Hardware-assisted Ransomware Detection using Automated Machine Learning

Zhixin Pan
Department of Electrical & Computer Engineering
Florida State University

Ziyu Shu
Department of Computer Science
Washington University in St. Louis

***Abstract*—Ransomware has emerged as a severe privacy threat, leading to significant financial and data losses worldwide. Traditional detection methods, including static signature-based detection and dynamic behavior-based analysis, have shown limitations in effectively identifying and mitigating ever-evolving ransomware attacks. In this paper, we present a machine learning-based framework that integrates both software-level scanning along with hardware-level microprocessor activity monitoring to enhance detection performance. Specifically, this paper offers three important contributions. (1) The proposed method incorporates adversarial training to address the weaknesses of conventional static analysis against obfuscation, along with a hardware-assisted dynamic analysis to reduce detection latency. (2) The proposed method employs a neural architecture search (NAS) algorithm to automate the optimization of machine learning models, significantly boosting generalizability. (3) Experimental results demonstrates that our proposed method improves detection accuracy by up to 10.2% and reduces detection latency by up to 4.8x speedup compared to existing approaches.**

I. Introduction

Cybersecurity has been one of the major concerns since the development of computers and networking technologies. Ransomware, a special type of malware attack characterized by its ability to encrypt critical data and demand ransom for its release, has become one of the most pervasive threats in data privacy. Figure 1 shows the common process of ransomware infection. The first phase focus on initial preparation and lay the groundwork for subsequent actions, including registration for persistent execution, loading encryption algorithms, as well as scanning the victim's system to identify target files to encrypt. The second phase typically involves a fast repetitive sequence of file encryption and display of the ransom demand.

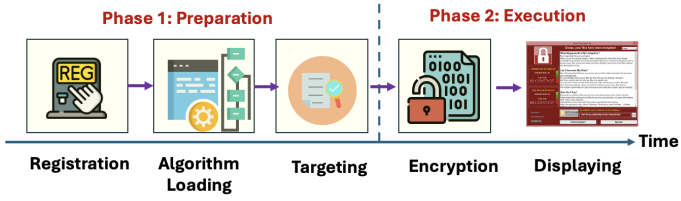


Fig. 1: The typical workflow of ransomware infection.

The rise of ransomware attacks has led to substantial financial losses. According to the study in [1], recent ransomware attackers have shifted their targets from individuals to larger corporations, resulting in higher ransom profits up to approximately $6 trillion. The study also pointed out the fact that ransomware attacks are not only growing in number but also leveraging more advanced evasion techniques to evade traditional detection mechanisms. As a result, there is an urgent demand for effective ransomware detection approach.

There are various works in the literature for ransomware detection, and they can be broadly classified into two categories: static analysis and dynamic analysis. Static analysis focuses on examining the source code or executable files before execution using signature-based patterns recognition. However, its effectiveness is limited by inability to account for runtime variables and suffer from high false positive rates (FPRs) [1]. Moreover, some evasive ransomware variants can bypass static analysis by inserting benign code to obscure program patterns [2]. Dynamic analysis, or behavioral analysis, monitors the application's behavior during runtime instead. It focus on capturing activities like file encryption and access patterns. Though dynamic methods provide promising accuracy, they often suffer from detection latency while ransomware detection is a time-critical task.

To address these challenges, we propose a hardware-assisted detection framework using machine learning that combines both static and dynamic analysis. Notice that prior to our proposed method, there are existing hybrid methods combining both category of approaches, and attempt to leverage the strengths of both sides. However, these methods face challenges to resolve conflicting results, and relying heavily on expert knowledge for model architecture design, leading to limited generalizability [1]. In response to these limitations, our approach offers several key innovations:

- **Multi-stage detection**: The proposed method perform both static analysis and dynamic behavior analysis in a hierarchical way to resolve conflicts. Dynamic monitoring is only activated if suspicious features are identified by static analysis, thereby reducing unnecessary overhead.
- **Hardware features**: The proposed method leverages multiple hardware features to improve efficiency. Both hardware performance counters (HPC) and embedded trace buffers (ETB) features are utilized by ML model to minimize detection latency and achieve early-termination of ransomware.
- **Automated model selection**: A neural architecture search (NAS) algorithm is applied to automatically optimize machine learning model architecture, which significantly enhances the detection system's generalizability across different ransomware variants.

978-3-9826741-0-0/DATE25/© 2025 EDAA

II. Background and Related Work

The study of detecting ransomware attacks have been increasing throughout recent years. As discussed in Section I, the majority of existing ransomware prevention techniques can be classified into two categories, static and dynamic analysis.

Static analysis is performed prior to the execution of a program, where the program's executable file or source code is examined to identify malicious content. For instance, studies such as [3]–[5] have employed deep file scanning techniques to prevent ransomware infections, while works in [6]–[8] applied linguistic analysis to the program's code for early detection. However, static analysis often suffers from high false positive rates (FPRs), since there are benign programs with similar binary patterns as ransomware (e.g, disk encryption programs), they are very likely to be misclassified as ransomware. Moreover, most static analysis methods are signature-based, detecting ransomware by matching binary/API call patterns. If the attacker intentionally obscures or camouflages the malicious code [9], signature-based tools often fail to detect it.

As for dynamic analysis, it involves executing the suspected ransomware to monitor its behavior in real-time. For example, [10] utilized the runtime performance traces to recognize high-frequency encryption behavior, leading to fast ransomware detection. In [11], the authors explored a detection framework based on dynamic changes of portable executable headers. Several other studies, such as [12], [13], have also proposed machine learning-based approaches to enhance the performance of dynamic analysis. Though achieving promising detection accuracy, they still face significant challenges. First, the majority of these methods did not prioritize the necessity of early-termination of ransomware, and often fail to specify how long it takes to terminate ransomware execution. Some approaches with long observation periods offer ransomware sufficient time to encrypt large amount of data regardless of detection outcome. Second, these approaches are typically tested on specific types of ransomware, limiting their generalizability. Figure 2 plots the performance of several existing algorithms across four ransomware variants, namely *WannaCry*, *Vipasana*, *Locky*, and *Petya*. The tested algorithms show inconsistent performance where detection rates often dropping below 70%.

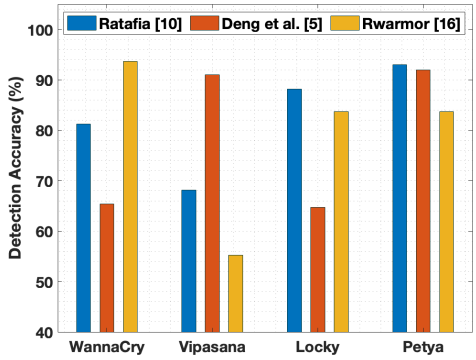


Fig. 2: The detection accuracy using various methods against four different ransomware variants.

There are also a few prior efforts using hybrid methods combining both static and dynamic analysis. For example, a two-stage ransomware detection methods was proposed in [14], which, however, is without early-termination of ransomware infection. In [15], a hybrid algorithm that focus on API call patterns was proposed. But the paper does not provide any insight into the frangibility of pattern-recognition against obfuscation. Another hybrid analysis approach, Rwarmor, was proposed in [16] which adopts the idea of static-informed dynamic analysis, but it still suffers from low generalizability issue, which will be further demonstrated in Section IV.

III. Proposed Method

A. Overview

To highlight the motivation for our proposed method, we summarize the key limitations of existing anti-ransomware techniques discussed in Section II, as follows:

- Vulnerability to obfuscation.
- Long detection latency.
- Low generalizability among variants.

To address these challenges, our proposed approach enables a synergistic integration of robust static detection framework and hardware-assisted dynamic analysis for efficient detection. Figure 3 presents an overview of the proposed workflow, modeled as a finite state machine (FSM).

In the first stage, the target program undergoes static analysis for early-stage detection. If a program is flagged as 'suspicious' at this step, it is passed to the dynamic analysis phase. This two-stage sequential framework offers several key advantages: it mitigates the issue of high false positive rates in conventional static analysis by labeling programs as suspicious rather than directly classifying them as malicious. Additionally, only programs flagged in the first stage proceed to runtime monitoring, reduceing unnecessary computational overhead. Next, based on the results of dynamic analysis, the program is either restored to its original state if deemed benign, or, if identified as malicious, our method terminates its execution and restores encrypted files using backups.

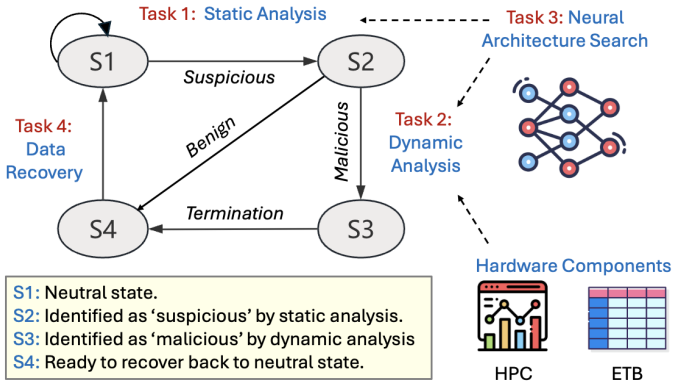


Fig. 3: State transitions of proposed framework.

The implementation of the workflow described above can be divided into four primary tasks:

- A **static analysis** phase, utilizing a boosted decision tree (DT) for early detection, where adversarial training

is applied to mitigate the vulnerability to obfuscation. (Section III-B)

- A **hardware-assisted dynamic analysis** phase, leveraging long-short term memory (LSTM) model to monitor runtime behavior with features from hardware components, including hardware performance counters (HPC) and embedded trace buffer (ETB).(Section III-C)
- A **neural architecture search** (NAS) algorithm to automate the selection and optimization of machine learning models for the first two tasks. (Section III-D)
- A post-detection task focuses on **data recovery**, where backups files are used to restore encrypted data. (Section III-E)

B. Adversarial training based Static analysis

The first task of proposed method is static analysis, and the workflow is illustrated in Figure 4. Initially, static features are extracted from the target programs. To enable a robust detection, our model examines multiple critical features, including API calls, opcode sequences, file metadata, along with PE File Header to capture malicious patterns. We split the benchmark dataset into training set and validation set with a ratio of 80/20. Next, we train the initial decision tree model using CART [17] method to ensure an effective baseline performance, then test it on validation set to identify misclassified instances (falsified labels). To strengthen the model's resilience against obfuscation, we further generate crafted adversarial samples by inserting benign code to obscure program patterns in order to simulate obfuscated ransomware. Both mislabeled and adversarial samples are then utilized for adversarial training, reducing model's susceptibility to obfuscation.

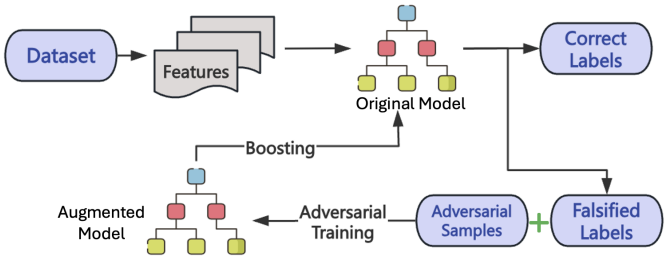


Fig. 4: The adversarial training based static analysis.

One key challenge in adversarial training is the difficulty of convergence due to the complexity of optimizing under adversarial conditions. To address this, our method leverages boosting [18] framework, where we build a sequence of lightweight models, $\mathcal{M} = \{\mathcal{T}_1, \mathcal{T}_2, ...\mathcal{T}_N\}$ with associating weight values $\{w_1, w_2, ..., w_N\}$s. In adversarial training phase, we construct a new augmented tree $\mathcal{T}_i$ that focuses specifically on the misclassified and adversarial examples, ensuring that the model prioritizes correcting previous errors. Then we incorporate the new tree into $\mathcal{M}$ and adjust their weights based on respective performance. The prediction of this aggregated model is eventually determined by a weighted voting among all decision trees. This strategy allows gradual improvement of robustness, while maintaining faster convergence. The algorithm is illustrated in Algorithm 1.

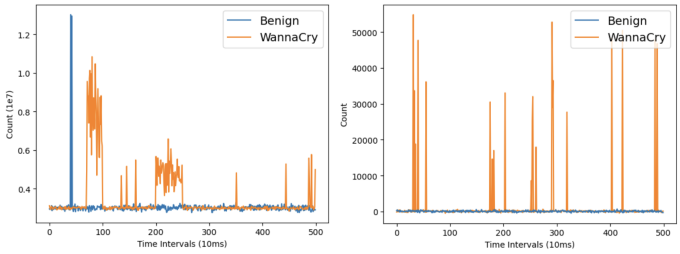
Algorithm 1: Adversarial training of boosting DTs

Input : Dataset($\mathcal{D}$), epochs (k)
Output: Model $\mathcal{M}$
Initialize: $\mathcal{M} = \emptyset$, $k = 0$
repeat
  Split $\mathcal{D}$ into training set $\mathcal{D}_t$ and validation set $\mathcal{D}_v$
  $\mathcal{T}_{(k)} = \text{train}(\mathcal{D}_t)$ ▷ Model training
  $\mathcal{D}' = \text{validate}(\mathcal{T}_{(k)}, \mathcal{D}_v)$ ▷ Falsified labels
  $\hat{\mathcal{D}} = \text{craft}(\mathcal{D})$ ▷ Craft adversarial samples
  $\mathcal{T}_{(k+1)} = \text{train}(\mathcal{D}' \cup \hat{\mathcal{D}})$ ▷ Adversarial training
  $\mathcal{D} = \mathcal{D} \cup \mathcal{D}' \cup \hat{\mathcal{D}}$ ▷ Dateset augmentation
  $\mathcal{M} = \mathcal{M} \cup \{\mathcal{T}_{(k)}\}$
  for *each* $\mathcal{T}_i \in \mathcal{M}$ **do**
    $acc_i = 1 - |\text{validate}(\mathcal{T}_{(k)})|/|\mathcal{D}|$
    $w_i = \frac{acc_i}{\sum_{i=1}^{N} acc_i}$ ▷ Weight Adjustment
  $k = k + 1$
until $k \geq max\_epoch$ *or convergence*;
Return $\mathcal{M}$

C. Hardware-assisted dynamic analysis

Once a target program is flagged as "suspicious" during the static analysis phase, it is prompted to dynamic analysis for a more thorough investigation. To achieve early termination, we incorporate hardware components to reduce the latency, as they can dump low-level information in a realtime manner, offering a significant speed advantage over software-based methods.

The first hardware component we use is hardware performance counters (HPCs), a set of built-in registers that monitor hardware events such as instruction counts, cache references, and branch misses. Notably, there is a natural correlation between certain hardware events and ransomware activities. For example, Figure 5 presents the difference between a benign program and WannaCry in terms of two HPC events, branch misses and cache misses. Obviously, substantial discrepancies can be observed. Intuitively, WannaCry's use of asymmetric encryption algorithms affects branching events, as well as its frequent file access patterns causes periodic cache miss spikes.



(a) Branch Misses (b) Cache Misses

Fig. 5: Comparison of HPC events.

Despite its utility, using HPCs to count and collect hardware events still introduces response delays. To overcome this limitation, we also integrate embedded trace buffers (ETBs) into our framework. ETBs capture realtime, cycle-by-cycle on-chip trace for designated process, offering more granular insights and faster speed than HPCs.

Nonetheless, detection with ETB comes at the cost of increased system overhead, making continuous monitoring with ETBs impractical. In order to capitalize on the strengths of both HPCs and ETBs, we adopt a hybrid realtime detection scheme in a sliding-window way. The program's execution is divided into time windows, during which different monitoring strategies are applied alternatively. Specifically, Figure 6 presents a typical workflow consisting of the following major activities:

- **Initial monitoring with ETBs**: In the initial window, we use ETBs for monitoring, since the program has already been deemed suspicious by the static analysis. The assumption is that it has a high likelihood of being ransomware, so ETBs are employed to detect any malicious behavior at earliest stage.
- **Adaptive monitoring with HPCs**: If the program passes the first ETB-based window without exhibiting malicious behavior, we assume it may either be benign or a ransomware using delayed infection tactics. To reduce the overhead of continuous ETB monitoring, we switch to HPC-based detection for the next window. The HPCs-based detection shall be extended to next window if no malicious behavior detected.
- **Proof monitoring with ETBs**: If HPC-based monitoring detects suspicious behavior in any window, ETBs are reactivated in the following window to confirms ransomware activity and quickly terminate the program. This secondary double-check helps avoiding false positives.
- **Rollback**: If ETB monitoring contradicts the HPCs' result in the subsequent window, we interpret this either as a false positive or as an attempt of the ransomware to evade detection through obfuscation. Either way, we roll back the possibly encrypted files to the previous state using backups, and switch back to HPC-based monitoring to reduce overhead.

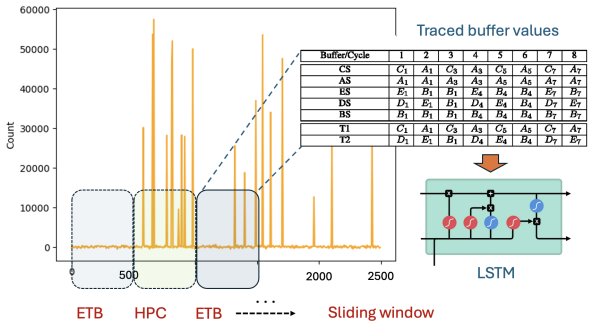


Fig. 6: The sliding-window realtime hardware events monitoring using HPC and ETB in an alternative way.

We have selected a window size of 500 ms to provide a balance between timely ransomware detection and minimal resource consumption. To effectively handle the sequential data collected from by HPCs and ETBs, we employ a long-short term memory (LSTM) model [19] as our machine learning classifier. The design of the model architecture is discussed in the following section.

D. Neural architecture search

Another key challenge in ransomware detection is the lack of generalizability across variants. We interpret this problem as a result of manually selected architecture and hyperparameters which lack flexibility.

To address this issue, we propose the use of Neural Architecture Search (NAS) to automatically design the model architecture. NAS is a type of automated method that searches for optimal architectures by exploring a search space of potential neural network configurations. It evaluates subsets within the search space, experimenting with various combinations of layers, resulting in an architecture that is fine-tuned to given task. In our work, we employ one-shot NAS to construct the optimal architecture, which proceeds in two main stages:

- **Supernet Training**: We start by constructing a supernet, an over-parameterized model that encompasses all potential architectural configurations, forming the search space. At each level, the supernet includes a variety of layer types, such as LSTM cells, fully connected layers, convolution, max_pooling, and activation functions.
- **Pruning**: The supernet is trained on the given dataset. During each iteration, the NAS algorithm identifies the less important components of the network by zeroing out nodes or connections and evaluating their impact on the output. Redundant parts are those that contribute minimally to the model's output and will be pruned.

An illustrative example is shown in Figure 7. This iterative pruning process progressively streamlines the architecture, producing a model that is both lightweight and highly efficient.

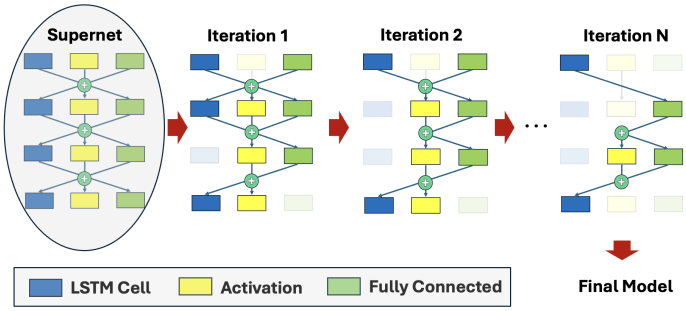


Fig. 7: Illustration of one-shot NAS for building model. We start with the over-parameterized supernet. At each training iteration, redundant nodes and connections are pruned (faded). The process continues until convergence.

E. Data Recovery

Although our proposed model allows for real-time detection and significantly reduces latency, some files may still inevitably get encrypted by ransomware. To mitigate this impact, we integrate a dynamic file backup and recovery mechanism that aligns with our detection model. This recovery program operates synchronously with the sliding-window dynamic monitoring process. During each time window, the system identifies recently accessed files and creates backup copies using system command (*rsync* in Linux os). If no ransomware

activity is detected during the window, the backup files are automatically deleted to conserve storage space. Otherwise, these backup files are used to restore any compromised data at the end of the time window.

IV. Experimental Evaluation

A. Experiment Setup

Benchmark: We conducted performance evaluations on a standard host Linux machine . Four ransomware variants *WannaCry*, *Vipasana*, *Locky*, and *Petya* were selected as benchmarks, and the SPEC integer benchmark [20] were used to present benign programs. Totally, the dataset is comprised of 500 program samples with evenly distributed labels. We also apply obfuscation techniques to craft adversarial samples. During program execution, hardware performance counter (HPC) values were collected using the *perf* tool at a sampling interval of 500ms, and embedded trace buffer (ETB) values were dumped via the UART port.

Implementation: The machine learning framework was implemented on the same host machine equipped with an Intel i9 3.70GHz CPU, 64 GB RAM, and an RTX 4080 GPU. We used PyTorch as the machine learning library. We train 200 epochs for the decision tree-based static analysis model, and 500 epochs for the LSTM model. The learning rate for the LSTM model was adjusted from 0.05 to 0.1 after 200 epochs to accelerate convergence. The dataset was split 80/20 for training and validation, and cross-validation was applied to reduce bias.

Approaches: To enable fair comparison, we compare the performance and efficiency of the following four approaches.

- **DRL:** A deep reinforcement learning based static analysis method [5].
- **Ratafia:** A dynamic analysis method using HPC values informed auto-encoders [10].
- **Rwarmor:** A hybrid detection framework using static-informed dynamic analysis [16].
- **Proposed:** The proposed hardware-assisted multistage detection method.

B. Detection Accuracy

Table I presents the performance comparison between proposed detection method and existing approaches across various ransomware benchmarks. We evaluated all four methods using accuracy (Acc), false positive rate (FPR), false negative rate (FNR), and F1 score (F1). Each row represents a specific ransomware benchmark. As we can see, DRL achieves an average accuracy of 78.5%, while Ratafia is at 83.9%. Both methods fall short compared to our proposed approach due to inherent limitations in conventional techniques. DRL, being a static analysis method, suffers from a high false positive rate, resulting in the highest FPR among the four methods. Ratafia as a dynamic analysis approach, relies solely on HPC events without accurate information from ETB, leading to a relatively high FNR. Rwarmor faces an even greater challenge, managing only 58.2% accuracy on Vipasana. This is due to Vipasana's unique behavior of encrypting files offline, without communication with control servers, making its dynamic behavior distinct from other ransomware. In contrast, our proposed method addresses this challenges using NAS (illustrated further in Section IV-E), achieving an average accuracy of 94.1%, representing up to a 10.2% improvement over existing approaches.

C. Case study: Adversarial Samples

We further generated adversarial samples by inserting benign and non-functional instructions into ransomware programs to obfuscate their behavior patterns, and assess the robustness of each method. The results are presented in Figure 8. As observed, most methods experience a significant decline in performance when exposed to adversarial samples. Notably, the accuracy of both DRL and Rwarmor occasionally drops below 60%, approaching the level of random guessing. DRL, as a signature-based detection method that relies on matching portable executable headers, and Rwarmor, a pattern recognition based framework, both struggle to differentiate between obfuscated ransomware and legitimate programs. In contrast, with the enhancement from adversarial training, our proposed method maintains an average accuracy of 85%.

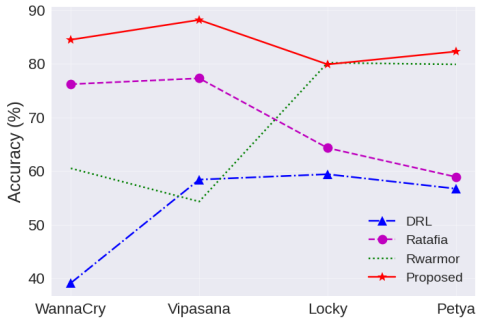


Fig. 8: Detection performance against adversarial samples

D. Case study: Detection Latency

Another critical performance metric is detection latency, which is the time elapsed from the beginning of encryption behavior by ransomware to its detection. This latency is directly proportional to the number of files encrypted before discovery. Table II records the average detection latency across all tested random samples. As indicated, our approach outperforms all other methods. DRL exhibits the slowest detection due to the computational overhead required by its reinforcement learning framework. While Ratafia demonstrates slightly better efficiency, our method achieves significantly lower latency than it. We attribute the improvement to the sliding-window monitoring technique with hardware assistance (Section III-C).

TABLE II: The comparison of detection latency.

Methods	DRL	Ratafia	Rwarmor	Proposed	Improv
Delay(ms)	49984	13480	29670	2784	4.8x

Specifically, Figure 9 presents detection processes of 6 samples by depicting their window-based monitoring procedures, and an upper histogram tracking total detection latency was attached. Notably, for sample 1 and 5, our method successfully detects ransomware in the first window through ETB-based

TABLE I: The comparison of detection performance.

Methods	DRL [5]				Ratafia [10]				Rwarmor [16]				Proposed				
Benchmarks	**Acc**	**FPR**	**FNR**	**F-1**	**Acc**	**FPR**	**FNR**	**F-1**	**Acc**	**FPR**	**FNR**	**F-1**	**Acc**	**FPR**	**FNR**	**F-1**	**Improv(%)**
WannaCry	67.1	17.5	15.4	0.62	82.3	9.3	8.4	0.90	93.9	4.1	2.0	0.97	95.2	2.7	2.1	0.98	1.3
Vipasana	91.3	5.8	2.9	0.96	73.1	13.5	13.4	0.84	58.2	12.2	29.6	0.73	94.1	3.6	2.3	0.97	3.2
Locky	65.8	25.2	9.0	0.79	88.4	4.7	6.9	0.94	83.1	9.2	7.7	0.91	93.2	3.8	3.0	0.94	4.8
Petya	89.9	7.4	2.7	0.94	92.1	2.8	5.1	0.96	82.4	8.2	7.6	0.91	94.1	3.3	2.6	0.96	2.0
Average	78.5	13.9	7.6	0.83	83.9	7.6	8.4	0.91	79.4	10.5	10.1	0.88	94.1	2.9	3.0	0.96	10.2

analysis. Even for ransomware variants employing delayed-infection strategies, such as sample 3, our method identifies the threat within six monitoring blocks by leveraging the combined HPC-ETB framework alternatively.

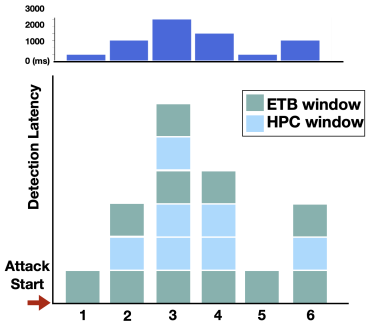


Fig. 9: Detection latency of 6 sample ransomware using sliding-window based dynamic analysis with HPC and ETB.

E. Case study: Generalizability

In this section, we closely examine the generalizability problem in ransomware detection. As presneted in Figure 2 and Table I, existing methods display inconsistent performance across different variants. To understand the reason, we visualized several selected HPC feature values in 3D space (Figure 10), where we could observe distinct feature clusters for each ransomware variant. For example, Vipasana exhibits unique patterns with low branch misses and low-level cache (LLC) references, due to its unique working pattern as discussed in Section IV-B. This highlights the challenge of designing a machine learning model that is sufficiently flexible to capture diverse feature patterns across different ransomware types, where existing methods relying on manually designed architecture often failed.

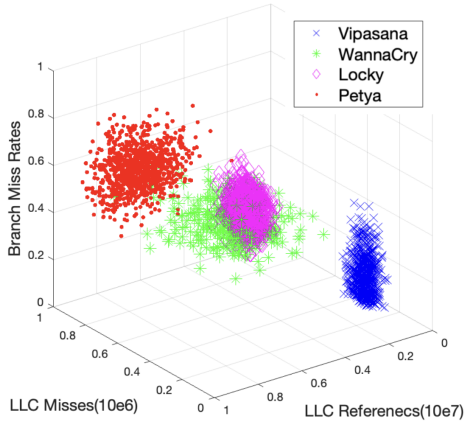


Fig. 10: Visualization of selected HPC feature values.

In contrast, the proposed method utilizes neural architecture search (NAS) to automatically generate the optimized model. Figure 11 illustrates a simplified version of this NAS-generated model. Intuitively, this model works by using blocks 0-2 as an initial "classifier" to roughly determine the ransomware variant, followed by different combination of layers that handle specific feature patterns subsequently, enabling effective detection across various ransomware types. Though seems intuitive, manually designing such an architecture would demand significant effort and domain knowledge. In contrast, our method achieves this automatically, without requiring manual intervention or extra cost.

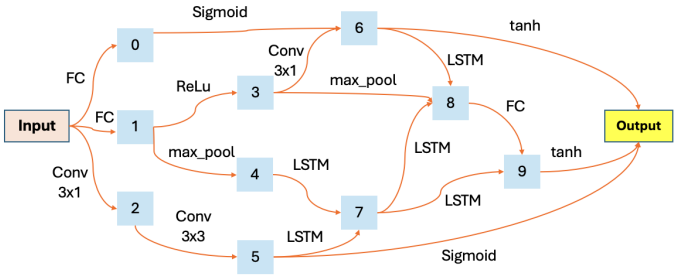


Fig. 11: A simplified demonstration of our model generated from neural architecture search (NAS) algorithm. Block 0-2 are working as initial classifiers to allocate different combination of subsequent layers based on different ransomware variants, in an purpose to capture distinct feature patterns.

V. Conclusion

Ransomware has become a significant threat to cybersecurity in recent years. Existing ransomware detection methods suffer from three major limitations: vulnerability to obfuscation, detection latency, and lack of generalizability. To address these limitations, this paper presents a novel machine learning-based framework for ransomware detection that combines software-level scanning along with hardware-level monitoring to enhance detection accuracy and reduce latency. By incorporating adversarial training, we strengthen static analysis against obfuscation techniques, while the use of sliding-window approach with HPCs and ETBs ensures real-time monitoring with minimal detection latency. Additionally, we utilize one-shot Neural Architecture Search (NAS) to automate the optimization of model architectures, enabling flexibility across ransomware variants. The experimental results show that our method achieves up to 10.2% improvement in detection accuracy and a reduction in latency by up to 4.8 times compared to existing approaches.

REFERENCES

[1] T. McIntosh, A. Kayes, Y.-P. P. Chen, A. Ng, and P. Watters, "Ransomware mitigation in the modern era: A comprehensive review, research challenges, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–36, 2021.

[2] M. N. Olaimat, M. A. Maarof, and B. A. S. Al-rimy, "Ransomware anti-analysis and evasion techniques: A survey and research directions," in *2021 3rd international cyber resilience conference (CRC)*. IEEE, 2021, pp. 1–6.

[3] B. A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, "Crypto-ransomware early detection model using novel incremental bagging with enhanced semi-random subspace selection," *Future Generation Computer Systems*, vol. 101, pp. 476–491, 2019.

[4] H. Faris, M. Habib, I. Almomani, M. Eshtay, and I. Aljarah, "Optimizing extreme learning machines using chains of salps for efficient android ransomware detection," *Applied Sciences*, vol. 10, no. 11, p. 3706, 2020.

[5] X. Deng, M. Cen, M. Jiang, and M. Lu, "Ransomware early detection using deep reinforcement learning on portable executable header," *Cluster Computing*, vol. 27, no. 2, pp. 1867–1881, 2024.

[6] A. Alzahrani, A. Alshehri, H. Alshahrani, R. Alharthi, H. Fu, A. Liu, and Y. Zhu, "Randroid: Structural similarity approach for detecting ransomware applications in android platform," in *2018 IEEE International Conference on Electro/Information Technology (EIT)*. IEEE, 2018, pp. 0892–0897.

[7] A. Cimitile, F. Mercaldo, V. Nardone, A. Santone, and C. A. Visaggio, "Talos: no more ransomware victims with formal methods," *International Journal of Information Security*, vol. 17, pp. 719–738, 2018.

[8] A. Karimi and M. H. Moattar, "Android ransomware detection using reduced opcode sequence and image similarity," in *2017 7th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 2017, pp. 229–234.

[9] S. Lee, S. Lee, J. Park, K. Kim, and K. Lee, "Hiding in the crowd: ransomware protection by adopting camouflage and hiding strategy with the link file," *IEEE Access*, 2023.

[10] M. Alam, S. Bhattacharya, S. Dutta, S. Sinha, D. Mukhopadhyay, and A. Chattopadhyay, "Ratafia: Ransomware analysis using time and frequency informed autoencoders," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2019, pp. 218–227.

[11] M. T. Nguyen, V. H. Nguyen, and N. Shone, "Using deep graph learning to improve dynamic analysis-based malware detection in pe files," *Journal of Computer Virology and Hacking Techniques*, vol. 20, no. 1, pp. 153–172, 2024.

[12] S. Gulmez, A. G. Kakisim, and I. Sogukpinar, "Xran: Explainable deep learning-based ransomware detection using dynamic analysis," *Computers & Security*, vol. 139, p. 103703, 2024.

[13] J. A. Herrera-Silva and M. Hernández-Álvarez, "Dynamic feature dataset for ransomware detection using machine learning algorithms," *Sensors*, vol. 23, no. 3, p. 1053, 2023.

[14] J. Hwang, J. Kim, S. Lee, and K. Kim, "Two-stage ransomware detection using dynamic analysis and machine learning techniques," *Wireless Personal Communications*, vol. 112, no. 4, pp. 2597–2609, 2020.

[15] F. Mercaldo, "A framework for supporting ransomware detection and prevention based on hybrid analysis," *Journal of Computer Virology and Hacking Techniques*, vol. 17, no. 3, pp. 221–227, 2021.

[16] M. A. Ayub, A. Siraj, B. Filar, and M. Gupta, "Rwarmor: a static-informed dynamic analysis approach for early detection of cryptographic windows ransomware," *International Journal of Information Security*, vol. 23, no. 1, pp. 533–556, 2024.

[17] R. J. Lewis, "An introduction to classification and regression tree (cart) analysis," in *SAEM*, vol. 14, 2000.

[18] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[19] A. Graves and A. Graves, "Long short-term memory," *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

[20] "Spec integer benchmark." www.spec.org/benchmarks.html.