMD*

pre-process experience replay data;
initialize state S, RL agent θ, reward policy π;
let $e \leftarrow$ design criteria (F1, AUC, latency, area);
let $d \leftarrow$ selected malware detector;
let $ML \leftarrow$ ML model;
while *training of RL system* **do**
  agent gets state s_t;
  agent predicts $y_{tk}^{-} = ML_t(s_t), k = 1, 2, ..., 6$ ML detectors;
  agent performs action
  $a_t \leftarrow max\{\pi(a_{t1}|s_t, y_{t1}^{-}, e_{t1}), ..., \pi(a_{t6}|s_t, y_{t6}^{-}, e_{t6})\}$,
    action is selected based on detection and cost analysis of all ML detectors;
  agent receives reward r_t and new state s_{t+1};
  update RL agent parameters θ;
  end

while *Inference of RL system* **do**
  for *zero-day test data:* $0 \rightarrow n$ **do**
    agent gets state s_t;
    agent perform action to select a detector d_t;
    agent assigns the best d_t to defend, gets $y_t^{-} = (a_t|s_t, \theta)$;
    if y_t^{-} == y_t *true label* **then**
      increment detection rate;
      record hardware overhead, selected detector d_t;
    else
  end
  calculate RL system detection metrics (F1, Accuracy, AUC) and overhead (latency, area) for all test data.
end

1) RL Environment: We customize the *Reinforced-HMD*'s RL environment based on OpenAI's Gym [32]. OpenAI Gym is an open-source interface that provides RL environments for researchers to develop and benchmark new algorithms. It also offers a structured interface for customizing our RL environment. The *Reinforced-HMD* environment consists of the following four key components and corresponding settings:

• *State* S consists of one set of experience replay data pre-processed inside the RL environment and available to the RL agent during run-time. It contains each row as a set of four HPCs data, the predictions (correct/incorrect predictions) from the six malware detectors, and their detection metrics (F1, AUC) and hardware overheads (latency, area). The state spaces (S) corresponds to six ML-based detectors regarding their experience replay data.

• *Action Spaces* A are a set of all valid actions given to the RL agent to choose from. Our RL environment has six discrete action spaces available to the agent. Each action corresponds to one of the six tested malware detectors. Various actions will result in different consequences for getting a reward or not, and how many rewards the agent can receive. The RL agent is inclined to select the detector d_t that receives the highest rewards among the action space.

• *Reward Policy* π is a rule provided to the RL agent to decide its best action at each step. The *Reinforced-HMD*'s environment provides a deterministic policy that maps $S \overset{R}{\rightarrow} A$, where S is the set of possible states from experience replay data. A is the action space containing all six detectors. The reward policy depends on the design criteria in our case. For the purpose of comprehensive analysis of metrics trade-offs, we define five performance and design criteria ($F1, AUC, AUC/Latency, F1/Area$, and $F1 * AUC/Latency * Area$) as various target measurements to motivate the RL agent to take action. The reward policy based on F1 leads the agent to select the detector with the highest F-measure among all correct-predicted ML models at time step t. The reward policy based on $(F1 * AUC)/(Latency * Area)$ will have a more balanced view to consider selecting the detector with a high detection rate and a low hardware overhead at time step t. The goal of the reward policy is to enable the RL agent to select proper actions to maximize total future rewards for accurate and cost-aware HMD process.

• *Experience Replay Data:* To simulate the interaction between the RL agent and environment, *Reinforced-HMD* uses an algorithm to collect all the possible outcomes for different malware detectors as the experience replay data. The presented pipeline reads each state, goes through each classifier for each state, and records the model detection performance (F1, AUC) and hardware overhead (Latency, Area). Then, we collect all the experience replay dataset for both known-test data (Train RL) and zero-day data (Test RL). As mentioned earlier, we consider three dataset partitions including train, known-test, and zero-day test. The training dataset is used to train and validate the ML detectors. These ML detectors are placed in the branches of the RL environment so that the RL agent can adaptively decide which branch to activate and use for cost-aware malware detection. Furthermore, the unknown-test dataset is a dataset that is set aside during the initial dataset split and is used to train the RL agent. In this setting, we train the RL agent such that each branch of the ML detector has not seen the data before so that the RL agent can learn from the new dataset. Once the RL agent is well-trained, the zero-day test data is used to examine the RL agent on an unseen RL environment. This is because the zero-day test dataset contains new malware types, and both malware and benign data have never been seen by either the ML detectors or the RL agent.

2) RL Agents: In *Reinforced-HMD* implementation, we defined two model-free, deterministic RL agents to interact with the environment. At each step, we trace the selected detector d_t chosen by RL agent. We let the selected ML model defend at each time step to evaluate the HMD performance. The goal is to determine if an adaptive selection of the optimal detector would result in increasing the detection rate while

reducing the resource and computational overheads.

Algorithm 3 Upper Confidence Bound (UCB)

Input: Experience replay data
Output: Agent's actions $A = [a_1, a_2, ..., a_6]$ is a set of detectors
-Set up observation space O, action space A, reward function R;
-Initialize the number of times of detector selected N_j for each detector all as 0; initialize episode length as 20;
-Initialize upper bound value as $1e^5$, let episodes $\tau = \lfloor \frac{length\ of\ training\ data}{episode\ length} \rfloor$;
while *episodes* τ *is not terminal* **do**
  for *step in each episode* **do**
    for *j=1,2,...,6 detectors* **do**
      1) calculate average reward $\bar{r}_j(n) = R_j(n)/N_j(n)$ if N_j is not zero, otherwise use the default value of $1e^5$
      2) calculate delta $\delta_j(n) = \sqrt{\frac{3}{2}\frac{\log(n)}{N_j(n)}}$
      3) update upper bound = $\bar{r}_j(n) + \delta_j(n)$
      4) $a \leftarrow$ detector j having maximum upper bound
      4) return $S^{'}$, r, a to the agent
    end
    Agent takes action a, record accumulated rewards
    Update current state $S \leftarrow S^{'}$.
  end
end

- Upper Confidence Bound (UCB): UCB is a deterministic algorithm that leverages a maximum confidence boundary to decide whether to explore or exploit the RL environment. As shown in Algorithm 3, it firstly assigns a high maximum confidence boundary of $1e^5$ for each branch that the RL agent shall learn from the data and gradually update it for each branch later when explorations progress. Initially, the RL agent randomly selects one of the branches to explore. Depending on how much reward this branch leads to, this branch's corresponding confidence interval is either increased or reduced. In the next round, the RL agent chooses the branch with the highest upper bound to explore and update rewards in the confidence interval.

At the end of round N, the UCB-based RL agent calculates the average reward $r_j(n)$ and $\delta_j(n)$, and updates the upper bound boundary interval $[r_j(n) - \delta_j(n), r_j(n) + \delta_j(n)]$ where $r_j(n)$ is the average reward of all rounds of N, $\delta_j(n)$ is the knowledge gained from the current round. Delta $\delta_j(n)$ encourages more exploration, and the average rewards $r_j(n)$ facilitates exploitation. When the RL agent has little knowledge of the best branch, it favors more explorations to search through all branches. As more rounds of exploration progress, more knowledge of the possible best branch is gained, the RL agent gradually shifts to exploitation to select the branch with the highest upper bound value. UCB greedily trials each detector on each round of exploration and evaluates the possible rewards according to the reward policy. Given different base classifier, UCB algorithm can almost explore the maximum potential rewards among the six malware detectors, with a relatively longer computation time. Since our environment is lightweight, the time used for executing UCB is slightly longer than Q-learning but very trivial. Detailed implementation is shown in Algorithm 3.

- Q-Learning: Q-Learning is an off-policy algorithm to determine the best action given the current state by keeping track of a Q-table that gets updated after each episode with its row corresponding to the state and its column to the action. An episode ends after a set of actions is completed. In the end, the Q-table suggests the optimal policy. We implement Q-learning as shown in Algorithm 4.

Algorithm 4 Q-Learning

(1) **Input:** Experience replay data
(2) **Output:** Agent's actions $A = [a_1, a_2, ..., a_6]$ is a set of detectors
- Set up observation space O, action space A, reward function R;
  - Initialize Q table $Q[s_i, a_j] \leftarrow 0$ for 20 steps/episode for 6 detectors, with the initial value N_j, the number of detector selected for each detector as 0;
  - Initialize learning rate α, discount factor γ, epsilon ϵ; let episodes $\tau = \lfloor \frac{length\ of\ training\ data}{episode\ length} \rfloor$;
  while *episodes* τ *is not terminal* **do**
(3) **for** *step in each episode* **do**
      Generate a random number $X \sim [0, 1]$;
      if $X > \epsilon$ **then**
        $a \leftarrow max(Q(s_i, a_j), j = 1, 2, ...6)$;
      end
      else
        $a \leftarrow X$;
      end
(4) Agent takes action a, record accumulated rewards;
(5) Update current state $S \leftarrow S^{'}$;
(6) Update Q table:
      $Q(s, a) \leftarrow Q(s, a) + \alpha[R + \gamma \cdot \max Q(s_{t+1}, a_{t+1}) - Q(s, a)]$;
  end
end

As shown in Algorithm 4, the RL agent deploys Q-table to track the maximum reward for the action it takes at each state. Initially, the Q value for each of the six branches is zero. The RL agent randomly selects a branch to explore. The parameter of epsilon ϵ helps the agent to decide on whether to explore or to exploit. The initial value for epsilon ϵ is a small value between [0,1]. In experiments, we found that 0.5 is the best value. The Q-function uses the Bellman equation as shown in line 6 of Algorithm 4 to update the Q-value at each step.

The Q value considers both the current reward R and the discounted future rewards. Current rewards are more important than future rewards when the RL agent makes a decision. This is an iterative process until all training data are explored. As the RL agent starts to explore the environment, the Q-function gives optimal approximations by continuously updating the Q-values in the table. We initially set the learning rate alpha α, the discount factor gamma γ, and epsilon each as a small value of [0,1], and we train the RL agent to learn the most optimal values of them. Our final alpha α is 0.35, gamma γ is 0.3, epsilon ϵ is 0.5.

3) Online Inference: We evaluated *Reinforced-HMD* on 1) sum of rewards, 2) RL system's malware detection accuracy, 3) RL system's detection and cost-effectiveness. Figure 3 right bottom shows a case study for inference and evaluation process. As shown, first each row of the four zero-day tests HPCs (state s_t) is fed into the RL environment. The RL agent loads all six ML-based detectors in different branches to run predictions. Among all six models, any ML which predicts the sample HPC correctly is eligible to receive a reward from the environment. If all ML models predict wrongly, the system assigns the least-costly ML (in terms of latency and hardware overhead) as the default detector with a reward granted.

In many situations, multiple ML models perform a correct prediction in which the RL system will evaluate detection and cost analysis for all six ML models to select the detector with the highest reward. The selected detector is then recorded one by one for all test data until the adaptive branching selection through the RL system is complete. Lastly, to evaluate the

effectiveness of the RL system, we use the recorded selection of malware detector at each time step to reproduce the defending process to obtain the result. We first load the ML model selected by the RL system at each time step and feed with the same row of four HPCs data to let each model run prediction. We record each prediction for all test data and calculate the RL system's evaluation metrics.

III. Evaluation Results

This section presents a detailed analysis of experimental results on the effectiveness of proposed RL-guided framework for adaptive and on-device zero-day malware detection.

A. Base ML-based Malware Detectors

Table II shows the base ML-based malware detectors' detection metrics and hardware overheads. Given the importance of analyzing hardware overhead of ML classifiers used for efficient on-device malware detection, we develop the ML-based malware detectors at the hardware level and analyze their associated area and latency overheads. We used Vivado HLS compiler to develop the HDL implementation of the classifiers on Xilinx Virtex 7 FPGA. The latency is considered as the number of clock cycles (cycles @10 ns) and the area overhead is calculated using the total number of utilized LUTs, FFs, and DSP units in the FPGA. We evaluate ML models in two ways. Firstly, we benchmark the RL-based method with ML-based and ensemble methods. Secondly, the six pre-trained MLs are available to the RL agents to select one ML at each step that returns the highest reward. This metric shows that the J48 decision tree can achieve the highest detection performance, 87% in F1-score and 89% in AUC, respectively. The results highlights the cost-inefficiency of MLP and LR algorithms with highest latency and area overhead among others. However, regarding latency, OneR is the fastest with 1 ms outperforming J48 (3 ms). Also, regarding the area, JRIP classifier outperforms both OneR and J48 classifiers.

TABLE II: Base classifiers' detection rate and overhead for unknown HMD.

ML Models	Accuracy	F1-score	AUC	Latency (ms)	Area
JRIP	0.91	0.86	0.89	2	156
J48	0.91	0.87	0.89	3	584
Logistic Reg.	0.62	0.14	0.50	59	11815
MLP	0.62	0.14	0.51	102	25667
OneR	0.70	0.46	0.63	1	292
RepTree	0.90	0.86	0.88	3	377

B. Reinforced-HMD Evaluation

1) Learning Performance: Figure 4 illustrates the training speed in episode rewards (left) and the sum of rewards (right) for the two RL agents UCB and Q-learning used in *Reinforced-HMD*, over the zero-test test episodes for the design criteria of $F1 * AUC/Latency * Area$. Since UCB is a greedy algorithm that exhaustively searches for the best-optimal detector at every step, it learns faster giving the most sum of rewards. Whereas, Q-learning is trained with a set of parameters, and the agent takes more episodes to explore. As shown, UCB is a more stable decision-maker to be used in *Reinforced-HMD* for selecting the optimal detector along with all episodes.

2) Performance vs. Cost-Efficiency Analysis against State-of-the-Art ML Models: Table III reports the detection performance and hardware overhead improvement of using our presented RL-guided framework as an adaptive and cost-aware model as compared with using the best classical ML model (J48) and widely used ensemble models. As seen, RL-UCB has a nearly perfect capability (99.65%) to select the most optimal malware detector that helps the RL system reaches a 96% detection rate in both F1-score and AUC, which is 1% higher than majority voting and a 9% up than J48. Since the RL system considers a balanced strategy across detection metrics and low hardware overhead, RL-UCB has much less hardware latency, 12 times lower than ensemble-based majority voting, and 44% lower than the light-weight J48 algorithm. Meantime, UCB consumes 36 times less hardware area than majority voting, 2.34 times less than J48.

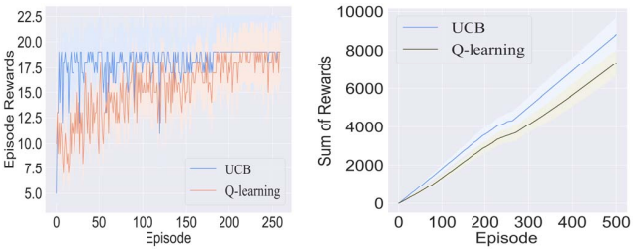

Fig. 4: Learning speed (left) and Sum of Rewards (right) of the two RL agents: UCB (blue) and Q-learning (orange). Both analysis are based on the design criteria of (F1*AUC)/(Latency*Area).

TABLE III: Analysis of the proposed RL-guided techniques with the best detector (J48) and ensemble-based methods for adaptive and cost-aware zero-day malware detection strategy.

Approach	Selection Probability	RL System F1-score	RL System AUC	HW Latency (ms)	HW Area
J48	N/A	0.87	0.89	3	584
Majority Voting	51%	0.95	0.95	28	6482
Stacking	25%	0.91	0.91	28	6482
RL-UCB	99.65%	0.96	0.96	2.08	175
RL-Q-learning	89%	0.93	0.93	6.1	1190

To accordingly account for malware detection rate and area overhead impact, in Figure 5 we show the trade-off analysis between the five tested ML-based detectors regarding their detection rate (F1-score) vs. models' hardware overhead (area). We use F1-score over area to determine the models that require small area and yet can detect the maliciousness of program with high performance. A classifier with a higher value in F1 and low value in area is considered more effective. As observed, an RL-based UCB is the best balanced solution with a high F1-score (96%) while occupying a minimal area outperforming other models. The majority voting has a high F1-score (95%) but with an increased area that makes the solution impractical for efficient on-device malware detection. Also, stacking method is the least optimal selector with 91% F1-score while consuming a high area on the host hardware.

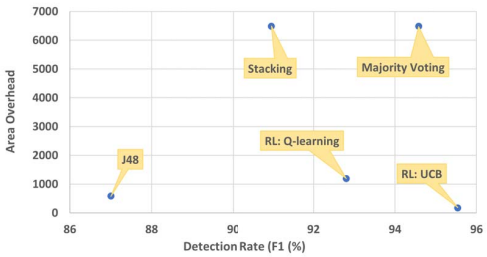

Fig. 5: Detection rate vs. area overhead comparison of RL (UCB, Q-learning), ensemble (majority voting, stacking) and best ML model (J48).

Figure 6 depicts the relationship between all methods' selection probability of the optimal detector versus the cost-efficiency measured across two design criteria of $F1/AUC$

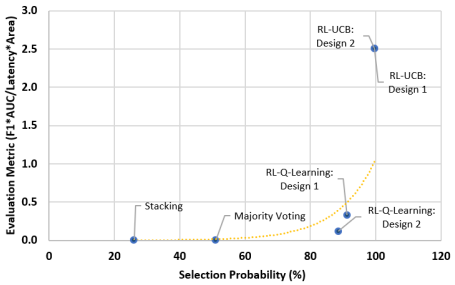

Fig. 6: The relationship between all RL and ensemble methods' selection probability of the optimal detector vs. system efficiency with two design criteria: F1/Area (Design 1) and (F1*AUC)/(Latency*Area) (Design 2).

and $(F1 * AUC)/(Latency * Area)$. The results show that the *Reinforced-HMD* based on UCB agent leads to a higher selection probability for both designs in choosing the most optimal malware detectors during run-time which leads to a higher cost-efficiency. Overall, UCB outperforms Q-learning by 11% in selection probability, which results in an increment of 20% efficiency measured in $(F1*AUC)/(Latency*Area)$. Moreover, RL-based UCB outperforms majority voting by nearly 49% to select the most optimal malware detector at run-time while delivering a five times higher cost-efficiency.

IV. Conclusion

Prior studies on Hardware-Assisted Malware Detection (HMD) have not presented a flexible and balanced solution that accounts for the detection performance and cost-efficiency trade-off analysis for accurate yet low-cost online malware detection. In this work, for the first time we have highlighted and tackled major issues with adaptive and cost-aware zero-day malware detection using low-level hardware events. We first propose a unified feature selection method based on a heterogeneous feature fusion technique to effectively determine the most prominent HPC events for on-device HMD. We further present *Reniforced-HMD*, a novel reinforcement learning-guided framework for adaptive and cost-aware hardware-assisted zero-day malware detection using an optimal number of hardware features. The results indicate that our novel framework obtains a superior detection performance (96% in both F1-score and AUC) for recognizing unknown malware using a limited number of hardware events facilitating an accurate, flexible, and low-cost on-device HMD.

V. Acknowledgment

This work is supported by the National Science Foundation under Award No. 2139034.

References

[1] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19.

[2] M. Lipp *et al.*, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, Aug. 2018, pp. 973–990.

[3] J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," in *ISCA'13*. ACM, 2013, pp. 559–570.

[4] A. Bettany *et al.*, "What is malware?" in *Windows Virus and Malware Troubleshooting*. Springer, 2017, pp. 1–8.

[5] H. Sayadi *et al.*, "Recent advancements in microarchitectural security: Review of machine learning countermeasures," in *MWSCAS'20*, 2020, pp. 949–952.

[6] A. Tang *et al.*, "Unsupervised anomaly-based malware detection using hardware features," in *RAID'14*. Springer, 2014, pp. 109–129.

[7] H. Sayadi *et al.*, "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in *Design Automation Conference (DAC'18)*, 2018, pp. 1–6.

[8] M. Ozsoy *et al.*, "Malware-aware processors: A framework for efficient online malware detection," in *HPCA'15*, 2015, pp. 651–661.

[9] H. Sayadi *et al.*, "2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection," in *Design, Automation Test in Europe Conference Exhibition (DATE'19)*, March 2019, pp. 728–733.

[10] B. Zhou *et al.*, "Hardware performance counters can detect malware: Myth or fact?" in *ASIACCS'18*, 2018, pp. 457–468.

[11] K. Basu, P. Krishnamurthy, F. Khorrami, and R. Karri, "A theoretical study of hardware performance counters-based malware detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 512–525, 2019.

[12] H. Sayadi *et al.*, "Stealthminer: Specialized time series machine learning for run-time stealthy malware detection based on microarchitectural features," in *GLSVLSI'20*, 2020, p. 175–180.

[13] S. M. P. Dinakarrao *et al.*, "Cognitive and scalable technique for securing iot networks against malware epidemics," *IEEE Access*, vol. 8, pp. 138 508–138 528, 2020.

[14] B. Singh *et al.*, "On the detection of kernel-level rootkits using hardware performance counters," in *ASIACCS'17*, 2017, pp. 483–493.

[15] H. M. Makrani *et al.*, "Adaptive performance modeling of data-intensive workloads for resource provisioning in virtualized environment," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 5, no. 4, pp. 1–24, 2021.

[16] H. Sayadi and H. Homayoun, "Scheduling multithreaded applications onto heterogeneous composite cores architecture," in *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*. IEEE, 2017, pp. 1–8.

[17] H. Wang *et al.*, "Hybrid-shield: Accurate and efficient cross-layer countermeasure for run-time detection and mitigation of cache-based side-channel attacks," in *ICCAD'20*, 2020.

[18] A. P. Kuruvila *et al.*, "Analyzing the efficiency of machine learning classifiers in hardware-based malware detectors," in *ISVLSI'20*. IEEE, 2020, pp. 452–457.

[19] K. N. Khasawneh *et al.*, "Ensemble learning for low-level hardware-supported malware detection," in *RAID'15*, 2015, pp. 3–25.

[20] L. Bilge and T. Dumitras, "Before we knew it: An empirical study of zero-day attacks in the real world," in *CCS'12*. ACM, 2012, p. 833–844.

[21] V. FrancoisLavet *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[22] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[23] S. Das *et al.*, "Sok: The challenges, pitfalls, and perils of using hardware performance counters for security," in *IEEE SP*, 2019, pp. 20–38.

[24] M. Helsely, "Lxc: Linux container tools," in *IBM developer works technical library*, 2009.

[25] M. R. Guthaus *et al.*, "Mibench: A free, commercially representative embedded benchmark suite," in *IISWC'01*, Dec 2001, pp. 3–14.

[26] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[27] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] B. C. Ross, "Mutual information between discrete and continuous data sets," *PLoS ONE*, vol. 9, 2014.

[29] F. Ferri *et al.*, "Comparative study of techniques for large-scale feature selection," in *Pattern Recognition in Practice IV*, ser. Machine Intelligence and Pattern Recognition, E. S. Gelsema *et al.*, Eds. North-Holland, 1994, vol. 16, pp. 403–413.

[30] A. Hashemi *et al.*, "Mfs-mcdm: Multi-label feature selection using multi-criteria decision making," *Knowledge-Based Systems*, vol. 206, p. 106365, 08 2020.

[31] J. P. N. Papathanasiou, *Multiple Criteria Decision Aid: methods, examples and python implementations*. Springer, 2019.

[32] G. Brockman *et al.*, "Openai gym," 2016.