Journal of Hardware and Systems Security
https://doi.org/10.1007/s41635-024-00146-6

Guarding Against the Unknown: Deep Transfer Learning for Hardware Image-Based Malware Detection

Zhangying He[1] · Houman Homayoun[2] · Hossein Sayadi[1]

Received: 11 May 2023 / Accepted: 21 February 2024

Abstract
Malware is increasingly becoming a significant threat to computing systems, and detecting zero-day (unknown) malware is crucial to ensure the security of modern systems. These attacks exploit software security vulnerabilities that are not documented or known in the detection mechanism's database, making it particularly a pressing challenge to address. In recent times, there has been a shift in focus by security researchers toward the architecture of underlying processors. They have suggested implementing hardware-based malware detection (HMD) countermeasures to address the shortcomings of software-based detection methods. HMD techniques involve applying standard machine learning (ML) algorithms to low-level events of processors that are gathered from hardware performance counter (HPC) registers. While these techniques have shown promising results for detecting known malware, accurately recognizing zero-day malware remains an unsolved issue in the existing HPC-based detection methods. Our comprehensive analysis has revealed that standard ML classifiers are ineffective in identifying zero-day malware traces using HPC events. In response, we propose *Deep-HMD*, a multi-level intelligent and flexible approach based on deep neural network and transfer learning, for accurate zero-day malware detection using image-based hardware events. *Deep-HMD* first converts HPC-based malware and benign data into images, and subsequently employs a lightweight deep transfer learning methodology to obtain a high malware detection performance for both known and unknown test scenarios. To conduct a thorough analysis, three deep learning-based and nine standard ML algorithms are implemented and evaluated for hardware-based malware detection. The experimental results indicate that our proposed image-based malware detection solution achieves superior performance compared to all other methods, with a 97% detection performance (measured by F-measure and area under the curve) for run-time zero-day malware detection utilizing soley the top four performance counter events. Specifically, our novel approach outperforms the binarized MLP by 16% and the best classical ML algorithm by 18% in F-measure, while maintaining a minimal false positive rate and without incurring any hardware redesign overhead.

Keywords Deep learning · Hardware-based malware detection · Transfer learning · Zero-day malware · Explainable machine learning

1 Introduction

For the past few decades, cybersecurity has been a significant concern worldwide due to its potential threat to information technology infrastructures. Malware, also known as malicious software, is a type of cyber-attack that often comes bundled with legitimate programs to deceive unwary users [1, 2]. Malware detection is a critical aspect of cybersecurity that helps organizations protect users' sensitive data, prevent system damage, and maintain business continuity. In recent years, there has been a growing concern regarding the limitations and performance issues associated with traditional software-based malware detection methods. Although traditional detection methods, such as anti-virus software, are capable of detecting known malware signatures, they are not very efficient in terms of complexity and computational overhead for the system [3, 4]. Updating these mechanisms with new malware variants requires a considerable amount of memory and hardware resources, making them less practical for emerging

✉ Hossein Sayadi
hossein.sayadi@csulb.edu

[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

Published online: 15 March 2024

computing platforms such as resource-constrained embedded systems and IoT devices. Additionally, these methods rely on static signature analysis, making them inadequate for detecting unknown attacks [5]. As the prevalence of malware increases, it is increasingly important to develop effective malware detection strategies as they serve as an early warning system to safeguard modern computer systems.

In recent studies, hardware-based malware detection (HMD) techniques have emerged as a promising alternative to overcome the limitations and performance issues associated with traditional software-based detection methods [2–4, 6–9]. These techniques utilize low-level hardware events, which are monitored through hardware performance counters (HPCs) registers. HPCs are the specialized registers present in the performance monitoring unit (PMU) of modern microprocessors that are designed to collect hardware events of running applications [10, 11]. HMD techniques can operate independently of the underlying operating system when it comes to capturing low-level events, making them more difficult for attackers to bypass detection measures. These events, such as cache memory hits/missess, branch mispredictions, TLB hits, etc. are hardware-level features that reflect the actual performance-related behavior of a running application on the target processor architecture. In addition, hardware-based malware detection techniques leverage standard machine learning (ML) algorithms that are trained on HPC events to create precise classifiers that can detect signatures of malicious software. Previous research in HMD has demonstrated the effectiveness of standard ML techniques in detecting known malware patterns. However, in this work, we have identified some significant challenges in existing HPC-based malware detection methods and propose a deep learning-guided approach to achieve accurate hardware-based detection of zero-day malware.

Challenge 1: Determining Key Hardware Events To ensure efficient hardware-based malware detection, it is crucial to identify the most significant low-level events. With modern microprocessors, there are a plethora of events, each serving a distinct function. This makes monitoring all of them result in high-dimensional data, which leads to increased computational complexity and delays. As a result, it is less feasible for effective HMD solutions. Therefore, as different HPC events are employed for various purposes, it is crucial to effectively specify the most appropriate hardware events to develop accurate ML-based countermeasures for malware detection.

Challenge 2: Detection of Zero-Day (Unknown) Malware Zero-day attacks take advantage of software security vulnerabilities that are potentially severe and undocumented in the detection mechanism's database [12]. The absence of signature history or a clear remediation strategy has made zero-day malware detection a persistent challenge for anomaly detection in securing modern computer systems in these works [13, 14]. Therefore, they need to be resolved as soon as being discovered in order to limit the security threats to the users. Current machine learning-based detection methods have overlooked the complex issue of zero-day malware detection, which makes them inherently inflexible and unscalable. As a result, incorporating new malware types would necessitate training new models, resulting in reduced efficiency and applicability of the solution.

Challenge 3: High False Positive Rate Malware detection methods based on machine learning, in their conventional form, have a high false positive rate, where benign applications are mistakenly classified as malware. This problem becomes even more critical when it comes to detecting unknown malware, as ML models often confuse benign software with malicious software. Our experiments with various standard ML algorithms reveal that existing HMDs wrongly detect benign applications as malware during zero-day tests with a significantly high false positive rate on average across different algorithms. Consequently, this challenge disrupts the accuracy and reliability of security countermeasures against emerging cyber-attacks that require to be addressed urgently.

In response to these challenges, we propose *Deep-HMD*, an accurate and salable deep neural network-based approach for effective known and unknown (zero-day) malware detection using processors' hardware events. In particular, after analyzing various types of malware and machine learning algorithms employed in HPC-based malware detection, our comprehensive examination demonstrates that the standard machine learning classifiers, which were widely utilized in previous studies, are ineffective in accurately recognizing the signature of zero-day malware with high detection performance and low false positive rates. The results demonstrate a significant performance drop in standard ML classifiers applied for hardware-based zero-day malware detection.

To address the limitations of existing ML-based malware solutions, we first determine the most notable hardware performance counter events for accurate HPC-based malware detection using an effective feature selection technique based on the mutual information (MI) method. Next, we present a novel deep transfer learning approach for hardware image-based malware detection. By utilizing deep neural network (DNN) and transfer learning, we achieve superior detection rate and effectiveness in detecting both known and previously unknown malware patterns. *Deep-HMD* first transforms HPC-based malware and benign data to images, and then leverages a lightweight deep transfer learning approach to attain a high malware detection performance for both known and unknown tests.

Contributions This work presents a substantial extension of the recent research [5], incorporating the principal contributions outlined below.

- We demonstrate the limitations (low detection rate and high false positive) of existing machine learning classifiers in defending against unknown attacks for hardware-based malware detection.
- We propose *Deep-HMD*, a multi-level deep neural network-based approach with transfer learning aid for accurate hardware-based known and unknown malware detection that first converts HPC-based malware and benign data to images using an effective 2D embedding image features conversion method.
- Next, *Deep-HMD* leverages a lightweight deep transfer learning approach to obtain a high malware detection performance despite using a small number of hardware events captured at run-time by existing hardware performance counter registers.
- To conduct a comprehensive analysis of known and unknown malware detection using hardware events, we implemented three deep learning-based and nine classical ML-based algorithms. In addition to *Deep-HMD*, we implemented a binarized neural network with a multi-layer perceptron (MLP) algorithm for image-based and a deep learning method for tabular HPC data, respectively, and compared their performance with *Deep-HMD*.
- To shed light on the explainability of the model, activation feature maps from different layers of *Deep-HMD* were properly visualized. These maps showcase the model's ability to accurately detect zero-day malware by observing the feature differences between malware and benign data. Furthermore, a reliability analysis was conducted to verify *Deep-HMD*'s robustness for predicting over medium and high uncertainty test data.
- *Deep-HMD* stands out as the first DNN-based methodology for accurate hardware image-based malware (known and zero-day) detection. It enables a lightweight and efficient transfer learning strategy on HPC-based data of new malware types in an image format, making it extensible and generalizable, reinforcing its effectiveness against emerging malware attacks.

The remainder of this paper is organized as follows. Section 2 presents an overview of related work and background on the topic of hardware-based malware detection using ML techniques and explainable ML. Section 3 introduces the details of the proposed methodology. Section 4 presents the evaluation metrics and experimental results analysis. Lastly, Section 5 concludes this study.

2 Background on Hardware Malware Detection

Hardware Performance Counters Modern microprocessors have hierarchical cache subsystems, processor pipelines, simultaneous multithreading, and out-of-order execution units, which significantly impact their performance. Modern microprocessors such as Intel, ARM, and AMD microprocessors have a performance monitoring module accessible through programmable hardware performance counters.

HPCs are versatile in their ability to record an array of low-level events that can have a significant impact on a processor's performance. These events include cache memory accesses and cache misses, translation lookaside buffer (TLB) hits and misses, branch mispredictions, and many more. By monitoring these events, developers can make informed decisions and fine-tune their software to achieve optimization goals, whether that be boosting performance, improving energy efficiency, or enhancing security.

HPCs vary in their availability across different processor platforms. For example, in Intel Ivy-bridge and Intel Broadwell CPUs, there are limitations on the number of counter registers, with only four available per processor core. This means that only four HPCs can be simultaneously captured on these architectures. Intel SandyBridge and Haswell architectures, on the other hand, offer a more generous allocation of eight general-purpose counters per core, providing greater flexibility for performance monitoring and optimization.

One of the notable features of HPCs is their ability to issue interrupts when a counter overflows. Additionally, these counters can be configured to start counting from a specific desired value, offering fine-grained control over the monitoring process. This level of programmability empowers developers to tailor their performance monitoring strategies to the precise needs of their applications, ensuring that they can extract the most relevant and valuable insights from the microprocessor's operation.

Hardware-Based Malware Detection Demme et al. [2] proposed the suitability of machine learning techniques applied to performance counter events for malware detection. The authors presented to use HPC data to detect malicious behavior patterns by developing machine learning techniques primarily on mobile operating systems such as Android. Tang et al. [4] discussed the feasibility of unsupervised learning using low-level HPCs features to detect specific attacks, while Ozsoy et al. [7] used sub-semantic features and learning algorithms to detect malware, suggesting changes in the microprocessor pipeline for real-time detection, which increased the overhead and complexity. Singh

et al. [6] developed an HMD method using ML algorithms trained on synthetic traces of hardware features to detect kernel rootkits. However, the work only focused on a limited set of synthetic datasets.

Recent works [3, 8] have highlighted that the number of HPC registers available is limited due to physical and cost constraints, which in turn limits the number of events that can be counted simultaneously. Therefore, they proposed effective ensemble learning and boosting techniques on weak standard ML classifiers to improve the performance of HMD by accounting for the impact of reducing the number of HPC features on the performance of ML-based malware detectors. Sayadi et al. [3] developed eight ML models and two ensemble classifiers (AdaBoost and Bagging), and compared them across various metrics including accuracy, robustness, and hardware overhead. Results showed that the proposed ensemble learning malware detection with just 2 HPCs outperformed standard classifiers with 8 HPCs by up to 17%, matching the robustness and performance of standard detectors with 16 HPCs while using only 4 events, enabling effective run-time hardware-assisted malware detection. Furthermore, the work in [8] presented 2SMaRT, a two-stage ML-based approach for specialized run-time malware detection. In the first stage, applications are classified into benign or specific malware classes using a multiclass technique. In the second stage, the authors optimize detection by employing tailored machine learning models for each malware class and enhance overall performance through effective ensemble learning. Some recent works [11, 15] addressed challenges of detecting advanced malware attacks such as modern morphic and stealthy (embedded) malware that is hidden within benign programs and proposed optimized traditional machine learning and deep learning-based techniques trained on HPC events to improve malware detection performance.

Performance Monitoring Tools To monitor application behavior and gather hardware-related events crucial for analyzing and optimizing application performance, previous studies have employed a variety of performance monitoring tools. These tools include Perf [16, 17], Pin [18], PAPI [19], Intel VTune [20], and Intel PCM [21]. All these tools are available for Linux systems while only Intel VTune and Intel PCM can monitor HPCs in Windows and macOS systems. Perf, PAPI, and Pin demand some knowledge of command lines for users due to the lack of a GUI interface. Perf tool is a Linux-based low-level performance monitoring tool that can instrument CPU performance counters, tracepoints, kprobes, and uprobes (dynamic tracing) [17]. Its monitoring granularity can be scaled to as fine as 10*ms* without the need for customization. The Pin tool is capable of capturing a wide range of program-specific ISA-dependent features, including instruction mix, instruction-level parallelism, register traffic, branch predictability, and more, enabling a thorough examination of application behavior [18].

Some hardware vendors provide their own proprietary tools or libraries for performance monitoring on different OSs. These tools are often tailored to their hardware and can be used to extract hardware events. For instance, macOS users can employ instruments, and Windows users have access to performance monitor. Furthermore, the Performance Application Programming Interface (PAPI) [19] is a cross-platform interface designed for monitoring hardware performance counters on processors equipped with specific registers for hardware events. PAPI is designed as a cross-platform interface, ensuring that it remains independent of any particular operating system. PAPI is intended to work on a variety of operating systems, including Linux, macOS, and Windows. This compatibility makes it a valuable tool for performance monitoring and analysis in diverse computing environments. Users can employ PAPI on the OS that best suits their specific requirements and system configuration. Furthermore, for the purpose of identifying and addressing performance bottlenecks in running programs, as well as for fine-tuning and debugging, Intel offers a licensed tool known as Vtune [20]. Vtune can efficiently record and display performance-related information, featuring a robust graphical user interface and comprehensive profiling capabilities, including HPC monitoring, call graphs, performance bottleneck analysis, and hotspot detection.

Additionally, Intel's Performance Counter Monitor (PCM) [21, 22] is a powerful performance monitoring unit implemented in Intel's processors, such as Xeon, Atom, and Xeon Phi. PCM enables the monitoring of performance and energy-related metrics in both Windows and Linux environments. Notably, PCM sets itself apart by supporting real-time monitoring of both core and uncore events, distinguishing it from tools like Perf and PAPI. Such cross-platform performance monitoring tools aim to work on multiple OSs often providing a degree of hardware event extraction on non-Linux operating systems. This highlights the adaptability and versatility of these tools, ensuring that performance analysis and optimization can be carried out effectively across a variety of computing environments. Additionally, in situations where hardware event monitoring is critical, virtualization or containerization solutions may be employed to run a Linux-based environment on top of the non-Linux OS. This allows the use of Linux-based performance analysis tools within a contained environment. Overall, it is important to note that with careful consideration of system-agnostic features and adaptable model architectures, hardware malware detection techniques hold the promise of being generalizable to other systems and operating systems with minimal modification, thus enhancing the broader cybersecurity landscape.

3 Proposed Methodology

This section presents the proposed multilayer deep transfer learning-based approach for accurate hardware image-based malware detection.

3.1 Experimental Setup

In our experiments, the benign and malware programs are profiled on an Intel Xeon X5550 machine. To effectively address the non-determinism and overcounting issues of HPC registers in hardware-based security analysis discussed in recent works [1, 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 with a sampling time of 10ms within Linux Containers (LXC) as an isolated profiling environment. More than 5000 benign and malware applications are executed for data acquisition. Benign applications include real-world applications comprising MiBench [24] and SPEC2006 [25], Linux system programs, browsers, and text editors. Malware applications, collected from and categorized by VirusShare and VirusTotal online repositories, comprise nine types of malware including worms, viruses, botnets, ransomware, spyware, adware, trojan, rootkit, and backdoor. Leveraging Linux containers in our experimental setup is advantageous because, unlike typical virtualization platforms like VMWare or VirtualBox, they offer direct access to real hardware performance counters data rather than emulating performance counter events. It is important to note that running malware within the container can potentially contaminate the environment, which may impact subsequent data collection. To mitigate this risk and ensure that collected data is not tainted by previous runs, the container is destroyed after each run. In total, we collected 16K samples of malware and 30K samples of benign applications. Among all malware, there are 7217 trojans, 2606 viruses, 1821 ransomware, 1787 spyware, 1588 botnets, 748 worms, 591 backdoors, 242 rootkits, and 229 adware samples. Table 1 reports a subset of sixteen deployed low-level features captured by HPC registers from the Perf tool under Linux in our experiments and their descriptions.

3.2 Feature Engineering

As highlighted before, feature analysis and selection (e.g., analyzing the importance of the hardware events) is an important step in developing accurate ML models for hardware-based malware detection. To address the *Challenge 1* of existing HMD methods, we first employ the mutual information (MI) method in information theory to analyze and rank the importance of the HPC events to the target label Y (benign/malware). Then, we analyze the top 16 features' correlations to each other to filter out the redundant features to reduce training costs and improve model performance. At last, we select the most prominent four features among the rest top-ranked features to fit the constraints of the number of available HPCs during run-time.

Table 1 Top 16 HPC features collected from Perf tool in our experiments and their descriptions

HPC event	Description
LLC-loads	Number of successful memory load operations that accessed the Last-Level Cache (L3 cache)
L1-dcache-load-misses	Number of cache lines brought into the Level 1 data cache due to cache misses
node-loads	Number of successful load operations to the DRAM, representing main memory access
mem-stores	Number of successful store operations in main memory, indicating data writes to main memory
dTLB-stores	Number of TLB lookups for data memory store operations, which are address translation requests for data writes
cpu/branch-instructions/	Number of branch instructions executed by the CPU
dTLB-loads	Number of TLB lookups for data memory load operations, indicating address translation requests for data reads
cpu/cache-references/	Number of references made to the CPU's caches, including cache hits and misses
cpu/branch-misses/	Number of branch instructions that were mispredicted by the CPU
cpu/instructions/	Number of instructions retired by the CPU, representing the total executed instructions
cache-references	Number of references made to the last level cache (LLC), which includes both cache hits and misses
instructions	Number of instructions retired by the CPU, representing the total executed instructions
branch-instructions	Number of branch instructions executed by the processor
branch-loads	Number of successful branch instructions, indicating branches that were taken
branch-misses	Number of all retired mispredicted branches, signifying branch prediction failures
msr/tsc/	Number of time-stamp counter (TSC)

3.2.1 Feature Importance Analysis

Mutual information (MI) measures the dependency between two variables. Regarding features X and label Y, the MI measure $I(X, Y)$ is obtained by estimating the marginal entropies $H(X)$, $H(Y)$, and the joint entropy $H(X, Y)$ as follows:

$$I(X, Y) = H(X) + H(Y) - H(X, Y) \quad (1)$$

For each data point i, the MI method computes I^i based on its neighboring data points. It first finds the k-closest neighbors falling inside of the distance to point i. Using $\psi(\cdot)$ as the digamma function, N is the total samples, Nx_i is the data sample falling within the distance d with k neighbors, and m_i is the total number of neighbors in the dataset. The estimated MI is defined as below:

$$I^i = \psi(N) - \psi(Nx_i) + \psi(k) - \psi(m_i) \quad (2)$$

In this work, we use MI to estimate the dependency between each HPC and the target label Y, which measures each HPC's contribution to predict label Y. To this purpose, We use Scikit Learn library's *mutual_info_classif* algorithm [26] to estimate MI from k-nearest neighbor statistics [27] to obtain the feature importance of each HPC to target label Y. Its output is the feature importance regarding each HPC to Y. We rank it and output a list of the top 16 features that contribute the most to distinguishing Y between malware and benign.

3.2.2 Feature Correlations Analysis

Our next step in the Feature Engineering process involves a thorough analysis of feature correlations, aiming to elucidate the relationships between pairs of hardware performance counter features. In instances where redundancy becomes apparent, we take action to eliminate such features from consideration within our feature list. Among the top 16 features, we employed Panda's Pearson's correlation coefficient method [28, 29] to conduct a comprehensive correlation analysis, which measures the linear relationship between each pair of HPC features for all data. We dropped the less significant HPC features that displayed a correlation coefficient exceeding a threshold of 0.9 with another HPC feature. Figure 1 shows the heatmap of the HPC features and their correlations. Notably, a vivid red color (excluding the diagonal) on the heatmap signifies the presence of redundant HPC features. For example, "dTLB-loads" was identified as redundant with respect to "dTLB-stores" and given its lesser contribution to the target label Y in comparison to "dTLB-stores," we opted to remove "dTLB-loads." Similarly, "cache-references" was deemed redundant in relation to "node-loads" and was considered less significant, leading to its exclusion from consideration.

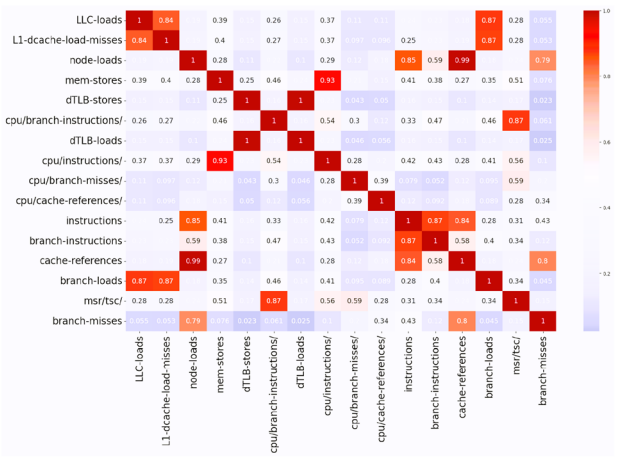

Fig. 1 Top 16 HPC features' correlations heatmap

3.2.3 Feature Selection

As the last step, among the top prioritized features (with horizontal and vertical axes showing the same sets of top 16 features) as shown in Fig. 2, we select the top four hardware events (from the blue-colored features) that show significant accumulated information gains (regarding label Y) to train a model, considering that most modern microprocessors' counters can only monitor a limited number of events at once during applications execution time [3, 8]. The selected four hardware events include $node - loads$, $LLC - loads$, $L1 - dcache - load - misses$, and $mem - stores$.

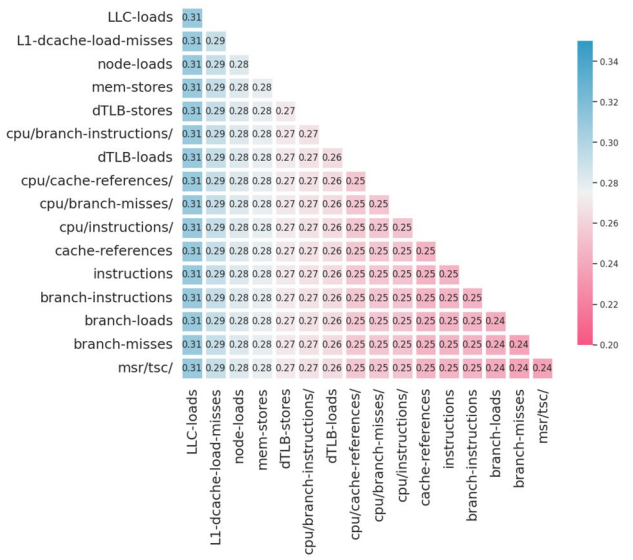

Fig. 2 Top HPC features' accumulated mutual information gain to Y. The more blue highlight, the more contribution of the feature to the label Y

3.3 Embedding Tabular Data

Some previous studies have utilized the embedding technique to convert non-image data types, such as text and byte sequences, into image-like representations as a preprocessing step. This approach is employed to harness the capabilities of deep neural networks (DNNs) for enhancing cybersecurity threat detection. The rationale behind this lies in the well-established performance of DNNs in both image processing and natural language processing (NLP). For instance, in the work by Raff et al. [30], a static analysis is conducted directly on byte programs extracted from Microsoft Windows Portable Executable (PE) files, without executing them. This analysis aims to transform raw byte sequences into higher-level representations, which are subsequently fed into an LSTM model. Notably, each raw byte sequence is treated as a lengthy sequence classification problem, spanning up to two million time steps. Similarly, in a study by [31], the authors take malware binaries, transform them into grayscale images, and then convert these images into sequence embeddings to be used with a recurrent neural network (RNN) for stealthy malware detection.

In the study by [32], the transformation of Android applications' API call sequences, derived from the structure of the API call graph, is undertaken. These sequences are converted into a low-dimensional numeric vector feature set, commonly referred to as embeddings. These embeddings are then utilized as inputs for a deep neural network (DNN). It is important to note that this approach primarily focuses on detecting malware that involves the invocation of malicious API(s) to execute malicious code. However, it's worth highlighting that this work adopts a static approach to extract the API call graph representation from malicious code, without actually executing the code. This static analysis approach does not capture the dynamic runtime scenario. Moreover, the research in [33] builds upon prior opcode sequence embedding work to solve the challenge of long opcode sequence problem, by converting sequential opcodes using low-dimensional opcode embeddings to discover the malicious patterns.

Notably, most of the previous approaches rely on static analysis without executing the malicious applications. This static analysis approach has its limitations, as it can be vulnerable to evasion techniques such as obfuscation and encryption employed by malicious actors to avoid detection. In contrast, our research takes a dynamic analysis approach. We collect hardware performance counter tracing data directly from running applications when malicious code is in action. Our unique contribution lies in the development of a lightweight algorithm to directly embed HPC data into image-like representations using only four numerical HPC values as features. This is a significantly smaller amount of information compared to PE byte sequences, opcode sequences, or API call graphs, making it challenging to train an effective DNN model directly. To address this challenge, we adopt a transfer learning approach and leverage a transferrable DNN architecture.

Furthermore, our proposed approach differs from previous works that often employ RNN and LSTM architectures, which are larger model architectures and can introduce substantial overhead due to the existing large data embeddings. In contrast, our paper explores the effectiveness of embedding only four numerical HPC events using our two-staged methodology. This unique approach is aimed at addressing the limitations of static analysis and exploring the potential of lightweight, transferable DNN architectures for hardware-assisted malware detection. This not only enhances the extensibility and generalizability of the proposed solution but also underscores its significance in diverse domains, including those involving resource-limited devices.

3.4 Machine Learning Classifiers

Standard ML Classifiers We examine the suitability of various standard machine learning classifiers for known and unknown malware detection. These ML models include RandomForest (RF), DecisionTree (DT), Gaussian Naive Bayes (GNB), Logistic Regression (LR), ExtraTreeClassifier (ExtraTree), RidgeClassifier (Ridge), K-Nearest Neighbor (KNN), Support Vector Machine (SVM), and BaggedDT. These ML models cover a diverse range of algorithms and the final predictor can be a binary classification model which is aligned with the malware detection task. It is worth noticing that standard MLs are most suitable for structured data rather than unstructured data.

Deep Neural Network (DNN) Traditional ML classifiers are primarily used to train on structured data such as tabular data stored in CSV files or relational databases. On the other hand, deep neural networks (DNNs) are most commonly used on unstructured data such as images and natural language processing. Computer vision-based DNN models can recognize hierarchical relationships in analyzing simple to complex features, and characterize the visual system as a hierarchical and feedforward system. While the neurons in the early layers of a DNN have small receptive fields and are sensitive to local features, they can capture more generalized patterns in deeper layers. Recent results of very deep neural networks from the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) have shown that the neural network can achieve a top 5 error rate of 3.57%. Inspired by computer vision applications, in this work, we leverage DNN to develop an accurate and intelligent malware detection framework based on image-based HPC data.

Transfer Learning Recent studies have reported that a well-trained DNN could transfer its knowledge of generalized features and feature extraction ability from one domain to another. The work in [34] highlights that transfer learning can also share the architecture-related parameters in a new field while maintaining a high-performance rate. While transfer learning has demonstrated its success in computer vision, its application to tabular-based data remains relatively unexplored. In a survey conducted by [35] on deep neural networks (DNN) and tabular data, it was observed that the potential of transfer learning in the context of tabular data remains an open question. To fill this research gap, our study in this paper is dedicated to investigating a transfer learning approach tailored to tabular hardware data, using only four performance counter-numerical values. Notably, existing literature suggests that despite its status as a state-of-the-art technique, deep neural networks may not perform as effectively as classical machine learning models in the realm of tabular data [36]. Notably, existing works [34] present that a CNN architecture can transfer its knowledge trained on the ImageNet domain to a new problem domain with high accuracy and stable results. Motivated by such advances, in this work we develop a transfer learning strategy combined with a DNN model in zero-day hardware-based malware detection. ImageNet is significantly different from malware and benign datasets, where ImageNet contains more generic images in everyday lives. In contrast, malware and benign datasets have numerical features from the processor's HPC events. In this paper, we utilize a tabular-based HPC-based dataset comprising malware and benign samples. Our primary objective is to investigate the potential of transfer learning in combination with DNN to enhance knowledge transfer and subsequently achieve superior detection performance compared to traditional machine learning approaches. Our work is the first in the field that explores the functionality and effectiveness of leveraging DNN network transferred from ImageNet to the tabular-like zero-day malware detection domain based on only a few hardware tracing events monitored at run-time.

3.5 Overview of *Deep-HMD* Framework

In this work, we propose *Deep-HMD* framework to address the *Challenges 1 & 2* of existing HMD methods and to overcome the limitations of standard ML classifiers in detecting zero-day malware. To this aim, we first explore the performance of standard ML classifiers trained with the most prominent HPC events. We train the ML models with default parameter settings as our base learners. We then examine the models across various metrics on the known test and zero-day test datasets. The ML classifiers are implemented using scikit-learn [26] and are used to analyze how well they can perform on known and zero-day test datasets. Next, given the weak performance of ML models (as we will show in Section 4), we present the *Deep-HMD* framework as the target hardware-based zero-day malware detector.

We investigate state-of-art deep learning model architecture trained over ImageNet Large Scale Visual Recognition Challenge (ILSVRC). ILSVRC uses the smaller portion of the ImageNet which consists of 1000 categories with a total of 1.3 million training images, 50,000 validation images, and 100,000 testing images. There are several advantages in using transfer learning. Firstly, the pre-trained model has already learned to recognize patterns from millions of images. Secondly, training from scratch requires a larger dataset, high training time, and trial-and-errors on parameter tuning. While using a pre-trained model and transfer learning can maintain high accuracy when fine-tuned in a new field. The work in [37] studied that among the many popular DNN networks, ResNet is an appropriate choice in network-based deep transfer learning. ResNet is a convolutional neural network that implements residual blocks of "skip connections" to alleviate the issue of vanishing gradient by setting up an alternate shortcut for the gradient to pass through. In addition, they enable the model to learn an identity function. This ensures that the higher layers of the model do not perform any worse than the lower layers. ResNet is also a simple architecture such that residual blocks do not add any major complexity to the network so that all the common optimization methods can be used in training residual networks. Thus, we implemented our proposed transfer learning scheme based on ResNet18 which has 18 layers in total. It is notable that in our exploration of various DNN architectures for transfer learning, we indeed considered more lightweight options like binary neural networks (BNN) and other convolutional neural networks (CNN). However, these alternatives did not meet the performance expectations for unknown malware detection. ResNet18 demonstrated the highest performance among all methods examined, underscoring its suitability as the preferred choice for our transfer learning scheme to boost up the performance of hardware-assisted zero-day malware detection.

3.5.1 Threat Model

Recent ML-assisted malware detection methods using HPC events have mainly considered two major validation methods including cross-validation and percentage split to examine the effectiveness of their models. 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 of the 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

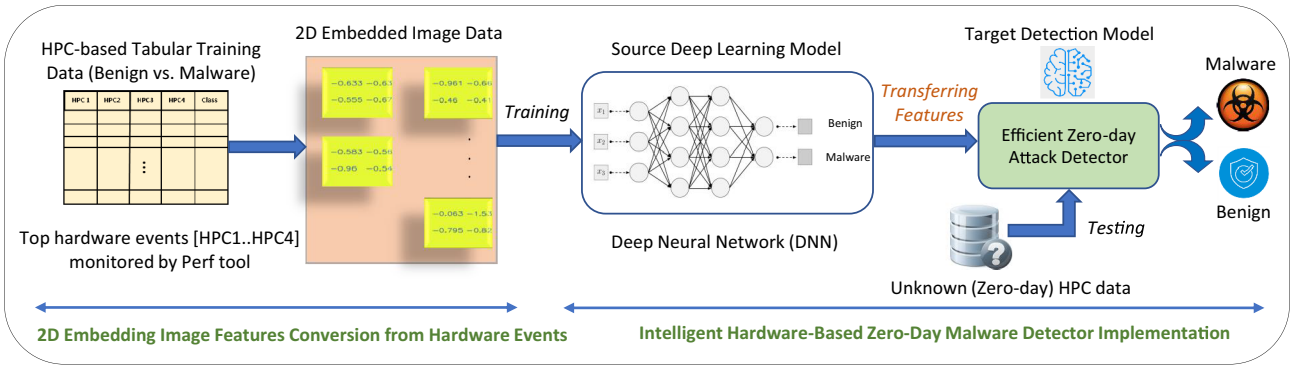

Fig. 3 Overview of *Deep-HMD*, the proposed zero-day malware detection framework using 4 selected hardware events

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 scenarios occurring in real-world applications in which the trained machine learning classifiers should have never seen the testing dataset.

According to our experiments, malware applications within the same family typically do not exhibit identical performance counter values. This variation arises because the values collected from HPC registers reflect the unique signature left by each application on the underlying processor and can differ between different applications. To streamline our analysis, we categorize all malware collectively into one category, encompassing all nine types, while designating a separate category for the HPCs of benign applications. This binary classification approach distinguishes between benign (0) and malware (1), aligning with our primary focus on binary classification (malware detection). We use the top four HPCs in Section 3.2.1 to form sub-datasets across the work.

To model the zero-day malware threat type in our experiments, among all nine malware types, we held out all four types of malware from rootkit, backdoor, virus, and ransomware as the target zero-day test data. These four types of malware are not presented in the training and known test datasets, thus, the zero-day malware set is totally unknown from the training dataset. For benign, we held out 30% of all benign data aside as a zero-day test benign dataset. We kept both malware and benign aside to imitate the zero-day testing in real-world scenarios where the malware is undocumented in the training database of the detection mechanism. The rest of the five types of malware including trojan, spyware, botnet, worm, and adware as well as the rest of the benign samples are considered for training and known test purposes, and we randomly split them into 70% for training and 30% for known testing. The difference between the known-test and zero-day-test in our experiments is that the known-test data contains the same malware types as the training dataset but with different unseen data and the zero-day-test data contains different malware types from the training dataset that are considered as new unknown attacks. After data are split, we relabel all types of malware as malware and leave benign as benign. Notably, our *Deep-HMD* framework uses the same datasets during training, known-test, and zero-day test same as all classifiers. Also, classical ML models use the tabular format data, and *Deep-HMD* employs the image data converted from corresponding tabular data.

3.5.2 Architecture of *Deep-HMD*

Deep-HMD is a multi-layer intelligent and salable framework that achieves an accurate and robust zero-day malware detection performance. The general overview of the *Deep-HMD* is depicted in Fig. 3. As seen, during the first level the *Deep-HMD* converts tabular malware and benign hardware events data (that are monitored from the underlying processor) to image formatted data using an effective 2D embedding image features conversion method (detailed algorithm and visual effects are introduced in the following section). A deep neural network can train on both tabular and image data. However, DNN-based architectures training on visual recognition tasks have achieved far-reaching performance with high top 1/top 5 accuracy and low top 5 error rates. Next, *Deep-HMD* leverages a high-performance ResNet architecture and transfer learning technique to recognize the zero-day malicious images at run-time. The work in [38] presented a method that projects features in tabular format into two-dimensional images before feeding images to fine-tune CNN models. However, there exist no similar experiments on the application of such methods in detecting the signatures of unknown malware and in particular with the emphasis on developing effective security countermeasures at the processors' hardware level. Our *Deep-HMD* method explores this space by employing a novel deep neural network and transfer learning training strategy on image-based hardware events data to achieve state-of-the-art performance for zero-day malware detection using a limited number of HPC events.

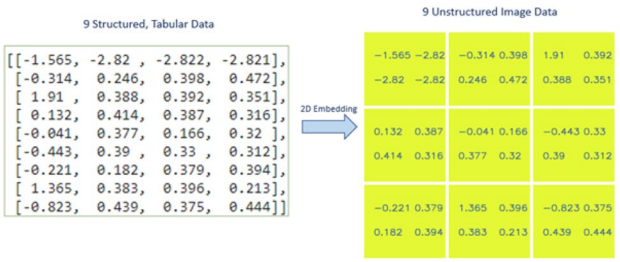

Fig. 4 Nine sample structured tabular data projected to nine two-dimensional images using 2D embedding image features conversion from hardware events

Stage 1 in *Deep-HMD* Our proposed *Deep-HMD* framework employs the encoding method described in Algorithm 1 to convert each row of 4 HPC tabular data to one image data represented by a two-dimensional Numpy array with the size of 256 × 256 and three color channels of RGB (red, green, and blue). The four HPC values represent the top four hardware events that are highly impactful to the target label of malware or benign. Mapping the same sequence of hardware events to an image uncovers potential hidden relationships and discrepancies within the four HPC events concerning malware or benign classification. Meanwhile, malware and benign follow discrepant patterns reflected by the four HPC values. These hidden features can be extracted by different convolutional layers of *Deep-HMD* to assist in training effective malware detection model. We normalized all rows of tabular HPC data using the standard scalar in Scikit Learn, which removes the mean and scales the data to unit variance. Next, we use OpenCV [39] library to evenly project the four numeric data to a 256 × 256 × 3 resolution image with equal spacing and no-overlapping as shown in Fig. 4.

Algorithm 1 Converting tabular data to 2D images

Input: HPC features in tabular format X = {x1,x2,x3,x4}
Output: image data equivalent to X
repeat
  forall *for each row of X={x1, x2, x3, x4}* **do**
    normalize HPCs
    set font size as 50, resolution as 256 x 256 x3
    set 2 columns 2 rows per image
    apply OpenCV's cv2.putText() to draw HPC number on image
    save converted image to folder
  end
until *all rows of fitted tabular data are converted to images*

The left image in Fig. 4 depicts nine rows of the 4 top HPCs in tabular format. We normalize them first according to all features' mean and standard deviation. Each row of the top 4 HPCs is projected to one image as described in Algorithm 1. The example on the right side shows that nine rows of tabular data are converted to 9 two-dimensional image data. Each image on the right side is associated with a target label of either malware or benign, and each resulting image is employed independently for training or inference purposes. Overall, in this stage, we draw the four features with equal sizes on the image with even spacing and ensure that they are not overlapped. Such images contain spatial relationships and hidden features among malware and benign applications resulted from a limited number of performance counter data (only 4 HPCs).

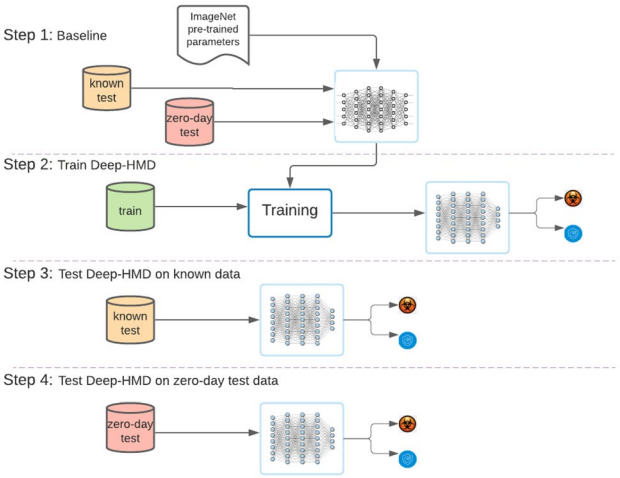

Fig. 5 Overview of training and testing in *Deep-HMD*

Stage 2 in *Deep-HMD* The second stage includes using the generated image data as inputs to train and test an accurate and effective DNN-based model for zero-day malware detection. We implement *Deep-HMD* with a customized transfer learning training strategy based on the ResNet18 architecture [40] during training. To further explain the work done in *Deep-HMD*, we describe stage 2 in several steps in the following subsection.

3.6 Training and Testing in *Deep-HMD*

Figure 5 demonstrates the overall training and testing process of *Deep-HMD* in four steps. Steps 1 and 2 are dedicated to the training process and steps 3 and 4 are for the testing phase.

In **Step 1**, we first examine how the baseline model of the *Deep-HMD* performs on our known and zero-day test datasets. To this aim, we load the ImageNet parameter of *Deep-HMD* in fast.ai, run two tests, and calculate

various metrics. Fast.ai [41] is a deep learning wrapper library with PyTorch as its underlined backbone that can quickly train high-performance deep learning models and supports application domains in computer vision, natural language processing, and tabular models. The baseline of *Deep-HMD* tests how much knowledge it learns from ImageNet on generic features such as lines and strokes. We found it can detect 92% of benign applications but only 6% of actual malware. This experiment indicates that the front layers of the pre-trained transferrable model trained on a sizeable ImageNet can be used as a feature extractor, and the extracted features are versatile according to Tan et al.'s study on transfer learning [37]. Notice we mention "transferrable'' because there is a relationship between model architecture and transferability [42]. The front layers of such pre-trained models learn the morphology of images in general and the morphology knowledge can be transferred to other visual recognition tasks with or without labeled data [43]. As shown in our baseline experiment where there is no training occurred, the model can detect 92% of benign as benign that further supports the understanding that a well-trained CNN can be an efficient feature extractor for unknown context. With fine-tuning model parameters on malware datasets, it continues to learn domain knowledge, particularly the pattern of malware from benign. In particular, we examine the possibility of transferring the knowledge of pattern recognition to a new domain with a limited number of training samples and fast training time.

During **Step 2**, we load the transferable pre-trained baseline model with the ImageNet parameter, remove the last layer of the network, replace it with a Softmax function that outputs binary classification as shown in Fig. 6, and train the model.

Algorithm 2 Training Process in *Deep-HMD*

Input: HPC features in tabular format X, and target label y
Output: *Deep-HMD* DNN Model for Binary Classification

Feature Selection:
  forall *tabular HPC-based dataset* **do**
    calculate MI by applying scikitlearn mutual_info_classif on all HPC features
    select top 4 features and fit to the whole dataset
    train/known test/zero-day test split on the fitted dataset
end
Image Conversion: - see Algorithm 1
while *training* **do**
  load batch data, resize images to 224 x 224 x3 and initialize ImageNet parameters
  while *validation loss >training loss* **do**
    for every 7 to 20 epochs:
      apply cyclical learning rate (clr) to find optimized learning rate (lr)
      apply found lr to the next training steps (7-20 epochs)
  end
  save model
end

Algorithm 2 describes the training process in *Deep-HMD*. In this stage, the generated two-dimensional images are resized to 224 × 224 × 3 and fed into the *Deep-HMD* network initialized with the ImageNet pre-trained weights. Firstly, we train it for five steps with a batch size of 64 to find

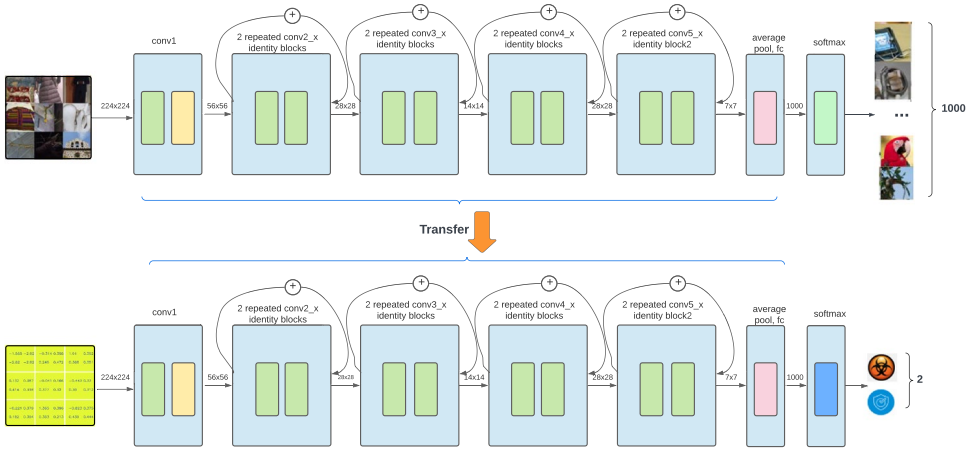

Fig. 6 *Deep-HMD* architecture to classify the HPC-based images into benign or malware. This figure shows how knowledge is transferred from a pre-trained transferrable CNN model (top) to *Deep-HMD* (bottom). It uses the backbone of ResNet 18 with a customized softmax layer and retrained on malware dataset. It inputs 224 × 224 × 3 ImageNet standard size and outputs binary classifications of malware or benign. In the middle of the hidden layers, it contains shortcut connections skipping one or more layers to avoid saddle points. Among these layers, 1 × 1 convolutions are utilized to reduce matrix multiplication, which directly reduces the number of parameters. The architecture is retrained on malware/benign data

an optimized learning rate for the next training cycle. In this step, generic features learned from ImageNet are transferred to the *Deep-HMD* network. In the customized training, we apply cyclical learning rates, oversampling, and weight decay techniques. We did not apply augmentation to the images. We use batch normalization right after each convolution and before activation. We use the cross-entropy loss function with a mini-batch size of 64. The learning rate starts from 0.0001, and we use fast.ai's learning rate finder for every 7 to 20 epochs to find the optimized learning rate to apply to the next training steps. We apply a weight decay of 0.01 and a momentum of [0.95, 0.85, 0.95] across the whole training process. We train the models up to 30 to 80 iterations. We monitor training loss and validation loss decreasing until the validation loss is close to the training loss. We save checkpoints periodically and use the best model for the testing phase.

3.6.1 Cyclical Learning Rates

DNN model learns from data and updates its weights from a gradient descent function. Gradient descents are often stuck at saddle points, which are the points in the gradient descent graph where some dimensions observe a local minimum, and others observe a local maximum. In general, an increased learning rate will allow the network to jump out of the saddle points to possibly enable the model to converge. However, it is also known that a larger learning rate might make the DNN diverge. Therefore, the learning rate strategy could have a significant impact on the effectiveness of model training. Various learning rate-finding strategies have been experimented with to solve this problem, such as Adaptive Learning Rate, SGDR, and CLR.

In our experiments, we employ a cyclical learning rate (CLR) technique [44] during the training process. For this purpose, we use the fast.ai wrapper library [41] with the plain PyTorch backbone and apply the cyclical learning rate technique to find the most optimal learning rate for every several training steps. The CLR algorithm is shown in Algorithm 3. We apply fast.ai's learning rate finder and set the minimum and maximum learning rates for 10 epochs. Then, we apply the best learning rate in the next training cycle. We set each training cycle with seven to twenty epochs depending on how soon we observe it stops learning by monitoring the validation loss and validation accuracy as depicted in Fig. 7. The validation accuracy starts to grow and stabilize from epoch 4, as shown in Fig. 7. As a result, the best-performing model is selected for the testing phase.

Algorithm 3 Applying cyclical learning rate

```
- Specify minimum (min_lr) and maximum (max_lr)
boundaries;
for a specified number of epochs do
    - increase lr linearly from min_lr to max_lr;
    - monitor validation loss to decrease, stops lr finding
    when the validation loss start to increase;
    - record validation loss and corresponding lr.
end
- Plot learning rate and validation loss in fastai [41];
- Select a fixed lr value that is before the minimum validation
loss;
- Use the lr to train for several epochs (1cycle policy [44]).
```

3.6.2 Over Sampling for Imbalanced Datasets

To overcome the challenges associated with the imbalanced dataset samples and remove the potential bias towards the majority class (benign in our case), we apply the oversampling technique during the training process. The oversampling technique involves duplicating examples from the minority class and adding them to the whole training dataset to create a more balanced training dataset. During each epoch, samples from the minority class are selected randomly with replacements. Once the training epoch completes, they are released back to the original dataset to be chosen again in the following training epochs. Doing so over the minority class dataset creates a more

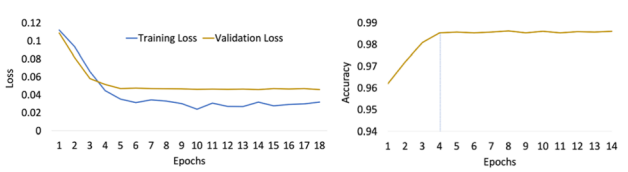

Fig. 7 Applied cyclical learning rate (CLR) during training. The left figure shows the training loss and validation loss, and the right figure depicts the validation accuracy during training

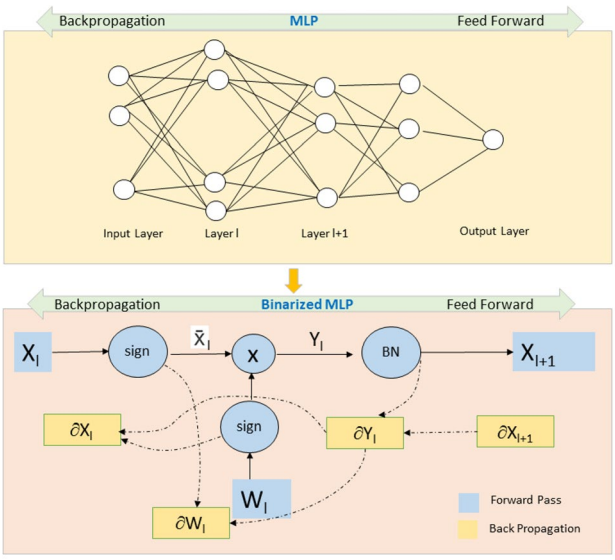

Fig. 8 MLP (top) and binarized MLP (bottom) illustrated weight gradients in the forward pass are replaced by 1-bit binary and backpropagated during training

balanced training dataset for all classes and helps with model convergence in *Deep-HMD*.

In **Steps 3** and **4**, we convert the same sets of HPC data stored in tabular format to images and store the images into sub-folders of malware or benign. In fast.ai, we load the pre-trained *Deep-HMD* model, resize the input image to 224 × 224 × 3, and then run in the batch size of 64 images to process the prediction. Algorithm 4 describes testing *Deep-HMD* on the zero-day dataset. Lastly, we accumulate the predictions, calculate the zero-day test metrics, and report the zero-day test results in the experimental results section.

Algorithm 4 Zero-day testing process in *Deep-HMD*

```
Input: HPC features in image format saved in zero-day
       dataset, and target label y for each image
Output: Detect every image associated application as
        malware or benign
repeat
    forall zero-day data do
        load Deep-HMD model
        resize images to 224 x 224 x3
        predict each image for either malware or benign
        calculate F-measure, AUC, TPR, FPR, TNR,
        FNR, Precision, Recall
        calculate unit latency for inference
    end
until testing done on all zero-day test data
```

3.7 Binary MLP

Our implementation of a binary MLP model follows the method proposed by Wang et al. [45], utilizing TensorFlow Keras with five layers, including the input and output layers. Similar to *Deep-HMD* we first convert each row of the top four HPC tabular data into a 32 × 32 × 3 image. We then train the binary MLP model in Keras with the converted image data on the training dataset. Figure 8 illustrates the architecture of the MLP, including the application of 1-bit binary to the weight gradients during the forward pass. It uses a binarized Sigmoid activation function in the last layer. After training, we evaluate the model's performance on an unknown zero-day test dataset. In addition, to regularize the model learning and reduce the impact of the weight scale, we apply L1 batch normalization and batch norm momentum in each hidden layer. We use the Adam optimizer and apply a learning rate scheduler with an initial value of 0.025. The learning rate scheduler automatically reduces the learning rate by a factor of 0.5 once the learning stagnates during training.

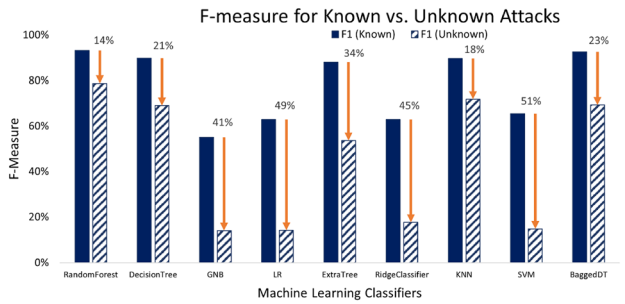

Fig. 9 Evaluation of standard ML classifiers for known and unknown (zero-day) malware detection

4 Experimental Results

In this section, we evaluate the experimental results and analyze the effectiveness of the proposed malware detection approach as compared to state-of-the-arts.

4.1 Classical MLs Performance

To further highlight the challenge of unknown malware detection, we have evaluated the standard ML classifiers that are widely used in state-of-the-art HMD methods considering both known and unknown conditions. The F-measure (F1-score) results for known and zero-day malware detection (with 4 HPC events) are shown in Fig. 9. As observed, the performance of standard ML models on zero-day attack detection substantially drops by more than 40% in GNB, logistic regression, ridge, and SVM classifiers. When examined by the unknown (zero-day) test data, the trained machine learning classifiers have never seen the testing malware types. Even for the most robust ML model, Random Forest, the F-measure for zero-day malware drops by 14% as compared with the scenario of detecting known malware. The results confirm the limitation of standard ML algorithms in recognizing the signatures of unknown malware using HPC events and further highlight the importance of proposing an effective mechanism to enhance the detection rate of the hardware-based zero-day malware detection.

4.2 *Deep-HMD* Performance

Table 2 reports the performance results of *Deep-HMD* versus binary MLP, a deep neural network model (tabular DNN) trained on HPC malware data (same data with *Deep-HMD* but in tabular format), and different ML-based detectors for zero-day and known malware detection using 4 HPC events. Note that both *Deep-HMD* and binary MLP are trained on image-based HPC data, while the tabular DNN and classical MLs are trained on HPC tabular data format. We selected various ML-based detectors that have widely been adopted in existing HMD techniques [2, 3, 6–8]. As the results indicate, our proposed *Deep-HMD* achieves 97% in F-measure, 97% in area under the curve (AUC), and 98% accuracy for unknown malware detection. In addition, it obtains 96% in

Table 2 Performance results of *Deep-HMD*, binary MLP, and tabular DNN, and various ML-based detectors for zero-day malware detection and known-test malware detection

Model	Acc	F1	AUC	TPR	FNR	TNR	P	R
Zero-day test								
Deep-HMD	0.98	0.97	0.97	0.99	0.01	0.97	0.96	0.99
Binary MLP	0.82	0.81	0.91	0.76	0.24	0.52	0.86	0.80
Tabuar DNN	0.66	0.16	0.65	0.08	0.92	0.99	0.88	0.09
RF	0.87	0.79	0.87	0.68	0.32	0.98	0.94	0.68
DT	0.81	0.69	0.79	0.58	0.42	0.94	0.86	0.58
GNB	0.62	0.14	0.40	0.09	0.91	0.92	0.38	0.09
LR	0.62	0.14	0.40	0.09	0.91	0.92	0.39	0.09
ExtraTree	0.74	0.54	0.70	0.41	0.59	0.93	0.78	0.41
Ridge	0.63	0.18	0.46	0.11	0.89	0.93	0.48	0.11
KNN	0.83	0.72	0.81	0.61	0.39	0.95	0.87	0.61
SVM	0.64	0.15	0.49	0.09	0.91	0.96	0.56	0.09
BaggedDT	0.82	0.69	0.82	0.56	0.44	0.97	0.92	0.56
Known test								
Deep-HMD	0.99	0.99	0.99	0.99	0.01	0.99	0.99	0.99
Binary MLP	0.86	0.82	0.83	0.84	0.16	0.87	0.8	0.88
Tabular DNN	0.82	0.68	0.83	0.52	0.48	0.99	0.97	0.52
RF	0.95	0.93	0.95	0.91	0.09	0.98	0.96	0.91
DT	0.93	0.90	0.91	0.90	0.10	0.94	0.89	0.90
GNB	0.75	0.55	0.69	0.44	0.56	0.92	0.74	0.44
LR	0.78	0.63	0.74	0.53	0.47	0.92	0.78	0.53
ExtraTree	0.92	0.88	0.90	0.89	0.11	0.93	0.87	0.89
Ridge	0.78	0.63	0.74	0.52	0.48	0.92	0.79	0.52
KNN	0.93	0.90	0.92	0.89	0.11	0.95	0.91	0.89
SVM	0.81	0.65	0.78	0.52	0.48	0.96	0.88	0.52
BaggedDT	0.95	0.93	0.94	0.90	0.10	0.98	0.96	0.90

precision and 99% in recall. Whereas, the best-performing standard ML classifier, Random Forest, can only achieve 79% in F-measure, 87% in AUC and accuracy, and 68% in recall when used for detecting unknown malware.

As illustrated by the results in Table 2 for known malware detection, our proposed *Deep-HMD* approach exhibits exceptional performance metrics, achieving an impressive 99% in F-measure, area under the curve (AUC), and accuracy, as well as a 99% true positive rate (TPR) and True Negative Rate (TNR). Notably, among all the traditional machine learning methods, Random Forest stands out with commendable results, boasting a 93% F-measure, 95% AUC, 95% accuracy, and a well-balanced TPR and TNR of 91% and 98%, respectively, in the known test dataset. However, a noteworthy observation is the significant decline in the detection performance of Random Forest (as well as other conventional ML models) when transitioning from known malware detection to unknown malware detection, as discussed in the preceding section. This discrepancy highlights a fundamental weakness of classical ML models when confronted with the task of identifying diverse and previously unseen zero-day malware samples. In essence, while these models excel at recognizing known malware patterns, they struggle to cope with the inherent unpredictability and diversity presented by unknown malware, making them less effective in real-world scenarios where novel threats continually emerge.

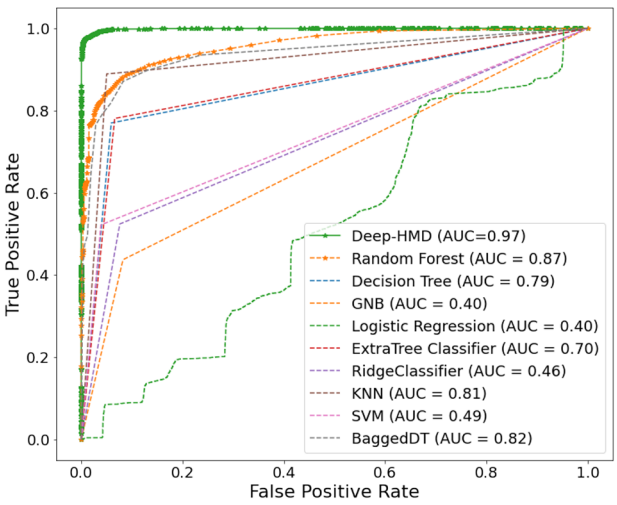

Fig. 10 ROC curves of *Deep-HMD* as compared with standard ML models for zero-day malware detection

Table 3 Performance and overhead results: *Deep-HMD* and binary MLP for zero-day malware detection

Model	F1	AUC	Latency (ms)	Model size(MB)
Deep-HMD	97.1%	97.4%	3.22	44.8
Binary MLP	81.3%	91.3%	2.85	13.8

Furthermore, when comparing *Deep-HMD* and tabular DNN, we notice that deep neural networks applied to tabular formatted data perform worse than classical machine learning methods. The results presented in Table 2 demonstrate that while tabular DNN is able to accurately detect benign data with a high true negative rate (99% TNR), it also mistakenly classifies benign data as malware with a very low true positive rate (8% TPR). Hence, the overall recall of the model to detect malware from all malware samples correctly is very low, at only 9%. A significant number of instances of malware are being falsely classified as benign. This raises concerns about the model's effectiveness in defending against cyberattacks.

Overall, the results demonstrate that our suggested intelligent hardware-based technique for detecting zero-day malware, known as *Deep-HMD*, is the most precise model out of all the classifiers tested. *Deep-HMD* not only achieves an F-measure of 97% on the unknown zero-day test but also provides a true positive rate of 99% and a false positive rate of only 1%. This is a significant improvement compared to the results obtained by the best standard ML (RF classifier in Table 2, zero-day test), which only delivers a true positive rate of 68% and a false positive rate of 32% (Fig. 10).

4.3 *Deep-HMD* vs. Binary MLP

Table 3 summarizes the evaluation results of *Deep-HMD* and binary MLP models for zero-day malware detection using 4 HPC events, including their performance and overhead (latency and memory footprint). The results demonstrate that *Deep-HMD* is a more robust malware detector with a higher detection rate, achieving 97.1% F-measure and 97.4% AUC in recognizing unseen zero-day malware. However, it has a larger model size to achieve such high accuracy. In contrast, the binary MLP has a much lower detection rate of 81.3% F-measure. However, it offers the advantage of being a lightweight DNN detector with a small model size of only 14 MB, and it still achieves a high AUC of 91.3% making it a potentially attractive option for resource-limited embedded systems.

4.4 Explainability Analysis

In this subsection, we explore the explainability of *Deep-HMD* methodology, uncovering how it learns distinctive

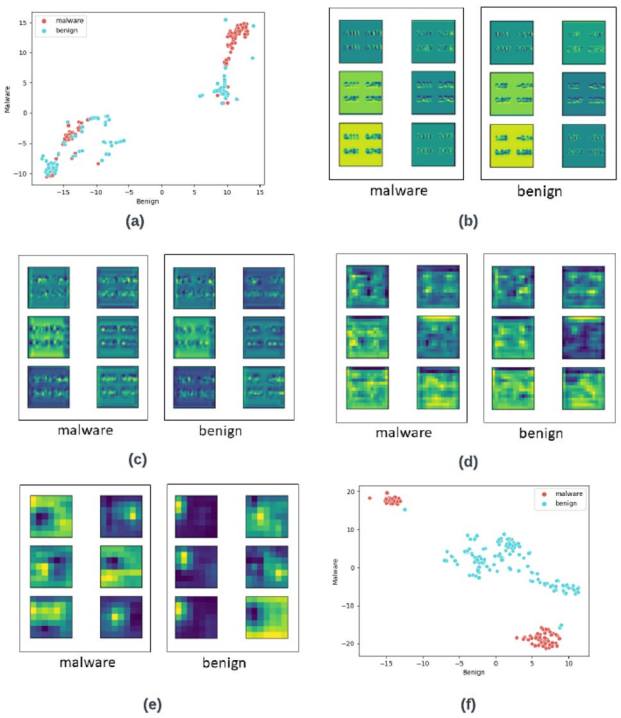

Fig. 11 *Deep-HMD* feature map during model training process. **a** original tabular data's feature distribution, **b** *Deep-HMD*'s first convolutional layer's feature map, **c** 8^{th} convolutional layer, **d** 12^{th} convolutional layer, **e** 16^{th} convolutional layer, **f** linear layer's feature distribution

features of malware and benign through a multi-layer image-based deep neural network than a one-stage-only tabular-based method. To achieve this, we randomly collected approximately 200 images each of benign and malware (generated from the corresponding tabular data with the same top 4 HPC features used across all experiments in this work) and utilized these to evaluate our *Deep-HMD* model. By plotting the feature maps at each layer of the model, we can gain insight into how the model distinguishes between malware and benign software. We extracted the feature maps for each convolutional and linear layer, allowing us to explore the unique features detected by the model at different stages of the learning process. The model produced 64 channels of feature maps at each convolutional layer, from which we selected a subset to illustrate and explain the differences in features detected between malware and benign software.

Figure 11 shows the distribution of the feature from the original four HPC tabular data feature space, to the model intermediate layers' feature map in *Deep-HMD*, until the very last linear layer before applying the SoftMax activation layer in the model. Sub-figures (a) and (f) are the features' data points that are projected in a two-dimensional feature space using the method of t-SNE [46]. t-SNE is an effective method to display feature spaces of various classes that visualizes high-dimensional data by giving each data point

Table 4 Uncertainty analysis of *Deep-HMD*

Test set	Uncertainty score	AUROC	AUPRC
Known test	48%	0.99	0.981
Zero-day test	96%	0.97	0.948

a location in a two or three-dimensional map. The rest sub-figures are feature maps extracted directly from selected convolutional layers. As shown, originally, the malware and the benign data are entangled altogether and could not be separated clearly, indicating that there is a non-linear hidden relationship between malware and benign data points (as shown in Fig. 11a). After the HPC tabular data are converted to image data and are fed to *Deep-HMD* neural network, the model applies non-linear convolutional functions and slides the feature space to multi-dimensional channels, and this computation continues in several convolution layers.

In Fig. 11b showing the first convolution layer, one can only observe the HPC tabular data value between malware and benign. The early layers in sub-figure-(c) focus on local features of the HPC-based images in different channels, making it difficult for humans to interpret the differences. In the convolution layer 12 as depicted in Fig. 11d, we observe a more abstract feature difference between malware and benign begin unfolding. In convolution layer 16 which is the last convolutional layer before applying linear layers and activation functions, as seen in Fig. 11e clear pattern differences between malware and benign are observed by the model. Finally, as shown in sub-figure (f), the linear layer demonstrates the t-distribution of the feature map, clearly distinguishing between malware and benign samples.

4.5 Reliability Analysis

To understand the robustness and generalization of the proposed method, we conduct a thorough reliability analysis and evaluate *Deep-HMD*'s predictive quality and its performance consistency involving uncertainty and robust generalization on unseen events. Tran et al. [47] suggested the reliability analysis of a model regarding uncertainty, robust generalization, and adaptation across a full range of datasets that have various contexts and characteristics such as open datasets, in and out-of-distribution datasets. In this paper, we focus on analyzing robust generalization on the unseen event that has a covariance shift (zero-day test) and subpopulation shift (known test). In our threat model, a zero-day test dataset has new malware types that the training dataset doesn't have; a known test is a subset with a similar feature space to the training dataset and is drawn from a larger population distribution. We implemented Mahalanobis distance [48] in scikit-learn [26] to measure the uncertainty score of the known test and zero-day test, which shows the similarities of two evaluated datasets. The score is the normalized distance of the feature spaces between the test dataset and the training dataset. Notice the training dataset itself has an uncertainty score of 4%. The Mahalanobis distance is calculated as below, where x is the data point to be evaluated, μ is the mean vector, and Σ is the covariance matrix:

$$D^2 = (x - \mu)^T \Sigma^{-1} (x - \mu) \tag{3}$$

We assess the detection performance of *Deep-HMD* by evaluating its prediction quality in terms of accuracy, F1, AUROC, and AUPRC (average precision). In Table 4, we present the results for AUROC and average precision. AUROC measures the model's ability to distinguish between malware (positive examples) and benign (negative examples) and is used to evaluate the discrimination performance. On the other hand, AUPRC indicates whether the model can correctly identify all the malware without mistakenly labeling too many benign samples as malware. We conducted a reliability analysis of our *Deep-HMD* model, considering both known and unknown test scenarios. The results indicate that as the uncertainty score increases from 48% in the known test to 96% in the zero-day test, the model's AUROC and AUPRC decrease by 4% and 7%, respectively. Despite the slight decrease in performance, the results highlight the robustness and reliability of our proposed multi-level deep transfer learning approach for HPC-based zero-day malware detection.

5 Concluding Remarks

In this paper, we investigated the efficacy of widely used machine learning classifiers for hardware-based zero-day malware detection. Our findings reveal that they are not effective in recognizing unknown malware patterns with high accuracy and low false positive rates. This challenge arises from the fact that the zero-day malware HPC data does not match any known attack applications' signatures in the existing database, making it a challenging problem to tackle. To address this issue, we proposed *Deep-HMD*, a multi-level DNN-based approach that utilizes a flexible transfer training strategy to detect both known and unknown (zero-day) malware at run-time with a small number of hardware events. Our experimental results demonstrated that *Deep-HMD* outperforms existing ML-based detection methods, achieving 97% in both F-measure and AUC metrics for recognizing unknown malware signatures with only 1% false positive rate. Moreover, *Deep-HMD* excels in known malware detection, obtaining 99% in both F-measure and AUC metrics. Our proposed intelligent solution is the first DNN-based method for accurate hardware-based known and zero-day malware detection, employing a lightweight and efficient

transfer learning strategy on HPC-based data presented in image format. It is extensible and generalizable, making it a well-suited solution for securing modern computing systems against emerging malware attacks.

Declarations

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

Competing Interests The authors declare no competing interests.

Author Contribution All authors contributed to the study conception and design. He and Sayadi prepared the main manuscript and performed the method, experimental setup, and analysis. Sayadi carried out the supervision of the work. All authors reviewed and revised the method and manuscript.

Data Availability Available upon request.

Ethical Approval Not applicable.

References

1. Das S, Werner J, Antonakakis M, Polychronakis M, Monrose F (2019) Sok: The challenges, pitfalls, and perils of using hardware performance counters for security. In: 2019 IEEE Symposium on Security and Privacy (SP), pp 20–38. https://doi.org/10.1109/SP.2019.00021
2. Demme J, Maycock M, Schmitz J, Tang A, Waksman A, Sethumadhavan S, Stolfo S (2013) On the feasibility of online malware detection with performance counters. In: Proceedings of the 40th Annual International Symposium on Computer Architecture. ISCA '13. Association for Computing Machinery, New York, pp 559–570. https://doi.org/10.1145/2485922.2485970
3. Sayadi H, Patel N, Sai Manoj PD, Sasan A, Rafatirad S, Homayoun H (2018) Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification. In: 2018 55th ACM/ESDA/IEEE design automation conference (DAC), pp 1–6. https://doi.org/10.1109/DAC.2018.8465828
4. Tang A, Sethumadhavan S, Stolfo SJ (2014) Unsupervised anomaly-based malware detection using hardware features. In: Stavrou A, Bos H, Portokalidis G (eds) Research in attacks, intrusions and defenses. Springer, Cham, pp 109–129
5. He Z, Rezaei A, Homayoun H, Sayadi H (2022) Deep neural network and transfer learning for accurate hardware-based zero-day malware detection. In: Proceedings of the great lakes symposium on VLSI 2022. GLSVLSI '22, pp 27–32. Association for Computing Machinery, New York. https://doi.org/10.1145/3526241.3530326
6. Singh B, Evtyushkin D, Elwell J, Riley R, Cervesato I (2017) On the detection of kernel-level rootkits using hardware performance counters. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. ASIA CCS '17. Association for Computing Machinery, New York, pp 483–493. https://doi.org/10.1145/3052973.3052999
7. Ozsoy M, Donovick C, Gorelik I, AbuGhazaleh N, Ponomarev D (2015) Malware -aware processors: A framework for efficient online malware detection. In: 2015 IEEE 21st international symposium on high performance computer architecture (HPCA), pp 651–661. https://doi.org/10.1109/HPCA.2015.7056070
8. Sayadi H, Makrani HM, Pudukotai Dinakarrao SM, Mohsenin T, Sasan A, Rafatirad S, Homayoun H (2019) 2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection. In: 2019 design, automation test in europe conference exhibition (DATE), pp 728–733. https://doi.org/10.23919/DATE.2019.8715080
9. Krishnamurthy P, Karri R, Khorrami F (2020) Anomaly detection in real-time multithreaded processes using hardware performance counters. IEEE Trans Inf Forensics Secur 15:666–680. https://doi.org/10.1109/TIFS.2019.2923577
10. Basu K, Krishnamurthy P, Khorrami F, Karri R (2020) A theoretical study of hardware performance counters-based malware detection. IEEE Trans Inf Forensics Secur 15(512–525). https://doi.org/10.1109/TIFS.2019.2924549
11. Sayadi H, Gao Y, Mohammadi Makrani H, Lin J, Costa PC, Rafatirad S, Homayoun H (2021) Towards accurate runtime hardware assisted stealthy malware detection: A lightweight, yet effective time series CNN-based approach. Cryptography 5(4). https://doi.org/10.3390/cryptography5040028
12. Bilge L, Dumitras T (2012) Before we knew it: An empirical study of zero-day attacks in the real world. In: Proceedings of the 2012 ACM Conference on CCS. CCS '12. ACM, New York, pp 833–844
13. Comar PM, Liu L, Saha S, Tan P-N, Nucci A (2013) Combining supervised and unsupervised learning for zero-day malware detection. In: 2013 Proceedings IEEE INFO COM, pp 2022–2030. https://doi.org/10.1109/INFCOM.2013.6567003
14. Gandotra E, Bansal D, Sofat S (2016) Zero-day malware detection. In: 2016 sixth international symposium on embedded computing and system design (ISED), pp 171–175. https://doi.org/10.1109/ISED.2016.7977076
15. Kuruvila AP, Kundu S, Basu K (2020) Analyzing the efficiency of machine learning classifiers in hardware-based malware detectors. In: 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp 452–457. https://doi.org/10.1109/ISVLSI49217.2020.00-15
16. Perf tools support for Intel Processor Trace. https://perf.wiki.kernel.org/index.php/Perf_tools_support_for_Intel%C2%AE_Processor_Trace. Accessed 1 Feb 2024
17. Perf: Linux Profiling with Performance Counters (2017). https://perf.wiki.kernel.org/index.php
18. Reddi VJ, Settle A, Connors DA, Cohn RS (2004) Pin: a binary instrumentation tool for computer architecture research and education. In: Proceedings of the 2004 workshop on computer architecture education: held in conjunction with the 31st international symposium on computer architecture, p 22
19. Mucci PJ, Browne S, Deane C, Ho G (1999) Papi: A portable interface to hardware performance counters. In: Proceedings of the department of defense HPCMP users group conference, vol 710
20. Reinders J (2005) VTune Performance analyzer essentials: measurement and tuning techniques for software developers. Intel Press, Engineer to Engineer Series
21. Performance monitoring events - intel. https://perfmon-events.intel.com/. Accessed 1 May 2023
22. Dementieve R, Willhalm T, Bruggeman O, Fay P, Ungerer P, Ott A, Lu P, Harris J, Kerly P, Konsor P, Semin A, Kanaly M, Brazones R, Shah R, Dobkins J (2022) Intel® performance counter monitor - a better way to measure CPU utilization. https://software.intel.com/content/www/us/en/develop/articles/intel-performance-counter-monitor.html. Accessed 1 May 2023
23. Zhou B, Gupta A, Jahanshahi R, Egele M, Joshi A (2018) Hardware performance counters can detect malware: Myth or fact? In: Proceedings of the 2018 on Asia conference on computer and communications security. ASIACCS '18. Association for Computing Machinery, New York, pp 457–468. https://doi.org/10.1145/3196494.3196515

24. Guthaus MR, Ringenberg JS, Ernst D, Austin TM, Mudge T, Brown RB (2001) Mibench: A free, commercially representative embedded benchmark suite. In: Proceedings of the fourth annual IEEE International Workshop on workload characterization. WWC-4 (Cat. No.01EX538), pp 3–14. https://doi.org/10.1109/WWC.2001.990739
25. Henning JL (2006) Spec cpu2006 benchmark descriptions. SIGARCH Comput. Archit. News 34(4):1–17
26. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in python. J Mach Learn Res 12(null):2825–2830
27. Kraskov A, Stögbauer H, Grassberger P (2003) Estimating mutual information. Physical review. E, Statistical, nonlinear and soft matter physics 69 6 Pt 2:066138
28. Pandas: User Guide. https://pandas.pydata.org/docs/index.html. Accessed 1 May 2023
29. McKinney (2010) Data structures for statistical computing in python. In: Walt M (ed) Proceedings of the 9th python in science conference, pp 56–61. https://doi.org/10.25080/Majora-92bf1922-00a
30. Raff E, Barker J, Sylvester J, Brandon R, Catanzaro B, Nicholas C (2017) Malware detection by eating a whole EXE
31. Shukla S, Kolhe G, Sai Manoj P, Rafatirad S (2019) Work-in-progress: Microarchitectural events and image processing-based hybrid approach for robust malware detection. In: 2019 International Conference on Compliers, Architectures and Synthesis for Embedded Systems (CASES), pp 1–2
32. Pektaş A, Acarman T (2020) Deep learning for effective android malware detection using API call graph embeddings. Soft Comput 24(2):1027–1043. https://doi.org/10.1007/s00500-019-03940-5
33. Kakisim AG, Gulmez S, Sogukpinar I (2022) Sequential opcode embedding-based malware detection method. Comput Electr Eng 98:107703. https://doi.org/10.1016/j.compeleceng.2022.107703
34. Kornblith S, Shlens J, Le QV (2019) Do better ImageNet models transfer better? In: 2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR), pp 2656–2666. https://doi.org/10.1109/CVPR.2019.00277
35. Borisov V, Leemann T, Seßler K, Haug J, Pawelczyk M, Kasneci G (2022) Deep neural networks and tabular data: A survey. IEEE Trans Neural Netw Learn Syst 21:1. https://doi.org/10.1109/TNNLS.2022.3229161
36. Shwartz-Ziv R, Armon A (2022) Tabular data: Deep learning is not all you need. Inf Fusion 81(C):84–90. https://doi.org/10.1016/j.inffus.2021.11.011
37. Tan C, Sun F, Kong T, Zhang W, Yang C, Liu C (2018) A survey on deep transfer learning. In: Kurkova V, Manolopoulos Y, Hammer B, Iliadis L, Maglogiannis I (eds) Artificial neural networks and machine learning – ICANN 2018. Springer, Cham, pp 270–279
38. Sun B, Yang L, Zhang W, Lin M, Dong P, Young C, Dong J (2019) Supertml: Two-dimensional word embedding for the precognition on structured tabular data. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp 2973–2981. https://doi.org/10.1109/CVPRW.2019.00360
39. Bradski G (2000) The OpenCV Library. Dr. Dobb's Journal of Software Tools
40. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR), pp 770–778. https://doi.org/10.1109/CVPR.2016.90
41. Howard J et al (2021) fastai. GitHub. https://github.com/fastai/fastai. Accessed 10 Oct 2023
42. Yosinski J, Clune J, Bengio Y, Lipson H (2014) How transferable are features in deep neural networks? In: Proceedings of the 27th international conference on neural information processing systems - Volume 2. NIPS'14, MIT Press, Cambridge, pp 3320–3328
43. George D, Shen H, Huerta EA (2018) Classification and unsupervised clustering of LIGO data with deep transfer learning. Phys Rev D 97:101501. https://doi.org/10.1103/PhysRevD.97.101501
44. Smith LN (2018) A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay
45. Wang E, Davis JJ, Moro D, Zielinski P, Lim JJ, Coelho C, Chatterjee S, Cheung PYK, Constantinides GA (2023) Enabling binary neural network training on the edge. ACM Trans Embed Comput Syst 22(6). https://doi.org/10.1145/3626100
46. Maaten L, Hinton G (2008) Visualizing data using t-sne. J Mach Learn Res 9(86):2579–2605
47. Tran D, Liu JZ, Dusenberry MW, Phan D, Collier M, Ren JJ, Han K, Wang Z, Mariet ZE, Hu H, Band N, Rudner TGJ, Singhal K, Nado Z, Amersfoort JR, Kirsch A, Jenatton R, Thain N, Yuan H et al (2022) Plex: Towards reliability using pretrained large model extensions. ArXiv abs/2207.07411
48. Lee K, Lee K, Lee H, Shin J (2018) A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In: Proceedings of the 32nd international conference on neural information processing systems. NIPS'18. Curran Associates Inc, Red Hook, pp 7167–7177