Hardware Performance Counters: Ready-Made vs Tailor-Made

ABRAHAM PEEDIKAYIL KURUVILA, The University of Texas at Dallas, United States
ANUSHREE MAHAPATRA and RAMESH KARRI, New York University, United States
KANAD BASU, The University of Texas at Dallas, United States

Micro-architectural footprints can be used to distinguish one application from another. Most modern processors feature hardware performance counters to monitor the various micro-architectural events when an application is executing. These ready-made hardware performance counters can be used to create program fingerprints and have been shown to successfully differentiate between individual applications. In this paper, we demonstrate how ready-made hardware performance counters, due to their coarse-grain nature (low sampling rate and bundling of similar events, e.g., number of instructions instead of number of add instructions), are insufficient to this end. This observation motivates exploration of tailor-made hardware performance counters to capture fine-grain characteristics of the programs. As a case study, we evaluate both ready-made and tailor-made hardware performance counters using post-quantum cryptographic key encapsulation mechanism implementations. Machine learning models trained on tailor-made hardwareperformance counter streams demonstrate that they can uniquely identify the behavior of every post-quantum cryptographic key encapsulation mechanism algorithm with at least 98.99% accuracy.

CCS Concepts: • **Security and Privacy → Security in Hardware**; *Embedded Systems Security;*

Additional Key Words and Phrases: Hardware performance counters, machine learning, post quantum cryptographic algorithms

ACM Reference format:
Abraham Peedikayil Kuruvila, Anushree Mahapatra, Ramesh Karri, and Kanad Basu. 2021. Hardware Performance Counters: Ready-Made vs Tailor-Made. *ACM Trans. Embedd. Comput. Syst.* 20, 5s, Article 65 (September 2021), 26 pages.
https://doi.org/10.1145/3476996

1 INTRODUCTION

Hardware Performance Counters (HPCs) are special purpose registers built into modern processors and monitor low-level micro-architectural events, e.g., branch-misses, cache-misses, and number of instructions executed. HPCs are typically used to monitor the performance of applications

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2021.
Authors' addresses: A. P. Kuruvila and K. Basu, The University of Texas at Dallas, Department of Electrical and Computer Engineering, 800 W Campbell Rd, Richardson, TX 75080; emails: {apk190000, kxb190012}@udallas.edu; A. Mahaptra and R. Karri, New York University, Department of Electrical and Computer Engineering, New York, NY 10003; emails: {am11019, rkarri}@nyu.edu.

1539-9087/2021/09-ART65 $15.00
https://doi.org/10.1145/3476996

at run-time. Every processor has a fixed set of ready-made HPCs. These ready-made HPCs were developed for software optimization and performance tuning. The number and type of available HPCs depends on the processor. As an example, a Raspberry Pi 3 Model B+, which utilizes an ARM Cortex A53 processor, supports 20 HPCs. However, a processor can only monitor a limited number of HPCs simultaneously, generally up to four at a time, depending on the processor [4, 12, 58].

HPCs have been extensively used in the context of embedded security. Prior research has studied the application of ready-made HPCs both as a side channel attack vector and as a defense technique in the context of system security. Information leakage via HPCs of applications has been exploited as covert channels [13, 35, 52, 58]. Previous research demonstrated the use of ready-made HPCs as a system-level defense, i.e., to distinguish Malware from benign applications [12, 28, 29, 58]. Furthermore, ready-made HPCs have also been utilized to detect microarchitectural attacks [22].

However, while ready-made HPCs have been utilized for application classification, they cannot adequately characterize the full unique behavior of monitored applications [11, 62]. This is because ready-made HPCs (e.g., instructions executed, cache misses, and branch misses) provide a coarse-grained characterization. They have low granularity, which inhibits them from thoroughly characterizing an application. To address this challenge, we propose tailor-made HPCs that capture fine-grained micro-architectural events, i.e., provide high granularity. While ready-made HPCs track the number of instructions, tailor-made HPCs can monitor the number of add, load, branch, boolean, and store instructions. Furthermore, tailor-made HPCs can record specific events such as a branch instruction followed by a load or an add instruction followed by a store. We explain tailor-made HPCs in more detail in Section 4.1. The profiling capabilities of tailor-made HPCs at a finer granularity enables characterizing the run-time behavior of the monitored applications more effectively than ready-made HPCs. The contributions of this paper are threefold:

(1) Demonstrate that ready-made HPCs are insufficient to uniquely identify algorithm implementations (i.e. low accuracy).
(2) Propose tailor-made HPCs for the monitored algorithm to yield hardware events (beyond the ready-made HPCs) that can identify the algorithms with a high accuracy.
(3) Develop Machine Learning (ML)-based classifiers to characterize algorithms using tailor-made HPCs.

Paper Roadmap

In order to evaluate the performance of both ready-made and tailor-made HPCs in distinguishing applications, we characterized the implementations of Post-Quantum Cryptographic (PQC) Key Encapsulation Mechanism (KEM) algorithms as a case study. These PQC KEM algorithms are being standardized by the National Institute of Standards and Technology (NIST) [37]. Section 2 describes the threat model and an overview of PQC KEM implementations. Section 3 motivates how ready-made HPCs cannot characterize the PQC KEM implementations. Section 4 proposes tailor-made HPCs to address this challenge. We collect dynamic stack traces of the considered algorithms and use ML to identify the sequences of instructions that the tailor-made HPCs should monitor. We train ML-based classifiers to fingerprint individual PQC KEMs. Section 5 presents experimental results and hardware implementation details of tailor-made HPCs. Section 6 discusses the implications of tailor-made HPCs. Section 7 describes the related work in this domain. Finally, Section 8 summarizes the key findings.

2 BACKGROUND

2.1 Overview of Post Quantum Computing Algorithms

In this paper, we utilize Post Quantum Computing (PQC) Key Encapsulation Mechanism (KEM) algorithms as exemplar applications to highlight the benefits of tailor-made HPCs vis-a-vis

Table 1. KEM Algorithms and Respective Class

PQC Algorithm	Crystals Kyber	NTRU	Saber	Classic McEliece	BIKE	Frodo-KEM	HQC	NTRU Prime
Type	Lattice	Lattice	Lattice	Code	Code	Lattice	Code	Lattice

ready-made HPCs. Traditional public key encryption algorithms such as RivestShamirAdleman (RSA) are vulnerable to quantum computers. The security of RSA is reliant on the fact that finding the prime factors of a composite number takes exponential time on classical computers. However, while integer factorization is computationally intractable on traditional computers, the development of Shor's algorithm enables solving this problem in polynomial time using quantum computers [48]. Consequently, the days of RSA as the ubiquitous Internet encryption system are numbered.

To address the inevitable threat from quantum computing, Post Quantum Cryptography (PQC) algorithms are being developed. PQC algorithms are either Digital Signature or KEM-based [3]. In digital signature algorithms, a sender utilizes a private key to sign the message while the receiver uses the sender's public key to verify the signature [49]. Consequently, digital signature algorithms are beneficial for message authentication. KEM algorithms are used to exchange session keys before a secure communication session using a symmetric key encryption is initiated. In traditional cryptographic protocols, the public key of the sender encrypts a message that is decrypted by a receiver's private key. The sender encrypts symmetric key K and the message M which are sent to the receiver. Second, the receiver decrypts the symmetric key K and utilizes it to decrypt M. Since attackers can reconstruct K when small, and realizing K yields M, KEM addresses these issues by generating a random number N. The symmetric key is produced through $K = \text{KDF}(N)$, where KDF is the key derivation function [3]. These functions are one-way cryptographic hash functions i.e., M can be obtained by having N but the reverse is not feasible. A KEM algorithm has three modules: KEM **Keypair generator** generates the symmetric key (i.e. the session key) from an input seed. KEM **encryption** encrypts the session key with the public key of the receiver. KEM **decryption** of the cipher text by the receiver uses their private key to recover the shared session key.

With the plethora of PQC algorithms being developed, the National Institute of Science and Technology (NIST) is running a competition to standardize PQC algorithms by soliciting and evaluating quantum-resistant public-key cryptographic algorithms. In the first round of the competition, 69 PQC algorithms were submitted for review consisting of 21 digital signature and 48 KEM-based submissions. Furthermore, these cryptographic algorithms consist of different designs. Of the participants, 28 were lattice-based, 20 were code-based, 10 were multivariate-based, 3 were hash-based, and 1 was isogeny-based. Subsequently, in the second round, only 26 PQC algorithms were selected to proceed. NIST intends to standardize a couple of algorithms, as different applications have tradeoffs in terms of utilized memory and speed. The competition is currently in round-three with seven finalists. The finalists consist of three Digital Signature and four Key Encapsulation Mechanism algorithms.

In this case study, we will characterize the PQC KEM implementations using HPCs, but we intend to extend our work in this paper to digital signature algorithms in the future. The KEM algorithms we consider are the four NIST round-three finalists: Crystals Kyber, NTRU, Saber, and Classic McEliece [6, 7, 20, 24]. Furthermore, NIST has provided a set of round-three alternative candidates which are PQC algorithms that did not proceed through the competition but could still be utilized. We consider four alternative PQC KEM algorithms: BIKE, Frodo KEM, HQC, and NTRU Prime [1, 2, 5, 36]. These KEM algorithms are either lattice or code-based. We will identify each algorithm and their type at run-time using ready-made and tailor-made HPCs. Table 1 depicts the type of each finalist and alternative algorithm.

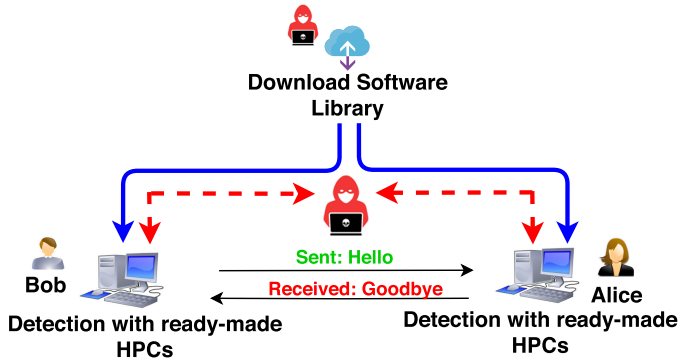


Fig. 1. Overview of Threat Model.

(1) **Lattice-based cryptography** solves lattice-based Shortest Vector Path (SVP) problem of NP-hard complexity. SVP approximates the minimal Euclidean length of a non-zero lattice vector. SVP constructions resist attacks by classic and quantum computers. These algorithms use large public and secret keys. Examples: Crystals Kyber, NTRU, Saber, Frodo KEM, and NTRU Prime.
(2) **Code-based cryptography** systems solve the Error-correction codes (ECC), in particular, binary Goppa Codes. The generating matrix of the code is hidden by scrambling and permuting the entries of the matrix. They require large public keys. Examples: Classic McEliece, BIKE and HQC.

2.2 Threat Model

Our threat model considers scenarios wherein the users (victims) procure software libraries for a specific enterprise. These libraries can range from a plethora of fields including high performance computing, machine learning, and data management. Since we utilize PQC algorithms as a case study in this paper, we particularly focus on a threat model in line with the Padding Oracle On Downgraded Legacy Encryption (POODLE) attack [54]. Figure 1 shows an example of the threat scenario considered in our paper. We envision the use of encryption algorithms, including PQC KEM algorithms in openSSL/TLS protocols. We assume the users have no knowledge of the internal algorithms while procuring the SSL/TLS protocol libraries. Open-source SSL/TLS implementations are available [38], contributed by untrusted developers (adversaries). For example, malicious developers may maliciously modify the libraries such that function calls to a standard encryption algorithm is replaced by an already broken scheme. Furthermore, adversaries may even replace with other PQC KEM algorithm candidates rejected by NIST. Application of such weak algorithms allows adversaries to recover the secret key, thus infiltrating the communication.

Moreover, anomaly detection is imperative for ensuring security in the case of evil maid attacks which bypass authentication protection schemes [10, 51, 55]. In an evil maid attack, an attacker with physical access can modify SSL/TLS protocol libraries. When victims access their web browsers using such modified SSL/TLS libraries, a man-in-the-middle (MitM) [8, 14] equipped with this knowledge can launch attacks on the replaced algorithm in use by running a finite number of queries needed to obtain the secret key. These aforementioned attacks are also feasible for other applications such as a high performance computing library being switched with a less efficient one. Lastly, we consider protecting against kleptographic attacks, which weaken cryptographic implementations without user knowledge for attackers to exploit. Common subversion techniques

include reducing the number of utilized bits, tweaking random number generators, and subverting the hash functions [44, 47]. Once the secret key has been recovered, the MitM can snoop in on future private outgoing data communications performed by the victims using their web browsers, thereby jeopardizing their privacy. Such threats can be mitigated if hardware-based real-time detection methods can effectively monitor a program and distinguish it from other applications.

We note that if the malicious adversary installed a Malware or Trojan, traditional ready-made HPC-based detection schemes are sufficient for identifying these attacks. Prior works have already shown that hardware-assisted Malware detection using HPCs can detect Malware with high confidence [12, 13, 30]. However, as we demonstrate in Section 3.3, anomaly detection methods using ready-made HPCs deployed in users' machines will be insufficient for distinguishing the PQC applications with high accuracy. Consequently, in this paper, we propose tailor-made HPCs that can effectively profile an application to ensure real-time anomaly detection, thereby certifying that users are utilizing the proper libraries for their specific enterprise.

3 STATE-OF-THE-ART: READY-MADE HPCS

This section presents a baseline characterization of PQC KEM algorithms using ready-made HPCs. In this study, we: (a) profile the ready-made HPCs when the algorithm is being executed, (b) extract features to train a ML model and (c) analyze the ML model's prediction performance.

3.1 Collection of Ready-Made HPC Measurement Data

In order to profile the PQC algorithms using ready-made HPCs, we use the software implementations of the KEM algorithms from NIST's official website [37]. In this study, we use a 64-bit Intel core i5-4210U processor, which supports 28 ready-made HPCs. We profile all the available 28 HPCs on this Intel platform for these KEM algorithms using the *Perf 5.4.44* tool [26] at a sampling frequency of 1 KHz. This frequency corresponds to the smallest available collection period, which produces high granularity and precise profiling of the algorithms that is not feasible at higher frequencies.

3.2 Feature Selection

To determine relevant HPCs for training ML models, we explore two feature selection techniques.

- **Extra trees feature selection** is expensive and aggregates multiple de-correlated decision trees and selects features while performing an initial classification [32].
- **Univariate feature selection** uses univariate statistical tests to identify features with the highest importance [33].

3.3 ML-based Characterization with Ready-made HPCs Fails

We implement three ML models—a base ML model trained with four HPCs as features (the processor limits monitoring of up to four counters at a time), and two ML models using Univariate and Extra Trees feature selection. The four HPCs used in the *Base* model are *Branch-instructions*, *Cache-misses*, *L1-dcache-load-misses*, and *Branch-load-misses*. For our models, we introduced a *None* category to label HPC data that are not consistent with the KEM algorithms. Consequently, we have five classes, four corresponding to the NIST Round 3 finalists (Cystals Kyber, Saber, NTRU, and Classic McEleiece), as explained in Section 2.1, and one for *None*. We obtained training and testing datasets for the *None* category by collecting ready-made HPCs from various programs such as sorting, MiBench, and CHStone benchmarks [15, 17, 18].

3.3.1 Ready-made Machine Learning Classifier Selection. Before training our models, it is imperative to select a suitable ML classifier. We trained four ML classifiers including Random

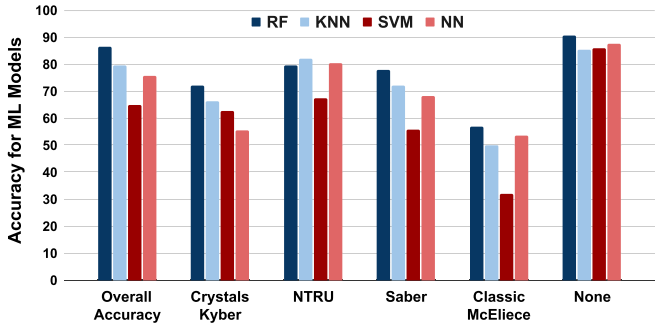


Fig. 2. Performance of ML Classifiers on KEM Finalists.

Forest (RF), Neural Network (NN), K-Nearest Neighbor (KNN), and Support Vector Machine (SVM) using our base HPCs and analyzed their furnished results. The ML classifiers and the utilized HPCs have previously been utilized in prior research for standard HPC-based Malware detection algorithms [28, 35]. Figure 2 shows the performance of the four ML classifiers in terms of accuracy. First, we evaluated the classifiers using a *heterogeneous dataset* consisting of HPC samples of all four KEM finalists shown as *Overall Accuracy* in Figure 2. Next, we tested the classifiers on separate *curated datasets* specific to each algorithm. Our heterogeneous dataset has 50,000 samples for each class, and the curated datasets have 10,000 samples for testing where the datasets represent multiple runs of the PQC algorithms for varying input seeds. When evaluating different machine learning classifiers, we experimented with various parameter modifications and different HPC utilization combinations. For parameter variations, we experimented with a different number of estimators and by varying neurons per layer. For results on the heterogeneous dataset, the highest *Overall Accuracy* was 86.54% achieved by the RF classifier. For the individual *curated datasets*, RF had the best accuracy in four of five classes, i.e., *Crystals Kyber*, *Saber*, *Classic McEliece*, and *None*. While the KNN classifier's accuracy of 82.13% was larger than RF's 79.45% for *NTRU*, RF outranked KNN for the majority of the algorithms. Hence, we use the RF classifier for the remaining experiments in this paper.

3.3.2 Characterizing KEM Algorithms with Ready-made HPCs. Similar to Section 3.3.1, we used a *heterogeneous dataset* for evaluating classifier precision and recall. We use separate homogeneous *curated datasets* for testing to assess classifier accuracy. These metrics determining a model's competency in correctly labeling each ready-made sample. For all succeeding experiments, we employ a *heterogeneous dataset* and a separate *curated datasets*.

Our results on ready-made HPCs for characterizing the KEM finalists are shown in Table 2. The precision and recall values for our *Base* model show that the classifier is unable to efficiently distinguish each algorithm. For example, the precision of *NTRU* was 92.40%, but it was only 77.49% for *Crystals Kyber*, incurring a 14.91% difference. Moreover, *NTRU's* recall of 93.45% is 15.26% more than *Crystals Kyber's* recall of 78.19%. Not only does the lower recall indicate that the samples of *Crystals Kyber* are being missclassified, but the smaller precision signifies that other algorithms are getting incorrectly labeled as *Crystals Kyber*. Both feature selection methods were successfully able to increase these values across all algorithms. *Crystals Kyber's* metrics increased by 11.97% and 9.15% for precision and recall using the HPCs recognized as significant via extra trees feature selection. When evaluating the accuracy of the *curated datasets*, the highest performance was realized by the extra trees feature selection. The classifier was able to distinguish 97.30% of *NTRU's* curated dataset, but it could only correctly identify 73.38% of HPC samples of *Classic McEliece*.

Table 2. Performance of Ready-made HPCs for Profiling the NIST PQC KEM Finalists

PQC Algorithm	Utilized Features	Crystals Kyber (%)	NTRU (%)	Saber (%)	Classic McEliece (%)	None (%)
Heterogeneous Dataset						
Base Precision	BI, CM, L1-DL, BLM	77.49	92.40	78.75	90.15	93.85
Base Recall		78.19	93.45	78.45	88.38	94.15
Univariate Precision	I, L1-DL, L1-DS, dTLB-L	89.20	96.90	85.71	96.56	95.73
Univariate Recall		86.12	98.07	88.14	97.03	94.68
Extra Trees Precision	L1-DL, L1-DS, BL, dTLB-L	89.46	97.51	87.48	96.03	94.84
Extra Trees Recall		87.64	98.50	88.11	96.57	94.63
Curated Datasets						
Base Accuracy	BI, CM, L1-DL, BLM	72.13	79.45	77.83	56.95	90.64
Univariate Accuracy	I, L1-DL, L1-DS, dTLB-L	78.33	76.95	84.64	54.54	94.30
Extra Trees Accuracy	L1-DL, L1-DS, BL, dTLB-L	79.41	97.30	86.27	73.38	95.08

Branch-instructions (BI), Cache-misses (CM), L1-dcache-loads (L1-DL), Branch-load-misses (BLM), Instructions (I), L1-dcache-stores (L1-DS), Branch-loads (BL), dTLB-loads (dTLB-L).

Table 3. Performance of Ready-made HPCs for Profiling the NIST PQC KEM Alternatives

PQC Algorithm	Utilized Features	BIKE (%)	Frodo-KEM (%)	HQC (%)	NTRU Prime (%)	None (%)
Heterogeneous Dataset						
Base Precision	BI, CM, L1-DL, BLM	81.26	93.26	80.55	90.03	93.11
Base Recall		74.46	96.48	77.34	84.23	94.58
Univariate Precision	L1-DS, BL, dTLB-L, dTLB-S	85.30	98.49	86.42	96.65	95.21
Univariate Recall		79.92	99.26	88.37	97.74	94.31
Extra Trees Precision	BLM, BL, dTLB-L, dTLB-S	84.13	98.06	86.54	93.59	95.58
Extra Trees Recall		81.12	98.82	85.60	95.37	94.50
Curated Datasets						
Base Accuracy	BI, CM, L1-DL, BLM	62.67	70.89	59.46	31.71	90.91
Univariate Accuracy	L1-DS, BL, dTLB-L, dTLB-S	60.17	95.21	52.25	56.64	94.94
Extra Trees Accuracy	BLM, BL, dTLB-L, dTLB-S	73.00	68.63	66.22	79.35	95.31

Branch-instructions (BI), Cache-misses (CM), L1-dcache-loads (L1-DL), Branch-load-misses (BLM), L1-dcache-stores (L1-DS), Branch-loads (BL), dTLB-loads (dTLB-L), dTLB-stores (dTLB-S).

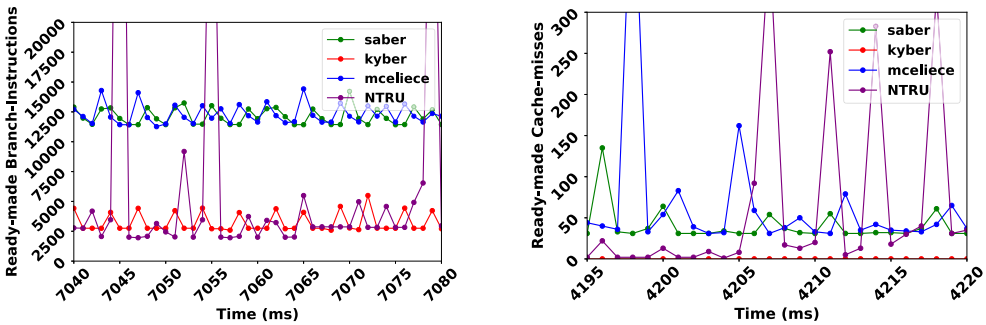
We repeated this experiment for the KEM alternative algorithms and report the results in Table 3. This model faced the same issues with furnishing satisfactory metrics for some algorithms while performing poorly for others. For the *BASE* model, *BIKE* had a precision and recall of 81.26% and 74.46%. However, *FrodoKEM* had much higher values with a precision of 93.26% and recall of 96.48%, which equates to a 12% and 22.02% difference, respectively. Univariate feature selection had the best overall improvement in model performance, but it failed to induce consistent classifier metrics across all algorithms. *BIKE's* recall only increased by 5.46% to 79.92%, but this value is far below *FrodoKEM* and *NTRU Prime* whose recall values are 99.26% and 97.74%, respectively. When evaluating *curated dataset* accuracy, the best performance was obtained using the HPCs identified through extra trees feature selection. The classifier was able to distinguish 97.30% of *NTRU's* curated dataset, but it could only correctly identified 73.38% samples of *Classic McEliece*. Thus, our experimental results lead us to conclude that ready-made HPCs are unable to uniquely identify the

Table 4. Performance of Ready-made HPCs for Profiling Code vs Lattice-based PQC KEM Algorithms

PQC Algorithm →	Utilized Features	Code (%)	Lattice (%)	None (%)
Heterogeneous Dataset				
Base Precision	BI, L1-DL, CM, BLM	83.80	91.93	92.34
Base Recall		75.79	95.12	89.67
Univariate Precision	BI, L1-DS, dTLB-L, dTLB-S	92.16	96.59	93.98
Univariate Recall		90.65	97.77	88.28
Extra Trees Precision	BM, L1-DS, BLM, dTLB-S	90.80	95.00	91.43
Extra Trees Recall		85.57	97.51	85.73

PQC Type	Code (%)			Lattice (%)			None (%)		
Feature Selection	**Base**	**Univariate**	**Extra Trees**	**Base**	**Univariate**	**Extra Trees**	**Base**	**Univariate**	**Extra Trees**
Curated Dataset									
Crystals Kyber	–	–	–	81.96	92.86	93.11	–	–	–
NTRU	–	–	–	89.56	63.14	89.45	–	–	–
Saber	–	–	–	83.26	89.28	90.92	–	–	–
FrodoKEM	–	–	–	94.87	94.60	79.71	–	–	–
NTRU Prime	–	–	–	91.70	35.33	83.34	–	–	–
Classic McEliece	58.76	61.94	72.82	–	–	–	–	–	–
BIKE	59.00	69.84	64.83	–	–	–	–	–	–
HQC	55.01	82.21	74.42	–	–	–	–	–	–
None	–	–	–	–	–	–	84.97	90.20	82.78

We Indicate Non-applicable Categories with a Hyphen. Branch Instructions (BI), Cache-misses (CM), L1-dcache-loads (L1-DL), Branch-load-misses (BLM), L1-dcache-stores (L1-DS), dTLB-loads (dTLB-L), dTLB-stores (dTLB-S), Branch-misses (BM).



(a) (b)

Fig. 3. Ready-made HPC Traces of (a) Branch-instructions and (b) Cache-misses for NIST PQC KEM Finalists.

characteristics of PQC KEM algorithms. We note that additional HPCs could certainly be employed for improving the accuracy. However, the number of HPCs that can be simultaneously monitored is dependent on the processor, typically a maximum of four depending on the processor architecture [4, 12]. Therefore, we limited the number of employed HPCs in our ML classifiers to four to respect this constraint. Consequently, we utilized feature selection techniques to ensure we used the best HPCs, as shown in Tables 2 and 3.

3.4 Classify Code vs Lattice KEMs using Ready-made HPCs

In this experiment, we investigate the effectiveness of ready-made HPCs in classifying code and lattice-based KEM algorithm implementations as shown in Table 4. Similar to Section 3.3.2, the base metrics are inconsistent with 83.80% precision and 75.79% recall for identifying code-based algorithms but 91.93% precision and 95.12% recall for lattice-based algorithms. Univariate feature selection was able to bolster these metrics to 92.16% precision and 90.65% recall for code-based and 96.59% precision and 97.77% recall for lattice-based. When assessing the *curated dataset*, we find that our model with univariate feature selection produced poor inconsistent performance metrics across all algorithms. For lattice-based algorithms, our model was able to identify *FrodoKEM* with a 94.60% accuracy, but it failed to label *NTRU Prime* as it only accurately predicted an abysmal 35.33% of samples. For code-based KEMs, the model accurately identified only 61.94% of the *Classic McEliece* dataset but classified *HQC* with an accuracy of 82.21%.

Our results show that ready-made HPCs cannot distinguish code-based KEMs from lattice-based ones with high performance metrics. To analyze the issues with ready-made HPCs for PQC KEM classification, we investigate HPCs *Branch-instructions* and *Cache-misses* as shown in Figures 3(a) and 3(b), respectively. These plots show the samples of *Branch-instructions* and *Cache-misses* for all four KEM finalists. The x-axis indicates the time in milliseconds, and the Y-axis corresponds to the respective HPC count at each timestamp. The traces in both figures indicate that these algorithms have a plethora of overlapping samples. Overlapping samples indicate that the HPC features will reduce the distinguishing power of the classifier. In Figure 3(a), the ready-made HPC traces of *Saber* and *Classic McEliece* indicate multiple samples that have similar HPC counts. *Crystals Kyber* and *NTRU* suffer from the same limitation. Figure 3(b) shows that the traces have samples of similar counts for *Saber* and *McEliece*. The cache misses of *Kyber* are significantly lower ($\backsim 0$) for many samples, indicating sparsity in their representations. Since the HPC counts of the algorithms are similar for multiple time intervals, these limitations induce inability to effectively distinguish data representations of algorithms effectively through a ML classifier. Consequently, utilizing additional ready-made HPCs and classifiers as an interleave framework wouldn't be able to bolster the results because the base classification accuracy for those models is already unsatisfactory. Interleave solutions are only a valuable scheme when the classifiers themselves furnish high performance. Therefore, for subpar models, the resolution would at best have a slight increase in accuracy, but the performance gains are still below ideal thresholds. Furthermore, an interleaved framework incurs additional power and area overhead with each additional classifier. Therefore, even if the resolution could be bolstered, the solution would not be as optimal. The overlapping samples and absence of clear boundaries elucidates the challenges that ready-made HPCs endure for KEM classification. The ML classifier struggles to use a classification boundary that accurately separates these algorithms, resulting in sub-optimal performance.

4 ADVANCING STATE-OF-ART: TAILOR-MADE HPCS

The limits of ready-made HPCs to accurately profile algorithm implementations motivates exploration of alternative approaches. We propose tailor-made HPCs to characterize the unique, fine-grain features of an algorithm implementation.

4.1 Tailor-made HPCs

Tailor-made HPCs count the sequences of assembly-level instructions at run-time. We performed a binary code analysis when developing our proposed methodology to analyze which instructions encompassed a majority of the directives at the assembly level. To capture a diverse set of instructions, we start with the following five instruction types: branch/jump (b), load (l), store (s),

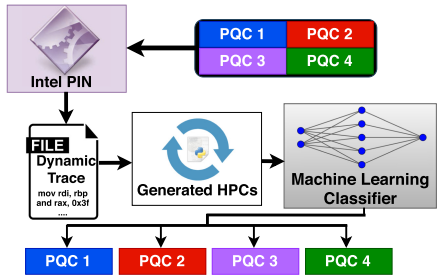


Fig. 4. KEM PQC Classification Using Tailor-made HPCs.

arithmetic (*a*), and boolean (*n*). Our tailor-made HPCs count the number of times these instruction types are encountered. These instructions are common in PQC algorithms since they contain a plethora of computations, conditional statements, and function calls. Moreover, they are frequently encountered in most programs as these aforementioned programming practices are ubiquitous in many application such as sorting algorithms, text editors, and web browsers. Consequently, utilizing these instructions for our tailor-made HPCs ensures that our proposed methodology is not unique to PQC algorithms and can be extended to other applications.

We implement tailor-made HPCs to track sequences of instructions in the form of "XY", where X and Y are two instruction types. These HPCs count the number of times instruction "X" is followed by the instruction "Y". The tailor-made HPC feature vector is: *b, l, s, a, n, ba, bs, bl, bb, bn, la, ls, ll, lb, ln, sa, ss, sl, sb, sn, aa, as, al, ab, an, na, ns, nl, nb* and *nn*. The tailor-made HPC "*ns*" counts the number of times a boolean instruction is immediately followed by a store instruction. Similarly, the tailor-made HPC "*la*" counts the number of times a load instruction is immediately followed by an arithmetic instruction. Employing these tailor-made HPCs can uniquely capture the run-time behavior of an application that is otherwise infeasible with ready-made HPCs.

To emulate tailor-made HPCs, we collect the dynamic traces of the KEM algorithms' implementations using the Intel PIN tool [34]. PIN is a dynamic binary instrumentation framework that provides a rich Application Programming Interface (API) to obtain assembly-level information, including instructions and contents of the general and special purpose registers. Since the dynamic trace is a sequence of assembly instructions representing the order of the program flow execution in the processor, we utilized a sampling interval, T_s, to collect the HPCs. Figure 4 presents our proposed flow for classifying PQC algorithms using tailor-made HPCs. A disassembler is utilized to obtain the dynamic trace from the PQC binary executable. Then, instructions are sampled from the dynamic trace until T_s instructions are encountered. In this paper, we set T_s to 10,000 as this permits capturing a significant amount of the KEM implementation behavior. A different T_s value could be utilized, albeit with a difference in the tailor-made HPC values. When T_s instructions are executed, the tailor-made HPCs are recorded and reset. We repeat this until all instructions are exhausted. The tailor-made HPCs are used to train ML classifiers to distinguis PQC KEM implementations.

To ensure high-quality classification performance, we propose one additional tailor-made HPC, which is the sample count of each tailor-made HPC sampled every T_s instructions. This value corresponds to the phase of execution flow of a KEM implementation. Labeling execution phases of an algorithm is crucial for a ML classifier to identify a tailor-made HPC observation, representing a sequence of executed instructions in an implementation. We add a tailor-made HPC called program phase (*pp*) into the mix. The *pp* HPC uses an instruction count register for fine-grained control on the sampling rate of the number of instructions counted. *pp* is incremented every T_s instructions. Thus, if $5 \cdot T_s$ instructions are encountered in the dynamic trace, then $pp = 5$.

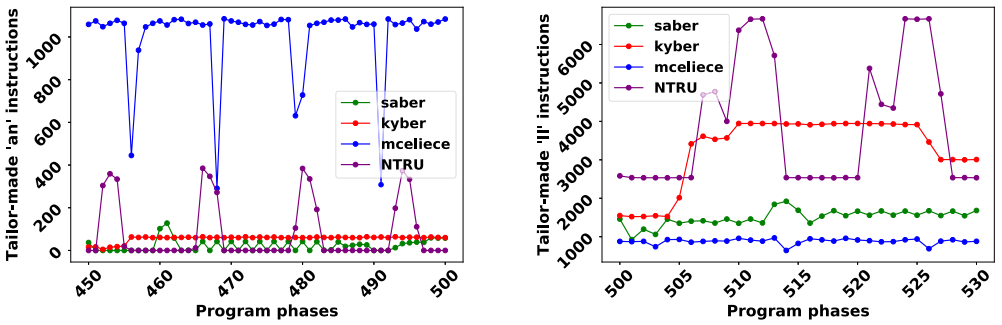


Fig. 5. Tailor-made HPC Traces of (a) "an" and (b) "ll" through the Various Phases of a Program of the NIST PQC KEM Finalists.

4.2 Advantage of Tailor-made HPCs

Figures 5(a) and 5(b) demonstrate the variations in tailor-made *an* and *ll* instruction sequence HPCs through the various program phases of the KEM finalists' implementations. Tailor-made HPC *an* trace is unique to Classic McEliece and distinguishes it from the other KEMs. Furthermore, *ll* instructions show behavior of KEMs delineated from each other through the several program phases. In contrast, as shown in Figure 3, there are large overlaps and small separations in the coarse-grain ready-made HPCs. These distinct traces help to increase the distinguishing power of the features for a classifier. More specifically, in Figure 3(a), Classic McEliece and Saber have similar ready-made HPC traces and Crystals Kyber and NTRU have similar ready-made HPC sample counts. Since tailor-made HPCs can capture the unique attributes and characteristics of an application, they provide a clear separation between the algorithms. Therefore, ML models trained on tailor-made HPCs furnishes superior classifier performance relative to ready-made HPCs.

Tailor-made HPCs additionally utilize less overhead than time series-based models. Deep learning models, such as Recurrent Neural Networks (RNN) have been used for time series classification by employing temporal traces for training. However, RNN incurs high computation, power, and area overhead. Our methodology provides a light-weight solution since we generate tailor-made HPCs from the dynamic runtime trace. Furthermore, by utilizing lightweight ML models, the classifiers use far less power and area resources as opposed to deep learning solutions.

Side-channel attacks exploiting ready-made HPCs to leak secret keys and confidential details of the program flow have dissuaded utilizing them for profiling cryptographic algorithms. Adversaries can monitor ready-made HPC cache events to determine a hit or miss [53]. Since tailor-made HPCs are generated by periodically sampling the dynamic assembly trace, an attacker would face a plethora of difficulties in exploiting this information to leak secret keys. This data is not ideal for key identification since assembly instructions are profiled. Furthermore, it is not possible to obtain defined register values in the dynamic assembly instructions as specific values are stored at specific memory locations accessed through a memory address. These addresses cannot be identified from the dynamic trace. Obtaining the dynamic trace offers no benefits as the register values remain unknown. This makes it near impossible to determine the key. It would be burdensome and tedious to iterate through the dynamic instructions to determine the secret key.

The need for tailor-made HPCs is due to the coarse-grained nature of ready-made HPCs. Our proposed methodology seeks to improve the characterization capabilities of HPCs through profiling at a more fine-grain abstraction level. Other solutions, such as ARM CoreSight, provides

hardware tracing, which allows profiling events such as interrupts, cross-triggers, and other system signals. However, the predicament with using ARM CoreSight hardware event tracing is that the execution behaviors obtained would have similar granularity to ready-made HPCs. We have previously shown that ready-made HPCs are too coarse-grained in Figure 3, where there is an abundance of overlapping samples. Consequently, the data monitored through CoreSight would also be too coarse-grained and incur the same characterization issues. Furthermore, CoreSight is only possible on ARM-based processors. Therefore, other architectures, such as x86 or high-performance computing environments, would not have access to these same execution behaviors. Our proposed tailor-made HPCs are applicable to all processor architectures for providing better fine-grained profiling capabilities that produce higher ML performance metrics.

While message sizes and execution times can be used as features for classification, they do not provide the real-time detection that tailor-made HPCs provide. Monitoring message sizes is not beneficial, as kleptographic attack modifications to algorithm implementations might not change the message size. Furthermore, code authentication and binary analysis methods, which are software-based solutions, are not robust and incur a plethora of computational overhead [12, 58]. Attackers can successfully subvert software-based solutions through obfuscation techniques such as injection of dummy statements, register reassignment, and instruction substitution [61]. Consequently, polymorphic and metamorphic applications, which can change their appearance between runs, will be able to go undetected by software-based solutions [12]. As a result, security solutions utilizing the hardware as a root of trust have evolved and are more robust than their software counterparts [4, 12, 27, 35]. Consequently, several technology companies, such as Intel and Microsoft, have made a transition into hardware-assisted detection as hardware features are much more difficult to jeopardize than their software equivalents [21]. Our proposed tailor-made HPCs are hardware-based primitives that provide fine-grain characterization, thereby enabling users to detect these malevolent subversion techniques. Since tailor-made HPCs are obtained from the dynamic assembly instructions, they can detect erroneous algorithm implementations faster than classifiers that require full execution. Tailor-made HPCs can detect anomalies in real-time, which is critical for securing sensitive information.

5 EXPERIMENTS

5.1 Experimental Setup

To demonstrate the effectiveness of tailor-made HPCs to characterize (PQC KEM) implementations, we used the Intel *PIN* tool [34] to collect the dynamic trace for each of the PQC algorithms. We wrote a *Python* script to sample the collected traces every 10,000 instructions and furnish the tailor-made HPCs. More details of the implementation are provided in Section 5.3. We trained the ML models using the *scikit-learn* library on this tailor-made HPC data. As discussed in Section 3.3.1, we employ the Random Forest model for classification (see Figure 2). We used 70% of the data for training and 30% for testing. Similar to the Section 3.3, we trained one ML model for monitoring the finalist KEM implementations and another ML model for the alternative KEMs. We initially developed a *Base* model utilizing all the tailor-made HPCs by not constraining them to only four. However, for a fair comparison and to validate the superior profiling capabilities of our proposed methodology, we devised separate models utilizing four tailor-made HPCs selected through univariate and extra trees feature selection methods [32, 33]. Furthermore, we utilized our tailor-made HPCs for classifying lattice versus code-based KEMs, and analyzed the behavior of individual PQC KEMs with a one-versus-all classification. Lastly, we evaluated the robustness of our proposed methodology against algorithm subversion techniques.

Table 5. Performance of Tailor-made HPCs for Profiling the NIST PQC KEM Finalists

KEM Algorithm	Utilized Features	Crystals Kyber (%)	NTRU (%)	Saber (%)	Classic McEliece (%)
Heterogeneous Dataset					
Base Precision	All Tailor-made HPCs	99.61	99.92	99.91	99.98
Base Recall		99.91	99.65	99.94	99.96
Univariate Precision	l, n, ll, *pp*	99.75	99.87	99.66	99.95
Univariate Recall		99.72	99.81	99.77	99.95
Extra Trees Precision	s, l, ll, *pp*	99.62	99.83	99.77	99.96
Extra Trees Recall		99.75	99.77	99.72	99.95
Curated Datasets					
Base Accuracy	All Tailor-made HPCs	99.93	99.68	99.93	99.97
Univariate Accuracy	l, n, ll, *pp*	99.72	99.85	99.75	99.97
Extra Trees Accuracy	s, l, ll, *pp*	99.81	99.85	99.77	99.97

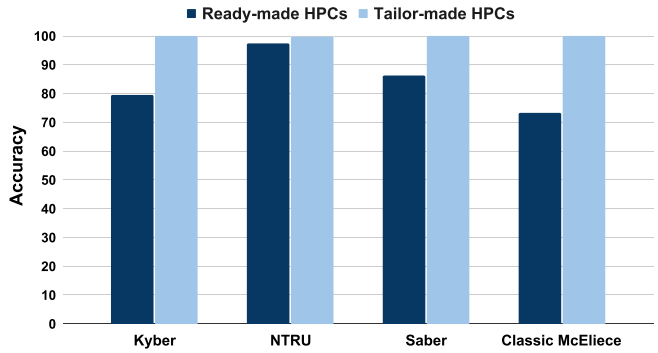


Fig. 6. Performance Comparison of Ready-made HPCs and Tailor-made HPCs for Classifying KEM Finalists.

5.2 Experimental Results

5.2.1 Characterizing KEM Algorithms with Tailor-made HPCs. Our experiment results for classifying the KEM finalists using tailor-made HPCs are shown in Table 5. Similar to Section 3.3, for all succeeding experiments, we provide the utilized features for each model, and we use a *heterogeneous* and *curated dataset* for evaluating our classifiers. We used imbalanced heterogeneous and curated datasets consisting of 40000, 60000, 65000, and 125000 samples for Saber, Crystals Kyber, NTRU, and Classic McEliece respectively. For our results utilizing tailor-made HPCs, we experimented with both imbalanced and balanced datasets, but they furnished similar performance metrics with their differences being extremely negligible (±0.1%). Consequently, we utilized imbalanced datasets as this reduced the training time as fewer samples were incorporated. When compared to ready-made HPCs from Table 2, tailor-made HPCs yield higher performance across all algorithms. *Crystals Kyber*, *NTRU*, *Saber*, and *Classic McEliece* have an average base precision of 99.85% and recall of 99.86%. In contrast, the average base recall of the corresponding model with ready-made HPCs is only 86.52%, which is 13.33% less. A high precision and recall indicate the uniqueness of the tailor-made HPCs in identifying each algorithm. For the curated datasets, Figure 6 presents the results of the performance comparison of ready-made and tailor-made HPCs. The best *curated dataset* accuracy for each model per algorithm is plotted. The ML model, utilizing tailor-made HPCs, furnished better classification accuracy across all algorithms. The biggest

Table 6. Performance of Tailor-made HPCs for Profiling the PQC KEM Alternatives

KEM Algorithm	Utilized Features	BIKE (%)	Frodo-KEM (%)	HQC (%)	NTRU Prime (%)
Heterogeneous Dataset					
Base Precision	All Tailor-made HPCs	99.51	99.98	99.60	99.94
Base Recall		99.71	99.95	99.54	99.88
Univariate Precision	a, b, aa, pp	98.72	99.97	97.62	99.92
Univariate Recall		98.24	99.95	98.61	99.63
Extra Trees Precision	a, n, al, as	97.69	99.97	95.70	99.93
Extra Trees Recall		96.33	99.93	97.63	99.75
Curated Datasets					
Base Accuracy	All Tailor-made HPCs	99.10	99.97	99.66	99.91
Univariate Accuracy	a, b, aa, pp	97.56	99.97	98.92	99.68
Extra Trees Accuracy	a, n, al, as	95.79	99.95	97.72	99.80

noticeable gain occurred with the *Classic McEliece* algorithm. The model with ready-made HPCs furnished an accuracy of 73.38% while the tailor-made HPC model furnished 99.97% for a 26.59% gain in total performance. As shown in Table 5, employing only four tailor-made HPCs is sufficient for maintaining exceptional classification performance.

We repeated this experiment on the KEM alternative algorithms and reported the results in Table 6. Comparing these results against results using ready-made HPCs in Table 3, confirms that tailor-made HPCs enable accurate classification for all algorithms. *BIKE*, *FrodoKEM*, *HQC*, and *NTRU Prime* have an average base precision of 99.75% and recall of 99.77%. Conversely, the ready-made HPC-based model only had an average base precision of 87.64% and recall of 85.41% which is 12.11% and 14.36% less performance. Figure 7 shows our results for the best *curated dataset* accuracy of ready-made and tailor-made HPCs for classifying PQC KEM alternatives. Every KEM implementation had improved results when utilizing tailor-made HPCs. The most substantial improvement is for *HQC* KEM. The accuracy when using ready-made HPCs was only 66.22% and this improved to 99.66% (a 33.44% improvement) by using tailor-made HPCs. Similar to the KEM finalists, four tailor-made HPCs are sufficient for furnishing the excellent performance metrics for the KEM alternatives. The experimental results confirm that employing tailor-made HPCs produces higher performing ML models than using ready-made HPCs. In addition to providing better metrics, our tailor-made HPC-based models performed far more consistently when evaluated on the curated datasets. For the ready-made curated dataset accuracy, changing the utilized HPCs could cause large classification variations. As an example, from Table 3, the base accuracy of *NTRU Prime* is 31.71% while the extra trees accuracy is 79.35%, a 47.64% difference. However, in the case of tailor-made HPCs, there is a negligible difference between curated accuracies. This confirms that fine-grain tailor-made HPCs are more distinct than coarse-grain ready-made HPCs.

5.2.2 Characterizing Code vs Lattice-based KEMs with Tailor-made HPCs. Similar to Section 3.4, we aggregated tailor-made HPCs for both KEM finalists and alternatives for a code versus lattice classification. These results are presented in Table 7. When analyzing the performance metrics and its ready-made HPC equivalent in Table 4, the tailor-made model had an almost perfect base precision and recall of 99.96%. However, the ready-made model only had a base precision of 89.35% and recall of 86.86%. These performance metrics prove that our tailor-made HPC-based model is far more competent at classifying the type of KEM. From Figure 8, we present the best *curated dataset* results per algorithm to further establish the superiority of tailor-made HPC-based models over

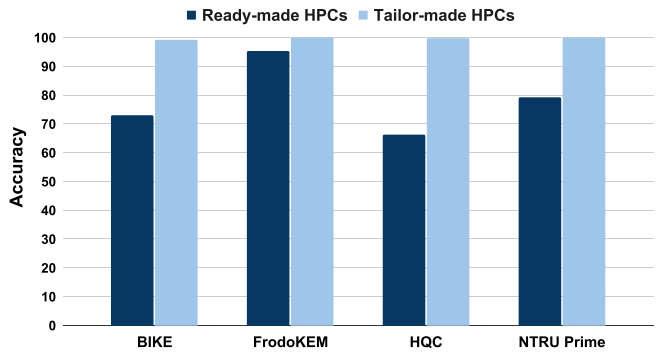


Fig. 7. Ready-made vs. Tailor-made HPCs for Classifying KEM Alternatives.

Table 7. Performance of Tailor-made HPCs: Code vs. Lattice KEM. Irrelevant Cells are Marked With a Hyphen

KEM Algorithm →	Utilized Features			Code (%)		Lattice (%)
Heterogeneous Dataset						
Base Precision	All Tailor-made HPCs			99.95		99.98
Base Recall				99.96		99.97
Univariate Precision	b, an, nn, na			98.04		96.18
Univariate Recall				93.28		98.91
Extra Trees Precision	b, sn, ss, pp			99.65		99.74
Extra Trees Recall				99.56		99.79
KEM Type Feature Selection	**Code (%)**			**Lattice (%)**		
	Base	**Univariate**	**Extra Trees**	**Base**	**Univariate**	**Extra Trees**
Curated Dataset						
Crystals Kyber	—	—	—	99.99	99.74	99.31
NTRU	—	—	—	99.98	99.95	99.96
Saber	—	—	—	99.92	91.52	99.48
FrodoKEM	—	—	—	99.98	99.95	99.98
NTRU Prime	—	—	—	99.92	99.70	99.85
Classic McEliece	99.96	88.91	99.91	—	—	—
BIKE	99.97	99.78	99.27	—	—	—
HQC	99.96	99.51	98.86	—	—	—

ready-made HPC-based. The largest difference in performance occurs for *BIKE* whose ready-made model furnished an abysmal 69.84% accuracy as opposed to the tailor-made model who had 30.13% more performance with an accuracy of 99.97%. Furthermore, when solely employing the tailor-made HPCs identified as the most impactful via extra trees feature selection, our lowest curated dataset accuracy was still 99.86%. This corroborates the benefits of fine-grained tailor-made HPCs.

Our proposed tailor-made HPCs provides an adaptable and secure solution that would be useful in ensuring that SSL/TLS implementations are executing the appropriate libraries. For each implementation, a RF model would be trained offline on the tailor-made HPCs of the appropriate algorithm to be detected. To handle legitimate updates in the algorithm, the classifier can simply

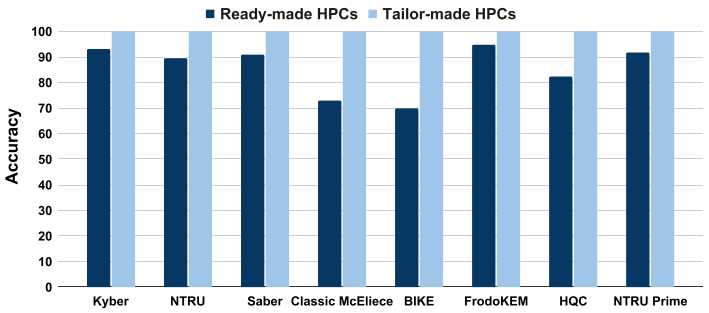


Fig. 8. Ready-made vs. Tailor-made HPCs for Classifying Code vs. Lattice.

be trained again with the tailor-made HPCs of the improved algorithm. We specifically utilized PQC KEM algorithms as a case study to validate the need for fine-grain profiling, but our tailor-made HPCs can be applied to a plethora of different scenarios and applications. Additionally, we can adjust the weights of the model and tune other parameters, such as the number of estimators and trees, to better fit the updated schemes. Specifically, backpropagation can be utilized to calculate the loss from the cost function and iteratively update the weights of the model [19]. Moreover, backpropagation can be employed to update only a single layer's weights, such as the final output layer, thereby further minimizing the computational bandwidth [9]. The model's weights can be modified through over-the-air updates by utilizing software APIs such as Keras [25]. This enables manually modifying the weights in each layer to address the legitimate updates. Consequently, there is negligible down time as the update is instantaneous and does not incur additional computational bandwidth. In this paper, we solely utilized the RF classifier. Therefore, retraining the classifier would only update the spilt nodes in the model. The changes to the prediction component's RTL would be minimal, as only the different decision thresholds would be updated.

We note that our false positive rate was extremely low as we had an accuracy range between 99.28% to 99.98%, as shown in Table 5 and 6. Consequently, when using our proposed framework, we can monitor if it determines an anomaly for multiple consecutive inputs, as this implies an anomaly has been detected. Therefore, false positives would not be detrimental to our detection scheme. The execution profile would incorporate trusted available algorithm implementations in which our tailor-made HPCs would profile for run-time anomaly detection. If a trusted library is not available, the respective algorithm implementation would not be utilized. For trusted execution profiles, our tailor-made HPCs would ensure that attackers cannot compromise them in order to leak sensitive information. Possessing real-time anomaly detection is imperative for securing sensitive information from adversaries.

5.2.3 Tailor-made HPCs One-vs-All. For analyzing the success rate of anomaly detection using tailor-made HPCs, Table 8 tabulates our results for classifying a single KEM against all other algorithms. This analysis highlights the uniqueness of tailor-made HPCs. By performing one-against-all classification, every KEM algorithm is identified using a unique subset of tailor-made HPCs shown in the *Best Features* column of Table 8. When analyzing the accuracy and recall for both feature selection methods, our highest furnished accuracy was 99.98%, and the largest recall was 99.99%. Furthermore, for each KEM implementation, our ML models were competent enough to furnish a minimum of at least 94.91% precision with just four tailor-made HPCs. These values indicate that the model has low false negatives, and using tailor-made HPCs allows for successfully distinguishing a KEM algorithm. This is because of the subsets of fine-grain HPCs that are unique to each algorithm. For example, the subset of tailor-made HPCs selected for Crystals Kyber: *ll, sl,*

Table 8. One-against-all Classification of PQC KEMs using Tailor-made HPCs

KEM ↓	Univariate				Extra Trees			
	Best Features	Acc. (%)	Prec. (%)	Recall (%)	Best Features	Acc. (%)	Prec. (%)	Recall (%)
Crystals Kyber	ll, sl, as, pp	99.82	98.88	99.44	l, sb, sl, as	99.82	99.39	98.92
NTRU	l, n, ll, pp	99.95	99.80	99.82	s, l, ll, sb	99.91	99.34	99.93
Saber	b, n, nb, pp	99.90	99.44	99.21	bn, sn, nb, pp	98.99	88.29	97.78
Classic McEliece	an, nn, na, pp	99.54	99.88	99.90	sn, nn, na, pp	99.94	99.84	99.90
BIKE	a, n, aa, pp	99.28	94.91	96.10	a, bb, la, aa	99.85	98.56	99.61
FrodoKEM	a, l, 11, pp	99.98	99.95	99.99	a, s, n, ll	99.98	99.94	99.99
HQC	b, ln, aa, pp	99.83	98.21	99.37	b, bb, ln, na	99.83	98.62	98.98
NTRU Prime	a, aa, al, pp	99.92	99.29	99.82	a, lb, aa, as	99.90	98.93	99.94

Table 9. Kleptographic Attack Detection in NIST PQC KEM Finalists

PQC Algorithm	Crystals Kyber			NTRU			Saber			Classic McEliece		
Section Targeted	Key_Pair()	Encrypt()	Decrypt()	Key_Pair()	Encrypt()	Decrypt()	Key_Pair()	Encrypt()	Decrypt()	Key_Pair()	Encrypt()	Decrypt()
Induced Modification	Swapped Variable Parameters	Changed Parameter Value	Adjusted Loop Length	Changed Arithmetic Operator	Swapped Function Parameters	Modified Assignment Statement	Changed Arithmetic Operator	Swapped Variable Parameters	Altered Loop Condition	Swapped Function Parameters	Changed Arithmetic Operator	Modified Loop Condition
Anomaly Detection	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

as, and *pp* do not repeat across all algorithms showing uniqueness. However, when inspecting the features identified through univariate feature selection, tailor-made program phase HPC (*pp*) is included as a feature for all algorithms highlighting its importance. The program phase is imperative as it provides a label of the current assembly execution phase of an application. Consequently, utilizing it as a feature enables a model to better distinguish between algorithms.

5.2.4 Robustness Against Kleptographic Attacks. It is certainly possible for the adversary to keep the same KEM implementation while inducing small, malicious modifications, i.e., kleptographic attacks. In order to demonstrate the fine-grain capabilities of our tailor-made HPCs and analyze the robustness of our methodology against these algorithm subversion attacks, we induced small pernicious changes in each PQC KEM algorithm. More specifically, we created an erroneous modification in the *Key_pair()*, *Encrypt()*, and *Decrypt()* functions for a total of 24 malicious variations as shown in Tables 9 and 10, for the finalists and the alternative algorithms, respectively. These functions encapsulate the three main components of the PQC KEM algorithms. Attackers commonly attempt to induce exploits through various diminutive malevolent changes including reducing the number of utilized bits, tweaking random number generators, and subverting the hash functions [44, 47]. Therefore, the changes induced in the algorithms range from swapping function parameters, changing loop lengths, modifying assignment statements, and altering arithmetic operations. We collected the tailor-made HPCs from the modified algorithms, and our utilized ML model was able to identify the anomaly for all malicious variations. Therefore, the robustness and fine-grain characteristics of our tailor-made HPCs are validated.

5.3 Hardware Overhead of Tailor-made HPCs

In order to assess the hardware overhead of tailor-made HPCs, we implemented them into a simple MIPS processor. A MIPS processor was chosen because (1) we do not have access to the 64-bit Intel core i5-4210U processor implementation, and, (2) since a MIPS processor incurs significantly less

Table 10. Kleptographic Attack Detection in NIST PQC Alternative KEM Finalists

PQC Algorithm	BIKE			FrodoKEM			HQC			NTRU Prime		
Section Targeted	Key_Pair()	Encrypt()	Decrypt()	Key_Pair()	Encrypt()	Decrypt()	Key_Pair()	Encrypt()	Decrypt()	Key_Pair()	Encrypt()	Decrypt()
Induced Modification	Swapped Variable Parameters	Changed Parameter Value	Adjusted Conditional Statement	Changed Arithmetic Operator	Swapped Function Parameters	Adjusted Loop Length	Swapped Variable Parameters	Changed Loop Count	Altered Arithmetic Operator	Modified Loop Condition	Swapped Function Parameters	Replaced Bitwise Operator
Anomaly Detection	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

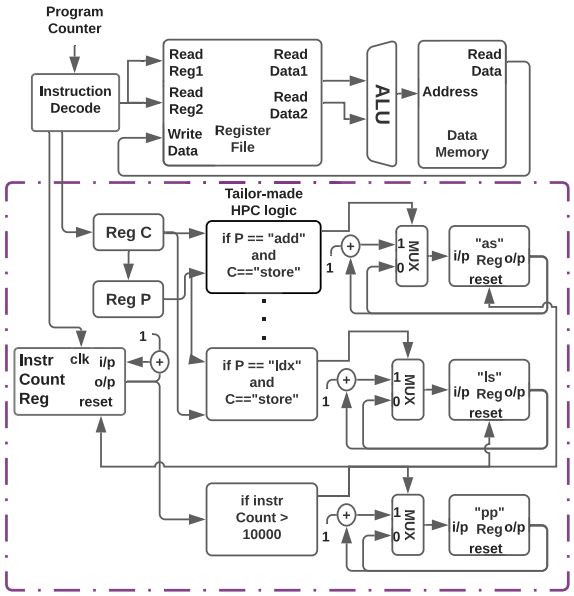


Fig. 9. Tailor-made HPC logic in a MIPS processor is shown in the dotted box. We show three HPCs: program phase HPC pp, HPC to track add→store (as), and HPC to track load→load (ll). The program phase bound (set to 10,000) can be adjusted.

area than an Intel processor, the area overhead for the HPC design units in the Intel processors will be correspondingly lower than a MIPS processor. Therefore, the obtained results provide a pessimistic estimate for area overhead.

Figure 9 shows a RTL diagram of the tailor-made HPC hardware logic in the MIPS processor. The hardware logic stores the current instruction (*Reg C*) and the previous instruction (*Reg P*) to check if a sequence of specific instructions (e.g., *Reg P* is an "add" instruction followed by a "store" in *Reg C*) are obtained. Every tailor-made HPC logic controls a multiplexer (MUX) that increments the respective HPC (e.g., *"as" Reg*). While HPC registers are updated using a conditional RTL logic block, adder and a MUX, program phase HPC (*"pp" Reg*) uses an instruction count register to count the number of instructions. If the instructions exceed 10,000 (can be adjusted), all HPC registers are reset and the *"pp"* register is incremented.

We rapidly prototyped the HPC logic and analyzed the overheads by synthesizing a high-level description of the MIPS processor using Xilinx Vivado High-level Synthesis (HLS) tool [56], which converts a C to RTL equivalent and targeted it to the Artix-7 FPGA. HLS enables us to understand the components that must be added into the architecture. We did not design any of the PQC algorithms using HLS, only the tailor-made HPCs for MIPS. While HLS is not ideal for security

Table 11. Hardware Area Overhead of Tailor-made HPCs for NIST PQC KEM Algorithms

KEM Algorithms	Crystals Kyber	NTRU	Saber	Classic McEliece	BIKE	Frodo-KEM	HQC	NTRU Prime
Utilized HPCs	ll, sl as, pp	l, n, nn, pp	b, n, nb, pp	an, nn, na, pp	a, n, aa, pp	a, l, ll, pp	b, ln, aa, pp	a, aa, al, pp
Area Overhead (%)	1.2	1.15	1.1	1.15	1.2	1.15	1.10	1.2

Table 12. Area and Power Overhead for Tailor-made HPCs

Design	lsi_10k Library				saed_90nm Library			
	Area (Sq. units)	Area Overhead (%)	Power (μW)	Power Overhead (%)	Area (Sq. units)	Area Overhead (%)	Power (μW)	Power Overhead (%)
MIPS Processor	126427	–	1.494	–	486952.24	–	13.602	–
MIPS + Four Tailor-made HPCs	130151	2.94	1.498	0.27	500093	2.69	14.071	3.45
MIPS + All Tailor-made HPCs	152543	20.65	1.515	1.41	643833.74	32.21	16.645	22.37

applications due to the potential side-channel leakage, this is not relevant in this analysis. We modified the decode unit of the MIPS processor to implement the tailor-made HPCs for each KEM from Table 8 and re-synthesized the modified MIPS processor with the HPC unit. Table 11 shows the hardware area overhead, in terms of LUTs, for supporting tailor-made HPCs for every NIST PQC KEM. The maximum overhead of 1.2% is for BIKE, which is negligible. When implemented on an Intel processor, this area overhead will be reduced considerably. There is no impact on the performance of the processor due to the HPC unit, since it resides outside the critical path. As shown in Table 5 and 6, only utilizing four tailor-made HPCs is sufficient for having a near perfect classification accuracy. While these overhead results are for the best tailor-made HPCs per each individual PQC algorithm, when implementing our design, we would not need to use all of them. Instead, we would utilize the four tailor-made HPCs that can classify the algorithms.

With our preliminary analysis of the required design flow, we develop a new RTL, separate from the RTL furnished by HLS, to obtain concrete power and area overhead values. We utilized the Synopsis Design Compiler for a logic synthesis of our tailor-made HPCs. We employed the lsi_10k and saed_90nm cell libraries. These are standard cell libraries available in our Synopsis Design Compiler tool. The area and power overhead numbers furnished from these utilized standard cell libraries will provide an indication of the incurred overhead. While other libraries could have been employed, the furnished power and area overhead numbers would be consistent with those obtained from our utilized libraries. Our results related to the power and area overhead acquired from our synthesis of the RTL are presented in Table 12.

We assume that the user is aware of which PQC algorithm is being utilized in the SSL/TLS implementations. As shown in Table 8, we utilized various groups of four tailor-made HPCs for ensuring our ML classifier furnished high performance metrics for one vs. all classifications. Consequently, the user will only monitor a specific set of HPCs based on the utilized PQC algorithm. As an example, from Table 8, for profiling *Crystals Kyber*, tailor-made HPCs *l*, *sb*, *sl*, and *as* should be utilized. However, to provide a through analysis, we have synthesized the MIPS processor when utilizing four tailor-made HPCs and all HPCs. When analyzing the incurred overhead of just four tailor-made HPCs, the area costs are only an additional 2.94% and 2.69% for the lsi_10k and saed_90nm libraries, respectively. Furthermore, the power overhead is negligible as the expenditure is in the micro-watt range. Unsurprisingly, incorporating all our tailor-made HPCs increases the utilized power and incurred area overhead. Both ARM and Intel processors have more complex architectures and incurs more area than MIPS. Therefore, when implementing tailor-made HPCs on an ARM or Intel processor, the overhead would be even smaller as the hardware overhead on a MIPS processor provides a pessimistic valuation.

Table 13. Transient Periods for Obtaining Execution Statistics

PQC Algorithm	Crystals Kyber	NTRU	Saber	Classic McEliece	BIKE	Frodo-KEM	HQC	NTRU Prime
Time	5.16μs	4.07μs	4.81μs	3.88 μs	5.31μs	3.03μs	6.81μs	3.23μs

In order to evaluate the time it takes to obtain the execution statistics, we measured the period of obtaining the tailor-made HPCs to the ML classifier furnishing a predication using the same data as in Tables 5 and 6. It should be noted the speed of the ML classifier is beyond the scope of this paper. The ML classifier can be implemented on a high-performance system, an embedded platform like a raspberry Pi, or a dedicated accelerator, like Google's Tensor Processing Unit (TPU), each with its own speed of operation. Therefore, depending on the user's resources, the latency of the ML operation can be determined. Table 13 presents our results for the four PQC KEM finalists and the four alternative algorithms. When analyzing the results, we can see that the transient period to obtain the execution statistics is within the 3 to 5μs range. As explained in Figure 9, our proposed tailor-made HPCs are not on the critical path. Moreover, our analysis of the time needed to obtain execution statistics corroborates our proposed methodology as a novel solution that provides fine-grain characterization capabilities and fast detection while intruding minimally on the processor architecture.

6 DISCUSSION

This study exposes the limits of ready-made HPCs in characterizing the behavior of implementations motivating fine-grain tailor-made HPCs. Following is a discussion as a Q&A.

Question: Only four tailor-made HPCs are utilized concurrently. Can additional tailor-made HPCs be accommodated?

Answer: Depending on the processor, a limited number of ready-made HPCs (about 4) can be concurrently monitored [4]. Ready-made HPC-based detection is constrained by this. Accordingly, results in Tables 2, 3, and 4 use four ready-made HPCs per classifier. Tailor-made HPCs face no such limitation. In Figure 9, we presented architectural adjustments and components that must be integrated into a processor to implement tailor-made HPCs. To add more tailor-made HPCs for real-time anomaly detection, we can append new logic to count the assembly instructions or sequence of instructions. In Table 12, we demonstrated that adding four tailor-made HPCs into the MIPS processor incurs 2.69% to 2.94% area overhead. Results in Tables 5, 6, and 7 that utilize tailor-made HPCs use four HPCs for a fair comparison with ready-made HPCs. Also, the four tailor-made HPCs yielded near perfect results obviating use of five or more HPCs.

Question: Tailor-made HPCs monitor either individual or and back-to-back instructions. Can this be extended to monitor triple or quadruple sequences? Will the improvements from these tailor-made HPCs become negligible because of optimized hardware such as out-of-order execution and multi-level caches?

Answer: Tailor-made HPCs can certainly be extended to include triple or quadruple assembly combinations. The inclusion of triple and quadruple tailor-made HPCs can be beneficial when profiling complex algorithms. A tailor-made HPC that monitors four arithmetic instructions in a row might prove beneficial. When four arithmetic instructions are detected, the count of arithmetic instructions is four, the count of two arithmetic instructions in a row is three, the count of three arithmetic instructions in a row is two, and the count of four arithmetic instructions in a row is one. Therefore, a longer custom-built HPC length of four arithmetic instructions captures more information than smaller lengths, but smaller ones are more frequent, yielding higher values. A tailor-made HPC to monitor four load instructions might be abundant in a dynamic trace of a

program, but four branch instructions may be uncommon. While tailor-made HPCs can be extended to include triples and quadruples, only some would be beneficial for profiling. For the PQC case study in this paper, individual and back-to-back assembly instructions was enough to build models that furnish high performance.

In Figure 9, we presented how our tailor-made HPC logic would be implemented. Our tailor-made HPC unit resides outside the critical path. In the MIPS processor, the instruction is fetched and decoded to understand the datapath that should be taken for handling the instruction. Concurrently, our HPC unit takes in this decoded instruction for producing our proposed tailor-made HPCs. Consequently, CPU optimization techniques, such as out-of-order execution and multi-level caches, would not be a deterrent to our methodology. Since we can extract the instruction in the decode stage before the optimized hardware techniques are incorporated, the tailor-made HPC trace is preserved. Regardless of the employed CPU optimization techniques, there can still be a plethora of instances where the dynamic trace would still contain specific assembly instruction sequences that would motivate utilizing triple and quadruple combinations for improved fine-grain characterization.

Question: The sampling interval, T_s, was set to 10,000. Why was this specific value used and how would using a smaller or larger number affect the tailor-made HPCs?

Answer: Ready-made HPCs are collected periodically at a specified interval (e.g. 1ms, 100ms, etc). To emulate such a periodic collection of tailor-made HPCs, we utilized a sampling interval, T_s. The value of T_s impacts the value and number of tailor-made HPC samples that are produced. When T_s is small, we get more tailor-made HPCs, but their values are small. Conversely, when T_s is large, the number of tailor-made HPCs is small and their values are high. In the former case, while an abundance of samples is beneficial, smaller values carry less information and leads to a plethora of overlapping samples. We set T_s to 10,000 to trade-off furnishing a sizable number of samples with ensuring that HPC values are substantial.

Question: Can tailor-made HPCs be used to profile other applications, and can they be employed for purposes outside of security? Is it possible that the benefit of tailor-made HPCs over ready-made HPCs is unique to the considered algorithm rather than a general issue? Are there some classes of domains for which a processor featuring tailor-made HPCs might be overkill?

Answer: Since tailor-made HPCs periodically monitor the dynamic assembly instructions being executed, they can be utilized to characterize any application. In this paper, we profiled PQC KEM algorithms as a case study to validate the coarse-grain versus fine-grain characteristics that exist in ready-made and tailor-made HPCs. We incorporated tailor-made HPCs into an anomaly detection framework for ensuring security in the SSL/TLS implementations.

In order to profile other applications, we would generate our tailor-made HPCs from the dynamic traces of these programs and train a ML classifier on this data. The utilized tailor-made HPCs could most certainly differ from the ones utilized in this paper. However, that is expected as the behavior and unique characteristics of one set of programs will differ from another set. Since our proposed methodology encompasses a plethora of instruction combinations, certain tailor-made HPCs may be better suited for profiling a specific application set. Additionally, feature selection techniques, such as the ones used in this paper, can be employed to select the best tailor-made HPCs for these application sets.

To demonstrate the capability of our proposed methodology on other applications, we collected ready-made and tailor-made HPCs for four programs from the MiBench benchmark: Fast Fourier Transform, Quicksort, String Search, and Basic Math. We then employed univariate and extras trees feature selection. For the ready-made HPCs, univariate furnished *CPU-cycles*, *Instructions*, *Ref-cycles*, and *L1-dcache-loads* while the best features from extra trees was *Bus-cycles*, *Ref-cycles*, *L1-dcache-stores*, and *L1-icache-load-misses*. For the tailor-made HPCs, univariate furnished *a*, *b*, *l*,

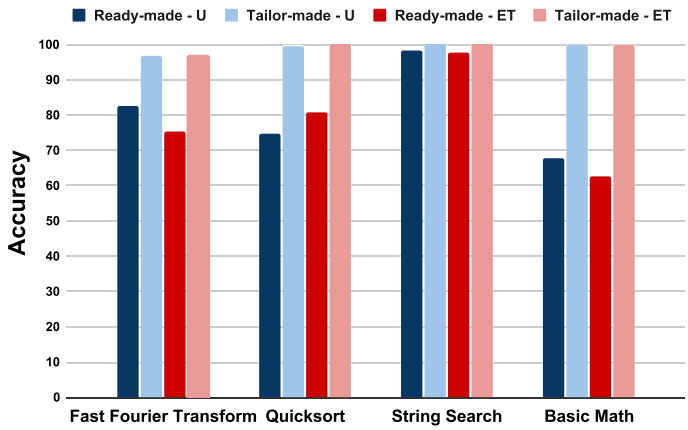


Fig. 10. Profiling Applications with Ready-made and Tailor-made HPCs. Univariate (U), Extra Trees (ET).

and *pp* and extra trees furnished *a*, *b*, *s*, and *pp*. We then trained separate Random Forest models utilizing the aforementioned features. Our results are presented in Figure 10. The best overall ready-made HPC accuracy was 80.84% when using univariate features. However, we had an average accuracy of 99.77% for univariate and 99.88% for extra trees when utilizing tailor-made HPCs. This validates that our proposed tailor-made HPCs can capture fine-grain characteristics for excellent profiling capabilities and is generalizable for any application. We note that prior research has shown that profiling assembly instructions, similar to our tailor-made HPCs, can be utilized for Malware and malicious firmware modification detection [31, 43].

The proposed tailor-made HPCs apply for other purposes, such as performance tuning and optimization. As an example, when minimizing the runtime and complexity of a program, it is desirable to reduce the amount of computations. Consequently, tailor-made HPCs can monitor the number of arithmetic instructions, enabling a user to tune their code to obtain the desired performance. There can certainly be specific situations where ready-made HPCs furnish high accuracy such as microarchitectural attack detection [16, 22]. However, utilizing tailor-made HPCs can still be beneficial, especially in critical systems where anomaly detection is imperative. We presented the coarse-grain issues of ready-made HPCs in Figure 3 and the fine-grain advantages of tailor-made HPCs in Figure 5. When comparing the results of Tables 2 and 3 to Tables 5 and 6, the tailor-made HPCs were able to significantly increase the accuracy for all PQC algorithms. The fine-grain characteristics of tailor-made HPCs enables ML models to furnish higher classification metrics than their ready-made counterparts. Therefore, while ML models trained on ready-made HPCs may furnish reasonable accuracy, utilizing tailor-made HPCs would boost this classification performance. This is critical in domains where safeguarding vital systems is of utmost importance.

Question: What compiler optimizations were used? What is the affect of different compiler optimizations? Can the different PQC-KEM be distinguished with different optimization levels?

Answer: We did not incorporate any specific compiler optimizations. If optimizations to reduce execution time and memory were employed, the dynamic trace of the application would change. Our proposed tailor-made HPCs are generated from the dynamic trace, and we assume the same optimization is employed for all collected traces. Therefore, regardless of which compiler optimization is used, the dynamic traces would still be unique, thereby yielding distinct tailor-made HPCs. Consequently, ML classifiers trained on tailor-made HPC data would be able to distinguish between the various PQC algorithms. However, in future, we intend to perform an analysis of the effects of compiler optimization on tailor-made HPC performance.

7 RELATED WORK

Since HPCs capture low-level micro-architectural events and characterize an implementation, they have been employed in a plethora of studies for classification including Malware detection. Malware detection using HPCs was first proposed in [35]. Later, researchers used ML models, such as Neural Networks and K-Nearest Neighbors, trained on HPCs for distinguishing implementations [12]. Additional studies explored HPC-based detection of kernel rootkits [58] and malicious firmware [57, 60]. Other HPC-based Malware detection schemes focused on securing vulnerable systems from hostile programs [29, 50]. Real-time monitoring of ready-made HPCs for malware detection in cyber-physical systems was proposed by [27]. The work in [46] proposed *2SMaRT*, which is a technique that selects the best ready-made HPCs via feature selection and utilizes a two-stage classifier for malware detection. Furthermore, *HPCMalHunter* was developed to produce behavioral vectors for applications through ready-made HPCs for real-time malware detection. Behavior-based Adaptive Intrusion detection in Networks (BRAIN) was developed and proposed in [23]. This framework utilizes ready-made HPCs to amalgamate network data, including statistics as well as profiled application information in order to identify Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks. Additional ready-made HPC-based Malware detection techniques have been proposed and discussed in [45, 59]. These techniques strive to secure and protect vulnerable systems from pernicious software programs. Furthermore, they attempt to leave the attacker with a tedious and onerous job of tampering with the trusted hardware architecture. Beyond ready-made HPCs, other hardware-based malware detection schemes were proposed in [63–65]. A moving target defense against adversarial attacks that manipulate HPC traces through induced perturbations was presented in [29]. An analysis of HPC-based ML classifier latency, power, and area was provided in [41]. A comparison of classification capabilities between ML models and the impact of the classifier parameters for Malware detection was presented in [28].

Studies used unique micro-architectural characteristics to distinguish a benign program from malware [42]. Starting from a feature space of 25 attributes, they created new feature spaces of branch, unigram, and bigram opcodes using feature reduction techniques such as the “discriminant feature variance" and “Markov blanket”. The system detected metamorphic worms and next-generation virus construction kit viruses with high accuracy, precision, and recall. Application of sub-semantic features for Malware detection was proposed by [39, 40]. An exploratory study of tailor-made HPCs and ML-based classifiers to detect morphing malware was reported in [43]. This is the first study of the trade-offs between ready-made and tailor-made HPCs.

8 CONCLUSION

This paper explored the limits of ready-made HPCs in characterizing algorithm implementations and proposed tailor-made HPCs as a simple and intuitive, yet novel way to address these challenges. The key takeaways of this paper are: Owing to their coarse-grain nature, ready-made HPCs are limited in uniquely characterizing algorithms. Tailor-made HPCs are effective in characterizing the algorithms, as they are fine-grained. In the PQC case study, tailor-made HPCs can identify a KEM PQC algorithm at run-time with a minimum of 98.99% accuracy. Tailor-made HPCs can identify if the algorithm is code or lattice-based with near 100% accuracy. The overhead of tailor-made HPCs is negligible (about 2.6% in area and no impact on timing), when implemented in a MIPS processor. One can extend the characterization and classification techniques to NIST PQC digital signature algorithms such as XMSS and Crystals Dilithium and to other applications. Moreover, tailor-made HPCs can be extended to monitor triplets and quadruplets when profiling complex algorithms.

REFERENCES

[1] Erdem Alkim. 2017. FrodoKEM. https://frodokem.org/.

[2] Nicolas Aragon et al. 2017. BIKE: Bit flipping key encapsulation. *NIST Submissions* 1, 1 (2017), 1–53.

[3] Kanad Basu et al. 2019. NIST post-quantum cryptography-a hardware evaluation study. *IACR Cryptol* 2019, 1 (2019), 47.

[4] Kanad Basu et al. 2019. A theoretical study of hardware performance counters-based malware detection. *IEEE Transactions on Information Forensics and Security* 15, 1 (2019), 512–525.

[5] Daniel J Bernstein et al. 2016. NTRU prime. *IACR Cryptol* 2016, 1 (2016), 1–63.

[6] Daniel J Bernstein et al. 2017. Classic mceliece: Conservative code-based cryptography. *NIST Submissions* 1, 1 (2017), 1–25.

[7] Joppe Bos et al. 2018. CRYSTALS-kyber: A CCA-secure module-lattice-based KEM. In *European Symposium on Security and Privacy*. IEEE, USA, 353–367.

[8] Franco Callegati et al. 2009. Man-in-the-middle attack to the HTTPS protocol. *IEEE Security & Privacy* 7, 1 (2009), 78–81.

[9] coral.ai. 2021. Retrain a classification model on-device with backpropagation | Coral. https://coral.ai/docs/edgetpu/retrain-classification-ondevice-backprop/overview.

[10] Dancho Danchev. 2019. 'Evil Maid' USB stick attack keylogs TrueCrypt passphrases | ZDNet. https://www.zdnet.com/article/evil-maid-usb-stick-attack-keylogs-truecrypt-passphrases/.

[11] Sanjeev Das et al. 2019. SoK: The challenges, pitfalls, and perils of using hardware performance counters for security. In *Symposium on Security and Privacy*. IEEE, USA, 20–38.

[12] John Demme et al. 2013. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 559–570.

[13] Sai Manoj Pudukotai Dinakarrao et al. 2019. Adversarial attack on microarchitectural events based malware detectors. In *Design Automation Conference*. IEEE, USA, 1–6.

[14] Simon Eberz et al. 2012. A practical man-in-the-middle attack on signal-based key generation protocols. In *European Symposium on Research in Computer Security*. Springer, USA, 235–252.

[15] Geeks for Geeks. 2020. Python Programming Examples - GeeksforGeeks. https://www.geeksforgeeks.org/python-programming-examples/searchingandsorting.

[16] Berk Gulmezoglu et al. 2019. FortuneTeller: Predicting microarchitectural attacks via unsupervised deep learning. *ArXiv* 1, 1 (2019), 1–16.

[17] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE International Workshop on Workload Characterization. WWC-4*. IEEE, USA, 3–14.

[18] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, and Hiroaki Takada. 2009. Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis. *Journal of Information Processing* 17 (2009), 242–254.

[19] Robert Hecht-Nielsen. 1992. Theory of the backpropagation neural network. In *Neural Networks for Perception*. Elsevier, USA, 65–93.

[20] Jeffrey Hoffstein et al. 1998. NTRU: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*. Springer, USA, 267–288.

[21] Intel. 2021. Intel Collaborates with Microsoft against Cryptojacking. https://www.intel.com/content/www/us/en/newsroom/news/intel-microsoft-scale-threat-detection-cryptojacking.htmlgs.zw9v32.

[22] Md Shohidul Islam et al. 2020. ND-HMDs: Non-differentiable hardware malware detectors against evasive transient execution attacks. In *International Conference on Computer Design*. IEEE, USA, 537–544.

[23] Vinayaka Jyothi et al. 2016. Brain: Behavior based adaptive intrusion detection in networks: Using hardware performance counters to detect ddos attacks. In *International Conference on VLSI Design*. IEEE, USA, 587–588.

[24] Angshuman Karmakar et al. 2018. Saber on arm. cca-secure module lattice-based key encapsulation on arm. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 1, 1 (2018), 243–266.

[25] Keras. 2021. Keras layers API. https://keras.io/api/layers/.

[26] kernel.org. 2020. Perf Events. https://www.kernel.org/doc/html/latest/admin-guide/perf-security.html.

[27] Prashanth Krishnamurthy et al. 2019. Anomaly detection in real-time multi-threaded processes using hardware performance counters. *IEEE Transactions on Information Forensics and Security* 15, 1 (2019), 666–680.

[28] Abraham Peedikayil Kuruvila et al. 2020. Analyzing the efficiency of machine learning classifiers in hardware-based malware detectors. In *Computer Society Annual Symposium on Very Large Scale Integration*. IEEE, USA, 452–457.

[29] Abraham Peedikayil Kuruvila et al. 2020. Defending hardware-based malware detectors against adversarial attacks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 1, 1 (2020), 1–13.

[30] Abraham Peedikayil Kuruvila et al. 2020. Defending hardware-based malware detectors against adversarial attacks. *arXiv preprint arXiv:2005.03644* 1, 1 (2020), 1–13.
[31] Abraham Peedikayil Kuruvila et al. 2021. Hardware-assisted detection of firmware attacks in inverter-based cyber-physical microgrids. *International Journal of Electrical Power and Energy Systems* 132, 1 (2021), 107–150.
[32] Scikit Learn. 2018. Feature Importance scikit-learn 0.23.2 documentation. https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html.
[33] Scikit Learn. 2018. Feature selection scikit-learn 0.23.2 documentation. https://scikit-learn.org/stable/modules/feature_selection.html.
[34] Osnat Levi. 2020. Pin Instrumentation Tool. https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html.
[35] Corey Malone et al. 2011. Are hardware performance counters a cost effective way for integrity checking of programs. In *Workshop on Scalable Trusted Computing*. ACM, USA, 71–76.
[36] Carlos Aguilar Melchor. 2017. HQC. http://pqc-hqc.org/index.html.
[37] NIST. 2020. Post-Quantum Cryptography | CSRC. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.
[38] OpenSSL. 2020. OpenSSL Cryptography and SSL/TLS toolkit. https://www.openssl.org/.
[39] Meltem Ozsoy et al. 2016. Hardware-based malware detection using low-level architectural features. *IEEE Trans. Comput.* 65, 11 (2016), 3332–3344.
[40] Meltem Ozsoy et al. 2015. Malware-aware processors: A framework for efficient online malware detection. In *International Symposium on High Performance Computer Architecture*. IEEE, USA, 651–661.
[41] Nisarg Patel et al. 2017. Analyzing hardware based malware detectors. In *Design Automation Conference*. IEEE, USA, 1–6.
[42] Jithu Raphel et al. 2017. Heterogeneous opcode space for metamorphic malware detection. *Arabian Journal for Sci. and Eng.* 42, 2 (2017), 537–558.
[43] Aditya. Rohan et al. 2019. Can monitoring system state+ counting custom instruction sequences aid malware detection?. In *Asian Test Symposium*. IEEE, USA, 61–615.
[44] Alexander Russell et al. 2016. Cliptography: Clipping the power of kleptographic attacks. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, USA, 34–64.
[45] Hossein Sayadi et al. 2018. Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification. In *Design Automation Conference*. IEEE, USA, 1–6.
[46] Hossein Sayadi et al. 2019. 2SMaRT: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection. In *Design, Automation and Test in Europe Conference*. IEEE, USA, 728–733.
[47] Bruce Schneier et al. 2015. Surreptitiously weakening cryptographic systems. *IACR* 2015, 1–26 (2015), 97.
[48] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review* 41, 2 (1999), 303–332.
[49] Deepraj Soni et al. 2020. Hardware Architectures for Post-Quantum Digital Signature Schemes.
[50] Adrian Tang et al. 2014. Unsupervised anomaly-based malware detection using hardware features. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, USA, 109–129.
[51] Alexander Tereshkin. 2010. Evil maid goes after PGP whole disk encryption. In *Proceedings of the 3rd International Conference on Security of Information and Networks*. ACM, USA, 2–2.
[52] Leif Uhsadel et al. 2008. Exploiting hardware performance counters. In *Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, USA, 59–67.
[53] Leif Uhsadel et al. 2008. Exploiting hardware performance counters. In *Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, USA, 59–67.
[54] Common Vulnerabilities and Exposures. 2014. CVE-2014-3566. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3566.
[55] Richard Wilkins et al. 2013. UEFI secure boot in modern computer security solutions. In *UEFI Forum*. Intel, USA, 1–10.
[56] Xilinx. 2019. Vivado High-Level Synthesis. https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.
[57] Wang Xueyang et al. 2015. Confirm: Detecting firmware modifications in embedded systems using hardware performance counters. In *International Conference on Computer-Aided Design*. IEEE, USA, 544–551.
[58] Wang Xueyang et al. 2015. Reusing hardware performance counters to detect and identify kernel control-flow modifying rootkits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 3 (2015), 485–498.
[59] Wang Xueyang et al. 2016. Hardware performance counter-based malware identification and detection with adaptive compressive sensing. *ACM Transactions on Architecture and Code Optimization* 13, 1 (2016), 1–23.
[60] Wang Xueyang et al. 2016. Malicious firmware detection with hardware performance counters. *IEEE Transactions on Multi-Scale Computing Systems* 2, 3 (2016), 160–173.

[61] Ilsun You et al. 2010. Malware obfuscation techniques: A brief survey. In *Conference on Broadband, Wireless Computing, Communication and Applications*. IEEE, USA, 297–300.
[62] Boyou Zhou et al. 2018. Hardware performance counters can detect malware: Myth or fact?. In *Asia Conference on Computer and Communications Security*. ACM, USA, 457–468.
[63] Liwei Zhou et al. 2016. Hardware-based workload forensics and malware detection in microprocessors. In *International Workshop on Microprocessor*. IEEE, USA, 45–50.
[64] Liwei Zhou et al. 2016. Hardware-based workload forensics: Process reconstruction via TLB monitoring. In *International Symposium on Hardware Oriented Security and Trust*. IEEE, USA, 167–172.
[65] Liwei Zhou et al. 2018. Hardware-assisted rootkit detection via on-line statistical fingerprinting of process execution. In *Design, Automation and Test in Europe Conference*. IEEE, USA, 1580–1585.

Received April 2021; revised June 2021; accepted July 2021