Balancing Security and Efficiency: System-Informed Mitigation of Power-Based Covert Channels

Jeferson González-Gómez, *Graduate Student Member, IEEE*, Mohammed Bakr Sikal, Heba Khdr, Lars Bauer, and Jörg Henkel, *Fellow, IEEE*

***Abstract*—As the digital landscape continues to evolve, the security of computing systems has become a critical concern. Power-based covert channels (e.g., thermal covert channel s (TCCs)), a form of communication that exploits the system resources to transmit information in a hidden or unintended manner, have been recently studied as an effective mechanism to leak information between malicious entities via the modulation of CPU power. To this end, dynamic voltage and frequency scaling (DVFS) has been widely used as a countermeasure to mitigate TCCs by directly affecting the communication between the actors. Although this technique has proven effective in neutralizing such attacks, it introduces significant performance and energy penalties, that are particularly detrimental to energy-constrained embedded systems. In this article, we propose different system-informed countermeasures to power-based covert channels from the heuristic and machine learning (ML) domains. Our proposed techniques leverage task migration and DVFS to jointly mitigate the channels and maximize energy efficiency. Our extensive experimental evaluation on two commercial platforms: 1) the NVIDIA Jetson TX2 and 2) Jetson Orin shows that our approach significantly improves the overall energy efficiency of the system compared to the state-of-the-art solution while nullifying the attack at all times.**

***Index Terms*—Countermeasures, covert channels, energy efficiency, machine learning (ML), security threats, security.**

I. Introduction

In TODAY's evolving digital landscape, the significance of security and data privacy in the modern computing environment cannot be overstated. Within this context, covert channel communication has been recently highlighted as an emerging security threat for modern computing systems [1]. In such a domain, power-based covert channels leverage the power consumption of a system to communicate information between malicious applications in a stealthy manner. The typical mechanism to modulate the power of a system is through intensive computation on the device's processing elements, such as CPU [2], GPU [3] or FPGA-based components [4]. Commonly, power-based covert channels are implemented as TCCs, where the processing power translates into temperature variations that are used as the medium for the communication.

In order to mitigate the threat of TCCs, and power-based covert channels in general, several detection and countermeasure techniques have recently emerged [5], [6], [7], [8], [9], especially for general-computing devices. Even in such platforms, the challenge of such techniques resides in effectively detecting and mitigating the attack while reducing the performance impact on the system. In the countermeasure domain for such channels, DVFS has been proposed [5], [6], as the default mechanism to tackle the attack. By dynamically switching between high and low frequencies for the processing device, the countermeasure technique affects the power consumption of the device and the system, and hence directly interferes with the covert communication. However, as it has been shown, reducing the frequency of the CPUs affects the performance of applications executing there. Current countermeasures do not consider the system information in the techniques, which can significantly affect its energy consumption and performance. For a many-core system, for example, when the attack is present at all times, the performance loss for a benchmark application set due to the countermeasure has been known to reach up to 25% [6], whereas the effect on the energy on the system due to the countermeasure has not properly been evaluated in the state-of-the-art. While energy efficiency might not be the most relevant metric for general-purpose computing platforms, for modern energy-constraint embedded systems it is a critical factor, hence we target our work to such devices.

To highlight the effect on performance and energy of the uninformed state-of-the-art DVFS countermeasure for power-based covert channels, especially on embedded systems, we show the following motivational example.

1) Motivational Example: Fig. 1 shows different scenarios for an application set executed on an NVIDIA Jetson TX2 embedded board. As shown, the device has two CPU clusters: 1) an ARM CPU cluster with a Quad-Core ARM Cortex-A57 and 2) a dual-core NVIDIA Denver 2 cluster. As initial state, five applications from the SPEC2006 [10] benchmark suite and a malicious TCC transmitter (named "tcc") are executed on the system. Shortly after beginning the execution, the malicious application's core is detected, using a proven detection technique (e.g., [8] and [9]). As a countermeasure, we first apply DVFS to the core where the malicious application executes,

Manuscript received 31 July 2024; accepted 1 August 2024. This article was presented at the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES) 2024 and appeared as part of the ESWEEK-TCAD Special Issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Jeferson González-Gómez.)*

Jeferson González-Gómez is with the Instituto Tecnológico de Costa Rica, Cartago 30101, Costa Rica, and also with the Chair for Embedded Systems, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany (e-mail: jeferson.gonzalez@kit.edu).

Mohammed Bakr Sikal, Heba Khdr, and Jörg Henkel are with the Chair for Embedded Systems, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany (e-mail: bakr.sikal@kit.edu; heba.khdr@kit.edu; henkel@kit.edu).

Lars Bauer resides in Karlsruhe, Germany.

Digital Object Identifier 10.1109/TCAD.2024.3438999

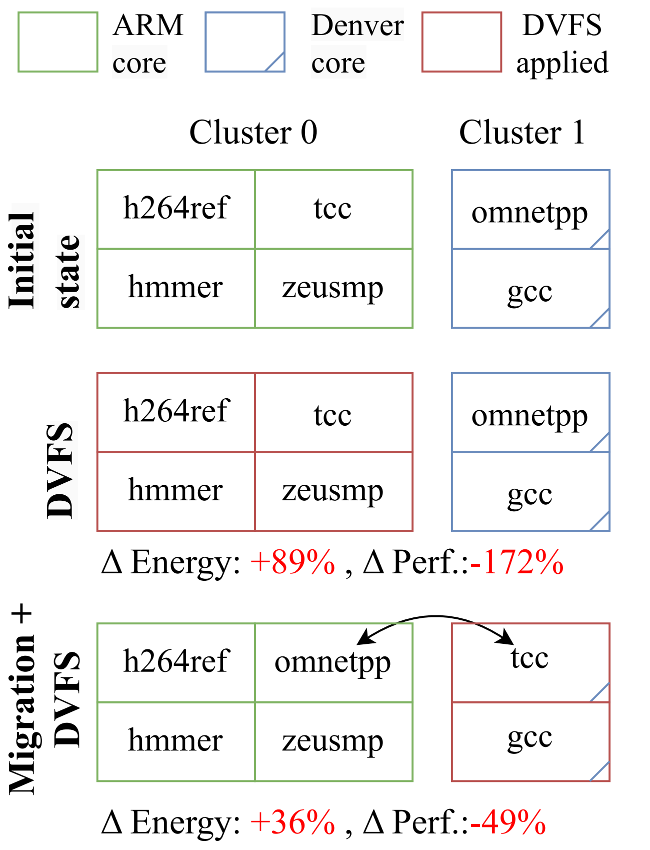

Fig. 1. Effect of applying migration and DVFS on the energy and performance (makespan) of the system.

as it is the state-of-the-art countermeasure technique [6]. Because of the clustering, the DVFS technique is applied to the whole ARM cluster, producing high performance and energy penalties in the system since all other applications executing in the same cluster are affected.

In order to show how this overhead can be reduced, we depict an alternative scenario, where we arbitrary migrate the malicious application from an ARM core to a Denver core, by dynamically exchanging the cores where *omnetpp* and *tcc* are executing, before applying DVFS. Because the offending application is now on the Denver cluster, we apply DVFS on that cluster, therefore affecting only one other application. By doing so, we are able to massively reduce the energy and performance overhead in the system.

As it is shown, while this arbitrary decision is able to reduce the overhead penalties in comparison to blindly applying DVFS, *the performance and energy penalties are still significant*. In a real setup, the dynamic state of the system (i.e., type of applications, CPUs' frequencies, system load, etc.,) creates a complex environment where the ideal execution scenario for the current application set is not easy to predict. Because of this, we propose to employ system information as input to the countermeasure technique in order to tackle the attack holistically and efficiently.

In this article, we focus on the challenge of mitigating power-based attacks in an energy-efficient manner. We seek to highlight this problem, which has not been done properly for embedded devices, and address it, by including information about the system as input to the countermeasure itself. We propose the use of *system-informed techniques* based on *the combination of DVFS and dynamic task migration* to mitigate power-based covert channels. By doing so, we are able to reduce the energy and performance penalties of the countermeasures on the real platform, while still mitigating the attack.

2) Contributions: The contributions of our work are the following.

1) We propose for the first time the use of system-informed techniques as countermeasures for power-based covert channels.
2) We devise new countermeasures to power-based covert channels from the heuristics and ML domain that combine for the first time DVFS and dynamic task migration to tackle the attack.
3) We deploy both our proposed countermeasures and the state-of-the-art DVFS approach on a real embedded platform. Our extensive evaluation demonstrates how our techniques mitigate the attack while significantly reducing both the energy and the performance penalties.

II. Related Works

A. Power-Based Covert Channels

In power-based covert channels, malicious applications manipulate the power consumption of a device to communicate information in a stealthy manner. While some approaches leverage the power directly as the medium for the communication for memory [11], CPUs [12] or cross devices communication [13], the most common power-based covert channels from a countermeasure perspective are TCCs, where the means for the communication is the temperature variation due to the power changes. Since the early implementation of TCCs on multicore systems [2], faster and more reliable covert channel implementations have appeared by leveraging new modulation and encoding mechanisms [14], [15]. These techniques have shown stable and improved transmission rates (bps) with significantly low-error rates between $1\%-11\%$ [1]. These types of power-based covert channels have spread to utilize different new resources, such as GPUs [3], 3-D multicore systems [16], and solid-state disks (SSDs) [17].

B. Detection Techniques

Detection techniques for power-based covert channels stem again from the TCC field. As detection mechanisms, approaches employ both time and frequency domain information about the performance of the cores, in instructions per cycle (IPC) or instructions per second (IPS), to determine which core acts as a transmitter. Since the covert channel requires the malicious transmitter application to increase and decrease the power through periodic patterns of intensive computation and idle states, the core where the malicious transmitter application executes can be identified by performance analysis [6]. With the advent of new attacks, detection approaches have evolved from threshold-based heuristics [6] to lightweight ML-based approaches [8], [9], which can accurately identify the core where the transmitter application is executed in fast response times (i.e., around 2 s for the frequency domain-based approach). As a base for our countermeasure technique, we assume that such a detector already exists in the system, as the aforementioned state-of-the-art approaches have shown great performance at detecting TCCs.

C. Countermeasures to Power-Based Covert Channels

As previously indicated, countermeasures to power-based covert channels are mostly focused on TCCs. Notably, since no detection technique is 100% accurate, simply halting a potentially offending application is not an option. The state-of-the-art for such attacks covers mainly noise and DVFS-based approaches. dynamic voltage and frequency scaling (DVFS) is a technique that has been historically used as a resource management mechanism to optimize power, temperature, aging, and energy efficiency in different domains [18], [19]. As a countermeasure to power-based covert channels, DVFS has been proven [3], [5], [6] as a successful mechanism to mitigate the attack. Scaling up or down the frequency of a processing element changes its power and temperature response, hence directly jamming the communication medium between transmitter and receiver in a covert channel. However, throttling the cores can produce significant performance degradation in the applications that execute there. Even in a simulation environment, when an attack is present at all times in a many-core environment, DVFS has been reported to produce 25% performance loss [6] in an application set. Nonetheless, DVFS remains the state-of-the-art countermeasure to power-based attacks, as it directly targets the root medium for the transmission: power consumption. The evaluation of DVFS as a countermeasure to power-based covert channels on embedded systems has not been done previous to this work. As we show in Section IV-D the performance loss of solely applying DVFS on an embedded system reaches around 70% on average, with some applications experiencing losses greater than 150% (see Fig. 1). We compare our countermeasure techniques against the β-based DVFS-only approach [6], as it remains the reference countermeasure technique.

Other countermeasures have employed power-based noise [7], [20] to interfere with transmission, resulting in a power overhead similar to that of the DVFS approaches. Although the performance overhead of these noise-based approaches in the system has not been evaluated, the jamming-noise approach requires periodic unnecessary processing on the core where the potential attacker executes at all times, which restricts the performance of other applications executing on the same core or cluster. Moreover, this approach has been shown to fail to mitigate enhanced attacks [6].

Task migration has previously been used as a standalone countermeasure to both side and covert channels. Specifically in many-core systems [21] proposed a task migration heuristic for side channel mitigation aimed at avoiding cache colocation between attacker and victim. Similarly, in [22], dynamic task migration has been employed to mitigate TCCs by increasing the physical distance between the transmitter and any potential receiver core. Although effective, this countermeasure assumes a multicore system in which the physical separation could be significant enough to produce heat transfer decay. However, increasing this separation distance might be impossible in an embedded system with just a few cores. Moreover, this countermeasure assumes that the receiver can only read the thermal sensor on its core to decode the signal. In practice, any user-space application is commonly allowed to read most of the thermal sensors of the system, which means that no matter the distance between the transmitter and the receiver, the transmission would still be possible. In contrast, our techniques employ dynamic task migration as a technique to optimize efficiency while the countermeasure mechanism remains as DVFS, which directly affects the power, hence mitigating the attack from its root.

In summary, no other countermeasure in the state-of-the-art for covert channels has leveraged task migration combined with DVFS to mitigate the attacks. Moreover, to the best of our knowledge, no other countermeasure in the state-of-the-art has employed a system-informed technique to tackle both security and efficiency.

D. Combining Task Migration and DVFS

The combination of task migration and DVFS to navigate the complex runtime dynamics has been proposed in the literature to achieve different optimization goals in nonsecurity domains. Through the joint usage of task migration and DVFS, Pourmohseni et al. [23] aimed to maximize the overall system performance under a temperature constraint. Targeting the same goal, a more recent work [24] proposed a cache contention-aware ML-based technique that also employs task migration and DVFS jointly. Using the same mechanism, Marinakis et al. [25] also aim at maximizing performance but under a power budget constraint. As shown by these recent works, the task of optimizing different system goals is not simple. The combination of task migration and DVFS has then proven to be a valid mechanism to achieve this optimization. In a similar way, we seek to bring this resource-management mechanism to the security domain for the first time, to mitigate power-based covert channels, while optimizing the energy efficiency of the system.

III. System-Informed and Efficient Mitigation of Power-Based Covert-Channel Attacks

As previously discussed in the motivational example in Section I, blindly applying DVFS, as done in the state-of-the-art, can lead to high performance and energy penalties. To tackle this problem, we intend to introduce system information into the covert channel migration strategies by combining task migration with DVFS. Each mitigation strategy attempts to address the following challenge at runtime: Once an attacker is detected, What is the best state the system should transition into (enforced by task migration) before applying DVFS, such that energy efficiency is maximized while the attack is mitigated? In the following subsections, we show the design and implementation considerations for our proposed techniques, which seek to address exactly this challenge.

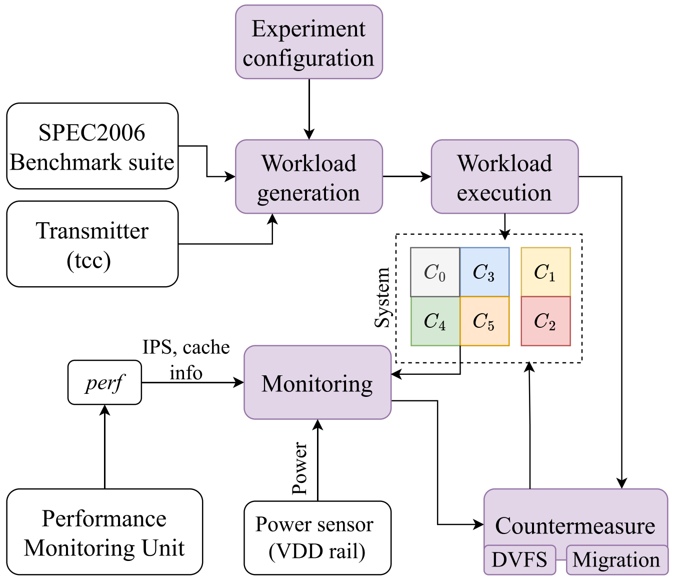

Fig. 2. Overview of the orchestration resource management application.

A. Enabling System and Application Awareness

As previously introduced, in this article we propose system-informed techniques to mitigate power-based covert channels through the combination of DVFS and dynamic task migration. To support the techniques, we implemented a resource management orchestration application. This orchestration application, depicted as an overview in Fig. 2, is in charge of generating the experiment parameters (e.g., workloads and initial mapping configurations), launching the applications, periodically monitoring the system metrics, selecting the countermeasure policy, and finally enforcing the technique by migrating the applications and applying cluster-level DVFS where required by the countermeasure technique.

For the application set, we generate random workloads consisting of combinations of applications from the SPEC2006 benchmark and a functional power-based covert channel transmitter in a one-application-per-core manner. Moreover, our monitoring tool periodically gathers execution metrics from the system, CPUs, and the cache, such as IPS, cache accesses, cache misses, and system power every 100 ms. We use *perf* [26] as the back-end mechanism to collect the performance counters information (both CPU and cache), and the platform's power, we read the board's power sensor accessible through Linux the file system.

B. Heuristic-Based Mitigation

To reduce the overhead in the system due to an uninformed countermeasure, we propose a simple yet effective technique that considers the performance of the cores to decide an efficient application mapping at run-time.

Our worst-performing cluster—best-performing core (WPCBPC) heuristic follows the principle of reducing the effect of performance penalty due to DVFS. It does so by moving the attacker application first to the cluster that has the worst performance. Then within that cluster, it selects the most performing core as the candidate for migration before

Algorithm 1: Our WPCBPC Heuristic

```
1  Input: $attack\_core$, $curr\_mapping$
   Result: $new\_mapping$: New application mapping
           configuration
2  $cluster0\_cores \leftarrow \{0, 3, 4, 5\}$;
3  $cluster1\_cores \leftarrow \{1, 2\}$;
4  $IPS\_cluster0 \leftarrow 0$;
5  $IPS\_cluster1 \leftarrow 0$;
6  $all\_IPS \leftarrow \{\}$;
7  for core in $all\_cores$ do
8      $all\_IPS$.push(getIPS(core)) ;        /* Gets the
       performance for each core */
9  end
10 for core in $cluster0\_cores$ do
11     $IPS\_cluster0$.push($all\_IPS[core]$) ;
       /* Performance for Cluster 0 */
12 end
13 for core in $cluster1\_cores$ do
14     $IPS\_cluster1$.push($all\_IPS[core]$);
       /* Performance for Cluster 1 */
15 end
16 $target\_cluster \leftarrow cluster1\_cores$;
17 if $average(IPS\_cluster0) < average(IPS\_cluster1)$ then
18     $target\_cluster \leftarrow cluster0\_cores$;
19 end
20 max $\leftarrow 0$;
21 $cid \leftarrow -1$;
22 for core in $target\_cluster$ do
23     if $all\_IPS[core] >$ max then
24         max $\leftarrow all\_IPS[core]$;
25         $cid \leftarrow core$ ;         /* Finds the best
           performing core */
26     end
27 end
28 $new\_mapping \leftarrow curr\_mapping$;
29 $moving\_app \leftarrow curr\_map[cid]$;
30 $new\_mapping[attack\_core] = moving\_app$;
31 $new\_mapping[cid] = curr\_mapping[attack\_core]$;
32 return $new\_mapping$;
```

enforcing the new application mapping dynamically in the next scheduling epoch. The full pseudo-algorithm for this technique is shown in Algorithm 1. In the algorithm, we first collect the performance information (IPS) from each core (lines 7–15). Then we identify the worst-performing cluster, by comparing the averages of the accumulated IPS (lines 16–19) over the last second of execution. The reason for selecting the worst-performing cluster as the host to the potential attacker is that the application of the subsequent DVFS policy will affect the overall performance the least. To further enforce this, we then select the best-performing core from that cluster (i.e., g the core with the highest IPS over the last second of execution) as the final candidate to which the malicious application should be migrated (lines 22 and 27). In this way, the best-performing application from the soon-to-be-affected cluster will not be affected by the performance penalty due to

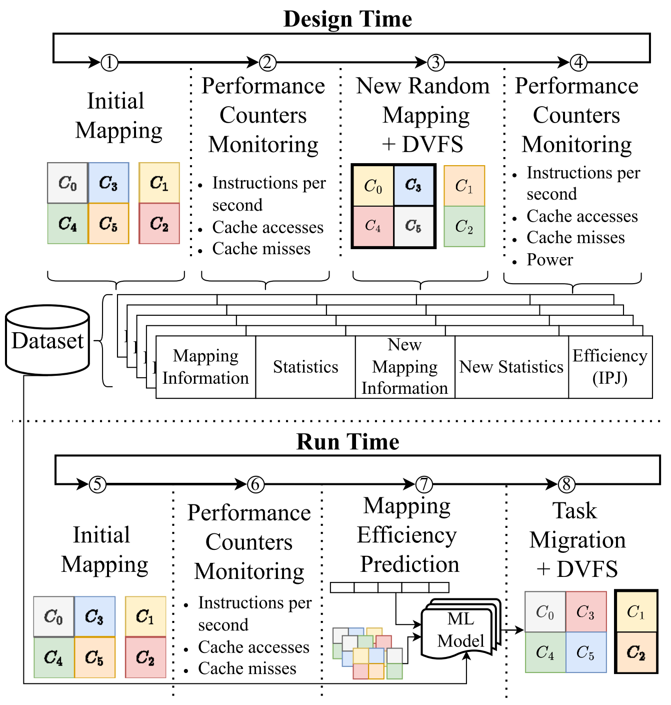

Fig. 3. Overview of the ML-based countermeasure techniques.

DVFS. Finally, we create the new mapping configuration in the system (lines 28 and 31) where the cores belonging to the attacker and the best-performing application on the candidate cluster have been exchanged.

C. *Machine Learning-Based Mitigation*

To further explore other system-informed techniques, in this section, we introduce some ML-based countermeasures to power-based covert channels. Through different supervised ML algorithms, we seek for our models to learn the behavior of the system when mitigating the attack, and predict the best-possible task migration scenarios at run-time. Instead of relying on heuristics, this approach attempts to quantify the impact of different mitigation strategies on the overall energy efficiency of the system.

Fig. 3 shows a high-level overview of our ML-based mitigation. Our approach follows a four-step process both for design and runtime. First, at design time, the process starts by generating a random workload from the SPEC2006 benchmark suite plus the malicious application, as an initial mapping ①. Then we start the execution of the workload and wait for a random delay (i.e., between 1 and 10 seconds) before starting the monitoring ② to encounter different execution phases for the applications. After that, we collect the performance (IPS) for each core, cache misses and accesses, and system power information periodically every 100 ms for a window of 1 s. When the collecting period expires, we create a new random mapping for the current workload and then apply DVFS to the cluster to which the malicious application will be moved ③. Then we set the new core's affinity to each application, which enforces the dynamic task migration to the workload following the new mapping. Finally, we again collect the statistics for the workload under the new mapping configuration ④. In addition to the mentioned metrics, we compute the energy efficiency (Instructions per Joule) obtained as a consequence of the migration and DVFS for the new mapping configuration. We repeated this process more than 5000 times, collecting over 180 individual data points per iteration. With the obtained metrics for all the iterations, we form a training data set where each row contains a representation of the original mapping, its statistics, the representation of the new mapping, the new statistics, and the obtained energy efficiency. This dataset is then used to train the ML models we employ for the countermeasure technique.

At run time, our techniques are applied in a continuous process which starts from an initial running mapping configuration ⑤. Then, we accumulate and collect the execution metrics over the most recent second of execution ⑥. After that, for each possible nonredundant mapping variation, we call the ML model to predict energy efficiency ⑦. It is important to note that in order to reduce the number of possible mapping predictions, we ignore mapping variants where all applications would reside in the same cluster but in different cores. Although technically different, these are *redundant mappings* in the sense that all applications are set to execute within the same DVFS domain, and the DVFS action would affect the same applications in the same manner. After the energy efficiency prediction is performed for all possible mappings, we select the new mapping variant that produces the highest-efficiency value as the new mapping configuration for the system. Finally, we enforce this new mapping configuration by applying task migration and then DVFS to the cluster where the malicious application is set to execute ⑧. This process is then repeated until the workload finishes the execution.

The following subsections describe in more detail each one of the steps involved in the design and implementation of the ML-based techniques.

1) Training Data Generation and Preprocessing: Following the steps depicted in the design-time phase of Fig. 3, we generate a dataset of ∼1M data points. The dataset first undergoes standardization and scaling in order to adjust the distribution of each feature to have a mean of zero and a standard deviation of one, thereby enhancing the model's ability to converge during training. The scaling parameters are saved for usage at runtime. Finally, we perform a random split of 80% / 20% training / testing of the data set to prepare for the model training phase.

2) Feature Selection and Model Training: The problem at hand is a *regression* problem that can be solved with various ML algorithms, e.g., decision trees, random forests (RFs), neural networks (NNs), etc., where the label is set as the energy efficiency of the system *after* migration. Therefore, we first train different *regressors* from the *scikit-learn* Python library [27] with their default parameters using the raw dataset in order to identify the most promising algorithm for this specific problem. Table I shows the root-mean-square error (RMSE), R2 and mean-absolute error (MAE) scores achieved by the different models, with the extreme gradient boosting (XGBoost), RF and NN models outperforming the other regressors. We then focus on training optimized models with each of the three selected algorithms, as follows.

We start with *the XGBoost model* since feature selection can be performed implicitly as part of its learning process. The algorithm identifies the most informative features through its tree-building mechanism, where it calculates a *gain* metric for each feature. This *gain* reflects the contribution of the feature to the model's predictive accuracy, with higher values indicating more importance and correlation with the efficiency label. To further refine the feature selection process, hyperparameter tuning is conducted through a grid search technique, aided by the *GridSearch* library from *scikit-learn*. The hyperparameters explored in the search include the number of estimators (up to 300), maximum depth of the trees (up to 9), learning rate, subsample ratio, and the column sample by tree. After exploring this search space, the following features are shortlisted for each core: cache accesses and misses, retired instructions, the encoded ID of the running applications, and the energy efficiency of the system *before* migration. The grouping of features per core is of particular importance, as *it guides the model to learn the individual characteristics of the core as part of its cluster*. The grid search yielded a final XGBoost model that used 10 estimators with a maximum tree depth of 6, which achieved a very high-prediction accuracy with MAE and RMSE scores of barely 0.19×10^9 and 0.31×10^9, respectively.

Based on the feature importance insights obtained from training the XGBoost model, the same list of features is maintained for training the *NN model*. The search for the model topology, including the depth and breadth of layers, is performed using the *Keras Tuner*. The nonlinear ReLU activation function is incorporated in each hidden layer to introduce nonlinearity and the *Adam* optimizer is used to effectively manage back-propagation and the learning rate during training. The final obtained NN model consists of 3 hidden layers with 32, 32, and 16 neurons. Though slightly less accurate than the XGBoost model, the NN model also achieved a very high-prediction accuracy of the energy efficiency label, with MAE and RMSE scores of 0.29×10^9 and 0.45×10^9, respectively.

Finally, with the same list of features as the two previous models, we train a *RF model,* by using *GridSearch* to explore a search space of parameters, including the number of trees in the forest and the maximum depth of each tree. The final model, which uses 100 trees, achieves a slightly higher-prediction accuracy compared to the NN model, with MAE and RMSE scores of 0.26×10^9 and 0.45×10^9, respectively.

TABLE I
PREDICTION ACCURACY OF DIFFERENT ML ALGORITHMS ON THE VALIDATION DATASET

Regression Model	**RMSE** (10^9)	**R² Score**	**MAE** (10^9)
Linear	0.47	0.76	0.31
Ridge	0.49	0.73	0.32
Lasso	0.58	0.63	0.39
Elastic Net	0.53	0.69	0.34
K-Nearest Neighbors	0.49	0.73	0.30
Decision Trees	0.64	0.54	0.38
Random Forest	0.45	0.77	0.26
Neural Network	0.45	0.76	0.29
XGBoost	0.31	0.89	0.19

IV. RESULTS AND DISCUSSION

A. Experimental Setup

1) Evaluation Platform: For our evaluation, we conducted experiments on *two real-world commercial embedded boards:* the NVIDIA Jetson TX2 and NVIDIA Jetson Orin Nano. The Jetson TX2 platform features a heterogeneous two-clustered architecture with a Quad-Core ARM Cortex-A57 and a Dual-Core NVIDIA Denver 2 64-bit CPU. The Jetson Orin Nano also has a two-clustered architecture, with one cluster consisting of a Quad-Core ARM Cortex-A74 and the other cluster having a Dual-Core ARM Cortex-A74. Besides the difference in the CPUs, the Jetson Orin boards features an extra cache level, i.e., a 4MB L3 shared cache for both clusters. These boards present a heterogeneous computing scenario, comprised of clusters and cores with different capabilities that follow the trend of modern high-end embedded devices, such as those in the automotive or mobile industry.

Both platforms run Ubuntu as the operating system (18.04.6 LTS on the Jetson TX2 and 20.04.6 LTS on Jetson Orin). Notably, while the different experiments are undergoing no other application is executing besides normal OS operation. Furthermore, we set the power management governor of the boards to "userspace," which avoids system-controlled changes in the CPUs' frequencies. Additionally, we restore the frequency level of the cores to the maximum value before executing each workload.

2) Benchmark Application Set: As the application set for our experiments, we use two benchmark suites. First, for training the ML-based models and general evaluation purposes, we employed 18 applications from the SPEC2006 benchmark suite, all using the intermediate (i.e., the so-called *"train"*) input size from the suite. The set includes applications both from the integer and floating point benchmarks. The full list is the following: *gcc, milc, bzip2, sphinx3, astar, lbm, bwaves, mcf, zeusmp, namd, h264ref, gobmk, povray, gromacs, cactusADM, omnetpp, hmmer, and leslie3d*. The remaining applications from the SPEC2006 suite were not used due to compilation or execution errors on the board. As a second application suite, we employ the full set (i.e., apps and kernels) from the PARSEC 2 benchmark [28], using the *simlarge* input size. These applications are exclusively used for evaluation purposes i,e., they not used for any training and hence are unseen to the techniques. In Section IV-E we employ these applications to show how our proposed system-informed techniques can perform well independently of the application characteristics with which where they were trained.

3) Malicious Application: The overview for both the malicious transmitter and receiver applications is shown in Fig. 4. The malicious transmitter is a C++ functional covert channel application that modulates the power of the system to transmit information. Similar to other power-based covert channels [5], [15], [16], we employ encoding and modulation mechanisms, such as return-to-zero and on-off-keying on the transmission. When encoding a bit of 1, the application continuously performs a compute-intensive kernel that increases the

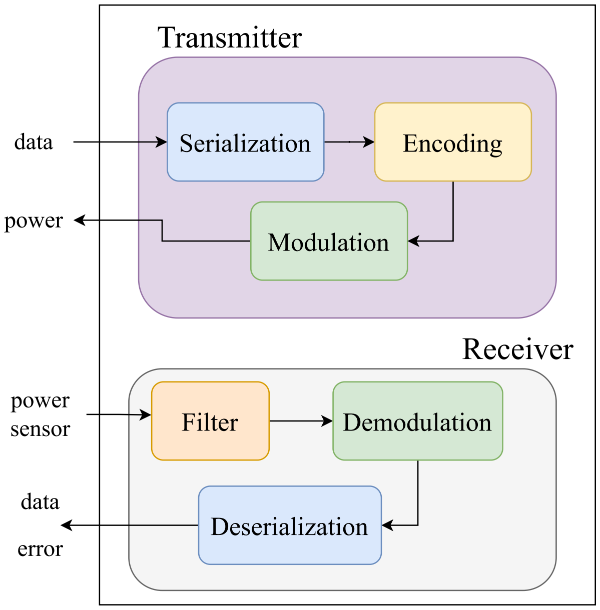

Fig. 4. Overview of the transmitter and receiver malicious applications.

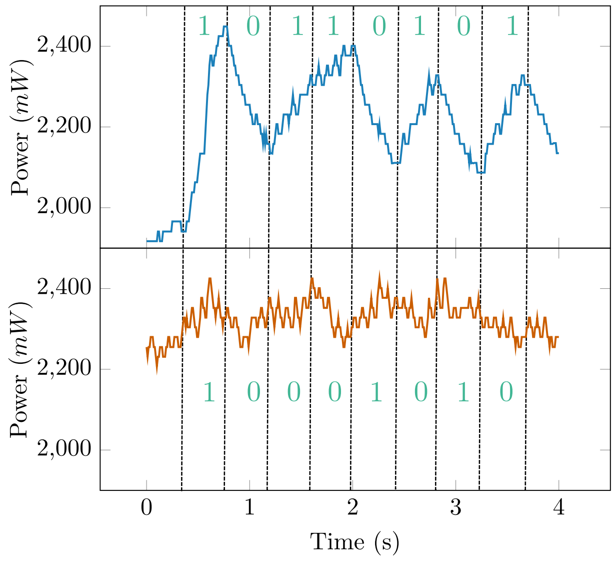

Fig. 5. Power signal of the Jetson TX2 platform during the transmission of a packet of 0xb5 at a maximum core frequency (up) and while applying DVFS to the attacking core (down). The annotated bits correspond to the decoded packet by the receiver module.

power consumption of the system. It consists of floating-point operations (i.e., square-root) combined with a busy-waiting loop. For a bit of 0, the malicious transmitter sleeps to reduce the power consumption.

To evaluate the communication, we implement a simple off-line receiver which upon saving the power measurements from the sensors, filters them and then decodes and de-serializes the bits employing a threshold-based approach as done on other approaches (e.g., [5]). For the purposes of the evaluation, the channel frequency is set around 15 Hz. Due to modulation, the transmission speed of the channel is approximately 2.67 bits per second, which is in the normal range for power-based covert channels (e.g., TCCs [2], [3]). As we show further in Section IV-C when no countermeasure is present in the system, the channel can communicate information reliably with low-error rates (i.e., less than 5%).

B. Baseline and Naive Policies

As a baseline for comparison with our proposed techniques, we implement *the state-of-the-art DVFS technique* from [6] (called simply "*DVFS*" in our experiments). This technique periodically toggles the frequency level of the CPUs from the highest value to a random low value, and vice-versa, to manipulate the power of the system and interfere with the attack. In our experiments, the high-value frequency is the maximum allowed frequency for the boards (i.e., 2000 MHz for the TX2 and 1500 MHz for the Orin). As low frequencies, we employ the four lowest levels available in the boards (345, 500, 625, and 806 MHz for the TX2 and 115, 192, 268, and 345 MHz for the Orin). We employ a β value of 9, as used in [6]. In our setup, this means that while DVFS is applied, the affected cores execute at the high frequency for 0.25 ms and then at the lower frequency for 2.25 ms. To further visualize the effect of the DVFS on the malicious transmitter application, in Fig. 5 we show the power signal from the Jetson TX2 platform for the transmission of a packet of $0xb5$ without the countermeasure (top) and while the DVFS countermeasure is active (bottom). As can be seen from the figure, the final decoded message is significantly affected by the changes in the power. We properly evaluate the transmission error rates produced by the different countermeasures techniques further in Section IV-C.

Furthermore, besides evaluating our system-information countermeasures, we develop two extra naive approaches and one semi-informed technique for comparison purposes. These approaches do not consider the system information explicitly but rather apply a fixed action.

The two naive techniques are fixed core on cluster 0 (FC0) and fixed core on cluster 1 (FC1). In the FC0 approach, we always migrate the malicious application to the first core within the 4-processor cluster. In the FC1 technique, we move it to the first core within the 2-processor cluster. Additionally, in our evaluation we include an extra heuristic. This straightforward semi-informed heuristc, which we name worst-performing core (WPC), finds the core with the lowest-IPS value and assigns the attacking application to that core regardless of the cluster organization. For these three additional policies, when other applications are executing in the newly selected core for the malicious application, we exchange the applications' cores so that the policy is always followed in the same manner as our WPCBPC heuristic. After the migration happens, we apply DVFS to the affected cluster to mitigate the attack.

All the experiments that follow include the state-of-the-art DVFS approach [6], the naive techniques, and our system-informed approaches for comparison purposes.

C. Covert Channel Mitigation

In this first experiment, we evaluate the effectiveness of the different countermeasure techniques to mitigate the attack by affecting the transmission. To do so, we sent a total of

TABLE II
AVERAGE RESULTS FOR THE BASELINE AND THE DIFFERENT COUNTERMEASURE APPROACHES UNDER 50 DIFFERENT WORKLOADS ON THE JETSON TX2 PLATFORM

Metric	Approach								
	Baseline	DVFS [6]	FC0	FC1	NN	RF	WPC	WPCBPC	XGB
Makespan (s)	210.12	358.63	407.7	284.81	280.47	301.26	282.51	269.38	268.03
Power (mW)	4999	4098	4009	4564	4609	4631	4489	4705	4759
Energy (J)	1050.48	1469.8	1634.64	1300.03	1292.63	1395.23	1268.32	1267.36	1275.63
EDP (Js)	220,727.79	527,116.38	666,442.94	370,260.59	362,544.76	420,326.52	358,313.13	341,403.66	341,909.21

TABLE III
AVERAGE RESULTS FOR THE BASELINE AND THE DIFFERENT COUNTERMEASURE APPROACHES UNDER 50 DIFFERENT WORKLOADS ON THE JETSON ORIN PLATFORM

Metric	Approach								
	Baseline	DVFS [6]	FC0	FC1	NN	RF	WPC	WPCBPC	XGB
Makespan (s)	183.36	296.83	418.89	234.44	239.8	191.26	291.76	184.92	213.22
Power (mW)	4960	4640	4368	4816	4918	5067	4723	5030	5080
Energy (J)	909.48	1377.2	1829.84	1129.08	1179.23	969.17	1377.87	930.22	1083.14
EDP (Js)	166,761.97	408,794.91	766,500.5	264,702.49	282,778.99	185,364.16	402,006.12	172,015.89	230,947.23

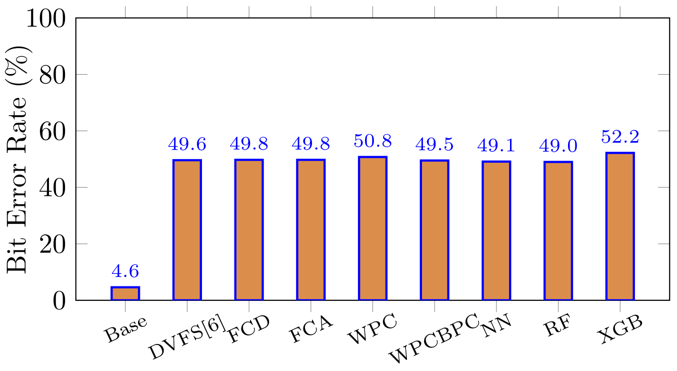

Fig. 6. BER for the transmitter attack with no countermeasure applied (base) and with each one of the evaluated countermeasure techniques on the Jetson TX2 platform.

800 bits as 8-bit packets using the malicious transmitter. We implement a simple receiver that reads the power of the system and decodes the information being transmitted.

Fig. 6 shows the results from this experiment. When no countermeasure is active (Base), the bit error rate (BER) from the transmission is very low (e.g., less than 5%). However, once the countermeasures are active, the BER increases drastically. Because of the transition to low frequencies in the DVFS, the power of the system tends to decrease, as seen in Fig. 5. This means that most of the bits of 1 would be interpreted as 0 while the bits of 0 are likely interpreted correctly. For a transmission with a balanced quantity of 1's and 0's, the expected error rate due to the countermeasure is then 50%. As seen in the figure, this is exactly the case for all of the techniques. Ultimately, this experiment shows that all the proposed countermeasures are effective for mitigating power-based covert channels.

D. Energy and Performance Penalty

In order to evaluate the energy and performance penalty of the different countermeasure techniques we devised an experiment where we generated 50 random workloads from the application set. After the workload generation, we simultaneously run the applications alongside the malicious transmitter application. The transmitter application executes at all times until the workload finishes. To replicate the behavior of a covert channel detector, we wait for a period of 1 s after the workload is launched, before triggering the countermeasure technique. Then, we continue to apply the countermeasure until the full workload has finished the execution. This process is repeated for all the countermeasure techniques: the state-of-the-art DVFS approach (DVFS), FC0, FC1, WPC, WPCBPC, NN, RF, and XGBoost (XGB). Notably, to keep fairness, all techniques are evaluated with the same workload set. The orchestration of the experiment and corresponding monitoring is done by the resource management application, as described in Section III-A. To reduce the effects of cached data in the experiments, we run all the workloads with one technique before moving to the next technique. The workloads are executed in the same order for all techniques. Additionally, we add delay of about 5 s between each workload to let the system return to a semi-idle state before a new execution.

Tables II and III show the averaged metrics obtained from the experiment for the baseline (no countermeasure applied) and all the different techniques on the two evaluation platforms. As a metric for performance, we report the average execution time of the whole workload from the moment we launch all the applications (done simultaneously) until the last application finishes its execution (i.e., makespan).

We measure the average power consumption of each run. Then, we compute the energy and energy-delay Product (EDP), as a measurement of the efficiency of the system. Notably, since the resource management orchestration application executes in the system concurrently with the workload the overhead in the system due to techniques is already included as part of the obtained metrics.

From the tables, it is clear that all the countermeasures affect negatively the energy and performance of the system. It is important to notice that this effect is expected as it is the cost of mitigating the attack. Notably, the power in the system due to countermeasures is overall reduced, as it can also be seen in Fig. 7. From the normalized power, it would seem as if the state-of-the-art DVFS and FC0 are the best approaches. This again is a product of the frequency reduction

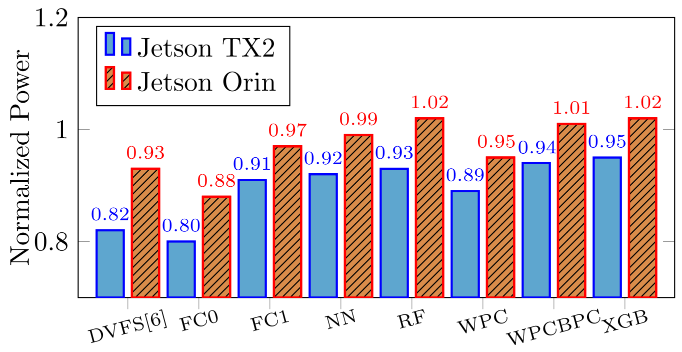

Fig. 7. Normalized power in the system due to the different countermeasures on both evaluation platforms.

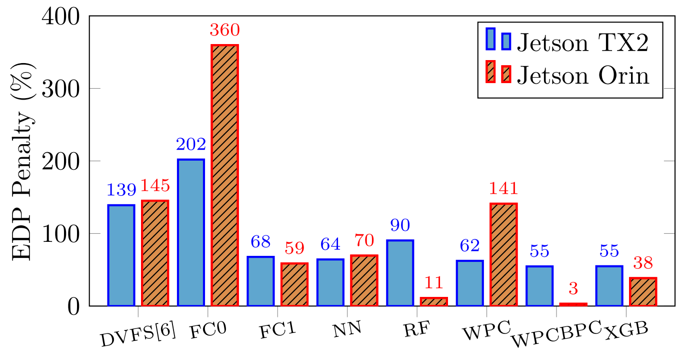

Fig. 10. EDP penalty in the system due to the different countermeasures on both evaluation platforms.

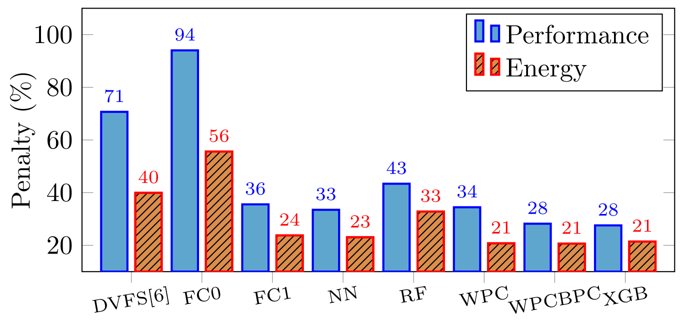

Fig. 8. Performance and energy penalty over the baseline implementation in the system due to the different countermeasure techniques on the Jetson TX2 platform.

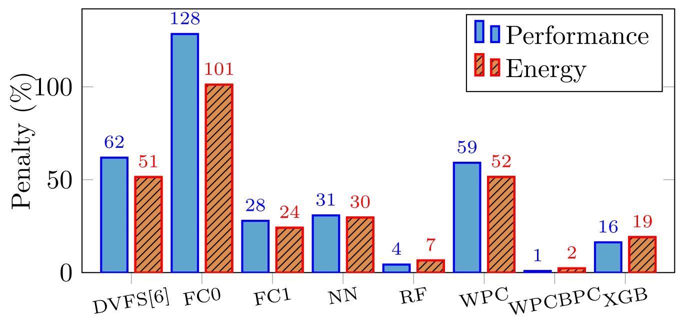

Fig. 9. Performance and energy penalty over the baseline implementation in the system due to the different countermeasure techniques on the Jetson Orin platform.

due to the DVFS mechanism. However, as our further results show, from an energy and performance point of view the case is exactly the opposite. As our following results indicate, the power consumption of the system while the countermeasure is active is not an indication of the efficiency of the system, especially considering the performance penalty.

To better dissect and analyze the impact on the system's efficiency due to the countermeasures, we plot the performance and energy penalty for both platforms in Figs 8 and 9. As can be seen, the state-of-the-art DVFS countermeasure has a high overhead of about 70% for the Jetson TX2 and about 62% for the Jetson Orin. This is a significant difference over the reported 25% for general purpose multicore system [6]. This means that the performance penalty due to DVFS countermeasure is significantly higher on an embedded system. This is an interesting effect that has not been reported before this work.

Moreover, the FC0 technique has the worst performance and energy penalty in both platforms. This outcome can be expected since this approach forces the DVFS to be applied to the bigger cluster, which consistently affects more cores (and applications) at all times when applying DVFS. On the other hand, the naive FC1 technique effectively reduces the performance and energy penalties when compared to the simple DVFS approach by affecting fewer cores.

More importantly, our system-informed approaches reduce the performance penalty by up to 40% and up to 60% for the Jetson TX2 and Orin, respectively, when compared to the state-of-the-art technique. From an energy perspective, our system-informed techniques reduce the penalty due to the DVFS state-of-the-art countermeasure by about 20% in the Jetson TX2 and up to 50% in the Jetson Orin. Moreover, when combining the effect of both energy and performance in EDP form, as can be seen in Fig. 10, it is clear that system-informed approaches are generally more energy-efficient than the reference and their naive counterparts. At their best, these techniques managed to reduce the EDP penalty by up to 84% and 142% for the TX2 and Orin boards, respectively, when compared to the simple state-of-the-art DVFS technique.

Our two other ML-based techniques resulted in slightly less EDP reduction compared to our XGB-based technique in the TX2 platform, while RF outperformed XGB on the Orin board. On the Jetson TX2 board, our NN reduced the penalty by about 70% while our RF reduced it by 48% compared to the simple state-of-the-art DVFS technique. Given the observed performance of the two models at design time (Table I), this result was not necessarily expected, as RF outperformed NN in terms of prediction accuracy. One possible explanation for this might be that the RF model experiences inefficient memory access due to the unpredictable traversal across its numerous decision trees, leading to irregular and intensive memory usage, thereby increasing the processor's energy demand for memory accesses. On the other hand, our NN benefits from more structured and regular memory access patterns, reducing memory bandwidth requirements, and minimizing data movement across the processor cores, thereby further conserving energy. This represents an example where the execution of the policy itself as part of the system changes the expected behavior. As a result, the accuracy superiority of RF was suppressed on this board, eventually leading to a longer

TABLE IV
AVERAGE RESULTS FOR THE BASELINE, STATE-OF-THE-ART, AND SYSTEM-INFORMED COUNTERMEASURES UNDER 25 UNSEEN WORKLOADS ON THE JETSON TX2 PLATFORM

Metric	Approach						
	Baseline	**DVFS [6]**	**NN**	**RF**	**WPC**	**WPCBPC**	**XGB**
Makespan (s)	60.34	113.87	86.98	99.66	86.22	79.06	79.06
Power (mW)	4820	3707	4186	4091	4212	4250	4409
Energy (J)	290.83	422.08	364.09	407.75	363.13	336.02	348.61
EDP (Js)	17,548.89	48,062.73	31,668.36	40,636.06	31,309.16	26,566.06	27,560.82

TABLE V
AVERAGE RESULTS FOR THE BASELINE, STATE-OF-THE-ART, AND SYSTEM-INFORMED COUNTERMEASURES UNDER 25 UNSEEN WORKLOADS ON THE JETSON ORIN PLATFORM

Metric	Approach						
	Baseline	**DVFS [6]**	**NN**	**RF**	**WPC**	**WPCBPC**	**XGB**
Makespan (s)	69.64	125.48	119.2	108.98	117.55	99.7	97.15
Power (mW)	4550	4263	4323	4396	4354	4381	4447
Energy (J)	316.87	534.95	515.26	479.06	511.8	436.78	431.99
EDP (Js)	22,066.8	67,125.85	61,419.26	52,207.57	60,162.34	43,546.54	41,967.37

makespan and higher-energy consumption for the application workload compared to the NN. Even when their execution affects the performance and energy consumption of the system, our system-informed approach still significantly outperform the blind state-of-the-art approach.

Interestingly, on the Jetson Orin board, RF outperforms NN as expected by the training. In fact, as Fig. 9 shows, RF and WPCBPC manage to produce at most 4% performance and 7% energy penalties in the system, which is a major improvement when compared to the state-of-the-art technique. In fact, this penalty is close to insignificant on this board, when compared to the case when no countermeasure is applied. We believe several factors contribute to this outcome. First, the Jetson Orin board features a unified 4MB L3 cache, which reduces the effect of the intense and irregular memory accesses the RF techniques had on the TX2 platform, which lacks an L3 cache. Moreover, the Orin board's homogeneous, modern, and more powerful CPUs further enhance performance, helping to achieve the expected results.

E. Generalization to Unseen Workloads

While our proposed system-informed heuristics are inherently application-agnostic (i.e., no application feature is considered in the migration logic), that might not necessarily be the case for the ML-based approaches. While we do not use features from the applications themselves as input to ML-models, since they are trained with execution traces from the SPEC2006 application set, it could be the case that the models are biased toward certain application behavior (e.g., memory or compute intensiveness).

In order to show the generality and the effectiveness of our ML-based techniques under a wider diversity of applications, we devised an experiment where we ran 25 completely new workloads, where each workload is fully comprised of apps never seen during training from the PARSEC 2 benchmark. In each one of these new workloads, all applications are selected randomly from the PARSEC 2 full application list. Additionally, we have ensured that each application from the set appears at least in one workload. The results from this test for both evaluation platforms can be seen in Tables IV and V.

For comparison purposes, we evaluate the baseline, the state-of-the-art approach and the system-informed countermeasure techniques. As shown for both platforms, even though new applications were unseen to the ML models during training, they still achieve a very good performance, which is consistent with our main experiments shown in Tables II and III. Furthermore, as shown in Table V, XGB has delivered the best performance out of the evaluated countermeasure techniques, surpassing even the WPCBPC heuristic in the Jetson Orin platform. This means that the ML-based techniques are not only able to successfully generalize to unseen applications, but can actually leverage the new workload characteristics to overperform the other approaches.

F. Runtime Overhead Analysis

As mentioned in Section IV-D, the overhead that each technique induces in the system from a performance and energy point of view is already included in the final result depicted in Table II, as the resource management orchestration application runs in the system alongside the workload for all the experiments. Moreover, the actual cost of task migration in the applications themselves is also included already in the reported metrics.

Nonetheless, in this section, we provide a more detailed analysis on the part the overhead produced in the system by each of the system-informed techniques. We omit the overhead of the naive approaches (i.e., DVFS, FC0, and FC1) as no processing is needed in the selection of new mapping to be enforced by task migration. Table VI shows the overhead of the system-informed techniques. As can be seen, the overhead due to the heuristics is significantly lower than the ML-based approaches since the computation needed to select the cluster and core needed for the migration is rather simple for both WPC and WPCBPC, and it only needs to be executed once at each acting epoch (of 1 s). The ML-based approaches, on the other hand, are called to predict the efficiency for each possible nonredundant mapping confirmation. For the

TABLE VI
OVERHEAD OF THE DIFFERENT SYSTEM-INFORMED TECHNIQUES ON THE BOTH EVALUATION PLATFORMS

Platform	Overhead (ms)				
	NN	RF	WPC	WPCBPC	XGB
Jetson TX2	128.64	30.37	0.02	0.12	49.09
Jetson Orin	78.57	113.91	0.02	0.02	19.52

configuration of our evaluation platforms, this represents a maximum of 15 nonredundant mapping configurations to be evaluated. The number reported in Table VI is the *accumulated overhead* of the ML-base techniques for all calls. This means that in the worst case, the overhead of the techniques is rather small at about 128 ms. As a final remark, it should be noted that even though the heuristic approaches have much less overhead than the ML techniques, XGB is able to surpass the heuristics in terms of performance for the workloads as seen in Tables II and V. In other words, the overhead difference between both approaches is balanced by the improvement the XGB technique produces in the workload, which is in the end the relevant metric to compare both approaches on this platform.

G. Machine Learning Versus Heuristics

As our experimental evaluation has shown, our system-informed approaches are effective at mitigating the attack while reducing the energy and performance penalty on the system.

While presenting techniques from both ML and heuristics domains, our intention in this article is not to indicate one *best* technique between the different approaches. On the contrary, as our results show, both approaches exhibit quite similar performance (the difference in EDP penalty in our main experiment between WPCBPC and XGB is less than 0.2%). We seek to show how both traditional and ML-based policies can effectively serve the purpose of an efficient countermeasure. Both approaches have advantages and disadvantages when used for this purpose. Both the WPC and our WPCBPC heuristics have low complexity and are very fast, as depicted in Table VI. These heuristics focus on optimizing performance, by reducing the negative effect of the DVFS mechanism. However, by only using IPS this approach does not consider the efficiency of the full system due to the current execution scenario. When dealing with diverse workloads, specially in a potentially more complex system (e.g., many-core), this information might not be sufficient to produce optimal results. The ML-based approaches, on the other hand, have a greater overhead when compared to the heuristics, but as just discussed in Section IV-F they compensate for this overhead by producing efficient execution scenarios. Moreover, the ML-based approach utilizes execution features to learn the behavior of the system, even hidden or nonmeasurable parameters. This means that with enough training, the approaches can be extended and adapted to perform well under diverse execution scenarios. Indeed, as we have demonstrated exactly this in Section IV-E, where the ML techniques were successfully able to generalize correctly to the new application set. Moreover, under this new execution scenario, the XGB model managed to outperform the best heuristic for the Jetson Orin board, showing the potential advantage of the ML approach versus the implemented heuristics.

By providing countermeasures from both heuristics and ML domains we presented two successful avenues to the problem of mitigating power-based covert channels in an efficient manner, Regardless of their domain, our system-informed techniques were able to defeat the state-of-the-art countermeasure, proving to be the better solution to the problem.

V. CONCLUSION

In this article, we have highlighted the performance and energy impact of traditional DVFS-based countermeasures to power-based covert channels on embedded systems. We have shown how the state-of-the-art DVFS method can produce up to 70% performance penalty on an embedded platform when the attack is present at all times, which differs greatly from the reported penalty for general purpose multi-/many-core systems. Moreover, we have proposed different techniques from the heuristic and ML domain that, for the first time, combine dynamic task migration and DVFS to mitigate such attacks in an efficient and system-informed manner, significantly reducing both energy and performance penalties. From our experimentation on the commercial NVIDIA Jetson TX2 and Jetson Orin embedded platforms, we were able to successfully reduce the EDP penalty due to the state-of-the-art DVFS-only countermeasure by more than 84% and 142%, respectively, proving that our system-informed techniques are a better approach to power-based covert channel mitigation.

REFERENCES

[1] I. Miketic, K. Dhananjay, and E. Salman, "Covert channel communication as an emerging security threat in 2.5D/3D integrated systems," *Sensors*, vol. 23, no. 4, p. 2081, 2023.
[2] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms," in *Proc. USENIX Conf. Security Symp. (SEC)*, 2015, pp. 865–880.
[3] J. González-Gómez, K. Cordero-Zuñiga, L. Bauer, and J. Henkel, "The first concept and real-world deployment of a GPU-based thermal covert channel: Attack and countermeasures," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2023, pp. 1–6.
[4] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "C^3APSULe: Cross-FPGA covert-channel attacks through power supply unit leakage," in *Proc. IEEE Symp. Security Privacy (SP)*, 2020, pp. 1728–1741.
[5] H. Huang, X. Wang, Y. Jiang, A. K. Singh, M. Yang, and L. Huang, "On countermeasures against the thermal covert channel attacks targeting many-core systems," in *Proc. Design Autom. Conf. (DAC)*, 2020, pp. 1–6.
[6] H. Huang, X. Wang, Y. Jiang, A. K. Singh, M. Yang, and L. Huang, "Detection of and countermeasure against thermal covert channel in many-core systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 2, pp. 252–265, Feb. 2022.
[7] J. Wang, X. Wang, Y. Jiang, A. K. Singh, L. Huang, and M. Yang, "Combating enhanced thermal covert channel in multi-/many-core systems with channel-aware jamming," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3276–3287, Nov. 2020.
[8] X. Wang et al., "Detection of thermal covert channel attacks based on classification of components of the thermal signal features," *IEEE Trans. Comput.*, vol. 72, no. 4, pp. 971–983, Apr. 2023.
[9] J. González-Gómez, M. B. Sikal, H. Khdr, L. Bauer, and J. Henkel, "Smart detection of obfuscated thermal covert channel attacks in many-core processors," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.
[10] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, 2006.

[11] T. B. Paiva, J. Navaridas, and R. Terada, “Robust covert channels based on DRAM power consumption,” in *Proc. 22nd Int. Conf. Inf. Security*, 2019, pp. 319–338.

[12] J. Haj-Yahya et al., “IChannels: Exploiting current management mechanisms to create covert channels in modern processors,” in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2021, pp. 985–998.

[13] M. Gross, R. Kunzelmann, and G. Sigl, “CPU to FPGA power covert channel in FPGA-SoCs,” Cryptology ePrint Archive, IACR, Bellevue, WA, USA, Rep. 2023/429, 2023. [Online]. Available: https://eprint.iacr.org/2023/429

[14] D. B. Bartolini, P. Miedl, and L. Thiele, “On the capacity of thermal covert channels in multicores,” in *Proc. Eur. Conf. Comput. Syst. (EuroSys)*, 2016, pp. 1–16.

[15] Z. Long, X. Wang, Y. Jiang, G. Cui, L. Zhang, and T. Mak, “Improving the efficiency of thermal covert channels in multi-/many-core systems,” in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, Apr. 2018, pp. 1459–1464.

[16] K. Dhananjay, V. F. Pavlidis, A. K. Coskun, and E. Salman, “High bandwidth thermal covert channel in 3-D-integrated multicore processors,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 11, pp. 1654–1667, Nov. 2022.

[17] T. Trochatos, A. Etim, and J. Szefer, “Security evaluation of thermal covert-channels on SmartSSDs,” 2023, *arXiv:2305.09115*.

[18] D. Ma and R. Bondade, “Enabling power-efficient DVFS operations on silicon,” *IEEE Circuits Syst. Mag.*, vol. 10, no. 1, pp. 14–30, 1st Quart., 2010.

[19] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, “Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs,” in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2014, pp. 1–6.

[20] P. Rahimi, A. K. Singh, and X. Wang, “Selective noise based power-efficient and effective countermeasure against thermal covert channel attacks in multi-core systems,” *J. Low Power Electron. Appl.*, vol. 12, no. 2, p. 25, 2022. [Online]. Available: https://www.mdpi.com/2079-9268/12/2/25

[21] J. Gonzalez-Gomez, L. Bauer, and J. Henkel, “Cache-based side-channel attack mitigation for many-core distributed systems via dynamic task migration,” *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 2440–2450, 2023.

[22] Q. Wu, X. Wang, and J. Chen, “Defending against thermal covert channel attacks by task migration in many-core system,” in *Proc. IEEE 3rd Int. Conf. Circuits Syst. (ICCS)*, 2021, pp. 111–120.

[23] B. Pourmohseni, S. Wildermann, F. Smirnov, P. E. Meyer, and J. Teich, “Task migration policy for thermal-aware dynamic performance optimization in many-core systems,” *IEEE Access*, vol. 10, pp. 33787–33802, 2022.

[24] M. B. Sikal, H. Khdr, M. Rapp, and J. Henkel, “Machine learning-based thermally-safe cache contention mitigation in clustered manycores,” in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.

[25] T. Marinakis, S. Kundan, and I. Anagnostopoulos, “Meeting power constraints while mitigating contention on clustered multiprocessor system,” *IEEE Embedded Syst. Lett.*, vol. 12, no. 3, pp. 99–102, Sep. 2020.

[26] “PERF: Linux profiling with performance counters,” Jun. 2009. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

[27] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.

[28] C. Bienia and K. Li, “Parsec 2.0: A new benchmark suite for chip-multiprocessors,” in *Proc. 5th Annu. Workshop Model., Benchmarking Simulat.*, 2009, p. 37.