Statistical Profiling of Micro-Architectural Traces and Machine Learning for Spectre Detection: A Systematic Evaluation

1st Mai AL-Zu'bi
Institute of Logic and Computation
TU Wien, Vienna, Austria
mai.al-zubi@tuwien.ac.at

2nd Georg Weissenbacher
Institute of Logic and Computation
TU Wien, Vienna, Austria
georg.weissenbacher@tuwien.ac.at

***Abstract*—Spectre attacks exploit features of modern processors to leak sensitive data through speculative execution and shared resources (such as caches). A popular approach to detect such attacks deploys Machine Learning (ML) to identify suspicious micro-architectural patterns. These techniques, however, are often rather ad-hoc in terms of the selection of micro-architectural features as well as ML techniques and frequently lack a description of the underlying training- and test-data.**

To address these shortcomings, we systematically evaluate a large range of (combinations of) micro-architectural features recorded in up to 40 Hardware Performance Counters (HPCs), as well as multiple ML algorithms on a comprehensive set of scenarios and datasets. Using statistical methods, we rank the HPCs used to generate our dataset, which helps us determine the minimum number of features required for detecting Spectre attacks with high accuracy and minimal overhead. Furthermore, we identify the best-performing ML classifiers, and provide a comprehensive description of our data collection, running scenarios, selected HPCs, and chosen classification models.

I. Introduction

The Spectre vulnerability [1] has resulted in some fundamental problems with modern hardware architectures: optimizations such as branch prediction and speculative execution allow an attacker to perform operations that would not occur during regular execution, and subsequently leak confidential information via side-channels (e.g., the cache). Short of deactivating these features entirely (at the cost of a substantial performance penalty), preemptive mitigation techniques in hardware and software remain elusive [2]. A popular alternative approach is to use ML to detect attacks at runtime by identifying suspicious micro-architectural behavior. Micro-architectural events such as cache eviction or branch misprediction are reflected in Hardware Performance Counters (HPCs), special-purpose registers found in all modern processors. Originally, HPCs were designed to monitor hardware-related events and provide real-time data on processor operations. Notably, this data can also be used to identify behavioral patterns indicative of a Spectre attack. This application of HPCs, however, is hampered by the fact that other applications may affect the same counters as an ongoing Spectre attack, thus increasing the false-positive rate.

Consequently, reliability and accuracy of the detection are determined by (the configuration of) the ML model and the selection of micro-architectural features that are being monitored.

Method	HPCs	Polling Rate	ML	Accuracy	Size
Abraham et al. [3]	1	100 ms	2	95%	N/A
Philipp et al. [4]	3	100 ms	1	99.23%	1564
Tong et al. [5]	4	100 ms	4	99.88%	186994
Congmiao et al. [6]	4	100	5	99.98 %	2400
Congmiao et al. [7], [8]	4	100 ms	3	99.23%	2400
Ahmad et al. [9]	5	100 ms	5	99.98%	100K
Oshana et al. [10]	6	N/A	6	N/A	16,000
Mushtaq et al. [11]	12	100 ms	12	99.99%	1M
Sayad et al. [13], [14]	16	N/A	5	100%	N/A
Wang et al. [12]	16	N/A	6	99.8%	N/A

TABLE I: Literature on HPC and ML-based Spectre detection

Existing work (see subsection I-A) often approaches this problem in a somewhat ad-hoc and haphazard way. Our contribution is to *systematically evaluate and identify the best-performing combinations of HPCs and ML algorithms for detecting Spectre attacks* with respect to effectiveness, computational efficiency, and implementation complexity. We comprehensively evaluate the effectiveness of up to 40 HPCs for Spectre detection by systematically assessing these HPCs under a variety of benign and malicious scenarios. We evaluate 10 ML algorithms (using standard metrics such as accuracy, precision, and score time) over these HPCs and their combinations and identify the most suitable HPCs and best-performing ML algorithms. Furthermore, we identify a small number of HPCs that strike a good balance between accuracy and complexity, resulting in a highly accurate detection model with minimal overhead.

A. State-of-the-Art

The detection of Spectre attacks using HPCs and ML algorithms has been the topic of numerous publications [3]–[14]

Abraham et al. [3], for instance, achieved up to 95% accuracy using a *single* HPC. Philipp et al. [4] utilized 3 HPCs to develop a detection model that detects two types of Spectre attacks with 99.23% accuracy. Ahmad et al. [9] and Congmiao et al. [6] demonstrated an accuracy of 99.98% using 5 and 4 HPCs, respectively, and Mushtaq et al. [11] achieved 99.99% by employing 12 HPCs. Sayad et al. [13], [14] achieved an astounding accuracy of up to 100% using 16 HPCs.

Table I summarizes these efforts. Notably, there is a large variation regarding the chosen HPCs (ranging from a single one to 16), the employment of ML models (varying from

a single to a comprehensive set of 12 ML algorithms), as well as workload scenarios and their sizes. Notably, some studies (such Congmiao et al. [6] and Philipp et al. [4]) rely on extremely small datasets. These variations complicate a meaningful comparison of these prior studies, calling for a comprehensive evaluation of these parameters in a single study.

B. Research Gap

Based on subsection I-A, we identified the following research gap which is addressed by our work:

1) **Limited Investigation of HPCs:** The studies in Table I focus on a limited set of HPCs. To our knowledge, no single paper has systematically compared HPCs; i.e., we lack an exhaustive evaluation of the suitability of (combinations of) HPCs for effective Spectre detection.
2) **Lack of Dataset Description:** Prior work often fails to explain why a specific combination HPC was chosen for the experiments or fails to provide a description of the evaluated scenarios and their generation. This prevents reproducibility and comparability of results across prior studies, necessitating a clear and detailed description of the underlying training and test datasets.
3) **Best-Performing Models:** Despite ML playing a significant role in this context, whether one [4] or multiple algorithms are deployed and evaluated [3], [6], [8]–[16], there is still a need to *systematically* select the best-performing algorithms across different datasets.

C. Research Contributions

To address this gap, our research offers a comprehensive investigation of a large variety of workload scenarios, micro-architectural features, and ML algorithms. Thus, it can serve as a foundation for future research in this field and provide a holistic understanding of the challenges in dataset generation, validation, and ML integration:

- We investigate the impact of micro-architectural traces from *different* HPCs on Spectre detection accuracy
- We increase the confidence in ML-based Spectre detection through a systematic experimental evaluation
- We evaluate a variety of ML algorithms to assess their suitability for Spectre detection
- We provide a reconfigurable dataset generator for micro-architectural traces, allowing efficient recording of different combinations of features (HPCs)
- We generate diverse datasets for a variety of benign and malicious scenarios, facilitating the training and evaluation of the proposed detection models
- We contribute towards reducing the overhead of Spectre detection methods
- We discuss significant factors influencing detection, allowing researchers to choose the optimal configurations for their detection models

II. Background

A. Speculative Execution

Modern processors deploy speculative execution techniques for optimization, performing computations tentatively in advance based on the probability that they lie on a program path that will eventually be executed. Speculative execution can be exploited to leak information and access sensitive data, triggering ongoing efforts to enhance mitigation techniques against vulnerabilities such as Spectre [1] and Meltdown [17].

B. Cache Memory

Caches are small, fast, and local memory cells that speed up access to frequently used data. Contemporary processors have multiple levels of cache memory, including per-core L1 instruction and data caches, per-core unified L2 caches, and a large L3 cache (also known as Last Level Cache). While caches speed up performance, they have also long been known to facilitate side-channel attacks (SCAs) that exploit the performance difference between cache hits and misses [18].

C. Side-Channel Attacks and Spectre

SCAs are complex cyberthreats that take advantage of unexpected data leaks in computer systems, where both attacker and victim may share resources. A prominent class of SCAs are cache-based attacks, which typically comprise three phases: initialization, encoding, and measurement. During initialization, the attacker resets the cache to a predictable state; in the encoding phase, the victim code performs cache accesses that depend on the secret the attacker wants to obtain; and finally, during the measurement phase, the attacker accesses specific cache blocks to measure the access time and determine which cache lines the victim code used. SCAs have commonly utilized some tactics to extrapolate the secret, including Flush+Reload [19], Evict+Time [20], and Prime+Probe [21]. Several attacks, such as Spectre [1], Meltdown [17], and Foreshadow [22] exploit these techniques, The major difference between them is how the secret is imported from the main memory to the cache. In the case of Spectre [1], these leaks are facilitated by cleverly combining speculative execution, branch prediction, and cache-based side channels. In order to trick victims into exposing their secrets, Spectre follows the following steps:

1) The attacker forces the victim to execute misleading program paths to trick the CPU into making incorrect branch predictions.
2) The attacker performs flush instructions to clear the cache, in order to force the program to retrieve any data from the main memory rather than the cache.
3) The attacker tricks the victim into performing speculative operations on sensitive (secret) data.
4) Since the cache is initially empty, this secret data is retrieved from the main memory.
5) The processor then realizes its misprediction and discards the results of the speculative computation; however, its side effects on the cache remain visible to the attacker.
6) The attacker decodes the secret using a high-precision timer to determine which values were read from main memory, exploiting that a *cache miss* takes longer than reading data from the cache (a *cache hit*).
7) Thus, the attacker retrieves a secret value from the cache.

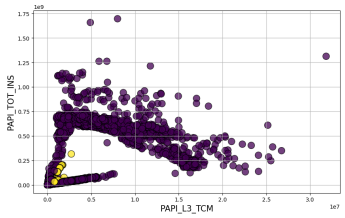


Fig. 1: Characteristics of benign processes vs. Spectre attacks

D. Performance Counters

Modern processors include special-purpose registers called HPCs that are used to track various processor events like clock cycles, branch misses, and cache hits. (The HPCs most relevant for our work are listed in Table IV.) Although initially designed for performance monitoring, these counters can also be used for security purposes [3], [6]–[8], [23], offering advantages such as a minimal effect on speed and resilience to modification by an attacker. In our setting, we systematically identify HPCs related to the phases of the Spectre attack (subsection II-C).

Lightweight profiling tools (Perf Linux https://perf.wiki.kernel.org/, PAPI [24]) allow measuring, recording, and monitoring of hardware and software events at the process and system levels. In prior studies, Perf Linux [3], [6]–[11] and PAPI [4], [9], [11] were used to collect events from HPCs. We deploy PAPI to track micro-architectural traces (cf. section III).

E. Machine Learning for Security

In security, ML is widely used for intrusion detection, malware analysis, and traffic anomalies (see, e.g., [25]). Its ability to identify complex patterns facilitates the accurate detection of Spectre attacks even when other (benign) processes affect the same HPCs and might even behave similarly to an attack. Indeed, Figure 1 (showing total instructions completed vs. level 3 cache misses) indicates that there is an overlap between the characteristics of a Spectre attack (in yellow) and benign processes (purple), making an analytical detection approach challenging. To minimize false positives and improve overall accuracy, an ML model needs to be carefully selected and trained. We evaluated 10 ML algorithms used in the publications in Table I (see subsection IV-B for details).

F. Evaluation Criteria

To systematically evaluate our Spectre detection models, we utilized common performance metrics. While prior work mostly focuses on accuracy due to its importance in avoiding wrong predictions, we also considered other metrics like recall to reduce false negative alarms, precision to avoid mistaking benign activities as attacks, and the F1-score, which reflects model performance regardless of class imbalance in our data (more benign labels than attacks). Finally, we evaluated the scoring time to ensure reliable and timely predictions. Table II provides an overview of our evaluation metrics.

Accuracy	Ratio of correct predictions to total predictions
Precision	Ratio of correct positive to the total positive predictions (high precision indicates a low false positive rate)
Recall	(Sensitivity) Ratio of correct positive predictions to all observations in the actual class
F1-Score	Weighted average of Precision and Recall
Score Time	Time required time to make predictions (shorter is better)

TABLE II: Performance metrics for evaluation of ML models

III. Methodology

We systematically identified the best-suited (and minimal) set of HPCs as well as the best-performing classifiers for detection (including the combinations of HPCs and classifiers considered in Table I) using the following approach:

1) **Validation of HPCs:** We initially identified and assessed 40 HPCs that reflect system-wide or cache-related events and that are potentially affected by Spectre based on expert knowledge, and we excluded HPCs not relevant to Spectre. Each HPC was assessed individually by generating datasets under various workloads (benign as well as attacks) to ensure their relevance for Spectre attacks. In this process, we noticed that some HPCs failed to reliably record data. These HPCs were removed from further experiments to guarantee the integrity and reliability of the generated datasets.
2) **Generating Datasets:** We used the validated HPCs and their combinations (from 1 to 8 HPCs, as the number of HPCs that can be monitored simultaneously is limited) to generate datasets. As some HPCs inhibit each other, we removed conflicting combinations that failed to record data from further experiments.
3) **HPCs Selection:** We ranked the remaining HPCs according to their feature importance (predictive power) and identified redundant HPCs. We employed the mRMR [26] technique to identify features that are least related to each other and most strongly correlated with the class; this algorithm operates in two steps:
   a) **Maximize Relevance** by measuring the Mutual Information (MI) from information theory, which shows how much information an HPC contributes when observing another (thus confirming that the selected HPCs are relevant to the class):

$$MI(X;Y) = \sum_{x,y} p(x,y) \log \left(\frac{p(x,y)}{p(x) \cdot p(y)} \right) ,$$

   where $p(x,y)$ is the joint probability distribution function and $p(x)$ and $p(y)$ are the marginal probability distribution functions of x and y, respectively.
   b) **Minimize Redundancy.** We deploy the MI to ensure that the selected HPCs are not strongly correlated, as two HPCs that share significant MI are redundant and can be predicted from each other.

$$Redundancy = \frac{1}{|S|^2} \sum_{s_i \in S} MI(s_i; f) ,$$

   where S is the current selected set of features, s_i is an individual feature within that set, and f is the new

Combinations	#HPC	Select
Individual events	1	KB
L3_TCM, BR_MSP	2	RW
BR_TKN, BR_PRC	2	mRMR
TOT_INS, L3_TCM, L3_TCA	3	RW
BR_TKN, BR_PRC, BR_CN	3	mRMR
TOT_INS, L3_TCM, L1_ICM, BR_MSP	4	RW
BR_TKN, BR_PRC, BR_CN, TOT_CYC	4	mRMR
L2_DCM, L1_ICM, L1_STM, L2_ICA, L2_ICM	5	RW
BR_TKN, BR_PRC, BR_CN, TOT_INS, BR_MSP	5	mRMR
L2_DCM, L1_ICM, L1_STM, L2_ICA, L2_ICM, L3_TCA	6	RW
L2_DCM, L1_ICM, L1_STM, L2_ICA, L2_ICM, L3_TCA, L2_TCM	7	RW
as above plus L3_ICA	8	RW

TABLE III: Examples of validated combinations, selected based on expert knowledge (KB), related work (RW), or mRMR.

feature. The mRMR criterion is defined as:

$$mRMR = \max_{f \in F \setminus S} \left[MI(f;C) - \frac{1}{|S|} \sum_{s_i \in S} MI(s_i;f) \right],$$

where F stands for the total set of features and C is the class label. In other words, mRMR identifies the most relevant but at the same time least redundant HPCs.

4) **HPCs Reduction:** To determine the minimal number of HPCs required to achieve high accuracy, we conducted two experiments. Firstly, we evaluated all datasets from step 2. We observed that accuracy did not increase for sets with more than 5 HPCs. Therefore, we decided to focus on combinations of up to 5 HPCs. Secondly, we generated new combinations by selecting the top-ranked HPCs from step 3 to generate datasets with 2 to 5 HPCs. To ensure a fair comparison, we generated new datasets under the same conditions and scenarios.
5) **Model Selection:** Finally, we generated datasets using the combinations of HPCs in Table III to assess 10 common ML algorithms with respect to the metrics in Table II.

IV. Experiments and Results

Our toolchain for data collection, analysis, and modeling was developed in Python and uses the libraries PyPAPI (https://flozz.github.io/pypapi/), TensorFlow and Keras (https://www.tensorflow.org/guide/keras), Numpy, and Sklearn, among others.

A. Data Collection and Analysis

The collection of reproducible and comparable datasets for a variety of workloads and scenarios with *differing* sets of HPCs poses a significant challenge. We implemented a reconfigurable dataset generator that enables us to monitor selected combinations of HPCs for a range of scenarios including various attack types (Spectre V1 and 2 [1]) and benign applications (e.g. SPEChpc 2021 [27], MiniSAT (http://minisat.se), MariaDB (https://mariadb.com/), OBS-Studio, Apache, and more), spanning light to intensive workloads and producing datasets in various sizes and dimensions. To improve generalization and avoid overfitting, we deployed the following tactics:

1) Generating two separate datasets—one for training and another for testing—instead of splitting a single dataset.

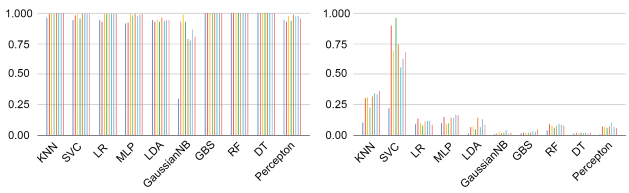


Fig. 2: Accuracy and Score-Time over Initial Datasets

2) Collecting learning and testing datasets under varying workloads. For example, our training dataset may include Firefox, Pycharm, and Mega, while our test dataset includes Chrome, Spyder, and Dropbox, respectively.
3) Excluding intensive workloads (such as MiniSAT) from the training dataset to assess the model's performance when MiniSAT was included.
4) Additionally, we collected 50 datasets of varying sizes ranging from 15,000 to 50,000 samples to analyze the influence of dataset sizes on model performance.

B. ML Algorithms and Models

We used cross-validation to compare various ML models based on the metrics in Table II across different datasets. We assessed a range of ML algorithms that were selected based on an in-depth literature review, including linear, non-linear, and non-parametric algorithms (Logistic Regression (LR), Support Vector Classifier (SVC), Perceptron, Decision Tree, Linear Discriminant Analysis (LDA), K-Nearest Neighbors (KNN) [6], [9], [11], [13], [28]), probabilistic models (Gaussian Naive Bayes (GaussianNB) [11], [12]), ensemble learning methods (Random Forest Classifier (RF) and Gradient Boosting Classifier (GBC) [11]), as well as artificial neural networks (Multilayer Perceptron Classifier (MLP) [6], [9], [12]–[14]). In our experiments, ML models that have consistently proven to be the optimal choices for similar detection tasks in prior studies (such as Decision Trees and ensemble learning methods [11], [13]) emerged as the best-performing models.

C. Experiments

All figures in this section use the following color coding to indicate the number of HPCs in the respective dataset: ■: 1, ■: 2, ■: 3, ■: 4, ■: 5, ■:6, ■: 7, and ■: 8.

*1) **Different Classifiers, Same Dataset**:* We compared the performance of 10 different classifiers (see subsection IV-B) on the same dataset using the classifier as the independent variable and performance metrics as the dependent variable. To ensure consistency, we always used the same k-fold cross-validation and controlled the random seed. We determined that the GBC, Decision Tree, and RF classifiers emerged as the best-performing models across all datasets in terms of Accuracy, Precision, Recall, and F1-score. On the other hand, GaussianNB, LDA, and MLP show the worst results across all metrics. The Decision Tree had the fastest score time among the best-performing algorithms, as shown in Figure 2.

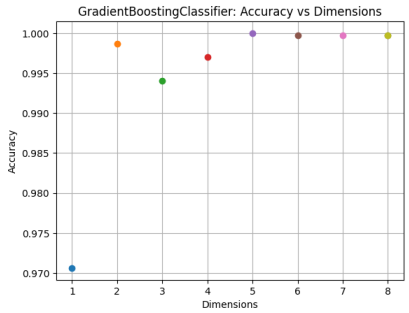


Fig. 3: Accuracy vs. Dimensions over Initial Datasets

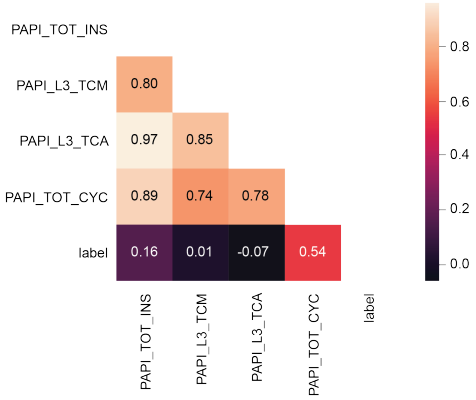


Fig. 4: Correlation Matrix

*2) **Individual Algorithm over All Datasets**:* We compared how the same algorithm performed across all datasets. Here, our independent variable is the dataset, and the dependent variable is performance metrics. Again, we kept the same k-fold cross-validation and controlled the random seeds. We generated datasets using different configurations ranging from a single HPC to combinations of up to 8. Notably, deploying more than 5 HPCs does not significantly improve accuracy, but increases model complexity as shown in Figure 3.

*3) **Ranking and Reduction of HPCs**:* A subsequent statistical analysis of the features (using Principal Component Analysis, Correlation Matrix as shown in Figure 4, and feature-importance, cf. section III) showed a strong correlation among certain HPCs (e.g. Total Cache Access and Total Cache Instructions in [4]), indicating potential for reducing the dimension of the datasets while maintaining a balance between effectiveness and efficiency of the model.

Table IV shows the HPCs ranked highest by our feature-importance and mRMR algorithms; these HPCs relate to overall execution branch behavior. (Table V provides a comprehensive overview of our ranking.) For systematic comparison, we used these HPCs and their combinations to generate new datasets and apply the same classifiers to these datasets as shown in Figure 6. Notably, GBC, Decision Tree, and RF emerged again as the best-performing classifiers, and Decision Tree was the fastest, as shown in Figure 6. The feature reduction technique was essential for identifying the minimum number of HPCs needed for detection with the best accuracy. Figure 5 shows that

Counter ID	Performance Counter
BR_CN	Conditional Branch Instructions Executed
BR_PRC	Conditional Branch Instructions Corr. Pred
BR_INS	Total Branch Instructions Executed
TOT_CYC	Total Cycles Executed
TOT_INS	Total instructions executed
BR_TKN	Conditional Branch Instructions Taken

TABLE IV: Top-Ranked HPCs by mRMR Algorithm

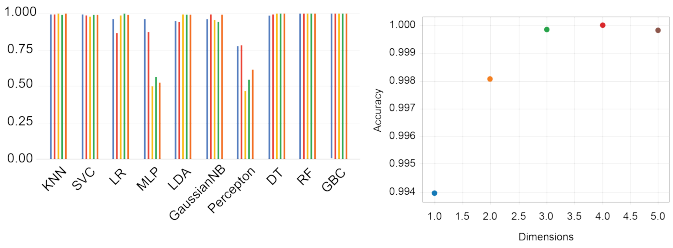


Fig. 5: Accuracy vs. Dimensions over Top-Ranked HPCs

3 top-ranked HPCs were already sufficient for Spectre detection with high accuracy.

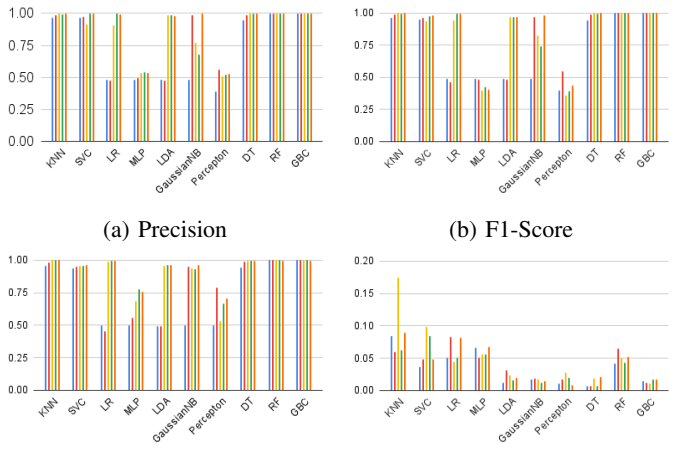
D. Significant Findings and Discussion

The most significant results of our systematic evaluation are:

- Employing top-ranked HPCs reflecting execution (`TOT-CYC`, `TOT-INS`) and branch behavior (`BR-CN`, `BR-PRC`, `BR-INS`, and `BR-TKN`) significantly improves accuracy. Notably, no cache-related HPCs are included in this set (see also Table V). We conjecture that the ratio of `TOT-CYC` and `TOT-INS` already enables the identification of high-latency cache flush instructions.
- Identifying highly correlated HPCs in combinations used in related work (Table I) enables feature reduction.
- Ensemble learning (GBC, RF) and Decision Trees outperformed other ML methods.
- Training and hardening the model with challenging datasets (cf. subsection IV-A) lowers false alarm rates.
- While workload scenarios had higher impact on detection performance than dataset size, both were of low impact.
- Fine-tuning the hyper-parameters of the algorithms improves the detection model's generalization and accuracy.

V. Conclusion

Our systematic evaluation yielded that Gradient Boosting, Decision Tree, and Random Forest classifiers outperform other classifiers under various conditions and scenarios. Decision Tree has the fastest processing time among the best-performing models. We also systematically identified instruction and cycle counters as well as counters reflecting branching behavior as the most appropriate features for detection. Employing challenging datasets enhances resilience against previously unseen attacks and lowers the number of false positive and false negative cases. Overall, our findings offer valuable insights for selecting ML algorithms, HPCs, and generating datasets for detecting attacks.



(a) Precision (b) F1-Score

(c) Recall (d) Score-Time

Fig. 6: Measurements over Ranked Features

Feature	Importance	Rank
PAPI_BR_CN	1.0000	7.5000
PAPI_L1_TCM	1.0000	7.5000
PAPI_TOT_CYC	1.0000	7.5000
PAPI_L3_DCA	1.0000	7.5000
PAPI_L2_TCA	1.0000	7.5000
PAPI_L2_STM	1.0000	7.5000
PAPI_L2_ICH	1.0000	7.5000
PAPI_BR_INS	1.0000	7.5000
PAPI_L2_DCA	1.0000	7.5000
PAPI_L2_DCR	1.0000	7.5000
PAPI_L1_DCM	1.0000	7.5000
PAPI_BR_TKN	1.0000	7.5000
PAPI_BR_PRC	1.0000	7.5000
PAPI_BR_NTK	1.0000	7.5000
PAPI_TOT_INS	0.8279	15.0000
PAPI_L2_TCM	0.5797	16.0000
PAPI_L2_DCM	0.5085	17.0000
PAPI_L3_ICA	0.5004	18.0000
PAPI_L3_TCA	0.4649	19.0000
PAPI_L3_TCM	0.3434	20.0000
PAPI_L2_ICA	0.3122	21.0000
PAPI_L2_ICM	0.2464	22.0000
PAPI_L1_STM	0.2052	23.0000
PAPI_L1_ICM	0.2004	24.0000
PAPI_BR_MSP	0.0645	25.0000

(a) **Ranked Features (based on feature importance)**

Feature	mRMR_Score	Rank
PAPI_BR_CN	0.1011	1.0000
PAPI_BR_PRC	0.0898	2.0000
PAPI_BR_INS	0.0828	3.0000
PAPI_TOT_CYC	0.0685	4.0000
PAPI_TOT_INS	0.0358	5.0000
PAPI_BR_TKN	0.0341	6.0000
PAPI_L2_TCA	0.0256	7.0000
PAPI_L2_DCA	0.0231	8.0000
PAPI_L3_TCA	0.0193	9.0000
PAPI_L2_ICH	0.0175	10.0000
PAPI_L3_TCM	0.0166	11.0000
PAPI_BR_NTK	0.0154	12.0000
PAPI_L1_DCM	0.0138	13.0000
PAPI_BR_MSP	0.0099	14.0000
PAPI_L3_DCA	0.0095	15.0000
PAPI_L2_DCM	0.0066	16.0000
PAPI_L2_ICM	0.0066	17.0000
PAPI_L1_STM	0.0065	18.0000
PAPI_L1_TCM	0.0062	19.0000
PAPI_L2_TCM	0.0056	20.0000
PAPI_L2_DCR	0.0056	21.0000
PAPI_L3_ICA	0.0047	22.0000
PAPI_L1_ICM	0.0044	23.0000
PAPI_L2_ICA	0.0044	24.0000
PAPI_L2_STM	0.0036	25.0000

(b) **Ranked Features (based on mRMR)**

TABLE V: Ranking of features based on different metrics

ACKNOWLEDGMENT

Supported by the Vienna Science and Technology Fund (WWTF) [10.47379/VRG11005] and the FWF (W1255-N23).

REFERENCES

[1] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: exploiting speculative execution," *Commun. ACM*, 2020.

[2] R. McIlroy, J. Sevcík, T. Tebbi, B. L. Titzer, and T. Verwaest, "Spectre is here to stay: An analysis of side-channels and speculative execution," *CoRR*, 2019.

[3] A. P. Kuruvila, S. Karmakar, and K. Basu, "Time series-based malware detection using hardware performance counters," in *HOST*. IEEE, 2021.

[4] J. Depoix and P. Altmeyer, "Detecting Spectre attacks by identifying cache side-channel attacks using machine learning," *Advanced Microkernel Operating Systems*, 2018.

[5] Z. Tong, Z. Zhu, Y. Zhang, Y. Liu, and D. Meng, "Attack detection based on machine learning algorithms for different variants of spectre attacks and different meltdown attack implementations," *CoRR*, 2022.

[6] C. Li and J. Gaudiot, "Online detection of spectre attacks using microarchitectural traces from performance counters," in *SBAC-PAD*. IEEE, 2018.

[7] ——, "Detecting malicious attacks exploiting hardware vulnerabilities using performance counters," in *COMPSAC*, V. Getov, J. Gaudiot, N. Yamai, S. Cimato, J. M. Chang, Y. Teranishi, J. Yang, H. V. Leong, H. Shahriar, M. Takemoto, D. Towey, H. Takakura, A. Elçi, S. Takeuchi, and S. Puri, Eds. IEEE, 2019.

[8] ——, "Detecting spectre attacks using hardware performance counters," *IEEE Trans. Computers*, 2022.

[9] B. A. Ahmad, "Real time detection of spectre and meltdown attacks using machine learning," *CoRR*, 2020.

[10] R. Oshana, M. A. Thornton, E. C. Larson, and X. Roumegue, "Real-time edge processing detection of malicious attacks using machine learning and processor core events," in *SysCon 2021*. IEEE, 2021.

[11] M. Mushtaq, J. Bricq, M. K. Bhatti, A. Akram, V. Lapotre, G. Gogniat, and P. Benoit, "WHISPER: A tool for run-time detection of side-channel attacks," *IEEE Access*, 2020.

[12] H. Wang, H. Sayadi, A. Sasan, S. Rafatirad, T. Mohsenin, and H. Homayoun, "Comprehensive evaluation of machine learning countermeasures for detecting microarchitectural side-channel attacks," in *GLSVLSI*, T. Mohsenin, W. Zhao, Y. Chen, and O. Mutlu, Eds. ACM, 2020.

[13] H. Wang, H. Sayadi, S. Rafatirad, A. Sasan, and H. Homayoun, "SCARF: detecting side-channel attacks at real-time using low-level hardware features," in *IOLTS*. IEEE, 2020.

[14] H. Sayadi, N. Patel, S. M. P. D., A. Sasan, S. Rafatirad, and H. Homayoun, "Ensemble learning for effective run-time hardware-based malware detection: a comprehensive analysis and classification," in *DAC*. ACM, 2018.

[15] J. Cho, T. Kim, S. Kim, M. Im, T. Kim, and Y. Shin, "Real-time detection for cache side channel attack using performance counter monitor," *Applied Sciences*, 2020.

[16] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, M. M. Yousaf, U. Farooq, V. Lapotre, and G. Gogniat, "Machine learning for security: The case of side-channel attack detection at run-time," in *ICECS*. IEEE, 2018.

[17] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, M. Hamburg, and R. Strackx, "Meltdown: reading kernel memory from user space," *Commun. ACM*, 2020.

[18] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *CRYPTO*, N. Koblitz, Ed. Springer, 1996.

[19] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *DIMVA 2016*, J. Caballero, U. Zurutuza, and R. J. Rodríguez, Eds. Springer, 2016.

[20] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *USENIX Security 15*, J. Jung and T. Holz, Eds. USENIX Association, 2015.

[21] M. Mushtaq, A. Akram, M. K. Bhatti, R. N. B. Rais, V. Lapotre, and G. Gogniat, "Run-time detection of prime + probe side-channel attack on AES encryption algorithm," in *GIIS 2018*. IEEE, 2018.

[22] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *USENIX Security*, W. Enck and A. P. Felt, Eds. USENIX Association, 2018.

[23] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Appl. Soft Comput.*, 2016.

[24] S. Browne, J. J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A portable programming interface for performance evaluation on modern processors," *Int. J. High Perform. Comput. Appl.*, 2000.

[25] G. Pang, C. Shen, L. Cao, and A. van den Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surv.*, 2022.

[26] H. Fang, P. Tang, and H. Si, "Feature selections using minimal redundancy maximal relevance algorithm for human activity recognition in smart home environments," *Journal of Healthcare Engineering*, 2020.

[27] J. Li, A. Bobyr, S. Boehm, W. C. Brantley, H. Brunst, A. Cavelan, S. Chandrasekaran, J. Cheng, F. M. Ciorba, M. E. Colgrove, T. Curtis, C. S. Daley, M. H. Ferrato, M. G. de Souza, N. Hagerty, R. Henschel, G. Juckeland, J. Kelling, K. Li, R. Lieberman, K. McMahon, E. Melnichenko, M. A. Neggaz, H. Ono, C. Ponder, D. Raddatz, S. Schueller, R. Searles, F. Vasilev, V. G. M. Vergara, B. Wang, B. Wesarg, S. Wienke, and M. Zavala, "Spechpc 2021 benchmark suites for modern HPC systems," in *ICPE*. ACM, 2022.

[28] Y. Zhang and Y. Makris, "Hardware-based detection of spectre attacks: A machine learning approach," in *AsianHOST*. IEEE, 2020.