AdvProtego: A Unified Framework to SafeGuard Deep Learning Models Against Side Channel Leakage

Xiaobei Yan*, Chip Hong Chang*, Shangwei Guo†, Tianwei Zhang*‡
* Nanyang Technological University, Singapore
† Chongqing University, China
‡ National Integrated Centre for Evaluation (NiCE), Singapore
xiaobei002@e.ntu.edu.sg, swguo@cqu.edu.cn, {ECHChang, tianwei.zhang}@ntu.edu.sg

***Abstract*—The widespread adoption of deep learning models exposes a common security threat: model stealing via side-channel attacks. An attacker can exploit unintended communication channels to recover model parameters or functionalities. Advanced side-channel analysis techniques based on machine learning further exacerbate such risks with higher attack efficacy and automation. Existing defense solutions fail to balance the effectiveness and overhead, and generalize to different platforms.**

We introduce AdvProtego, a novel unified framework to protect deep learning models against various forms of side-channel attacks. Drawing inspiration from adversarial attacks in machine learning, AdvProtego crafts delicate noise to obfuscate side-channel information with minimal performance overhead. However, it is challenging to apply this idea for mitigating model stealing, due to the complexity of defense goal specification and noise control. AdvProtego overcomes these challenges with a modular three-tier design: (1) At the frontend, it repurposes Neural Architecture Search and adapted adversarial attack techniques to identify optimal perturbations that effectively degrade attacker's recovery capability. (2) At the middleware, it translates the perturbations into executable commands, ensuring compatibility with diverse platforms. (3) At the backend, fine-grained noise generation units are implemented to inject crafted noise into side channels precisely. Extensive evaluations across diverse devices and scenarios show the superiority of AdvProtego over existing solutions in effectiveness, robustness, and transferability.

I. Introduction

The widespread adoption of deep learning models in critical applications underscores their transformative potential as well as the growing concerns about security vulnerabilities. Among these threats, model stealing attacks have received significant attention. The adversary's objective is to steal sensitive information (e.g., network architectures, configurations, parameters) about the victim model, thereby jeopardizing the confidentiality and intellectual property of deep learning assets. Prior works have shown that side-channel attacks (SCAs) can be adopted to achieve model stealing. SCAs exploit unintended information leaks from the underlying computing platforms, such as power consumption [44], cache access timing [26], and electromagnetic emanations [20], to extract critical model attributes. Their effectiveness has been further amplified by the integration of machine learning (ML) techniques into side-channel analysis, enabling the processing of noisy signals with high accuracy, robustness, and automation [46], [18], [26], [20], [31]. The rapid evolution of such attacks underscores the urgent need for innovative and effective defenses.

There are two possible directions to mitigate the above threat. First, we could fundamentally eliminate potential side-channel leakage at the source using strategies like constant execution [3], [2], [19], [11], side-channel obfuscation [33], [23], [48], [7], and domain isolation [25], [43], [17], [39], [32], [52]. While effective against conventional SCAs, these approaches face notable limitations in the context of modern SCAs targeting deep learning models. On the one hand, such attacks often exploit higher-order statistical features powered by machine learning algorithms, which can substantially undermine the effectiveness of obfuscation-based mechanisms. On the other hand, constant execution and domain isolation methods incur noticeable performance and resource impacts, which are particularly prohibitive for computation- and resource-intensive deep learning applications. Second, we can redesign the model architecture with obfuscation to prevent model information leakage [51], [24], [27]. However, these defenses typically require extensive model redesign or retraining, making them both time- and cost-inefficient. Moreover, they are impractical in scenarios where the defender lacks the expertise or privileges to modify the deployed models.

To address the above limitations, this paper presents AdvProtego, a novel unified framework to mitigate model stealing attacks via different sorts of side channels. The basic idea is to leverage the *adversarial attack* technique in machine learning to obfuscate side-channel leakage. Adversarial attack is a prominent threat to machine learning models [41], where the attacker injects carefully-crafted imperceptible perturbations into the input to mislead the victim model. In our context, since the attacker exploits machine learning to analyze side-channel signals and steal the model, the defender can also add such perturbations into the runtime execution to deceive the attacker into recovering wrong information.

Some preliminary studies have investigated the feasibility of adopting adversarial attacks for side-channel mitigation. For instance, a poster in CCS'2019 [35] proposed to add adversarial perturbations to power side-channel traces to protect cryptographic applications. Another workshop paper [22] used adversarial techniques to mitigate application fingerprinting through hardware performance counter side channels. However, significant gaps remain when applying this idea to safeguard deep learning models. First, different from cryptography or application fingerprinting, model stealing has totally different attack goals. **How to formulate the corresponding**

defense objectives in adversarial perturbation optimization is overlooked. Second, existing studies only focus on one specific side channel. The work [35] only experimented on the datasets without actual deployment. **How to generate accurate, fine-grained execution noise that corresponds to the desired adversarial perturbation on diverse hardware platforms presents another challenge**.

`AdvProtego` encompasses a set of innovative approaches from different system aspects to address these two challenges. First, we introduce two effective defense objectives for the defender to choose. (1) *Model Similarity Reduction*: this aims to increase the model stealing errors with the perturbed side-channel trace. (2) *Model Utility Reduction*: this aims to mislead the attacker to extract a model with the worst performance. To achieve these objectives, we design adapted adversarial algorithms and Neural Architecture Search (NAS) to identify the corresponding perturbation efficiently. Second, we present a general pipeline for generating and injecting precise and controllable runtime noise in the real-world environment. This pipeline consists of multiple critical steps, including noise quantization, calibration, trace collection, and noise injection, shedding light on the implementation over different computing platforms.

`AdvProtego` adopts a modular design at frontend, middleware and backend levels. It is general to cover different types of side channels and platforms. Without loss of generality, we implement and evaluate `AdvProtego` with three scenarios: FPGA power side channels, CPU cache side channels, and GPU memory side channels. Extensive experiments validate the advantages of `AdvProtego` across several aspects, including mitigation effectiveness, resource and computation overhead, defense transferability and robustness.

II. Preliminary

A. ML-assisted Side-channel Analysis

The advance of machine learning (ML) algorithms has propelled side-channel analysis [40]. Attackers build ML models to automate the extraction of sensitive information from complex side-channel traces, which cannot be achieved with manual efforts. Typically, such attacks unfold in two phases.

1) **Offline Profiling Phase**: The attacker executes the target application over a secret set $\mathbf{s} = \{s_1, ..., s_n\}$ on either the same or similar platform. He collects N side-channel traces $\mathbf{T_{i,n}}$ corresponding to each secret s_i. Using such data, he constructs an ML model $f : \mathbf{T_{i,n}} \mapsto s_i$, establishing a correlation between the side-channel traces and secrets.
2) **Online Exploitation Phase:** With f developed from the previous phase, the attacker can exploit the acquired knowledge at runtime. By utilizing an additional set of q traces $T'_1, ..., T'_q$ captured from the targeted device, the attacker can infer the secret $s_i^{'}$ by evaluating $s_i^{'} = f(T'_i)$.

ML-assisted side-channel attacks offer several advantages. (1) They can retrieve information even when discernible patterns in the side-channel trace are absent. This is particularly relevant in cases where computations of the victim program occur in parallel, making manual pattern analysis infeasible. (2) ML models can seamlessly integrate all the information contained within a single trace, as they excel at handling high-dimensional data. In contrast, traditional methods necessitate the identification of specific points of interest to narrow down the information for the attack [38]. (3) ML models exhibit greater robustness against noise in the side-channel trace compared to conventional statistical techniques. This robustness enhances their efficacy in real-world scenarios.

Due to these attractive features, machine learning has also been adopted to analyze side-channel signals for model stealing. A common practice is to train a sequence-to-sequence (seq2seq) model, which maps the side-channel sequence to the model operation sequence. This has been realized in power [46], cache [26] and memory [20] side-channel attacks, where the attacker leverages a seq2seq model to precisely reconstruct the model architecture based on the sequences of power levels, critical function calls, and memory activities.

B. Review of Existing Defense Solutions

The severity and practicality of side-channel-based model stealing attacks necessitate the development of effective and efficient countermeasures. Prior research has proposed various strategies to achieve such protection, as categorized below.

Mitigating Side Channels. There are several strategies to reduce side-channel information leakage. (1) *Constant execution* [3], [2], [19], [11] aims to disrupt the correlation between sensitive information and side-channel leakage, ensuring uniform execution regardless of the secret. (2) *Side-channel obfuscation* conceals side-channel patterns by injecting noise, making it challenging for the attacker to extract meaningful information. Noise can be applied to the temporal dimension [33], [23], [7] or the amplitude dimension [48], [7]. (3) *Domain Isolation* [25], [43], [17], [39], [32], [52] prevents resource contention and interference by allocating exclusive resources to the victim, thereby preventing side-channel leakage.

However, these defenses have notable limitations for defeating model stealing attacks. First, the obfuscation mechanism is insufficient against ML-assisted side-channel analysis. Particularly, temporal-based obfuscation introduces random delays between instruction executions, which is ineffective in protecting deep neural networks due to their shift-invariance properties [4]. Amplitude-based obfuscation, which injects noise to degrade the signal-to-noise ratio (SNR), could by invalidated by sophisticated signal processing techniques and repeated measurements to filter out noise. Domain isolation and constant execution, while robust in specific scenarios, often demand significant computation and resource investments, limiting their practicality in real-world applications.

Mitigating Model Extraction. We can redesign deep learning models to reduce the attacker's extraction accuracy. Researchers introduce the model architecture *obfuscation* strategy, to prevent the attacker from extracting the correct model information [51], [24], [27]. This strategy has several limitations: (1) For each protected architecture, it needs to identify the corresponding obfuscation target, which is time- and cost-inefficient. (2) The defender is required to redesign or even re-

train the models. This is infeasible in some scenarios, where the model users have no expertise or privileges for model modifications. (3) As these solutions perform obfuscations only at the software level rather than the hardware level, the perturbed side-channel trace is restricted by the model accuracy requirement, which can limit the defense effectiveness. For instance, ObfuNAS [51] achieves only around 1% to 3% accuracy degradation for the attacker's extracted model, but also deteriorates the victim's model accuracy by about 1%; NeurObfuscator [24] even enhances the accuracy of the extracted models by 2.5% in some circumstances.

Injecting Adversarial Noise as a Defense. The high-level idea of leveraging adversarial perturbations as a defensive mechanism has been explored in a few earlier works. Earlier studies such as Picek et al. [35] proposed the use of adversarial perturbations to obscure cryptographic power traces, while Inci et al. [22] applied similar ideas to defend against application fingerprinting via hardware performance counters. Gu et al. [15] employed a differential evolution strategy to generate adversarial AES traces, and more recent works [34], [6] explored GAN-based obfuscation for specific domains like cryptography or audio processing. Yet, all these studies share two fundamental limitations. (1) *Simulation-only evaluation*: they were verified only on synthetic datasets or idealized environments without real hardware deployment, overlooking critical implementation constraints such as quantization, calibration, and injection control. (2) *Distinct domain focus*: they target cryptographic or application-fingerprinting workloads, whose execution flow and attack models are far simpler than deep learning inference pipelines, where multi-layer interactions and ML-based side-channel analysis make both attacks and defenses significantly more complex.

`AdvProtego` is the first framework to operationalize adversarial perturbation–based defense for AI model protection in real-world systems. Our distinct innovations include: (1) *new defense objectives and formulations*: explicitly defining Model Similarity Reduction and Model Utility Reduction for machine learning–assisted model stealing, which differ fundamentally from cryptographic obfuscation goals; (2) *algorithmic realization*: designing and integrating adapted adversarial attack algorithms (masked FGSM/PGD) with reversed NAS search to achieve quantized, hardware-feasible perturbations; (3) *cross-platform implementation*: bridging simulation and real execution via a complete pipeline, validated across CPU, GPU, and FPGA with empirical measurements.

C. Threat Model and Defense Requirements

In this defense-oriented work, we deliberately adopt a conservative threat model by (1) assuming a strong adversary capable of launching both physical and remote side-channel attacks, and (2) constraining the defender's capabilities, with no administrative control over the underlying infrastructure. Such a setting can broaden the applicability of our proposed defense, making it suitable for diverse environments. This threat model has been commonly adopted in existing works [42], [46], [26], [20], which proves to be realistic and practical.

Specifically, an external attacker aims to use certain types of side channels to steal victim's deep learning models. He performs the ML-assisted analysis: gathering side-channel data in the profiling phase to build a model f, and reconstructing victim's model architecture from the real-time side-channel traces T in the exploitation phase: $\mathrm{M} = f(T)$.

The defender is also a non-privileged user, and possibly could be the target model's owner. He deploys the defense solution on the same system to prevent model stealing. Note that the defender is non-privileged in the sense that he does not have administrative control of the host (no kernel/BIOS/vendor firmware modifications). However, he can still create observable execution effects to affect co-located attackers. Examples we target in the paper include: (1) Cloud / multi-tenant FPGA services (e.g., Amazon F1 instances [1]). Users typically upload and run their custom on-chip modules within their allocated region without privileged access. The defender can leverage this legitimate design capability to deploy controlled noise sources, without requiring access to privilege-level resources. This method has been widely used in the side-channel literature [42], [46], [26], [20] and in our FPGA case. (2) User-level processes on CPUs/GPUs. On general-purpose servers and GPUs, a co-located user process can intentionally trigger specific library calls, memory accesses, or CUDA kernels to generate cache and memory activity. These actions are performed entirely at user privilege and are how our CPU and GPU noise injectors operate in the evaluated scenarios.

The defense needs to satisfy the following requirements:

- *Effectiveness*. The defense can effectively disrupt attacker's side-channel observation such that his extracted model is significantly different from what he expects to obtain.
- *Computation efficiency*. The defense solution incurs minimal impact on the computation of the target platform, in terms of resource consumption, model latency and accuracy.
- *Implementation-friendly*. The defender only needs to implement a lightweight defense module at the same privilege level as the victim model. He does not need to modify or customize the target model.
- *Generalization*. The defense is general and adaptable to different types of models, platforms and side channels, ensuring seamless transplantation to various environments.
- *Black-box defense*. The defender does not know attacker's ML model f. Instead, he can adopt a local surrogate model different from f as reference for the defense design.

III. AdvProtego

A. Defense Objectives

The basic idea of `AdvProtego` is to leverage adversarial attack techniques [41] to craft noise δ and inject it into the runtime executions. By carefully adjusting both the scale of the noise and the injection moments, it can disrupt attacker's side-channel observations $T + \delta$ and mislead his extraction model f. Specifically, the defender aims to identify a command sequence $c = [c_1, \ldots, c_n]$ and execute each one c_i at the moment i. This will result in the desired noise δ and disrupt the attacker's analysis.

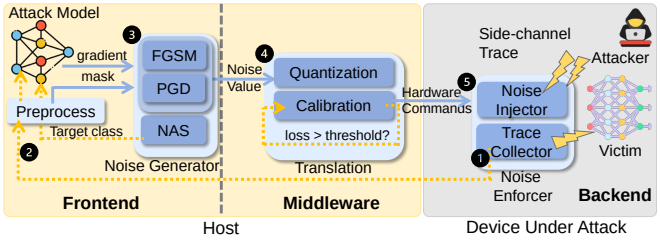


Fig. 1: AdvProtego framework overview and workflow.

Different from the protection of cryptographic applications [35] or mitigating process fingerprinting [22], we need to design the new defense objectives for defending against model stealing attacks, which will guide the noise optimization subsequently. Considering the diverse defense scenarios and implementation feasibility, we propose two objectives:

1) **Model Similarity Reduction**: the defender aims to make the attacker's extracted model as distinct from the correct one as possible. This can be formulated as follows:

$$\arg\max_{\delta} L(f(T+\delta), M) \tag{1}$$

where L represents a distance function that measures the model difference, $f(T+\delta)$ represents the attacker's extracted model from the side-channel trace, and M is the actual victim's deep learning model.

2) **Model Utility Reduction**: the defender aims to make the attacker's extracted model have as low accuracy as possible. This objective can be formulated as follows:

$$\arg\min_{\delta} \mathtt{Acc}(f(T+\delta)) \tag{2}$$

where ACC measures the testing accuracy of a model.

B. System Overview

Figure 1 displays the overview of AdvProtego, consisting of three major stages: frontend, middleware and backend.

The frontend is responsible for calculating the desired noise for different defense objectives. It includes two key modules: Attack Model and Noise Generator. The Attack Model takes preprocessed input traces to provide gradients and parameters that guide the Noise Generator. The Noise Generator employs adapted adversarial attack algorithms with masking mechanism to identify the optimal noise. The mask ensures noise is generated only in permissible locations while adhering to the backend's noise injection capabilities. It is worth noting that at this stage, the generated noise remains in the raw form, unsuitable for direct implementation on the target platforms.

The middleware stage includes a Noise Translator Module, mapping the raw noise values into a corresponding sequence of executable commands on the platforms. The translation is required because the noise calculated in the frontend defines the intended effect but not the hardware-specific behavior.

The backend comprises a Noise Injector and Trace Collector module. Trace Collector captures the side-channel traces in the offline phase and forwards them to the frontend to guide the noise calculation. Noise Injector injects the calculated scale of noise into the execution at the calculated moment.

Workflow. In correspondence with ML-assisted side-channel analysis, AdvProtego also operates in two distinct phases.

In the *offline* phase, the defender begins by gathering side-channel traces through Trace Collector in the backend (❶). Since raw traces are often noisy and voluminous, a preprocessing module is employed to average the data points and analyzes permissible noise injection locations, generating a masking template for the Noise Generator. Using these preprocessed traces and model-specific attributes, the defender trains a surrogate Attack Model (❷). This model could be different from attacker's actual model f; the adversarial noise calculated from it can exhibit strong transferability to counter a variety of potential attacks. The surrogate model is then used to extract gradient information from the victim model's side-channel traces. Together with noise parameters such as the injection locations, such information is fed into Noise Generator, which calculates the desired noise tailored for specific defense goals (❸). The Noise Translator maps the generated noise to executable commands that can drive the underlying hardware components (❹).

In the *online* phase, the defender activates Noise Injector to inject the calculated noise during model inference (❺). The noise can effectively disrupt the attacker's ability to observe and analyze the victim's side-channel traces. By continually injecting carefully-crafted noise, AdvProtego provides robust protection against a wide range of side-channel attacks.

C. Frontend

The frontend of AdvProtego leverages collected side-channel traces to train a surrogate Attack Model, which is subsequently utilized to generate adversarial noise fulfilling different defense objectives. For *Model Similarity Reduction*, the optimization goal in Equation 1 is analogous to untargeted adversarial attacks, and we use an adapted version of FGSM [12] over the surrogate model to generate the adversarial noise. For *Model Utility Reduction*, we apply an adapted Neural Architecture Search (NAS) algorithm to obtain the worst-performing model M^w, and convert the optimization goal in Equation 2 to the following:

$$\arg\min_{\delta} L(f(T+\delta), M^w) \tag{3}$$

This is analogous to targeted adversarial attacks, which drives the attacker's predicted model $f(T+\delta)$ towards the target searched one M^w, by minimizing the above loss function. We use an adapted version of PGD [28] over the surrogate model to optimize the desired perturbation δ.

1) Adapted Adversarial Attacks: Directly applying the standard adversarial attack solutions in conventional ML tasks [12], [28] for side-channel trace analysis could yield suboptimal results, stemming from two key reasons. (1) Side-channel traces inherently contain large amounts of execution noise and uncertainty. Even for the same setting, the collected traces can vary significantly, making perturbations generated from one trace ineffective for others. This variability reduces the transferability and reliability of the generated adversarial noise. (2) The noise injection mechanisms in different platforms impose restrictions on both the type and location of the noise. For

Algorithm 1: Adapted Universal Masked Adversarial Attack Algorithm

Input: Attack Model $\mathcal{A}$, Trace Set $\mathcal{T} = \{T_1, ..., T_i\}$, Mask $\mathcal{M}$, Adversarial Algorithm Adv
Output: Adversarial Noise δ

Initialize δ to all zeros;
foreach *trace* T_i *in* $\mathcal{T}$ **do**
  $\delta \leftarrow \delta \odot \mathcal{M}$;
  $T_i^{'} \leftarrow T_i + \delta$;
  $T_i^{'} \leftarrow Adv(\mathcal{A}, T_i^{'})$;
  $\delta \leftarrow T_i^{'} - T_i$;
  Clamp δ as backend requirements;
return δ

instance, many side-channel traces cannot be injected with negative noise values. In GPU side-channel attacks, injecting noise for memory read/write operations is easy, while it is far more challenging to obfuscate kernel execution latency.

To address these issues, we introduce two key adaptations to existing adversarial attack methods, as detailed in Algorithm 1. First, instead of generating individual perturbations for each trace, we create a universal perturbation by accumulating the ones from all traces, enhancing its transferability. Second, we define a mask $\mathcal{M}$ to indicate the permitted locations of the side-channel trace for noise injection. This mask ensures that perturbations are only applied to feasible regions of the trace. The Hadamard product ($\odot$) is used to apply the mask, ensuring compliance with hardware constraints. We adopt the Connectionist Temporal Classification (CTC) loss to guide the perturbation optimization process. CTC is well-suited for measuring the distance between sequential data like our model architecture encodings. Its actual definition depends on the defense objective. For *Model Similarity Reduction*, the CTC loss is calculated between the attacker's predicted sequence and ground-truth model sequence. For *Model Utility Reduction*, the loss is calculated between the predicted sequence and the sequence of the identified worst-performing model from the NAS phase. In both scenarios, this calculated CTC loss is then backpropagated to the input to compute the precise perturbations that will alter the side-channel signals to achieve the desired defensive effect.

2) Adapted NAS Algorithm: NAS is a mature automated ML technology that searches for the model architecture with the best performance [29], [53], [30]. In our defense objective of Model Utility Reduction, we want to identify the worst architecture as the target to deceive the attacker. To this end, we reverse the metric used in NAS to guide the search process to the opposite direction. This can be formulated as follows:

$$\pi^* = \arg\max_{\pi_i \in \Omega} \mathcal{L}(W_{\pi_i}; X_{val}) \tag{4}$$

where W_{π_i} is the network parameters associated with the model π_i, Ω is the search space, X_{val} is the validation dataset, $\mathcal{L}(\cdot)$ stands for the loss function, and π^* is the target model we expect to find out.

Researchers have proposed different algorithms to achieve efficient and effective NAS. We opt for NAS-RL [53] to implement our defense. It is based on reinforcement learning, and uses the reward (i.e., accuracy on the validation set acc_{val}) to guide the search process. Therefore, we adjust the reward from acc_{val} to $1 - acc_{val}$, to find the worst model.

D. Middleware

In this stage, the Noise Translator converts the generated noise from the frontend into platform-specific commands that could cause the desired defense effect. The translation undergoes two processing steps.

Quantization. The calculated noise using adversarial attack algorithms is continuous, while the noise we can inject into the execution is normally discrete. Therefore, we need to quantize the noise into discrete levels controllable by the Noise Injector in the backend. The number of levels is determined by the specific noise injection mechanisms. A greater number of levels enables more precise trace manipulation, allowing the injected noise to produce effects more closely aligned with expectations. However, this also places higher demands on the backend to control the noise. Formally, this quantization can be modeled as a constrained projection problem. Adversarial noise $\delta = [\delta_1, ..., \delta_n]$ is quantized to fit the range R of the noise injector. Using the formula

$$\text{round}\left(\frac{(\delta_i - \delta_{\min})(\delta_{\max} - \delta_{\min})}{R}\right),$$

we discretize each value, and then use a mapping table M to convert them into hardware commands: $M[\text{round}((\delta_i - \delta_{\min})(\delta_{\max} - \delta_{\min})/R)]$.

Calibration. This step establishes a mapping table from platform control commands to actual noise effects. This can be achieved in an iterative manner. Formally, we first identify a set of possible command patterns that could inject execution noise. For each pattern r_i, we measure the corresponding noise level n_i, and form a pair $R_i = [r_i, n_i]$. We further sort these pairs by n_i to construct the mapping table M, where each pattern r_i maps to a discrete noise level n_i. We keep adjusting the command patterns to minimize the discrepancies between the measured noise level $\hat{n_i}$ and recorded noise level n_i in M. A loss value, defined as $\mathfrak{L} = \sum_{i=1}^{n} |n_i - \hat{n_i}|$, evaluates such inconsistencies at each iteration. Calibration continues until the loss falls below a predefined threshold, ensuring accurate and reliable noise injection.

E. Backend

This stage is responsible for collecting the side-channel traces, and injecting the noise into the side-channel trace at runtime by executing the identified commands. These are achieved by the following two modules, respectively.

Trace Collector. This module gathers the execution traces in the offline phase to build the surrogate model. Depending on the specific platforms and side channels, `AdvProtego` adopts diverse profiling tools to collect such information.

Noise Injector. In the online protection phase, this module executes the identified commands at the identified moments to inject the desired noise. The noise is indistinguishable from the

side-channel trace by the attacker, while remaining transparent to the victim's execution. The implementation of this module is also dependent on the computing platforms and side channels in consideration. It could be integrated into the hardware, or deployed as a separate process co-locating with the victim.

IV. Implementation

For FPGA power side channels, the implementation utilizes a Time-to-Digital Converter (TDC) as the trace collector to capture voltage fluctuations and Ring Oscillators (ROs) as the noise injector. Because RO circuits often introduce unpredictable undershoot and overshoot effects in power readouts, the noise translator relies on an iterative calibration scheme. This process maps the raw adversarial noise into discrete levels by evaluating and sorting actual TDC measurements against specific RO enable patterns, ensuring the continuous adjustment of hardware commands until the noise injection error is minimized to an acceptable threshold.

In CPU cache side channels, the framework monitors specific function invocations (such as `sgemm_incopy` and `sgemm_oncopy` in OpenBLAS APIs) using the Flush-Reload technique. Since cache attacks typically detect the binary presence of a function call rather than its exact execution count, the noise translator clamps the generated adversarial noise to binary values and outputs straightforward injection commands consisting of a time delay and a target function (d_i, c_i). The noise injector then executes these functions at the designated intervals via a dynamically loaded library, effectively producing the desired cache obfuscation without the need for complex hardware calibration.

For GPU memory side channels, trace collection is performed using profiling tools that monitor DRAM read/write volumes alongside kernel execution latency. Because manipulating exact execution timing on a GPU is inherently difficult, the noise translator masks latency perturbations to zero and formats the noise strictly into designated read and write data volumes (W_i, R_i, T_i). To inject this noise, the framework deploys custom dummy CUDA read and write kernels designed with loop dependencies and shared memory reduction operations to prevent compiler optimization. Concurrently, an iterative calibration process accounts for runtime optimizations, guaranteeing that the simulated memory operations precisely match the intended adversarial noise.

Across all platforms, the frontend remains unified and platform-agnostic, while the middleware translates the noise into hardware-specific commands, enabling consistent, low-overhead protection against diverse side-channel modalities.

V. Evaluation

A. Experiment Setup

Testbed. For FPGA power side channels, we use a Xilinx Zynq-7000 SoC ZC706 FPGA board (xc7z045ffg900-2) with Nvidia Deep Learning Accelerator (NVDLA) [5]. The ARM processor on the board runs Ubuntu 16.04 OS, supporting NVDLA alongside the driver for Power Monitor and Noise Injector. Hardware implementation is carried out using Vivado 2019.1. For CPU cache side channels, experiments were conducted on an AMD EPYC 7313P Processor. We utilize PyTorch v2.3.0 and OpenBLAS v0.3.20 as the linear algebra library. For GPU memory side channels, experiments were conducted on an Nvidia GeForce RTX 4080 GPU.

To train the surrogate model and generate adversarial noise for all experiments, we employ PyTorch v1.13 and CUDA v11.6. This training process was conducted on a server equipped with Nvidia GeForce RTX 3090 GPUs.

Datasets and Models. For FPGA power side channels, due to the memory resource constraint of the adopted FPGA board (e.g., 1 GB memory and lack of advanced operators), we mainly consider the image classification tasks over the MNIST and CIFAR-10 datasets. This choice is consistent with prior side-channel works [42], [31], [21]. Since the adversarial attack is a fundamental feature to AI models and tasks regardless of their scales, we believe our solution can be applied to larger models with stronger hardware support, which will be our future work. For each dataset, we run the evaluation on 200 randomly generated models. These models have random numbers (within the range of [2, 16]) of network layers. The composition of each layer is also determined through random selection. The selection space includes 12 convolution layers (with the kernel size of 2, 3, 4, 5 and output size of 10, 20, 30), 4 pooling layers (with the kernel size of 2, 3, 4, 5), 5 fully-connected layers (with the output size of 100, 200, 300, 400, 500), 1 ReLU layer, and 1 Softmax layer. The initial pre-training of these models is conducted using Caffe. Subsequently, calibration is performed utilizing TensorRT, followed by compilation through the NVDLA compiler. These models are generated on the host computer and subsequently executed on the FPGA using NVDLA runtime.

For CPU cache and GPU memory side channels, we use image classification tasks on the ImageNet and Caltech256 [14] datasets. The evaluation encompasses 1000 randomly generated models with network layers ranging between 6 and 112. The composition of each model is determined by a random selection of layers from a broader pool, which includes convolutional layers, fully connected layers, pooling layers, batch normalization layers, dropout layers, ReLU layers, and structural elements such as concatenation and addition layers. At each step, we randomly select from block types including sequential, addition, and concatenation blocks. Sequential block configurations are chosen from commonly used patterns such as (Conv, ReLU), (FC, ReLU), and (Conv, ReLU, Pool), optionally incorporating batch normalization.

Attacks and Defense Baselines. We choose three representative attacks as our defense targets, including a FPGA power attack [46], CPU cache attack [26] and GPU memory attack [20]. We also select five SOTA defense solutions as baselines, including three noise injection mechanisms [50], [23], [36] and two model architecture obfuscation mechanisms [51], [24].

Metrics. We adopt two evaluation metrics to demonstrate the effectiveness of `AdvProtego`. For *Model Similarity Reduction*, the model architecture is represented as a variable-length sequence where each element denotes a layer type. Following

prior works [46], [26], we use the Layer Error Rate (LER). It is defined as $LER = L(s', s)/\|s\|$, where $\|s\|$ is the length of sequence s, and $L(s', s)$ is the edit (Levenshtein) distance between the two sequences s' and s. In the following evaluation, *LER-to-label* denotes the LER between attacker's extracted architecture and victim model's ground-truth sequence, while *LER-to-target* denotes the LER between attacker's extracted architecture and defender's searched model with the worst performance. A larger *LER-to-label* indicates lower model similarity, thus higher defense effectiveness. For *Model Utility Reduction*, we train each extracted model under the same conditions and measure the accuracy drop compared to the original model; a higher drop indicates stronger defense.

B. Overhead

For FPGA power side channels, we analyze the resource utilization on the FPGA board. In our implementation, each RO utilizes only 4 LUTs. With each set containing 64 ROs and a total of 32 such sets, our design efficiently employs only 8,192 LUTs. Factoring in peripheral circuits such as the AXI bus, the aggregated resource utilization for our design stands at 8,251 LUTs and 170 flip-flops (FF). A comprehensive comparison of overhead across various defense solutions from prior works is presented in Table I. The "Area Increased" column shows the ratio of the number of LUTs and flip-flops used by the protective measures to the corresponding counts in the circuits under protection. The "Utilization" column computes the proportion of LUT and flip-flop counts relative to the available resources on the FPGA. For schemes targeting the AES accelerator, the resource usage is estimated based on the implementation in [13], where 2,640 LUTs and 1,682 FFs are utilized (as the information is not disclosed in these papers). The table demonstrates that among all schemes, `AdvProtego` exhibits the least resource consumption in relation to the circuit under protection. To estimate the power consumption, we rely on the Vivado power report to gauge the power usage of the Noise Unit. Its estimated power consumption amounts to 0.001W, constituting less than 1% of the total power consumption (1.737W).

TABLE I: Overhead for power side channels

Solution	LUT	FF	BRAM	Utilization	Target	Area Increased
[47]	4608	1152	0	4.10%	AES	175% / 69.0%
[48]	321	258	0	0.70%	AES	12.1% / 15.3%
[9]	8000	6499	163	8.63%	BNN	430% / 580%
[10]	9818	7709	163	10.43%	BNN	540% / 680%
`AdvProtego`	8251	170	0	1.28%	DNN	**10.1% / 0.1%**

For CPU cache and GPU memory side channels, we analyze the additional latency (ΔLatency), increased floating-point operations per second (ΔFLOPS), and the accuracy impact (ΔAcc) of each scheme. Table II summarizes these comparisons. For NeurObfuscator [24], we compare the configuration with a LER similar to `AdvProtego`, as reported in [24]. The FLOPs data for ObfuNAS [51] is based on experiments conducted with ImageNet. The configurations of these schemes align with those listed in Table III. As shown in Table II, `AdvProtego` consistently achieves the lowest overhead across all metrics, introducing no additional latency or accuracy degradation. The negative ΔLatency (–0.05%) indicates a marginal reduction in the measured inference latency, likely caused by statistical variation across repeated runs rather than an actual performance improvement. Similarly, the zero entries in ΔFLOPS show that `AdvProtego` imposes no computational overhead. The negative accuracy variation (–1%) in ObfuNAS reflects a 1% degradation in the protected model's accuracy, whereas `AdvProtego` maintains full accuracy (0% change). These results are expected since `AdvProtego` operates as an independent module alongside the AI inference pipeline, thus introducing minimal overhead compared to prior defenses.

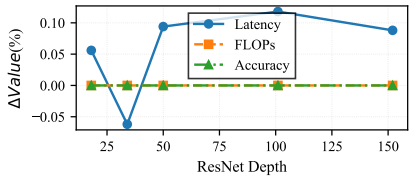


Fig. 2: Cost of `AdvProtego` for increased model scales.

TABLE II: Overhead for cache and memory side channels

Defense	ΔLatency (%)	ΔFLOPS (%)	ΔAcc (%)
NeurObfuscator[24]	2	53	-
ObfuNAS[51]	-	25	-1
CATalyst[25]	0.7	N.A.	N.A.
`AdvProtego` (CPU)	-0.05	0	0
`AdvProtego` (GPU)	0.09	0	0

To further evaluate the efficiency of `AdvProtego` under sustained workloads and large-scale models, we conduct additional experiments over the ResNet family (ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152). Figure 2 shows the results of latency, FLOPs and accuracy degradation, which have no measurable increase after applying `AdvProtego`. The measured values remain statistically indistinguishable from the unprotected baseline, with any minor fluctuation within system-level variance. This further confirms that `AdvProtego` introduces negligible runtime or computational overhead even for large, sustained workloads. This is because the perturbation generation and injection modules are implemented as independent components that operate orthogonally to the inference process. As such, the defense operations of `AdvProtego` run asynchronously and do not interfere with the computation flow of the target model, ensuring scalability and transparency even in deep architectures.

C. Effectiveness

Model Similarity Reduction. We analyze the adversarial noise generated by `AdvProtego` through its Noise Injector across different platforms. Table III compares the LER of `AdvProtego` and other defense methods. For fair comparison, we set a maximum RO set enable number as 8 for all defenses implemented on FPGA, and the maximum memory read/write amount is limited to 1KB for experiments on GPUs. We observe that the LERs of our delicate adversarial noise are significantly higher than other methods, indicating its higher effectiveness in reducing the similarity of stolen model.

Model Utility Reduction. Table III shows the evaluation results. "Acc. Drop (Searched)" indicates the accuracy decrease

TABLE III: LER and accuracy for Model Utility and Similarity Reduction

Defense Methods	Dataset	Platform	LER (%)	Acc. Drop (Searched)	Acc. Drop (Extracted)
[50]	MNIST	FPGA	75.6	-	-0.7%
[36]	MNIST	FPGA	70.5	-	-0.8%
[23]	MNIST	FPGA	95.4	-	-1.9%
[33]	MNIST	FPGA	92.3	-	-2.1%
[50]	CIFAR10	FPGA	84.5	-	-8.5%
[36]	CIFAR10	FPGA	89.1	-	-2.7%
[23]	CIFAR10	FPGA	49.2	-	-8.8%
[33]	CIFAR10	FPGA	53.5	-	-6.3%
[50]	ImageNet	GPU	56.3	-	-
[36]	ImageNet	GPU	57.7	-	-
[50]	ImageNet	CPU	52.2	-	-
[36]	ImageNet	CPU	61.8	-	-
[24]	ImageNet	All	-	-0.4%	-
[51]	ImageNet	All	-	0.1%	-
[35]	ImageNet	GPU	1.6	-	-
AdvProtego	MNIST	FPGA	**144.1**	2.9%	**1.6%**
AdvProtego	CIFAR10	FPGA	**128.3**	20.3%	**17.4%**
AdvProtego	ImageNet	CPU	137.2	13.6%	9.9%
AdvProtego	ImageNet	GPU	**156.7**	13.6%	**10.3%**
AdvProtego	Caltech256	CPU	115.2	28.8%	16.1%
AdvProtego	Caltech256	GPU	131.9	28.8%	19.4%

of the identified target model relative to the victim's original model. "Acc. Drop (Extracted)" refers to the actual accuracy degradation of attacker's extracted model relative to victim's original model when noise is incorporated into the side channel. A larger accuracy drop signifies a greater degradation of the extracted model and higher defense effectiveness.

We observe that the accuracy drop in attacker's extracted model closely aligns with the target accuracy identified during the search phase. This indicates that AdvProtego generates and injects noise with high precision, achieving the intended obfuscation effect. Furthermore, the extracted model's accuracy drop is significantly larger compared to all the other defense methods, confirming that AdvProtego effectively reduces model utility. The main reason is that existing defenses [50], [36], [23], [24] mainly focus on distorting attacker's observations, while ignoring the utility of the extracted models. Hence, the attacker can still extract a good model.

In Table IV, we measure the similarity of the extracted model to the NAS-searched one ("LER to target") and to the victim one ("LER to victim"). We observe that LER to target is significantly lower than LER to victim, indicating that the perturbations indeed drive the extracted model towards the searched worst-performing model.

TABLE IV: Similarity of the extracted model to the victim and searched models.

Defense Methods	Dataset	LER to victim	LER to target
[50]	MNIST	94.6%	356.3%
[36]	MNIST	73.3%	153.7%
[23]	MNIST	98.3%	257.4%
[50]	CIFAR	37.1%	263.7%
[36]	CIFAR	74.9%	140.0%
[23]	CIFAR	74.3%	124.8%
AdvProtego	MNIST	98.1%	44.3%
AdvProtego	CIFAR	75.3%	69.1%
AdvProtego	ImageNet	96.0%	51.7%

D. Ablation Study

To further justify the necessity of each design component in AdvProtego. We perform a comprehensive ablation study. Specifically, we remove individual modules from the full pipeline and re-evaluate both defense objectives. All experiments are conducted on the GPU platform using ImageNet.

The results in Table V shows that removing any component significantly degrades protection performance. For Model Similarity Reduction, without accumulation across samples, LER drops from 156.7% to 65.6%; omitting the mask reduces it to 76.6%; and removing quantization and calibration almost completely destroys effectiveness (LER = 17.1%). For Model Utility Reduction, removing NAS notably weakens the extracted model degradation (Acc. Drop (Extracted) = 0.8% versus 10.3% for the full version). Similarly, removing accumulation, masking, or quantization/calibration causes noticeable defense strength reduction.

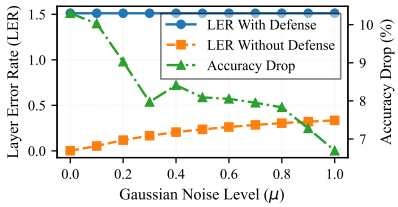


Fig. 3: Impact of environmental noise on defense effectiveness.

TABLE V: Ablation Study. LER is for Model Similarity Reduction; Acc. Drop is for Model Utility Reduction.

Type	**LER**	**Acc. Drop (Searched)**	**Acc. Drop (Extracted)**
Full	156.7%	13.6%	10.3%
No NAS	–	13.6%	0.8%
No accumulate	65.6%	13.6%	3.4%
No mask	76.6%	13.6%	6.5%
No quantization & calibration	17.1%	13.6%	1.5%

Overall, these experiments confirm that all components in AdvProtego contribute meaningfully to its robustness and scalability. Without NAS, AdvProtego fails to intentionally mislead extraction toward low-utility targets; without accumulation, masking, or quantization, the generated perturbations lose transferability and practical deployability.

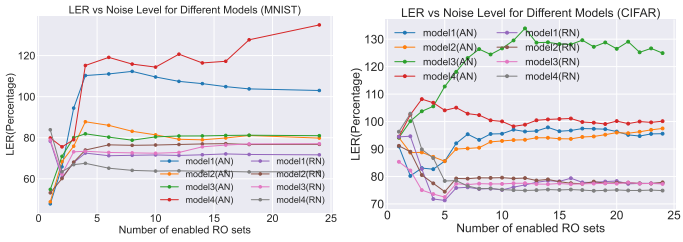
E. Robustness Against Environmental and Process Noise

In practical deployments, side-channel traces are inevitably affected by environmental factors such as temperature drift, voltage fluctuation, and process variation. To examine the robustness of AdvProtego under such conditions, we conduct a noise-resilience experiment by injecting Gaussian noise into the side-channel traces. Specifically, we vary the mean value of Gaussian noise $\mu = [0, 0.1, 0.2, ..., 1]$ and evaluate the resulting traces under two defense settings.

Figure 3 illustrates the results. For Model Similarity Reduction, AdvProtego maintains a high LER across all noise levels, showing negligible performance degradation even as μ increases. For Model Utility Reduction, we report the corresponding accuracy degradation ("Acc. Drop") of attacker's extracted model under the same conditions. While minor fluctuations occur under extreme noise conditions, the overall trend remains stable.

F. Transferability

To measure the transferability of the adversarial noise from defender's surrogate model to attacker's model, we construct five models with varying structures . Model 0 is the surrogate model used by the defender to generate adversarial noise, while models 1-4 are used by the attacker for side-channel analysis. We mainly conduct experiments on FPGA. We diversify the noise intensity by adjusting the number of RO enabled sets, for analyzing the impact of varying levels of interference on the transferability and overall defense effectiveness.



(a) Results for MNIST dataset (b) Results for CIFAR10 dataset

Fig. 4: LERs for different attack models. "AN": adversarial noise from `AdvProtego`; "RN": random noise

Model Similarity Reduction. We measure the LERs between the extracted and victim models. The results are shown in Figure 4. In both datasets, `AdvProtego` induces significantly higher error rates in the extracted models compared to using random noise across four distinct models. This underscores the efficacy of `AdvProtego` against diverse attack models.

Model Utility Reduction. We proceed to test the transferability of the Model Utility Reduction defense strategy. The evaluation results for this defense objective are shown in Table VI. "Acc. Drop (Extracted)" represents the actual accuracy drop of the attacker's extracted model compared to the victim's model when the identified adversarial noise is implemented. A higher extraction accuracy drop indicates a greater degradation of the extracted model, thus higher defense effectiveness. The results clearly show that the extracted accuracy of the attacker's model is lower than the victim's model accuracy for all the four cases. In model 2, the defense result is even better than using the surrogate model (model 0) for extraction. This confirms the transferability and effectiveness of `AdvProtego` against different attack models.

Sensitivity to Surrogate Model Quality. The robustness of `AdvProtego` can be affected by defender's surrogate model. To evaluate this, we measure the defense performance by varying the surrogate model's accuracy and representational fidelity. Specifically, we select a set of models with different training completeness and generalization capabilities: Model-0 corresponds to the well-trained surrogate adopted in our main experiments, while Model-5∼7 represent weaker surrogates—either under-trained or exhibiting lower extraction accuracy relative to the victim's model.

TABLE VI: Transferability results for Model Utility Reduction

Model	Dataset	Acc. Drop (Extracted)
1	MNIST	0.7%
2	MNIST	3.0%
3	MNIST	0.8%
4	MNIST	0.8%
1	CIFAR10	10.5%
2	CIFAR10	15.2%
3	CIFAR10	12.0%
4	CIFAR10	10.4%

TABLE VII: Defense sensitivity to surrogate model quality

Model	**Attack LER(%)**	**Defense LER(%)**	**Acc. Drop (Searched(%))**	**Acc. Drop (Extracted(%))**
0	0.1	156.7	13.6	10.3
5	14.3	140.1	13.6	8.7
6	38.2	80.3	13.6	4.9
7	80.4	77.9	13.6	4.1

Table VII shows the results: the surrogate model quality is quantified by "Attack LER", where a lower value indicates that the surrogate can more accurately extract the victim's architecture. As the surrogate model quality decreases, we observe a gradual reduction in the defense's performance, but the degradation remains within reasonable bounds. Similarly, the attacker's extracted model's accuracy degradation decreases from 10.3% to 4.1%, suggesting that while weaker surrogates produce less optimized perturbations, they still meaningfully disrupt model extraction.

G. Resilience against Adaptive Attacks

We explore the scenarios where the attacker is aware of our defense mechanisms and attempts to bypass them.

Adversarial attack mitigation as adaptive attacks. A potential adaptive attack is to apply existing mitigation solutions against adversarial attacks to circumvent `AdvProtego`. However, as adversarial attacks are still an unsolved threat [37], these solutions still exhibit some limitations to undermine adversarial perturbations. For evaluation, we mainly consider three advanced techniques. (1) Sample resizing and rescaling [45]. This involves adding random resizing and padding layers at the beginning of the target model. These layers alter the spatial positioning of adversarial perturbations to make them ineffective. (2) Randomized smoothing [8]. This entails training a neural network with Gaussian data augmentation to create a smoothed classifier. (3) Bit-depth reduction [16]. This involves simple quantization to remove small adversarial variations in the input. We reduce the input side-channel traces to 8 bits. We conduct experiments on FPGA with MNIST to evaluate if `AdvProtego` can resist those operations when employed by the attacker to process the power traces.

Figure 5 presents the results of Model Similarity Reduction. Table VIII shows the results of Model Utility Reduction. In conclusion, these adaptive attacks do not perform as effectively as they do for conventional AI tasks , making it challenging to leverage these countermeasures to circumvent our defense.

Noise isolation as adaptive attacks. An adaptive attacker may attempt to isolate the injected noise from natural execution behavior by exploiting its statistical or spectral characteristics. To evaluate the resilience of `AdvProtego` against such attack, we quantitatively compare the distinguishability between injected noise and normal side-channel trace across three representative filtering methods: (1) *spectral filtering* (FFT-based low-pass), (2) *statistical filtering* (median smoothing), and (3) *adaptive filtering* (a hybrid of Gaussian and spectral filtering). We experiment with the GPU platform over ImageNet, and adopt LER to measure the defense effectiveness.

The results show that the filtered traces—spectral (LER: 1.23), statistical (LER: 0.96), and adaptive (LER: 0.89)—exhibit only marginal improvement compared to the defense baseline without filtering (LER: 1.57). These findings demonstrate that adaptive filtering fails to effectively recover the hidden signal. This is because the injected noise closely matches the statistical characteristics of natural execution: both share the same hardware pathways and contribute jointly to observable side-channel variations.

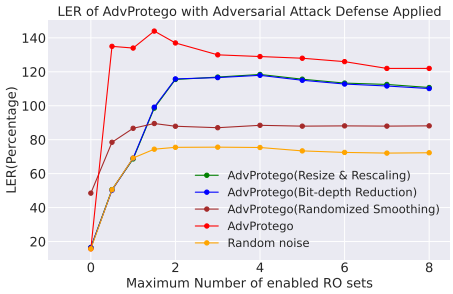


Fig. 5: LERs for random noise and AdvProtego with adversarial attack defense mechanisms applied.

TABLE VIII: LER and accuracy for Model Utility Reduction with adversarial attack defense mechanisms applied.

Defense	LER to label	LER to target	Acc. Drop (Searched)	Acc. Drop (Extracted)
No defense	67.2%	15.3%	2.9%	2.2%
Resize & rescaling	73.5%	46.3%	2.9%	1.2%
Randomized smoothing	68.7%	5.9%	2.9%	2.5%
Bit-depth reduction	76.1%	46.5%	2.9%	1.3%

VI. Discussion

A. Extension to Other Tasks and Models

In this paper, we mainly evaluate AdvProtego on the protection of computer vision tasks. However, its design is general. AdvProtego is architecture-agnostic because its core operation is built upon the attacker's surrogate model rather than the protected model itself. The victim model simply produces side-channel traces; the actual perturbations are computed to mislead the attacker's extraction model, regardless of the internal structure of the victim network. Consequently, AdvProtego is not inherently tied to convolutional architectures or the vision modality. In principle, as long as the attacker relies on a trainable extraction model that can be adversarially perturbed, AdvProtego can be adapted. This is consistent with prior evidence that adversarial attack techniques generalize to modern architectures, e.g., Vision Transformers [49].

Our choice to evaluate on vision workloads aligns with the current state of side-channel model-stealing research. To the best of our knowledge, most published side-channel extraction attacks to date target deep learning platforms executing vision models. The massive scale, distributed execution, and quantized/sharded deployment of LLMs make side-channel leakage modeling extremely challenging under current measurement resolutions. Thus, our focus on CV workloads reflects where practical attacks currently exist and can be meaningfully defended. Extending protection to LLMs and multimodal architectures represents an important next step as future work.

B. Design of Objective Functions

AdvProtego separates two complementary defense objectives: Model Similarity Reduction and Model Utility Reduction, because they reflect distinct attacker goals and each requires a different optimization and hardware-aware translation strategy. The former targets structural reconstruction errors useful against sequence-based extractors, while the later leverages reversed NAS to directly minimize attacker test accuracy. Combining these objectives into a single weighted loss is possible but could increase optimization sensitivity and, importantly, be projected by quantization / mask constraints into infeasible injection commands. We perform some evaluations and observe that the defense with the unified objective function achieves weaker performance than the ones focusing on either single objective. Therefore, it is suggested to adopt the two-objective design to obtain robust, transferable perturbations under realistic hardware constraints. The defender should selects the objective based on the actual security requirement.

C. Impact of Continual Model Updates

In real-world deployments, machine learning models may be periodically updated through fine-tuning or additional retraining to improve performance or adapt to new data. Such model evolvement does not affect the defense. On the one hand, model updates normally only alter the model weights but not the network architectures. They induce negligible changes in the side-channel trace distribution, which targets the model architecture. Therefore, moderate weight updates after redeployment have limited influence on either the attack or defense. On the other hand, the core defense of AdvProtego operates on the defender's surrogate model instead of the deployed model itself. When the target model is retrained, fine-tuned, or partially adapted post-deployment, the defender can retrain or update the surrogate with minimal cost to approximate the new architecture or data distribution. The whole process is lightweight and automated, without requiring redesigning the hardware injectors or retraining the full defense pipeline. This design choice decouples the defense from any single frozen model instance and supports periodic retraining of protection signals.

VII. Conclusion

We present AdvProtego, a novel framework to defeat side-channel-based model stealing. It crafts delicate execution noise to obfuscate side-channel observations. Our contributions include the followings: (1) proposing two defense objectives and formulating them in the context of adversarial perturbation optimization. (2) introducing an end-to-end modular design with innovative algorithms and techniques to achieve practical and unified protection. We implement AdvProtego in three representative attack environments to demonstrate its excellence and superiority over existing solutions.

VIII. Acknowledgement

This research is supported by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity Research & Development Programme (Development of Secured Components & Systems in Emerging Technologies through Hardware & Software Evaluation <NRF-NCR25-DeSNTU-0001>). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the view of National Research Foundation, Singapore and Cyber Security Agency of Singapore.

REFERENCES

[1] Amazon F2 web site. https://aws.amazon.com/cn/ec2/instance-types/f2/.

[2] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. On subnormal floating point and abnormal timing. In *2015 IEEE Symposium on Security and Privacy*, pages 623–639. IEEE, 2015.

[3] Daniel J Bernstein, Tanja Lange, and Peter Schwabe. The security impact of a new cryptographic library. In *Progress in Cryptology–LATINCRYPT 2012: 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings 2*, pages 159–176. Springer, 2012.

[4] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures: Profiling attacks without pre-processing. In *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 45–68. Springer, 2017.

[5] Giuseppe Cesarano. Fpga implementation of a deep learning inference accelerator for autonomous vehicles.

[6] Jung-Woo Chang, Ke Sun, David Xia, Xinyu Zhang, and Farinaz Koushanfar. Eveguard: Defeating vibration-based side-channel eavesdropping with audio adversarial perturbations. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 4534–4552. IEEE, 2025.

[7] Kwen-Siong Chong, Jun-Sheng Ng, Juncheng Chen, Ne Kyaw Zwa Lwin, Nay Aung Kyaw, Weng-Geng Ho, Joseph Chang, and Bah-Hwee Gwee. Dual-hiding side-channel-attack resistant fpga-based asynchronous-logic aes: Design, countermeasures and evaluation. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(2):343–356, 2021.

[8] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *international conference on machine learning*, pages 1310–1320. PMLR, 2019.

[9] Anuj Dubey, Rosario Cammarota, and Aydin Aysu. Bomanet: Boolean masking of an entire neural network. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[10] Anuj Dubey, Rosario Cammarota, Vikram Suresh, and Aydin Aysu. Guarding machine learning hardware against physical side-channel attacks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(3):1–31, 2022.

[11] Apostolos P. Fournaris and Odysseas Koufopavlou. Protecting crt rsa against fault and power side channel attacks. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 159–164, 2012.

[12] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[13] Joseph Gravellier, Jean-Max Dutertre, Yannick Teglia, and Philippe Loubet-Moundi. High-speed ring oscillator based sensors for remote side-channel attacks on fpgas. In *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8. IEEE, 2019.

[14] Gregory Griffin, Alex Holub, Pietro Perona, et al. Caltech-256 object category dataset. Technical report, Technical Report 7694, California Institute of Technology Pasadena, 2007.

[15] Ruizhe Gu, Ping Wang, Mengce Zheng, Honggang Hu, and Nenghai Yu. Adversarial attack based countermeasures against deep learning side-channel attacks. *arXiv preprint arXiv:2009.10568*, 2020.

[16] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens Van Der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

[17] Yanan Guo, Andrew Zigerelli, Youtao Zhang, and Jun Yang. Ivcache: Defending cache side channel attacks via invisible accesses. In *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, pages 403–408, 2021.

[18] Naina Gupta, Arpan Jati, and Anupam Chattopadhyay. Ai attacks ai: Recovering neural network architecture from nvdla using ai-assisted side channel attack. Cryptology ePrint Archive, Paper 2023/368, 2023. https://eprint.iacr.org/2023/368.

[19] Yi-Liang Hong, Yui-Kai Weng, and Shih-Hsu Huang. Hardware implementation for fending off side-channel attacks. In *2021 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, pages 1–2, 2021.

[20] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 385–399, 2020.

[21] Weizhe Hua, Zhiru Zhang, and G Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *ACM/ESDA/IEEE Design Automation Conference*, 2018.

[22] Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Deepcloak: Adversarial crafting as a defensive measure to cloak processes. In *Proceedings of the 2019 Workshop on DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security*, pages 1–12, 2019.

[23] Jonas Krautter, Dennis RE Gnad, Falk Schellenberg, Amir Moradi, and Mehdi B Tahoori. Active fences against voltage-based side channels in multi-tenant fpgas. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[24] Jingtao Li, Zhezhi He, Adnan Siraj Rakin, Deliang Fan, and Chaitali Chakrabarti. Neurobfuscator: A full-stack obfuscation tool to mitigate neural architecture stealing. In *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 248–258. IEEE, 2021.

[25] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby B Lee. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *2016 IEEE international symposium on high performance computer architecture (HPCA)*, pages 406–418. IEEE, 2016.

[26] Xiaoxuan Lou, Shangwei Guo, Jiwei Li, Yaoxin Wu, and Tianwei Zhang. Naspy: Automated extraction of automated machine learning models. In *International Conference on Learning Representations*, 2021.

[27] Yukui Luo, Shijin Duan, Cheng Gongye, Yunsi Fei, and Xiaolin Xu. Nnrearch: A tensor program scheduling framework against neural network architecture reverse engineering. In *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 1–9. IEEE, 2022.

[28] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[29] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*, pages 293–312. Elsevier, 2019.

[30] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

[31] Shayan Moini, Shanquan Tian, Daniel Holcomb, Jakub Szefer, and Russell Tessier. Remote power side-channel attacks on bnn accelerators in fpgas. In *Design, Automation & Test in Europe Conference & Exhibition*, 2021.

[32] Soo-Jin Moon, Vyas Sekar, and Michael K Reiter. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *Proceedings of the 22nd acm sigsac conference on computer and communications security*, pages 1595–1606, 2015.

[33] Mahya Morid Ahmadi, Faiq Khalid, Radha Vaidya, Florian Kriebel, Andreas Steininger, and Muhammad Shafique. Shield: An adaptive and lightweight defense against the remote power side-channel attacks on multi-tenant fpgas. *Available at SSRN 4459334*, 2023.

[34] Hyoungwook Nam, Raghavendra Pradyumna Pothukuchi, Bo Li, Nam Sung Kim, and Josep Torrellas. Friendlyfoe: Adversarial machine learning as a practical architectural defense against side channel attacks. In *Proceedings of the 2024 International Conference on Parallel Architectures and Compilation Techniques*, pages 338–350, 2024.

[35] Stjepan Picek, Dirmanto Jap, and Shivam Bhasin. Poster: When adversary becomes the guardian–towards side-channel security with adversarial attacks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2673–2675, 2019.

[36] Raghavendra Pradyumna Pothukuchi, Sweta Yamini Pothukuchi, Petros G Voulgaris, Alexander Schwing, and Josep Torrellas. Maya: Using formal control to obfuscate power side channels. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 888–901. IEEE, 2021.

[37] Javier Rando, Jie Zhang, Nicholas Carlini, and Florian Tramèr. Adversarial ml problems are getting harder to solve and to evaluate. *arXiv preprint arXiv:2502.02260*, 2025.

[38] Mark Randolph and William Diehl. Power side-channel attack analysis: A review of 20 years of study for the layman. *Cryptography*, 4(2):15, 2020.

[39] Read Sprabery, Konstantin Evchenko, Abhilash Raj, Rakesh B Bobba, Sibin Mohan, and Roy Campbell. Scheduling, isolation, and cache allocation: A side-channel defense. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 34–40. IEEE, 2018.

[40] François-Xavier Standaert, François Koeune, and Werner Schindler. How to compare profiled side-channel attacks? In *International Conference on Applied Cryptography and Network Security*, 2009.

[41] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[42] Shanquan Tian, Shayan Moini, Adam Wolnikowski, Daniel Holcomb, Russell Tessier, and Jakub Szefer. Remote power attacks on the versatile tensor accelerator in multi-tenant fpgas. In *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2021.

[43] Zhenghong Wang and Ruby B Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 34th annual international symposium on Computer architecture*, pages 494–505, 2007.

[44] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 393–406, 2018.

[45] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

[46] Xiaobei Yan, Xiaoxuan Lou, Guowen Xu, Han Qiu, Shangwei Guo, Chip Hong Chang, and Tianwei Zhang. Mercury: An automated remote side-channel attack to nvidia deep learning accelerator. In *2023 International Conference on Field-Programmable Technology (ICFPT)*, 2023.

[47] Yuan Yao, Pantea Kiaei, Richa Singh, Shahin Tajik, and Patrick Schaumont. Programmable ro (pro): A multipurpose countermeasure against side-channel and fault injection attack. *arXiv preprint arXiv:2106.13784*, 2021.

[48] Fan Zhang, Zhiyong Wang, Haoting Shen, Bolin Yang, Qianmei Wu, and Kui Ren. Darpt: defense against remote physical attack based on tdc in multi-tenant scenario. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 559–564, 2022.

[49] Jianping Zhang, Yizhan Huang, Weibin Wu, and Michael R. Lyu. Transferable adversarial attacks on vision transformers with token gradient regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16415–16424, June 2023.

[50] Mark Zhao and G Edward Suh. Fpga-based remote power side-channel attacks. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 229–244. IEEE, 2018.

[51] Tong Zhou, Shaolei Ren, and Xiaolin Xu. Obfunas: A neural architecture search-based dnn obfuscation approach. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[52] Ziqiao Zhou, Michael K Reiter, and Yinqian Zhang. A software approach to defeating side channels in last-level caches. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 871–882, 2016.

[53] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.