Architectural Whispers: Unveiling Machine Learning Models with Frequency Throttling Side-Channel Fingerprinting

Najmeh Nazari, Chongzhou Fang, Hosein Mohammadi Makrani, Behnam Omidi[2], Mahdi Eslamimehr[3], Setareh Rafatirad, Avesta Sasan, Hossein Sayadi[4], Khaled N. Khasawneh[2], and Houman Homayoun
University of California Davis, [2]George Mason University, [3]UCLA, [4]CSULB
Corresponding author: nnazari@ucdavis.edu

ABSTRACT

Machine Learning (ML) security practices include hiding ML model architectures to protect intellectual property and prevent attacks. We introduce a novel fingerprinting attack using frequency throttling-based Side-Channel Attack (SCA) to detect an ML model's architecture family by converting power side-channel data into timing variations. This method involves using adversary kernels and a time series ML classifier to discern the architecture from execution time patterns during model operation. We achieved up to 96% accuracy in identifying known ML models' architecture families under Ring 0 privileges and we demonstrated its effectiveness across different platforms. Moreover, our code is publicly available [1].

ACM Reference Format:
Najmeh Nazari, Chongzhou Fang, Hosein Mohammadi Makrani, Behnam Omidi[2], Mahdi Eslamimehr[3], Setareh Rafatirad, Avesta Sasan, Hossein Sayadi[4], Khaled N. Khasawneh[2], and Houman Homayoun . 2024. Architectural Whispers: Unveiling Machine Learning Models with Frequency Throttling Side-Channel Fingerprinting. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3657388

1 INTRODUCTION

The significance of ML model architecture knowledge cannot be understated, as it serves as a gateway to numerous security threats, including model inversion attacks and adversarial attacks [17]. In response, the academic community is engaged in the exploration of various channels that could potentially leak information [9, 16].

In this study, we present a breakthrough exploration of ML model architecture fingerprinting attacks through the analysis of power side-channel leakage. Specifically, our approach introduces a novel form of ML model fingerprinting attack that harnesses the passive examination of frequency-throttling side-channel information. This information, recently discovered in [12], is accessible even at the user-space level. We explore the potential of leveraging the frequency throttling SCA and converting the power side channel to timing information to detect the architecture of an ML model performing inference tasks. Our proposed approach overcomes the limitations of telemetry data by employing a black-box attack strategy. This involves the development of a robust supervised classifier using timing information obtained from adversary kernels running concurrently with popular ML models.

Unlike telemetry data, which can potentially be restricted or filtered through software controls, frequency-throttling side-channel leakage cannot be neutralized by imposing access limitations or applying filtering-based mitigation patches. This is due to the fact that any malicious software can monitor its execution time by reading system timers. It is important to note that even low-precision timing information (in order of 100 microseconds) is enough to have high-accuracy fingerprinting results. Furthermore, unlike preceding work [16], our proposed fingerprinting attack does not require access to the victim's system-level trace information such as CPU, GPU, and memory utilization, marking a distinctive contribution in our approach. An additional advantage of our proposed approach, when compared to contention-based timing side-channel (such as execution port) techniques [7], lies in the use of frequency-throttling side-channel leakage which effectively operates across different CPU cores. Moreover, it does not depend on the Simultaneous Multi-Threading (SMT) feature of CPUs to achieve high-accuracy results [18]. This aspect is crucial in cloud environments where SMT features could be disabled on the shared instances [1].

Our proposed ML fingerprinting attack process consists of two key steps: *1) Gathering Timing Information:* We collect the execution time of an adversary kernel while a given ML model is actively running on the same processor, performing inference on a batch of inputs. This allows us to observe the unique timing patterns associated with each ML model architecture. Furthermore, this step involves initiating the frequency throttling side-channel effect and converting it into timing information. *2) Machine Learning Classification:* Following the acquisition of the timing data, we construct an effective supervised classifier customized to discern and accurately identify the architecture of the victim's ML model.

Considering the versatility of ML models, we investigate the transferability of our proposed attack, particularly with unseen model variants—those belonging to the model's family that were not present in our classifier's training set. Our findings showcase the effectiveness of our approach in accurately identifying a known ML architecture family with 98% accuracy. Additionally, when applied to architectures that were never encountered during the training process, our attack still achieved a commendable 89% accuracy in identifying the corresponding architectures' family. Furthermore, to demonstrate the platform portability of the proposed ML fingerprinting attack, we targeted three prominent platforms, including two Intel CPUs, and an AMD CPU paired with an Nvidia GPU.

[1]Access at: https://tinyurl.com/dac24-980

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3657388

2 BACKGROUND AND RELATED WORKS

Frequency throttling, a power management technique adopted on Intel and AMD CPUs, is a result of the dynamic adjustment of the CPU frequency when the execution of a workload causes specific electrical or thermal system parameters to surpass predefined limits. In such situations, the power management architecture reactively takes action to throttle the CPU frequency to a lower value, ensuring the system operates within safe and sustainable conditions [11], while Dynamic Voltage and Frequency Scaling (DVFS) proactively adjusts the voltage and frequency of a CPU in response to real-time workload demands. This study specifically concentrates on the reactive limits that trigger the frequency-throttling side channel as a new source of side-channel information.

Previous studies have utilized various side-channel information to target different attack objectives, including model architecture identification [16], model inputs [6], and parameters extraction [10]. The typical sources of side-channel leakage consist of cache [22], memory access [10], electromagnetic (EM) emission and power [21], timing [6], and GPU statistics [19].

Wei et al. [19] developed an attack called Leaky DNN, which exploits the GPU context-switching penalty to extract the victim's model architecture and it requires access to shared GPU profiling information. In [16], the authors proposed employing shared embedded GPU memory traces on edge devices to identify the model architecture. However, their method can be circumvented with a straightforward memory isolation technique.

Hong et al. [8] utilized the Flush+Reload SCA to extract the DNN architecture by observing specific function calls during inference. On the other hand, Torrellas et al. [22] employed the Flush+Reload and Prime+Probe techniques to monitor special functions, enabling them to infer the model architecture. These techniques are prone to cache partitioning. Cache-based fingerprinting attacks require high-precision timers and access to shared cache. Several defenses have been already implemented against these types of attacks [15]. Batina et al. [4] utilized power and electromagnetic (EM) side-channel to reverse engineer models' architecture. However, collecting EM/power traces requires physical access to the target device.

Table 1 shows a comparison of our work to existing works. Patwari et al. [16] predicts the DNN architecture family based on shared memory usage of CPU and GPU, but this method is vulnerable to isolation defenses and cannot differentiate between specific model architectures. In contrast, EZClone [20] identifies model architectures using GPU kernel features via the PyTorch profiler and, therefore is susceptible to access restriction defenses.

Unlike prior research efforts, our innovative fingerprinting technique can passively and remotely gather timing information exclusively about its own kernel execution time, while a black-box victim ML application operates on a separate core. Consequently, it obviates the need for any GPU profiling data or shared resource usage information. Moreover, we are the first work to fingerprint the transformer-based models (BERT, and ViT).

Table 1: Comparison with related works

Methodes	GPU profiling trace	Shared memory info	Isolation resistance	Access restr. resistance	Target of prediction (Accuracy)
[19]	Yes	Yes	×	×	Model (95.2%)
[8]	No	Yes	×	✓	Model (97.4%)
[22]	No	Yes	×	✓	Model (No report)
[16]	No	Yes	×	✓	Family (99%)
[20]	Yes	No	✓	×	Family/Model (100%)
Our work	**No**	**No**	✓	✓	**Family/Model (96%)**

3 THREAT MODEL

Our proposed approach operates under a closed-world scenario; thus, the attacker possesses knowledge of the potential ML model architectures and the processors running them. Leveraging this awareness, an attacker can create a labeled dataset to train its classifier using a supervised learning technique, in an offline manner.

The primary objective of the adversary in our attack is to determine the victim's ML model architecture (fingerprinting), enabling them to launch a more potent downstream attack. For fingerprinting, the attacker adopts a remote and passive approach, solely measuring its carefully crafted adversary kernel's execution time. If frequency throttling is triggered, the timing information reveals the model architecture.

Our attack is based on four fundamental assumptions:

1) The attacker should have the knowledge of victim's platform. The adversary can use a similar processor to run ML models and collect timing information to mimic the victim's CPU behavior.

2) The victim processor runs one ML model at a time and the inference task is performed on a random batch of inputs (1000 in our case). This is to ensure that the collected timing information reflects the targeted model being executed. For each execution, a new batch of inputs will be loaded. Hence, the probability of using the same image multiple times (for both training and testing) is low. In this way, we make sure that fair sets of inputs are fed to models to prevent input bias on models' behavior.

3) The adversary assumption is that the victim ML model is a derivative of the known families. However, the attacker does not require any prior knowledge of hyper-parameters. The models can even be fine-tuned for specific tasks.

4) The x86 architecture uses four privilege levels (Rings 0 to 3) to implement a form of protection called "privilege level separation". The adversary can operate as a user-space attacker with Ring 3 privilege or as an adversary hypervisor/root-user with Ring 0 privilege, while the victim could be an application running on the same processor (they are not required to run on the same core).

Various isolation techniques can be employed on the system under attack. To initiate frequency throttling, a Ring 3 attacker can run a stressor code alongside the victim application, surpassing the system's default power consumption limits. Alternatively, a Ring 0 attacker can modify the reactive limit of the processor package, prompting the system to trigger throttling activity.

4 PROPOSED METHODOLOGY

4.1 Attack Overview

In our research, we utilize timing information to fingerprint widely used ML model architectures deployed from *Hugging Face* [3]. A typical ML model inference pipeline involves the following steps: 1) Loading the ML model hyperparameters, 2) Loading the inputs to apply the ML model, and 3) The inference task, that entails performing a forward pass of the input through the model to obtain predictions. It is essential to highlight that in our fingerprinting attack, we deliberately mimic a real-world scenario by solely relying on runtime information collected during step 3 of an ML application.

This means that in our experiments the ML model is already loaded into the main memory of the processor and prepared for inference. However, previous works, have relaxed this assumption [16].

Our approach comprises two main phases including the Offline phase and the Online phase. 1) Offline Phase: During this phase, the adversary runs the crafted kernels along with the victim ML model, captures timing information, processes them (normalization and smoothing), and creates a labeled dataset. The collected dataset is then used to train a fingerprinting classifier. 2) Online Phase: In the online phase, the actual attack is executed. The adversary employs the pre-trained fingerprinting classifier from the Offline phase for runtime ML architecture fingerprinting.

4.2 Selection of ML Model Architectures

During the offline phase, the attacker is tasked with creating a dataset that associates the necessary timing information with each known model. For this study, we utilize the *Hugging Face* library and its pipelines. Consequently, we select a pool of machine learning models suitable for image and text classification tasks. Our selection is based on the most commonly used families. To develop a realistic inference pipeline (loading data, loading model, and performing inference), we select the batch of inputs from popular datasets to make sure the dataset used for each model is consistent across models performing the same tasks. For image classification and image segmentation, we used ImageNet-1K and COCO validation sets, respectively. For the text classification task, we used the Yelp review dataset. However, the images used for the $vit\text{-}age\text{-}classifier$ are the images with a person from the COCO dataset.

Table 2 shows the models and families chosen for our research. These selected models serve as the foundation for building the dataset, which will be instrumental in training the ML classifier to effectively identify each ML model's architecture during the online phase of our fingerprinting attack. After selecting the models, the next step in building the dataset is to find a set of features for each model that can be used later for architecture fingerprinting.

4.3 Adversary Kernels

Advanced Vector Extensions (AVX) instructions are power-intensive and can significantly raise a processor's power consumption. Additionally, different types of AVX instructions have varying execution times. This motivates us to use a combination of AVX instructions as an adversary kernel alongside the ML application. With this approach, we can consider the execution time of each kernel as a feature. It is essential to highlight that AVX support is not a mandatory prerequisite, and our methodology remains applicable to platforms that do not possess this parallelism feature. In such instances, a combination of alternative instructions, including *RDRAND*, *RDSEED*, *XSAVE*, *XRSTOR*, and AES-NI instructions (*AESENC*, etc.), can be utilized.

The pathway to identifying each ML model architecture involves establishing a correlation between the execution time of adversary kernels and the architecture of the ML models. However, finding this correlation is not straightforward unless frequency throttling is triggered. For this, our approach relies on the influence of power side channel leakage, which directly affects the timing. This method

Table 2: Pool of ML models from Hugging Face

#	Model set 1	Family	Task
1	google/vit-base-patch16-224	VIT	Image Classification
2	facebook/deit-tiny-patch16-224		
3	google/vit-hybrid-base-bit-384		
4	nateraw/vit-age-classifier		
5	facebook/convnext-large-224	convnext	
6	microsoft/resnet-50	ResNet	
7	microsoft/resnet-101		
8	microsoft/resnet-152		
9	google/mobilenet_v2_1.4_224	Mobilenet	
10	google/mobilenet_v2_0.75_160		
11	Matthijs/mobilenet_v1_1.0_224		
12	inception_v3.tf_adv_in1k	Inception	
13	inception_v3.gluon_in1k		
14	vgg11.tv_in1k	VGG	
15	vgg13.tv_in1k		
16	vgg19.tv_in1k		
17	nvidia/mit-b0	MiT	Image Segmentation
18	nvidia/mit-b2		
19	distilbert-base-uncased-finetuned -sst-2-english	BERT	Text classification
20	textattack/bert-base-uncased-QNLI		

Table 3: AVX Instruction sets used

	Instruction 1	Instruction 2
Kernel 1	_mm256_mul_ps	_mm256_mul_ps
Kernel 2	_mm256_div_ps	—
Kernel 3	_mm256_mul_ps	_mm256_div_ps
Kernel 4	_mm256_add_ps	_mm256_sub_ps
Kernel 5	_mm256_fmadd_ps	_mm256_fmadd_ps
Kernel 6	_mm256_permute_ps	_mm256_permute_ps

allows us to ensure that the execution time of the kernel is dependent on the power consumption of other applications running alongside, thereby achieving our objective. This dependency enables us to effectively fingerprint each model based on the unique timing patterns of kernels.

In our approach, we employed a set of six different kernels (implemented in C), each containing a loop with two instructions, as outlined in Table 3. For example, Kernel 4 consists of *add* and *sub* instructions performing the operation on 256-bit vectors containing floats (*ps* stands for packed single-precision). Within each kernel, the loop executes AVX instructions 10,000 times. At the beginning of the loop, the adversary kernel reads the system time, and upon completion of the loop, it calculates the time taken to finish the iteration. This time measurement represents one point of timing data. The kernel repeats this process 20,000 times, resulting in a time series of data consisting of 20,000 data points, which is then saved to a file. Consequently, for each ML model, we possess six traces of timing information obtained from our adversary kernels. These six time-series data collectively form a training sample point for each ML architecture. The collection of these samples creates our dataset. To generate the train set and test set, we use an 80:20 train-test split ratio. We divided our dataset to include 160 samples per model for training and 40 samples per model for the test time.

4.4 Converting Power Side-Channel to Timing information

To convert the power side channel to the timing information, we propose a method that utilizes reactive limit-induced throttling to introduce variations in the execution time of the program. For

our specific attack, we induce variations in the execution time of our adversary kernels based on the ML model architecture. In this way, we can effectively convert the power side channel into timing information. This adaptation is crucial to the success of our fingerprinting attack, as it allows us to uniquely identify each ML model architecture based on its power consumption behavior and corresponding timing signatures.

Figures 1 illustrate our proposed approach to convert the frequency throttling side effect into model-dependent runtime information. Let's consider two models, M1 and M2. The model can run on a core separate from our adversary program. Although the adversary cannot directly monitor the execution time of the ML models, it can measure the execution time of its own kernels even with a low-resolution timer.

In the provided examples, the adversary runs three different kernels, each with its unique power profile and execution time. When executing these kernels in parallel with the ML application and if the package power limit is not exceeded, the execution time of each kernel remains the same, regardless of the running model.

However, when the adversary reduces the power limit (Ring 0 privilege) or introduces a power stressor (Ring 3 privilege) to trigger frequency throttling, the execution time of each kernel becomes dependent on the running ML application. This is due to the varied behavior exhibited by each ML model's architecture, which influences their power profiling. Consequently, this leads to time variations in the execution time of the kernels. By observing these time variations in the kernel's execution time, the adversary can determine the specific ML model architecture being executed.

Figure 2 demonstrates how we can exploit the conversion of the power side channel to the timing information for the ML architecture fingerprinting task. To utilize this capability, the adversary kernel must run concurrently with the victim's ML application, and although they do not necessarily have to be on the same core, they should run on the same system. The adversary then triggers frequency throttling by either impacting the reactive limit or running a stress code.

For Ring 0, we followed a specific procedure for each processor. We initially obtained the minimum system power (using the *powerstat* command) when running the given models and kernels. Next, we set the reactive limit to a value lower than the obtained minimum power to ensure that the processor would activate frequency throttling. The new reactive limit is calculated as follows:

$Reactive\_Limit = \alpha \times Min(Sys\_Power(Model_i, Kernel_j))$

where α represents a parameter within the range of (0, 1), which we determined experimentally to achieve the most distinct behavior.

For Ring 3, the reactive limit remains configured at its default value. To induce frequency throttling, we needed to execute the stressor program on a minimum of 2/3 of the processor's idle cores. To mitigate any potential interference caused by the stressor program with the operation of the ML application and our adversary kernel, it was essential to restrict its interactions with I/O and minimize its utilization of the main memory and shared cache. To achieve this goal, we employed a specialized matrix multiplication application that was executed in a continuous loop, performing operations on small arrays. The dimensions of these arrays were carefully chosen so that they could fit within the L1 and L2 cache, thereby reducing the need for accessing shared memory resources.

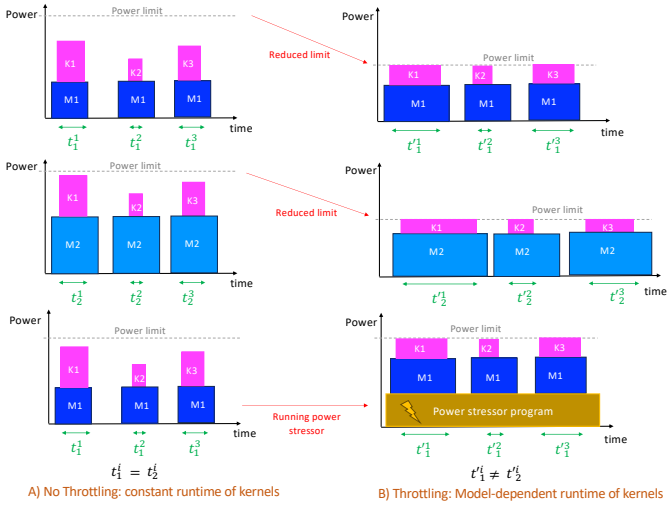

Figure 1: Proposed approach to convert power side-channel to timing information. t_m^k represents the execution time of Kernel k while running alongside Model m.

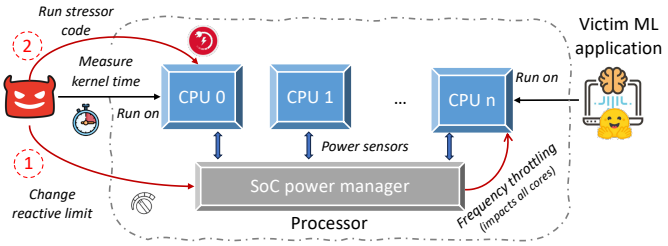

Figure 2: Exploitation of frequency throttling effect. The adversary triggers frequency throttling either by reducing the reactive power limit or running a stressor program.

This approach ensured minimal interference and maximized core utilization, effectively minimizing any disruptions to the primary tasks at hand.

Subsequently, the adversary can run the kernel and measure its execution time. By analyzing these timing patterns obtained during the inference task, we effectively identify distinct timing signatures associated with each model architecture that serves as the basis for identifying the ML models during the online phase.

4.5 Training the Fingerprinting Classifier

The objective of our fingerprinting classifier is to use the collected timing information as input to predict the ML model's architecture and its family when running alongside our adversary kernels. For our classification task, we leverage the *sktime* Python library [2] to construct a machine-learning classifier. This library is a recent addition to the open-source ecosystem, offering compatibility with scikit-learn and providing specialized functionalities for time series tasks. We used the *ROCKET* [5] model, which stands out for its high accuracy and relatively fast performance in handling time series. The dataset for this model must be formatted as follows: 1) Input Samples: The dataset should be in a 3D *numpy* array with

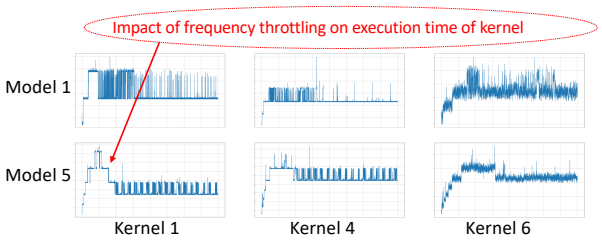

Figure 3: Samples from the dataset for Ring 0 (time series data for 3 kernels running alongside 2 different models).

dimensions (instance, feature, time point). Each instance represents a sample, and each sample contains features observed over different time points. In our study, each kernel's timing trace is considered a feature. Hence we have 6 features over 20,000 time-steps per sample. 2) Labels: The corresponding labels for the input samples should be provided in a one-dimensional *numpy* array. Each element in this array represents the label for the corresponding instance in the input samples. This format ensures that the data is properly structured for input to the *sktime*. The hyperparameter for the *ROCKET* model is the number of random convolutional kernels that we set it to 3,000. Figure 3 illustrates six examples of data, each representing the normalized execution time (Y-axis) of a kernel running concurrently with a specific model for 20,000 consecutive data points (X-axis).

Although the attacker uses the input data on the same device as the victim, it needs to be generalized to handle slight variations. To achieve this, we normalize the timing values between 0 and 1.

5 EXPERIMENTAL RESULTS AND EVALUATION

In our experimental setup, we specifically chose to utilize Intel Xeon CPUs and AMD EPYC processors sourced from x86 machines. To demonstrate the portability of our proposed attack to different platforms, we evaluated the effectiveness of our fingerprinting strategy on two distinct processors with different configurations: 1) Intel Xeon W-2135 CPU, operating at 3.7 GHz frequency, and having a Thermal Design Power (TDP) of 140 Watts. 2) Intel Xeon E5-2650 V2 CPU, running at 2.6 GHz frequency, and having 95 Watts TDP. 3) AMD EPYC 7302 CPU, running at 3.0 GHz with 155W TDP; This platform is equipped with one NVIDIA GeForce RTX 3070 GPU. All systems were equipped with Ubuntu 20.04.6 LTS and Python 3.11. To ensure optimal performance during the experiments, we configured the OS scaling governor to the "performance" mode. In this mode, the clocks are locked at their maximums, unless the power limit is exceeded. This setting is ideal for ML applications on the CPU with low latency.

5.1 Parameter Tuning

To tune the hyperparameter α for Ring 0 attacker, we performed an experiment by sweeping α, and the results are shown in Figure 4.

Based on the optimal accuracy of classifiers, we accordingly set α to 0.8, 0.75, and 0.65 for CPU1, CPU2, and CPU3-GPU, respectively. Once the classification model is trained, the attacker is ready for online deployment. While the ML application is running on the victim's device, the adversary concurrently runs the kernels

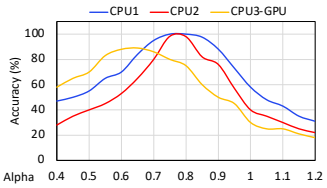

Figure 4: Classifiers' accuracy for various α on Ring 0

Table 4: Pool of ML models as unseen models [3]

#	Model set 2	Family	Task
1	ahishamm/vit-base-isic-patch32	VIT	Image Classification
2	microsoft/resnet-18	ResNet	
3	microsoft/resnet-34		
4	resnet50.tv_in1k		
5	facebook/convnext-large-224-22k-1k	convnext	
6	google/mobilenet_v1_0.75_192	Mobilenet	
7	inception_resnet_v2.tf_ens_adv_in1k	Inception	
8	vgg16.tv_in1k	VGG	
9	nvidia/mit-b1	MiT	Image Segmentation
10	distilbert-base-uncased	BERT	Text classification
11	bert-base-go-emotion	BERT	

and measures their execution time. To ensure that frequency throttling is occurring, the adversary may utilize a power stressor or manipulate the reactive limit to trigger throttling.

By collecting the timing information during the execution of the kernels, the trained classifier can be effectively used to pinpoint the architecture of the ML model or identify its family. The unique timing patterns captured during the online phase act as the fingerprint that the classifier uses to accurately distinguish and classify the ML models based on their underlying architecture.

5.2 Transferability Analysis

In order to assess the transferability of our attack and investigate its ability to classify unseen models (that are not included in the training dataset), we introduced a new set of 11 models from the selected families (referred to as Model Set 2, as presented in Table 4). The performance of our developed classifier on both Model Set 1 and Model Set 2 is detailed in Table 5. We reported the accuracy of both model classification and family classification.

Model classification accuracy is applicable only to Model Set 1, where the models were used during the training. The results indicate that we achieved an accuracy of over 98% in identifying the architecture's family for the known models in Model Set 1. Furthermore, our classifier successfully classified the family of unknown models in Model Set 2 with an average accuracy of 89.5%. These findings demonstrate the effectiveness of our proposed classifier in classifying both known and unknown model families, underscoring the transferability and portability of our attack.

In Figure 5, we present the confusion matrices corresponding to the family classifier from the CPU3-GPU platform on model set 2. Our analysis reveals that the MiT family demonstrates the most robust classification accuracy. Conversely, the ConvNext family exhibits the least accurate performance. We also note that the VGG architecture exhibits exceptional distinguishability, positioning itself as the second-best performer. Furthermore, our analysis highlights a distinct pattern of misclassifications primarily occurring between

Table 5: Classifiers accuracy results-rounded (%)

	Family classification						Model classification					
Platform	CPU1		CPU2		CPU3-GPU		CPU1		CPU2		CPU3-GPU	
Privilege	R0	R3	R0	R3	R0	R3	R0	R3	R0	R3	R0	R3
Set 1	100	93	99	91	90	83	99	90	95	84	88	77
Set 2	92	86	91	84	89	81	—	—	—	—	—	—

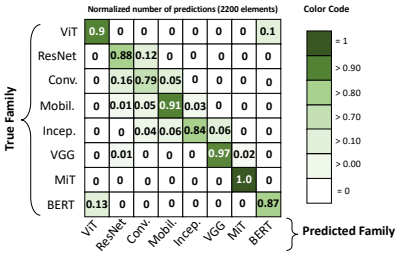

Figure 5: Family classification's confusion matrix: CPU3+GP, model set 2 (200 samples per model), Ring 0

the ViT and BERT models. This pattern is not surprising, given that both ViT and BERT belong to the transformer-based model family, leading to similarities in their classification characteristics.

5.3 Downstream Adversarial Attacks

The objective of adversarial attacks is to deceive ML classifiers by introducing imperceptible perturbations to inputs, undetectable by humans. In this context, we illustrate that our proposed model fingerprinting attack improves the performance of adversarial attacks by transforming the semi-black-box setting into a white-box setting, utilizing knowledge obtained through fingerprinting. In the semi-black-box setting, the attacker constructs a substitute model to generate adversarial examples, with little knowledge about the ML model. In this scenario, the attacker generates adversarial examples using an ensemble of models to enhance their success.

As a case study, we employ pre-trained models from PyTorch and the Adversarial Robustness Toolbox (ART) library [14], for conducting adversarial attacks. Within our experiments, we designate $microsoft/resnet$-101 as the target victim model. Specifically, we utilize the DeepFool attack [13] from the ART library in three distinct scenarios. We use 2,000 images from the ImageNet validation set. In the initial scenario, adversarial examples are generated by using a random ensemble of models. In the second scenario, the attack is orchestrated utilizing models from the same family as the victim. Finally, adversarial examples are crafted specifically for the architecture of the victim model.

Table 6 provides a summary of our findings. The baseline accuracy of the victim model on benign examples is 81.7%. Our results indicate that adversarial examples generated from unrelated model families exhibit lower effectiveness. Leveraging the adversarial samples generated from the ResNet family results in a notable 41% decrease in the accuracy of the victim model. Notably, when considering adversarial examples tailored to ResNet-101, the victim model's accuracy experiences a substantial drop of 68%. Our results show that with the knowledge of the ML architecture, an adversary can generate much more effective adversarial examples.

Table 6: Accuracy drop of models by adversarial examples

Scenario	Models used for adv. examples generation	Accuracy drop (%)
1	VGG19, Inception3, Mobilnet V2	12%
	Convnext_base, VGG13, DenseNet 121	17%
	ResNet 50, VGG16, Convnext_small	23%
2	ResNet 18, ResNet 34, ResNet 152	41%
3	ResNet 101	68%

6 CONCLUSION

This work presents a novel approach to ML models' fingerprinting. By exploiting the frequency throttling side effects, we extract timing patterns through the execution of crafted adversary kernels. Leveraging the time series feature, we develop a classification model to identify the ML model architecture. The experimental results demonstrate the effectiveness of our proposed attack achieving a classification accuracy of 96.3% for known ML model families and 90.6% accuracy for previously unseen models with Ring 0 privilege. By relying on timing information and avoiding direct access to the ML models, we open avenues for stronger remote downstream adversarial attacks on the cloud.

ACKNOWLEDGMENTS

The work in this paper is partially supported by National Science Foundation grants CNS-2155002 and CNS-2155029.

REFERENCES

[1] [n. d.]. *https://docs.aws.amazon.com/whitepapers/latest/security-design-of-aws-nitro-system/the-ec2-approach-to-preventing-side-channels.html.*
[2] [n. d.]. *https://github.com/sktime/sktime.*
[3] [n. d.]. *https://huggingface.com.*
[4] Lejla Batina et al. 2019. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium.*
[5] Angus Dempster et al. 2020. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery* 34, 5 (2020).
[6] Gaofeng Dong et al. 2019. Floating-point multiplication timing attack on deep neural network. In *SmartIoT.* IEEE.
[7] Ben Gras et al. 2020. ABSynthe: Automatic Blackbox Side-channel Synthesis on Commodity Microarchitectures.. In *NDSS.*
[8] Sanghyun Hong et al. 2018. Security analysis of deep neural networks operating in the presence of cache side-channel attacks. *arXiv* (2018).
[9] Xing Hu et al. 2020. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *25th ASPLOS.*
[10] Weizhe Hua et al. 2018. Reverse engineering convolutional neural networks through side-channel information leaks. In *55th DAC.*
[11] 2009 Intel. [n. d.]. Power Management in Intel Architecture Servers.
[12] Chen Liu et al. 2022. Frequency throttling side-channel attack. In *CCCS.*
[13] Seyed Moosavi et al. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE CVPR.*
[14] Maria-Irina Nicolae et al. 2018. Adversarial Robustness Toolbox v1. 0.0. *arXiv preprint arXiv:1807.01069* (2018).
[15] Daniel Page. 2003. Defending against cache-based side-channel attacks. *Information Security Technical Report* 8, 1 (2003), 30–44.
[16] Kartik Patwari et al. 2022. Dnn model architecture fingerprinting attack on cpu-gpu edge devices. In *7th EuroS&P.* IEEE.
[17] Zirui Peng et al. 2022. Fingerprinting deep neural networks globally via universal adversarial perturbations. In *CVPR.*
[18] Daniel Townley and Dmitry Ponomarev. 2019. Smt-cop: Defeating side-channel attacks on execution units in smt processors. In *28th PACT.* IEEE.
[19] Junyi Wei et al. 2020. Leaky dnn: Stealing deep-learning model secret with gpu context-switching side-channel. In *50th DSN.* IEEE.
[20] Jonah Weiss et al. 2023. EZClone: Improving DNN Model Extraction Attack via Shape Distillation from GPU Execution Profiles. *arXiv* (2023).
[21] Yun Xiang et al. 2020. Open dnn box by power side-channel attack. *IEEE Transactions on Circuits and Systems II* 67, 11 (2020).
[22] Mengjia Yan et al. 2020. Cache telepathy: Leveraging shared resource attacks to learn {DNN} architectures. In *29th USENIX Security Symposium.*