The ISC Int'l Journal of Information Security

ISeCure

October 2025, Volume 17, Number 3 (pp. 1–17)

http://www.isecure-journal.org

Presented at the ISCISC'2022 in Tehran, Iran.

Securing Deep Learning Hardware: A Survey of Side-Channel Vulnerabilities and Countermeasures **

Zahra Mohammadi [1], Mona Hashemi [1], and Siamak Mohammadi [1,*]

[1] *School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran.*

ARTICLE INFO.

Keywords:
Side-Channel Attacks, Deep Learning Models, Model Reverse Engineering, Intellectual Property, Side-Channel Protection, Model Security

Type:

doi:

ABSTRACT

As deep learning models are increasingly deployed in critical sectors such as healthcare, finance, and security, ensuring their protection against emerging threats has become crucial. Among these threats, side-channel attacks (SCAs) represent a particular challenge since they can extract sensitive information such as model architectures, parameters, and even user inputs without requiring direct access to the model. By leveraging the physical and micro-architectural properties of the hardware, attackers can compromise systems. This survey begins by classifying leakage sources and attacker objectives, then analyzes representative studies that demonstrate practical side-channel exploits against deep-learning hardware. It also reviews existing defenses aimed at mitigating these vulnerabilities and concludes by outlining key open research challenges and potential future directions.

1 Introduction

Deep learning models have become fundamental to a wide array of modern applications, including cloud computing, mobile platforms, Internet of Things (IoT) devices, and critical infrastructure. Their exceptional accuracy and performance have led to widespread deployment across both data centers and resource-constrained edge devices. However, the underlying hardware used to run these models such as Central Processing Units (CPUs), Graphics Processing Units (GPUs), Field-Programmable Gate Arrays (FPGAs), and custom accelerators can unintentionally expose sensitive information through physical and micro-architectural side channels.

Side-channel leakages, such as power consumption, memory access patterns, timing differences, and electromagnetic (EM) emissions, can be exploited by adversaries to extract critical model information. These include architecture details, model parameters, or even private user inputs [1–3]. Such attacks do not rely on software vulnerabilities but instead observe and analyze low-level physical behaviors of hardware during model execution.

This survey provides a comprehensive overview of the landscape of hardware side-channel vulnerabilities in deep learning models. We present a structured taxonomy of SCAs, analyze representative attack techniques, and review defensive strategies developed to mitigate these threats. By categorizing attacks based on leakage sources, attacker capabilities, and objec-

* Corresponding author.

**The ISCISC'2025 program committee effort is highly acknowledged for reviewing this paper.

Email addresses: zahramohammmadi@ut.ac.ir, Hashemi.mona@ut.ac.ir, smohamadi@ut.ac.ir

ISSN: 2008-2045

tives, this paper aims to shed light on the emerging security challenges and motivate the development of more resilient machine learning hardware systems.

1.1 Deep Learning Model as Intellectual Property (IP)

Modern deep learning models represent far more than simple lines of code—they are valuable forms of IP, developed through extended efforts involving expert engineering, costly training procedures, and often the use of proprietary datasets. Constructing a competitive model typically requires large-scale data collection and annotation, careful design of neural network architectures, and thorough hyperparameter tuning. For instance, training large-scale models such as Generative Pre-trained Transformers like GPT-3 and GPT-4 is estimated to cost over $100 million, with expenses often offset through monetization strategies like paid services or Application Programming Interfaces (APIs) [4].

Beyond financial investment, these models encapsulate strategic knowledge that can serve as a major competitive advantage. Many are fine-tuned using proprietary data—ranging from user behavior to medical records—which enables them to outperform publicly available or open-source alternatives. If an attacker successfully steals such a model, they can reproduce its functionality without incurring the original development costs. This constitutes IP theft and poses a serious threat to the developer's market position [5].

Furthermore, deep learning models can unintentionally retain portions of their training data, especially when overfitting or fine-tuning occurs on small, sensitive datasets. This raises serious privacy concerns in domains such as healthcare, finance, and biometrics, where data leakage could lead to regulatory violations or facilitate reconstruction attacks such as membership inference and model inversion [5].

These risks are not merely theoretical. In a recent controversial case, it was suspected that DeepSeek had copied models developed by OpenAI without explicit permission [6]. While no definitive evidence of model theft has been confirmed, the case raised serious concerns about the potential for unauthorized use of proprietary artificial intelligence (AI) systems. Such incidents highlight the growing importance of protecting deep learning models as strategic assets.

1.2 Threat Landscape for Deep Learning Hardware

The widespread deployment of deep learning across cloud platforms, edge devices, and embedded systems has significantly expanded the risk of SCAs. The hardware used to execute these models has historically been designed with a primary focus on performance, not security or side-channel resilience. As a result, these devices may unintentionally expose sensitive information through hardware-level leakage. Adversaries can exploit observable signals such as power consumption, EM radiation, cache behavior, and timing differences to extract confidential details, including model parameters and user inputs [1–3, 7–9].

One of the most prominent vulnerabilities emerges in multi-tenant cloud environments, where computational resources like CPUs and GPUs are shared among numerous users, some of whom may not be trusted. This setup is typical in commercial platforms such as Amazon Web Services (AWS) and Microsoft Azure [10]. In such settings, attackers can apply techniques such as timing analysis, context-switch monitoring, and cache probing to recover model parameters or reconstruct inputs, all without breaching standard hardware isolation protocols [1, 10–14].

Another critical concern lies in edge deployments, including applications such as smart cameras, drones, autonomous vehicles, and IoT sensors. These devices are often physically accessible and therefore particularly susceptible to power analysis and EM SCAs. Attackers can connect probes to power lines or observe EM fields to infer internal model information or reconstruct private inputs such as images. The physical nature of these attacks makes them difficult to detect and mitigate—especially in devices with limited processing capabilities, minimal tamper resistance, or cost-sensitive designs [2, 5, 15–17].

Taken together, these scenarios reveal a growing and diverse attack surface for deep learning hardware. From EM attacks on consumer drones to cache-based inference in large-scale data centers, adversaries have multiple vectors through which they can bypass software-layer protections and compromise the confidentiality of deployed models.

1.3 Research Motivation

While SCAs have been extensively studied in the context of cryptographic hardware, their implications for deep learning systems remain comparatively underexplored. The hardware platforms that execute these models pose unique challenges that are not fully addressed by traditional security models [18]. One critical distinction lies in the nature and scale of the underlying secrets. Unlike cryptographic keys, which have short and fixed-length representations, deep neural networks may contain millions or even billions of floating-point parameters. The leakage of these parameters can differ significantly across layers, depending on both model architecture and input data,

which complicates detection and mitigation [18].

Another difficulty stems from the dynamic and irregular execution patterns of deep learning workloads. Variable input dimensions, mixed-precision arithmetic, compiler optimizations, and hardware-level scheduling all introduce significant noise and unpredictability into side-channel traces. These factors increase the complexity of constructing effective attacks and designing robust defenses, particularly in multi-tenant or real-time execution environments.

At the same time, hardware innovation is advancing faster than security evaluation can keep pace. New classes of AI hardware such as tensor cores, neuromorphic chips, and photonic processors are being introduced into the market before their side-channel vulnerabilities are fully understood.

Beyond technical concerns, these challenges also pose serious economic risks. As Machine Learning as a Service (MLaaS) platforms and embedded AI solutions generate billions of dollars in revenue, a successful SCA could result in IP theft, privacy breaches, and major financial losses. Yet, most existing work focuses on algorithmic threats such as adversarial examples or cryptographic protections, while overlooking hardware-level leakage channels.

This survey aims to bridge that gap by offering:

- A unified taxonomy of hardware SCAs on deep learning models, highlighting leakage sources, attacker capabilities, and attack goals.
- A review of representative attacks reported in the literature.
- An overview of existing defense mechanisms and runtime mitigation strategies.
- An analysis of current research gaps and future directions, including potential threats to transformer-based models and other emerging AI frameworks.

1.4 Literature Search Methodology

To provide a comprehensive overview of hardware side-channel vulnerabilities in deep learning models, we employed a systematic literature search strategy. Initially, several key survey papers in the domain were reviewed to understand its breadth; this field encompasses diverse hardware platforms (such as CPUs, GPUs, FPGAs, and custom accelerators) and various leakage sources that are exploited to extract three primary categories of model information (architecture, parameters, and inputs), each of which holds significant value for adversaries. Based on these insights, the taxonomy presented in Figure 1 was developed. Subsequently, priority was given to leakage sources; for each source, papers with high citation counts or foundational works were first identified and studied, followed by more recent (up-to-date) contributions, and finally, studies focusing on different hardware targets (such as cache-based side-channel attacks on both CPUs and GPUs) were selected to ensure diverse coverage. The search primarily focused on papers published in the last five years (2020–2025), as this is an emerging field with most relevant research falling within this timeframe. Inclusion criteria encompassed papers that specifically addressed side-channel attacks against hardware platforms used in deep learning models and provided empirical evidence or theoretical analyses of attack methods, vulnerabilities, or countermeasures. Exclusion criteria eliminated papers primarily centered on non-hardware attacks or information extraction from non-deep learning models. This approach, while comprehensive, concentrates on approximately 30–40 representative studies to maintain analytical depth and avoid redundancy.

The rest of this paper is organized as follows: Section 2 introduces deep learning hardware platforms, identifies major leakage sources, and presents a taxonomy of SCAs. Section 3 reviews several representative attacks categorized by their leakage type. Section 4 surveys defense mechanisms and detection techniques. Section 5 outlines open research challenges and future directions. Finally, Section 6 concludes the paper.

2 Background

Before exploring specific SCA techniques and defense mechanisms, this section provides the necessary background and introduces a structured taxonomy. SCAs exploit unintentional information leakage from physical or micro-architectural behaviors of hardware such as timing variations, power consumption, EM emissions, or cache activity to extract sensitive data. Unlike conventional software-based attacks that target code-level vulnerabilities, SCAs operate by passively or actively monitoring low-level hardware behavior during model execution.

This section begins by surveying the landscape of deep learning hardware platforms and identifying prevalent sources of information leakage. It then outlines typical attacker capabilities and threat models, followed by a classification of attack objectives and the ways in which extracted information may be exploited. To provide a practical understanding, representative attack primitives such as power analysis and cache-based methods are also introduced.

Figure 1 presents a comprehensive taxonomy of hardware SCAs on deep learning models, categorized according to key dimensions such as leakage sources, attack methodology, access level, target hardware, and attacker objectives. While this taxonomy draws

inspiration from general SCA classifications in prior surveys, such as the one proposed in [2], it is specifically tailored to deep learning systems. For instance, unlike general SCA taxonomies that emphasize broad categories like logical vs. physical exploited properties or profiling phases across various cryptographic implementations, our taxonomy highlights deep learning specific elements, including hardware platforms optimized for neural networks (e.g., GPUs, FPGAs, and edge AI chips) and attacker goals focused on extracting model architecture, parameters, or inputs. This adaptation addresses the unique vulnerabilities in machine learning hardware deployments.

2.1 Deep Learning Hardware

To meet the growing computational demands of deep learning, specialized hardware platforms are widely adopted. However, each of these platforms introduces distinct vulnerabilities and potential side-channel leakage vectors that have been explored in various studies.

•**CPUs** are commonly used for neural network inference, particularly in scenarios where flexibility, energy efficiency, or cost-effectiveness is prioritized over raw performance. Due to their extensive use of shared memory resources, especially the Last-Level Cache (LLC), CPUs are highly susceptible to cache-based SCAs. For instance, studies such as *DeepCache* and *Cache Telepathy* have demonstrated that techniques like *Flush+Reload* can accurately recover the architecture and parameters of convolutional neural networks (CNNs) running on CPU-based platforms [10, 19]. Additionally, the work in [20] demonstrates a stealthy inference attack using cache-based side-channel on CPUs to extract the model outputs, while the article *DeepTheft* [16] exploits power side channels to infer model architectures on CPUs.

•**GPUs**, with their massively parallel architecture optimized for high-throughput numerical operations, are ideal for both training and batch inference of deep learning models. Despite their advantages, GPUs exhibit significant side-channel vulnerabilities due to parallel execution, predictable scheduling patterns, and distinctive power consumption signatures. For example, the *Leaky DNN* attack exploited timing variations during context switching to extract detailed architectural information from models running on shared GPU infrastructure [11]. Similarly, the *BarraCUDA* attack leveraged EM emissions from GPU operations to recover the weights of convolutional layers with high precision [21]. Additionally, the work in [22] demonstrates the exploitation of side-channel information from GPUs to extract optimized Deep Neural Network (DNN) architectures, using techniques like timing variations during inference. Similarly, the article *Spy in the GPU-box* [23] highlights a covert channel attack across multi-GPU systems, where cache contention across GPUs was used to infer sensitive data, including model information from a remote GPU.

•**FPGAs** are reconfigurable and flexible platforms that support rapid prototyping and the deployment of custom accelerators for deep learning inference tasks. Likewise,custom accelerators—such as tensor processing units and neuromorphic chips—use fixed-function hardware to optimize computational efficiency and energy consumption. However, both FPGAs and custom accelerators may expose side-channel vulnerabilities through predictable patterns in power usage and EM emissions during intensive computation. For instance, researchers [24] demonstrated that memory access patterns in CNN accelerators can be exploited to reverse-engineer the architecture and parameters of encrypted models. Moreover, another work [25] showed that FPGA-based CNN accelerators could leak input image information through power side-channels, enabling attackers to reconstruct sensitive inputs without requiring knowledge of the model's internal parameters. These examples underscore the critical need for dedicated defense mechanisms tailored to FPGA-based and custom deep learning hardware implementations. Additionally, [26] presents a novel remote power attack on Cloud FPGAs that allows attackers to steal sensitive input data of DNN models through power side-channels. [27] introduces a method to recover inputs from neural network accelerators on FPGAs using generative CNNs, highlighting the effectiveness of power-based side-channel attacks for input recovery. [28] leverages a three-dimensional power surface to recover input images, showing the advanced capabilities of power analysis in FPGA-based DNN accelerators. Finally, [29] proposes the *NNLeak* attack, which successfully extracts full DNN models by combining power analysis with multi-stage correlation techniques, demonstrating the feasibility of these attacks on AI-oriented accelerators.

2.2 Sources of Side-Channel Leakage

SCAs exploit indirect leakages that result from hardware execution to extract sensitive information. These leakages stem from the physical or micro-architectural behavior of hardware and may vary across different platforms. This subsection categorizes the primary leakage sources exploited in attacks targeting deep learning hardware.

•**Cache Leakage** occurs when attackers exploit access patterns and timing behaviors within the shared cache hierarchy—particularly in CPUs and GPUs.

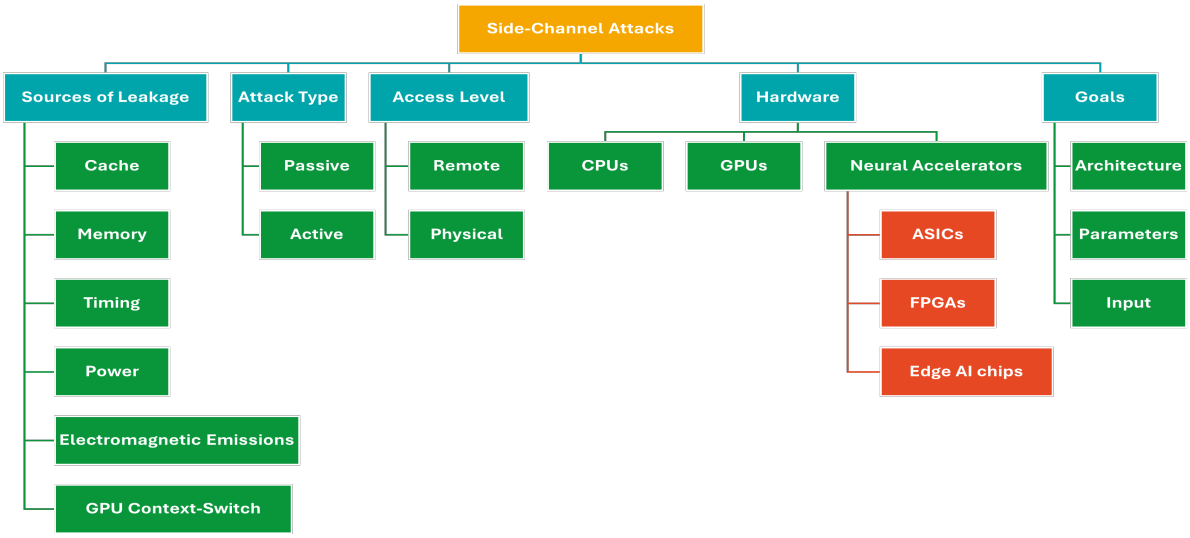

Figure 1. Overall taxonomy of hardware SCAs on deep learning models

These attacks determine whether specific memory blocks remain cached after victim execution, thereby revealing neural layer activations or tensor usage. Techniques such as *Flush+Reload* and *Prime+Probe* have been widely used to recover model architecture information. For instance, the *DeepCache* and *Cache Telepathy* and other studies demonstrate how CNN structures can be reconstructed by probing shared libraries (e.g., BLAS) during inference on shared CPU platforms [10, 12, 19, 30, 31].

•**Memory Leakage** results from observing the behavior of the physical memory subsystem, including buffer activity. Attacks in this category trace memory access patterns to extract neural network parameters. For example, [24] shows how CNN accelerators implemented on custom hardware can be reverse-engineered by analyzing memory access traces.

•**Timing Leakage** is based on measuring inference execution time under varying input conditions. Differences in latency may stem from layer types (e.g., convolution vs. pooling), branching conditions, or optimization paths. Attackers can leverage these timing variations to infer the model's depth, structural features, or even properties of the input data [12]. Additionally, [32] demonstrates a timing attack on sparsity-aware neural networks, where timing side-channel information from optimizations like skipping sparse multiplications is used to predict the model's classification outcomes, revealing sensitive architectural and input data details.

•**Power Leakage** exploits variations in power consumption during model execution. By monitoring these changes using power sensors or side-channel monitors, attackers can correlate specific patterns with operations such as multiply–accumulate (MAC). Simple power analysis (SPA) can reveal layer boundaries, while differential power analysis (DPA) can extract parameters like weights or input bits. The study in [5] demonstrates how DPA combined with power correlation techniques can recover convolutional weights.

•**EM Leakage** results from unintended radio-frequency emissions during hardware operation. These signals are strongly correlated with switching activity in circuits and can be captured using near-field probes. EM SCAs have proven especially effective in embedded architectures such as GPUs and low-power accelerators. For example, the *BarraCUDA* attack [21] utilized EM leakage from NVIDIA GPUs to extract convolutional weights, successfully reconstructing model parameters with high accuracy.

•**Context Switch Leakage** targets multi-tenant GPU systems, where multiple workloads share the GPU through time-sliced execution. When the GPU switches from one process to another, residual register states and timing artifacts may unintentionally leak sensitive information. The *Leaky DNN* attack demonstrates this by recovering architectural details and layer dimensions through delays observed during context switches in shared GPU environments [11].

2.3 Adversary Capabilities and Access Methods

This subsection categorizes threat models based on two primary dimensions: the adversary's mode of interaction with the system (passive vs. active) and their method of access (physical vs. remote). These classifications help determine which types of attacks are realistic in different deployment environments, such as cloud servers or edge devices.

•**Passive vs. Active Attacks:** In passive SCAs, the adversary silently observes unintentional leakages—such as power consumption, cache behavior, timing differences, or EM emissions—to extract sensitive information. These attacks are typically stealthy and difficult to detect. In contrast, active SCAs deliberately interfere with the system to induce or amplify information leakage. Techniques may include fault injection, clock/voltage manipulation, or thermal tampering. Active attacks can expose system states that would otherwise be inaccessible but often leave detectable traces, making them easier to identify [1, 2].

•**Physical vs. Remote Access:** Physical attacks assume that the adversary has direct access to the hardware device—such as placing a probe on a smart camera or IoT sensor, or measuring emissions from a chip in a laboratory environment. This level of access enables high-resolution monitoring of EM signals or power traces. On the other hand, remote attacks are executed entirely through software interfaces or virtual environments without requiring physical contact. These attacks are common in shared cloud infrastructures, where adversaries can exploit cache-based side channels or delays introduced by GPU context switching to extract sensitive information [1, 2].

2.4 Attack Objectives and Potential Post-Extraction Abuses

In hardware SCAs targeting deep learning models, adversaries typically aim to extract valuable internal information from neural networks—such as the model's architecture, parameters, or input data. Each of these targets holds significant value for attackers and can enable a range of subsequent malicious activities.

Attack Objectives:

• **Architecture:** Architecture extraction involves revealing the internal structure of a neural network, including the number and types of layers (e.g., convolutional, fully connected, pooling), the number of neurons or filters in each layer, activation functions, and inter-layer connections. Knowing the architecture effectively gives the adversary the model's blueprint, which facilitates further attacks such as model inversion [2, 5].

• **Parameters:** Parameters refer to the numerical weights and biases learned during training. These values directly influence how inputs are transformed at each layer and ultimately determine the model's outputs. Recovering these parameters allows the adversary to replicate the model's behavior completely [2, 5].

• **Inputs:** Inputs include the actual data used during training or inference, such as sensitive images, textual data, or private records (e.g., medical scans or financial transactions). Extracting input data poses serious privacy risks and may lead to the exposure of confidential or regulated information [2, 5].

Potential Post-Extraction Abuses:

• **Evasion:** In evasion attacks, adversaries craft adversarial examples—inputs that appear benign to humans but cause the model to produce incorrect results. For instance, a minor pixel change in a cat image could cause it to be misclassified as a dog. These attacks undermine the reliability and robustness of deployed models [33].

• **Poisoning:** Poisoning involves inserting malicious or misleading data into the training set to corrupt the learning process. This can cause the model to behave incorrectly or embed hidden backdoors. For example, attackers may inject modified stop sign images that later cause misclassification in real-world settings. This compromises the model's integrity and may lead to unsafe decisions [33].

• **Membership Inference:** Membership inference seeks to determine whether a particular data point was used in training. This can severely breach privacy, especially in domains like healthcare or finance. For example, an attacker may infer whether someone's medical record was part of the training dataset, raising compliance and ethical concerns [33].

• **Model Inversion:** Model inversion attacks aim to reconstruct input data using model outputs or internal behaviors. For instance, an attacker with access to the output of a facial recognition model might regenerate a recognizable image of a face. This poses a serious privacy risk for models trained on sensitive or biometric data [33].

• **Model Stealing:** In model stealing attacks, adversaries attempt to duplicate the neural network either by extracting its architecture and parameters or by creating an approximate replica based on observed outputs. Stolen models allow attackers to offer similar services without training costs, undermining the original developer's competitive edge and causing direct financial harm [33].

Table 1. Effect of each attack objective on corresponding post-extraction abuses [5].

Objective	Evasion	Poisoning	Membership Inference	Model Inversion	Model Stealing
Arch.	●	◐	○	○	●
Params.	●	●	●	●	●
Input	◐	◐	○	○	◐

Table 1 illustrates the relationship between dif-

ferent attack objectives and their potential post-extraction abuses. It visualizes how the extraction of architecture, parameters, or input data can enable various adversarial activities such as evasion, poisoning, membership inference, model inversion, and model stealing. The level of impact is represented using filled, half-filled, and empty circles to denote strong, moderate, or limited relevance, respectively.

2.5 Representative SCAs

This section presents a detailed overview of SCA techniques frequently reported in the literature. Specifically, we focus on two prominent categories of leakage exploited in these attacks: power and EM analysis, and cache-based analysis. These approaches have repeatedly demonstrated their effectiveness in compromising deep learning hardware.

2.5.1 Power and EM Analysis Attacks

Power consumption and EM emissions are among the most commonly exploited physical side channels. When deep learning models are executed on hardware, switching activity within logic units and memory components produces distinct power and EM patterns. Among numerous analysis techniques, SPA and DPA have emerged as two of the most effective and widely adopted methods. This section describes how each technique operates and how they are employed to extract model-level details such as architecture and parameters in the deep learning context.

• **SPA** involves direct visual inspection of a single power or EM trace to identify distinguishable operational patterns. It takes advantage of the fact that different computational operations (e.g., convolution, fully connected layers, activation functions) typically produce unique power signatures. By visually examining recorded traces, an attacker can infer specific operations executed by a neural network without the need for complex statistical techniques. For example, SPA can reveal the boundaries between layers in a neural network, such as identifying the end of convolution operations and the start of activation functions (e.g., ReLU, Softmax), simply by observing fluctuations in power during inference. This enables adversaries to reconstruct a deep learning model using minimal effort and without requiring multiple measurements.

The following steps illustrate a typical SPA attack aimed at extracting a model's architecture:

(1) *Trace Collection:* The attacker passively captures a power or EM trace during model inference on target hardware, such as an FPGA.
(2) *Visual Inspection:* The attacker visually inspects the collected trace to identify unique signal patterns or spikes. For instance, convolutional layers often show repetitive, easily distinguishable patterns due to intensive MAC operations.
(3) *Layer Identification:* Based on transitions between distinct patterns (e.g., from repetitive convolution to simpler activation), the attacker identifies layer types and boundaries.
(4) *Model Reconstruction:* Once operations and their sequence are identified, the attacker reconstructs the network's architecture—inferring the number of layers, layer types (e.g., convolution, activation, pooling), and their approximate order.

This step-by-step approach highlights SPA's simplicity and effectiveness, showing how adversaries can uncover critical architectural information using basic visual inspection techniques.

• **DPA** extends SPA by statistically correlating multiple power or EM traces with hypothesized numerical values—typically weights, biases, or inputs. Instead of analyzing a single trace, the attacker collects hundreds to millions of traces from repeated inferences on controlled, known inputs and applies statistical tests (e.g., Pearson correlation) to determine which guessed values best explain the measured leakage. Studies show that with sufficient traces and precise alignment, DPA can recover highly accurate weight values from deep learning models. A typical DPA attack targeting model weights proceeds as follows:

(1) *Trace Collection:* The attacker feeds a set of known inputs into the target hardware and records a power trace for each inference.
(2) *Segmentation and Alignment:* Each trace is segmented so that each portion aligns with a distinct MAC operation or layer.
(3) *Hypothesis Generation:* For a candidate weight value and known input, the attacker computes a predicted leakage signal.
(4) *Statistical Correlation:* Correlation between each candidate's leakage vector and real trace samples is computed; the weight with the highest correlation is assumed correct.
(5) *Iterative Recovery:* The process continues for subsequent weights, using previously recovered ones, until a full layer is reconstructed.
(6) *Validation and Refinement:* The model is re-executed with new inputs to verify if the recovered parameters yield acceptable outputs; otherwise, more traces are collected or alignment is adjusted.

Because DPA yields precise numerical parameters, it enables direct model replication and facilitates fur-

ther attacks such as membership inference or model inversion. Its primary drawbacks are the heavy trace collection overhead and the need for fine-grained synchronization. However, when these challenges are addressed, DPA stands out as one of the most powerful and threatening hardware side-channel techniques against deep learning models.

2.5.2 Cache-Based Attacks

Many hardware platforms use multi-level cache architectures. These caches often leave measurable timing footprints when lines are flushed, loaded, or reloaded. Cache-based SCAs exploit such timing leakages to infer a victim's memory access patterns without requiring direct access to their memory. This subsection outlines three widely adopted techniques used to exploit caches, each accompanied by a step-by-step explanation to illustrate its operation in practice. As illustrated in Figure 2, a typical cache-based side-channel scenario involves a co-located attacker core and a victim core that contend for a shared LLC. By priming, flushing, or probing specific cache sets, the attacker distinguishes cache hits from misses and thereby infers the victim's memory-access footprint.

•**Flush + Reload:**This technique applies when the attacker and victim share memory pages (e.g., shared libraries). It uses the `clflush` instruction to flush cache lines and measures reload time to determine if the victim accessed them [34].The main steps of this attack are:

(1) *Shared memory mapping:* The attacker maps a memory region shared with the deep learning model.
(2) *Flush:* Specific cache lines are flushed using `clflush`.
(3) *Victim execution:* The model may reload those lines.
(4) *Reload and timing:* The attacker reloads the lines and measures access latency.
(5) *Analysis:*
   - Low latency ⇒ victim used the line.
   - High latency ⇒ line was unused.
(6) *Repeat:* To reconstruct the model's memory access pattern.

• **Prime + Probe:**Unlike Flush + Reload, this technique does not require shared memory. It fills a cache set (prime), waits for the victim to execute, then re-accesses the same set (probe) to measure eviction [10]. General steps of this attack are:

(1) *Prime:* The attacker fills cache sets using their own data.
(2) *Victim execution:* May evict the attacker's data.
(3) *Probe:* The attacker re-accesses the same addresses.
(4) *Analysis:*
   - High latency ⇒ victim accessed that cache set.
   - Low latency ⇒ victim did not access it.
(5) *Repeat:* To build a temporal access map of the model.

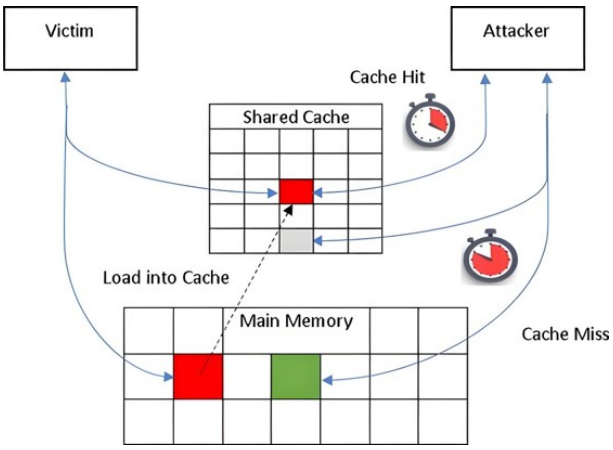

Figure 2. Generic overview of cache side-channel attacks [35].

3 Hardware SCAs on Deep Learning models Case Studies

This section provides a comprehensive review of recent Hardware SCAs targeting deep learning models. We analyze a selection of representative attacks published in the literature and categorize them based on their underlying leakage sources: power and EM emissions, cache and timing behaviors, memory access patterns, and context-switching mechanisms in GPUs. Each subsection presents a focused analysis of impactful studies, discussing their threat models, attack goals, methodologies, and key outcomes. The reviewed works highlight how various side-channel techniques can effectively extract sensitive information from deep learning systems and underscore the pressing need for robust countermeasures.

3.1 Power and EM-Based Attacks

Power and EM SCAs exploit physical signals emitted by hardware during computation to extract sensitive information about deep learning models. These signals, directly stemming from electrical activity during neural network execution, can reveal detailed insights about a model's architecture, parameters, or even its input data. The accuracy of such attacks largely depends on factors such as measurement precision, alignment of recorded traces, and noise reduction techniques. Despite the deployment of both software- and hardware-level protections, modern deep learning platforms—including GPUs and FPGAs—remain vulnerable to these forms of side-

channel leakage [5, 21, 25, 36]. In what follows, we review two representative studies that demonstrate the feasibility of recovering neural network internals via power and EM side channels.

3.1.1 SoK: Extracting Neural Network Secrets via Physical Side Channels

An attack [5] that focuses on side-channel leakage arising from power and EM emissions with the goal of recovering confidential information from deep neural networks, including model architecture, trained parameters (weights and biases), and sensitive inputs. The attacks are evaluated on a commercial FPGA-based accelerator from Xilinx.

Attack Methodology:

• Architecture Extraction (via SPA):

(1) *Trace Collection:* Power and EM signals are recorded during the neural network's execution on known inputs.
(2) *Layer Identification:* Using SPA, distinct patterns for each layer type are identified:
- Convolution: repetitive patterns with characteristic peaks.
- Activation functions such as ReLU (sharp spikes), Softmax (smooth curves), Tanh, and Sigmoid (distinct power profiles).

(3) *Layer Counting:* For simpler models, the number of layers can be deduced from trace repetitions. This becomes harder on parallel hardware like FPGAs.
(4) *Hyperparameter Estimation:* Execution time patterns and model type allow inference of kernel sizes, strides, and neuron counts.

•Parameter Recovery (via DPA):

(1) *Assumptions:* The attacker knows the model architecture and can supply chosen inputs.
(2) *Weight Guessing:* Candidate values (e.g., 16-bit precision) are iteratively tested.
(3) *Power Simulation:* Theoretical traces are generated based on weight guesses.
(4) *Correlation Analysis:* Pearson correlation between simulated and real traces identifies correct weights.
(5) *Error Propagation Issue:* Early weight errors distort deeper layers, making high accuracy critical. Even minor noise can cause significant accuracy degradation.

• Input Recovery (via DPA and Signal Analysis):

(1) *DPA-Based:*
- Requires known architecture and parameters.
- Input values (e.g., pixel intensities) are guessed, simulated, and matched to measured traces.

(2) *Model-Free Classification:*
- EM traces from known input classes are used to train a classifier.
- New inputs are classified based solely on trace features; for MNIST, accuracy reaches approximately 88

As summarised in Figure 3, the attack unfolds in three main stages: architecture extraction, parameter recovery, and input reconstruction. This attack is especially effective against quantized and binary neural networks due to the smaller search space for weight values.

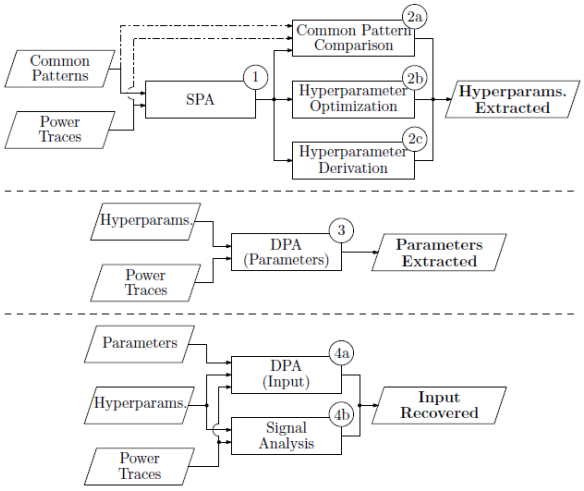

Figure 3. Overview of the multi-stage SCA pipeline [5].

3.1.2 BarraCUDA: Electromagnetic Leakage from GPUs Reveals Neural Network Parameters

BarraCUDA [21] presents a SCA based on EM leakage targeting neural networks deployed on NVIDIA Jetson Nano GPUs. The primary goal is to extract sensitive model parameters, including weights and biases.

Attack Methodology:

This attack uses Correlation EM Analysis (CEMA), a variant of DPA, to incrementally recover convolutional layer parameters:

(1) *Threat Model:* The attacker has physical access to the device, knows the model architecture, and can feed chosen inputs.
(2) *Measurement Setup:*
- An EM probe is precisely placed near the GPU core.

- Around 15 million traces are recorded over 10 days using repeated inferences with known inputs.

(3) *Leakage Identification (via TVLA):*
- Statistical Test Vector Leakage Assessment (TVLA) pinpoints locations in the EM traces where weights or biases influence the signal.

(4) *Weight and Bias Extraction (via CEMA):*
- Inputs are known; possible weight values are hypothesized.
- For each guess, expected EM traces are simulated.
- Pearson correlation between simulated and real traces reveals the correct parameters.

The results reveal that even commercial, closed-source GPU platforms (like NVIDIA's TensorRT) are susceptible to advanced SCAs based on EM leakage.

3.2 Cache and Timing-Based Attacks

Cache based SCAs exploit observable timing variations and memory access patterns during the execution of deep learning models on CPUs and GPUs. By carefully analyzing the behavior of shared hardware resources—especially the cache hierarchy in CPUs—these attacks can infer sensitive information about the model, including its architecture and layer configurations. Such attacks are particularly effective in shared computing environments, such as cloud infrastructures, where attackers often share physical resources with victims [10, 12, 19, 30, 31]. Below, we review two representative works that successfully demonstrate the extraction of neural network details via cache and timing behaviors.

3.2.1 Cache Telepathy: Leveraging Shared Resource Attacks to Extract Deep Neural Network Architectures

The study in [10] focuses on recovering the architecture of deep neural networks in shared environments, such as cloud services and MLaaS platforms. The attacker runs on the same CPU as the victim and shares cache resources, enabling effective exploitation of cache side channels. The attack, known as Cache Telepathy, combines two classical cache side-channel techniques, Prime+Probe and Flush+Reload, to gradually extract structural details of the target network. It is demonstrated on popular architectures like VGG-16 and ResNet-50, using linear algebra libraries such as OpenBLAS and Intel MKL.

Attack Methodology:

(1) *Inferring Matrix Multiplication Dimensions*:
- GEMM operations underpin the execution of most neural layers.
- By monitoring function calls like `itcopy`, `oncopy`, and `kernel`, the attacker can infer matrix dimensions involved in each layer.
- Example: if matrices A $(m \times k)$ and B $(k \times n)$ produce C $(m \times n)$, repeated access patterns reveal m, n, and k.

(2) *Mapping Dimensions to Network Parameters*:
- Fully-connected layers: matrix dimensions reveal neuron counts.
- Convolutional layers: dimensions are mapped to kernel size, padding, and stride using known relationships.
- Example: A recovered filter matrix of size 64×576 with input depth $D_i = 64$ implies a 3×3 kernel, since $\sqrt{576/64} = 3$.

(3) *Reconstructing Network Architecture*: Sequential analysis of GEMM calls and their constraints enables layer-by-layer recovery, including shortcut connections.

Cache Telepathy significantly reduces the architecture search space; for instance, the search space for VGG-16 drops from 5.4×10^{12} to just 16 candidates, and for ResNet-50 from 6×10^{46} to only 512 possibilities.

3.2.2 DeepCache: Revisiting Cache SCAs on Optimized Deep Neural Network Executables

Another study [19] that targets cache-based side-channel attacks against optimized executable files of deep neural networks, aims to infer model architecture remotely in shared cloud settings, even when the attacker has no access to shared libraries or model code. DeepCache addresses the challenges posed by compiler-level optimizations that reduce the effectiveness of traditional cache attacks.

Attack Methodology:

(1) *Collecting Cache Traces via Prime+Probe*:
- By repeatedly executing Prime+Probe, the attacker collects binary traces representing cache access over time.

(2) *Segmenting Traces and Identifying Layer Boundaries*:
- Using Autoencoder + Conv-LSTM (Long Short-Term Memory), the method detects significant shifts in cache patterns, which indicate transitions between layers.

(3) *Extracting Features from Segmented Traces*:
- Each trace segment is encoded into a numeric vector via contrastive learning.
- Similar layers yield similar feature vectors; dissimilar layers yield distinct ones.

(4) *Inferring Layer Types and Parameters*:

- The extracted vectors are compared against a reference database to determine the layer type (e.g., convolution, pooling, fully-connected) and key parameters like kernel size and stride.

DeepCache reveals that compiler optimizations, while improving performance, introduce regular patterns that can still be exploited. These findings highlight a critical vulnerability in how optimized executables are deployed, posing serious risks to model intellectual property. The complete DeepCache leakage–recovery pipeline is shown in Figure 4.

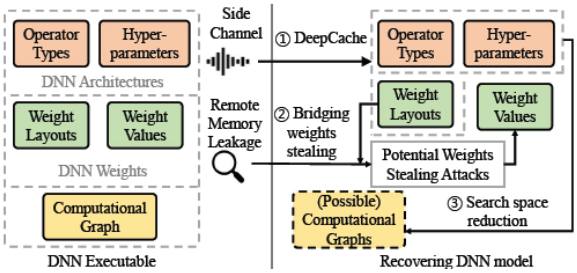

Figure 4. DeepCache pipeline [19].

3.3 Memory Access Pattern-Based Attacks

Memory access pattern-based attacks analyze how deep learning models interact with main memory such as DRAM to extract sensitive information including model architecture, layer dimensions, and data layout. These attacks do not require direct access to memory contents; rather, they leverage observable temporal or spatial memory access patterns to reverse-engineer the internal structure of the model. Below, we review a representative work that demonstrates the practical feasibility of recovering neural network architectures and parameters via memory behavior.

3.3.1 Reverse Engineering Convolutional Neural Networks through Side-Channel Information Leakage

The study in [24] introduces a novel attack on CNNs deployed on custom accelerators. The main objective is to reconstruct the confidential network architecture and trained parameters (such as weights), even when data remains fully encrypted in memory. The attack utilizes side-channel leakage from memory access behavior in two distinct stages: structure recovery and weight extraction.

Attack Methodology:

- **Reconstructing the Network Structure:**

(1) *Identifying Layer Boundaries via RAW Dependency:* During CNN execution, the output feature map (OFM) of a layer is first written to memory, then immediately read as the input feature map (IFM) by the next layer. By tracking such write-then-read transitions at the same memory address, the attacker can identify layer boundaries.

(2) *Classifying Accessed Data Types:* Each CNN layer involves three types of data with distinct memory behaviors:
- Filter weights – read-only, constant throughout.
- Input feature maps – read at the start of a layer.
- Output feature maps – primarily written, rarely read in the same layer.

The attacker differentiates these based on observed read/write access patterns.

(3) *Estimating Layer Dimensions:* These data blocks (IFMs, OFMs, filters) are stored contiguously in memory. By observing memory access granularity and continuity, the attacker estimates their sizes and thereby infers layer-wise dimensional parameters.

(4) *Solving Structural Equations:* With IFM and OFM dimensions known, the attacker formulates equations describing their relationship with filter size, stride, and padding. Solving these yields precise structural details of each layer.

(5) *Refining with Execution Time Analysis:* By measuring the runtime of each layer and leveraging the correlation between execution time and the number of MAC operations, the attacker filters out implausible layer configurations to converge on the most accurate structure.

- **Recovering Weights via Zero Pruning-Based Inference:**

(1) *Analyzing Pruned Output Patterns:* Many networks skip storing outputs that evaluate to zero to optimize inference. By crafting inputs and tracking transitions from zero to non-zero outputs, the attacker obtains clues about the weight-to-bias ratio.

(2) *Locating Zero-Crossing Points via Binary Search:* The attacker incrementally perturbs inputs to pinpoint exact thresholds where outputs transition from zero to non-zero. Each transition yields a constraint equation linking weights and biases.

(3) *Inferring Weight-to-Bias Ratios:* By aggregating multiple such equations, the attacker can deduce the ratio of each weight to its corresponding bias, significantly narrowing the uncertainty about the model parameters.

The study demonstrates that using access patterns alone, adversaries can reconstruct networks like AlexNet and SqueezeNet with high fidelity—reducing the architectural search space for AlexNet from an intractable number to just 24 candidates. Moreover, weight-to-bias ratios were inferred with remarkable precision using zero-crossing analysis. These findings highlight a critical threat: even with encrypted data, off-chip memory access patterns can leak sufficient information to compromise the intellectual property and integrity of deep learning models. This underscores the need for countermeasures such as access obfuscation or randomization.

3.4 Context Switch-Based Attacks on GPUs

Context switch-based SCAs in GPUs exploit residual information generated during task-switching events in multi-tenant environments to extract sensitive insights about deep learning models. In shared infrastructures such as MLaaS, multiple users may concurrently utilize the same GPU. During context switching, the GPU performs task scheduling and saves/restores execution states, potentially leaking information such as layer execution sequences, input/output dimensions, and operation types [1, 11]. This subsection examines a representative study that empirically demonstrates how such leakages can be exploited to infer a model's architecture.

3.4.1 Leaky DNN: Stealing Deep Learning Model Secrets via GPU Context-Switch Side Channels

The study in [11] introduces a novel SCA designed to extract confidential information from deep neural networks executing on GPUs. The core idea is that in shared environments—such as public clouds—an attacker can utilize context switch-induced latency artifacts to infer detailed structural and hyperparameter-level information, including the types and sequence of layers.

Attack Methodology:

(1) *Threat Model*:
- The attacker and victim share the same GPU instance in a cloud environment.
- The attacker has no access to the victim's model or code but can execute custom CUDA kernels on the shared GPU.

(2) *Attack Slowdown via Spy Kernels*:
- The attacker executes multiple lightweight spy kernels concurrently to increase the execution time of the victim's layers.
- This extended execution window allows for higher-resolution sampling of context-switch latency, improving attack accuracy.

(3) *Trace Collection and Analysis*:
- Using the CUPTI profiler, the attacker collects timing traces related to context switch overhead.
- These traces encode indirect clues about the victim model's structure and layer specifications.

(4) *Layer Classification Using LSTM Models*:
- To identify layer types (e.g., convolutional, fully connected, pooling), the attacker trains LSTM models, well-suited for analyzing sequential time-series data.
- The collected latency traces are input to the trained models for classification.

(5) *Hyperparameter Inference with Dedicated LSTM Models*:
- Separate LSTM networks are trained to infer specific hyperparameters, including neuron counts, kernel sizes, number of filters, and strides.

(6) *Error Correction Using Structural Priors*:
- The attacker applies common architectural design rules (e.g., convolution layers typically precede pooling layers) to refine layer sequences and correct misclassifications.

The results show that the proposed attack can successfully reconstruct the architectures of well-known models like ZFNet and VGG16 with high precision. These findings underscore a critical vulnerability in multi-tenant GPU environments and highlight the need for effective defensive mechanisms to preserve model intellectual property in such contexts.

4 Defensive Strategies Against SCAs

This section reviews major defense mechanisms and detection frameworks proposed to mitigate hardware SCAs targeting deep learning models. For each class of attack—including power-based, EM, and cache-based threats—we discuss specific countermeasures implemented at both hardware and software levels. Additionally, we review a runtime detection technique aimed at identifying cache-based side-channel threats.

To provide a concise overview, Table 2 summarizes the main defense mechanisms discussed in this section. Each method is mapped to the type of SCA it targets, along with the associated performance, power, or area overheads when known. This structured comparison highlights the trade-offs between security and efficiency, and illustrates how certain countermeasures—such as hiding/balanced logic and dummy operations—extend protection beyond power, EM, and cache-based leakages to timing and context-switching channels as well.

Table 2. Mapping of Defense Mechanisms to Attack Types and Their Overheads

Defense Mechanism	Targeted Attack Type(s)	Overhead (Area/Power/Latency/Accuracy Impact)
Masking	Power, EM	Latency ↑, moderate power overhead
Hiding and Balanced Logic	Power, EM, Timing, Context Switching	Area ↑, latency ↑
Randomized Execution / Shuffling	Timing, Cache-based	Latency ↑, power ↑; entropy ↑ but throughput ↓
Noise Injection	Power, EM	Power ↑, may reduce efficiency
Physical Shielding / Filtering	Power, EM	Area ↑, cost ↑; minimal latency but expensive
Cache Partitioning	Cache-based	Memory overhead ↑, performance ↓ in shared systems
Cache Randomization / Line Remapping	Cache-based	Latency ↑, moderate slowdown
Dummy Operations / Decoy Accesses	Cache-based, Timing, Context Switching	Execution time ↑, high computational overhead
Runtime Detection	Cache-based	Moderate latency depending on monitoring intensity

4.1 Countermeasures Against Power and EM Leakage

Power and EM SCAs exploit physical emissions generated during model execution to infer sensitive information, such as architecture details or model weights. To mitigate such leakages, researchers have proposed various techniques designed to obfuscate the correlation between internal computations and observable physical signals. Key approaches include:

•**Masking:** This technique involves randomizing sensitive data—such as weights or intermediate outputs—by combining them with random masks. As a result, power or EM leakage becomes statistically independent from actual data values. These masks are removed after computation to restore correct outputs. Masking is especially feasible for digital arithmetic operations, such as multiply–accumulate units. For example, the approach in [37] applies masking to protect the internal states of deep learning models. Additionally, [38] presents a masked hardware accelerator for feed-forward NNs with fixed-point arithmetic, protecting against side-channel analysis through hardware-level masking.

• **Hiding and Balanced Logic:** This approach modifies hardware designs so that power consumption and EM emissions become data-independent. In balanced logic implementations, a fixed number of bits switch in each operation, producing uniform power patterns across different inputs. Such uniformity eliminates the attacker's ability to correlate leakage with specific computations.

• **Randomized Execution and Shuffling:** By randomizing the execution order of operations—such as the arrangement of neurons or filters—the power trace associated with each inference becomes unpredictable. This increased entropy disrupts the attacker's ability to align traces with specific layers or input features, thereby mitigating inference-based extraction [39].

•**Noise Injection:** Here, artificial noise is deliberately added to power consumption or EM signals to obscure the original signal. This noise may be random, periodic, or continuous. Although simple and effective in practice, noise injection increases energy consumption. In [40], a ring oscillator is used to inject noise and protect model computations.

• **Physical Shielding and Filtering:** These methods involve physical modifications to the hardware. For instance, shielding layers (e.g., metal enclosures or absorbing materials) can prevent EM emissions from propagating externally. Similarly, analog filters may be employed to smooth out transient variations in power signals. These techniques are commonly used in military and industrial applications where security requirements are stringent.

4.2 Countermeasures Against Cache-Based Leakage

Cache-based SCAs exploit timing behaviors and memory access patterns within multi-level cache hierarchies (e.g., L2, LLC) to infer sensitive model information. Several architectural and system-level defenses have been proposed to address these vulnerabilities:

• **Cache Partitioning:** In this method, cache lines are partitioned and assigned to different processes or virtual machines to prevent cross-core eviction. This isolation ensures that the attacker cannot observe cache usage by the victim, thereby disrupting attacks such as Prime+Probe [41].

•**Cache Randomization and Line Remapping:** This approach alters the mapping between memory

addresses and cache sets dynamically at runtime. By preventing fixed memory-to-cache correspondence, it becomes significantly harder for attackers to predict or track victim access patterns, thereby neutralizing attacks like Flush+Reload and Prime+Probe [41].

•**Dummy Operations and Decoy Accesses:** These countermeasures insert non-functional memory accesses or spurious instructions into program execution to distort cache traces. For example, random loads/stores, loop repetitions, or dummy function calls can obfuscate the real memory usage of the model, misleading the attacker and increasing their analysis complexity [30].

While these defense mechanisms can significantly reduce side-channel risks, they often introduce substantial overhead that can negatively impact system performance. Therefore, such countermeasures should be carefully designed and deployed to balance security and efficiency without compromising the overall functionality of the system.

4.3 Runtime Detection of Cache-Based SCAs

Beyond architectural countermeasures, runtime detection techniques provide an additional layer of defense by monitoring system behavior during execution. These methods aim to identify anomalous patterns indicative of SCAs and respond in real-time. For instance, Profiling tools such as `perf` (a standard Linux utility) are commonly used to monitor low-level hardware events including `cache-misses`, `cache-references`, `branch-mispredictions`, and `instructions`. These performance counters offer insight into cache-related behavior and can expose attack patterns such as those seen in Flush+Reload or Prime+Probe attacks [42–44]. A general workflow of run-time detection mechanisms includes:

- *Performance Monitoring:* System-level metrics are collected periodically or after specific code blocks using profiling tools like `perf`.
- *Feature Extraction:* Features such as cache miss rates, LLC access frequency, or load/store ratios are extracted to capture cache behavior.
- *Machine Learning Classification:* Classifiers such as Support Vector Machines (SVMs), Random Forests, or Neural Networks are trained to distinguish between normal and attack-induced behavior.
- *Online Detection and Response:* The trained model is deployed for real-time monitoring. Upon detecting suspicious activity, the system can trigger alerts, log incidents, or terminate compromised processes.

Although most existing runtime detection approaches have focused on cryptographic scenarios (e.g., AES or RSA), their principles are readily applicable to machine learning contexts—especially in multi-tenant or cloud environments where model co-location is common. Extending these detection strategies to protect IP in Deep learning models represents a promising avenue for future work.

5 Future Research Directions

In recent years, SCAs targeting deep learning hardware have received increasing attention. Nevertheless, several important research gaps remain unaddressed. This section outlines key future research directions that can enhance our understanding of these threats and facilitate the development of more robust defense mechanisms.

• **Attacks on Emerging Deep Learning Architectures:** While most existing studies have focused on CNNs, modern architectures such as Transformers and LSTMs—used widely in natural language processing, time-series analysis, and computer vision—remain largely unexplored in the context of side-channel vulnerabilities. Moreover, scenarios involving Federated Learning and Distributed Learning, where models are trained or deployed across multiple devices, present new challenges for leakage analysis and require deeper investigation.

•**Hybrid Hardware–Software Attack Surfaces and Defense Strategies:** Existing research on SCAs has largely focused on either hardware-based or software-based perspectives in isolation. However, in practical scenarios, attackers often exploit a combination of leakage vectors—such as timing characteristics, cache activity, power usage, and software-level cues (e.g., shared libraries). This highlights the need for integrated frameworks capable of analyzing and mitigating threats that span both hardware and software layers. Advancing such cross-layer defense mechanisms remains a key direction for future investigation.

• **Robust Model Watermarking Techniques:** With the increasing adoption of MLaaS platforms, the protection of model IP has become more urgent. Model watermarking—embedding digital signatures within model parameters or designing specific responses to secret trigger inputs—offers a promising means to assert ownership. However, existing watermarking schemes have rarely been evaluated under SCA scenarios. Developing watermarking techniques that are resilient to power, cache, or timing-based leakage would significantly strengthen IP protection in machine learning systems.

6 Conclusion

This report has shown that deep learning models remain vulnerable to a broad range of hardware SCAs. These attacks exploit physical indicators such as power consumption, cache access patterns, and timing fluctuations to reconstruct model architectures, extract weights, and even recover sensitive inputs. A review of recent research reveals that such threats are effective not only on CPUs and GPUs but also on custom hardware accelerators. Although a variety of defense strategies have been developed, significant gaps still remain in achieving comprehensive protection. As such, the development of robust and adaptive defense mechanisms at both hardware and software levels is essential to safeguarding the security and intellectual property of deep learning systems.

References

[1] Y. Liu and et al. A Survey on Side-Channel-based Reverse Engineering Attacks on Deep Neural Networks. In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 312–315. IEEE, June 2022. .

[2] M. Méndez Real and R. Salvador. Physical Side-Channel Attacks on Embedded Neural Networks: A Survey. *Applied Sciences*, 11(15):6790, July 2021. .

[3] M. Isakov, V. Gadepally, K. M. Gettings, and M. A. Kinsy. Survey of Attacks and Defenses on Edge-Deployed Neural Networks. *arXiv*, 2019. .

[4] D. Meyer. The Cost of Training AI Could Soon Become Too Much to Bear. *Fortune*, April 2024. https://fortune.com/2024/04/04/ai-training-costs-how-much-is-too-much-openai-gpt-anthropic-microsoft/.

[5] P. Horváth, D. Lauret, Z. Liu, and L. Batina. SoK: Neural Network Extraction Through Physical Side Channels. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, SEC '24, USA, 2024. USENIX Association. .

[6] M. Sweney and D. Milmo. OpenAI 'reviewing' allegations that its AI models were used to make DeepSeek. *The Guardian*, January 2025. https://www.theguardian.com/technology/2025/jan/29/openai-chatgpt-deepseek-china-us-ai-models.

[7] S. Mittal, H. Gupta, and S. Srivastava. A Survey on Hardware Security of DNN Models and Accelerators. *Journal of Systems Architecture*, 117:102163, 2021. .

[8] S. Picek, G. Perin, L. Mariot, L. Wu, and L. Batina. SoK: Deep Learning-based Physical Side-channel Analysis. *ACM Computing Surveys*, 55(11):1–35, 2023. .

[9] T. Nayan, Q. Guo, M. A. Duniawi, M. Botacin, S. Uluagac, and R. Sun. SoK: All You Need to Know About On-device ML Model Extraction - The Gap Between Research and Practice. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, pages 1–18. USENIX Association, 2024. .

[10] M. Yan, C. W. Fletcher, and J. Torrellas. Cache Telepathy: Leveraging Shared Resource Attacks to Learn DNN Architectures. In *Proceedings of the 29th USENIX Conference on Security Symposium*, SEC'20, USA, 2020. USENIX Association. .

[11] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. Al Faruque. Leaky DNN: Stealing Deep-Learning Model Secret with GPU Context-Switching Side-Channel. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 125–137. IEEE, June 2020. .

[12] Y. Liu and A. Srivastava. GANRED: GAN-based Reverse Engineering of DNNs via Cache Side-Channel. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pages 41–52. ACM, Nov. 2020. .

[13] Z. Gao, J. Hu, F. Guo, Y. Zhang, Y. Han, S. Liu, H. Li, and Z. Lv. I Know What You Said: Unveiling Hardware Cache Side-Channels in Local Large Language Model Inference (Version 3). *arXiv*, 2025. .

[14] A. Adiletta and B. Sunar. Spill The Beans: Exploiting CPU Cache Side-Channels to Leak Tokens from Large Language Models (Version 1). *arXiv*, 2025. .

[15] X. Hu and 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. ACM, Mar. 2020. .

[16] Y. Gao, H. Qiu, Z. Zhang, B. Wang, H. Ma, A. Abuadbba, M. Xue, A. Fu, and S. Nepal. DeepTheft: Stealing DNN Model Architectures through Power Side Channel. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3311–3326. IEEE, 2024. .

[17] A. Chaudhuri, S. Shukla, S. Bhattacharya, and D. Mukhopadhyay. "Energon": Unveiling Transformers from GPU Power and Thermal Side-Channels (Version 1). *arXiv*, 2025. .

[18] K. Lee, M. Ashok, S. Maji, R. Agrawal, A. Joshi, M. Yan, J. S. Emer, and A. P. Chandrakasan. Secure Machine Learning Hardware: Challenges and Progress [Feature]. *IEEE Circuits and Systems Magazine*, 25(1):8–34, 2025. .

[19] Z. Liu, Y. Yuan, Y. Chen, S. Hu, T. Li, and

S. Wang. DeepCache: Revisiting Cache Side-Channel Attacks in Deep Neural Networks Executables. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, pages 4495–4508. ACM, Dec. 2024. .

[20] H. Wang, S. M. Hafiz, K. Patwari, C.-N. Chuah, Z. Shafiq, and H. Homayoun. Stealthy Inference Attack on DNN via Cache-based Side-Channel Attacks. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1515–1520. IEEE, Mar. 2022. .

[21] P. Horvath, L. Chmielewski, L. Weissbart, L. Batina, and Y. Yarom. BarraCUDA: Edge GPUs Do Leak DNN Weights. *arXiv preprint*, Dec. 2023. https://arxiv.org/abs/2312.07783.

[22] Y. Sun, G. Jiang, X. Liu, P. He, and S.-K. Lam. Layer Sequence Extraction of Optimized DNNs Using Side-Channel Information Leaks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(10):3102–3115, 2024. .

[23] S. B. Dutta, H. Naghibijouybari, A. Gupta, N. Abu-Ghazaleh, A. Marquez, and K. Barker. Spy in the GPU-box: Covert and Side Channel Attacks on Multi-GPU Systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23)*, pages 1–13. ACM, 2023. .

[24] W. Hua, Z. Zhang, and G. E. Suh. Reverse Engineering Convolutional Neural Networks Through Side-channel Information Leaks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, June 2018. .

[25] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. 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. ACM, Dec. 2018. .

[26] S. Tian, S. Moini, D. Holcomb, R. Tessier, and J. Szefer. A Practical Remote Power Attack on Machine Learning Accelerators in Cloud FPGAs. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023. .

[27] L. Huegle, M. Gotthard, V. Meyers, J. Krautter, D. R. E. Gnad, and M. B. Tahoori. Power2Picture: Using Generative CNNs for Input Recovery of Neural Network Accelerators through Power Side-Channels on FPGAs. In *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 155–161. IEEE, 2023. .

[28] L. Wu, L. Wu, Z. Ba, and X. Zhang. An Input Recovery Side-Channel Attack on DNN Accelerator with Three-Dimensional Power Surface. In *2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 1–11. IEEE, 2025. .

[29] Y. Gao, H. Ma, M. Yan, J. He, Y. Zhao, and Y. Jin. NNLeak: An AI-Oriented DNN Model Extraction Attack through Multi-Stage Side Channel Analysis. In *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6. IEEE, 2023. .

[30] S. Hong and et al. Security Analysis of Deep Neural Networks Operating in the Presence of Cache Side-Channel Attacks. *arXiv preprint*, 2018. https://doi.org/10.48550/ARXIV.1810.03487.

[31] S. Hong, M. Davinroy, Y. Kaya, D. Dachman-Soled, and T. Dumitraş. How to 0wn NAS in Your Spare Time. *arXiv preprint*, 2020. . https://doi.org/10.48550/ARXIV.2002.06776.

[32] S. Maji, K. Lee, and A. P. Chandrakasan. SparseLeakyNets: Classification Prediction Attack Over Sparsity-Aware Embedded Neural Networks Using Timing Side-Channel Information. *IEEE Computer Architecture Letters*, 23(1):133–136, 2024. .

[33] G. Wang, C. Zhou, Y. Wang, B. Chen, H. Guo, and Q. Yan. Beyond Boundaries: A Comprehensive Survey of Transferable Attacks on AI Systems. *arXiv preprint*, 2023. https://doi.org/10.48550/ARXIV.2311.11796.

[34] J. Sharma, S. S. Ojha, and R. Dubey. Exploring Flush+Reload Side Channel Attack Vulnerabilities: Detection and Countermeasures. In *2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS)*, volume 17, pages 717–723. IEEE, 2023. .

[35] A. Albalawi. On Preventing and Mitigating Cache Based Side-Channel Attacks on AES System in Virtualized Environments. *Computer and Information Science (CIS)*, 17(1):9, Feb. 2024. .

[36] L. Batina, S. Bhasin, D. Jap, and S. Picek. CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, pages 515–532, USA, 2019. USENIX Association. .

[37] A. Dubey, R. Cammarota, and A. Aysu. MaskedNet: The First Hardware Inference Engine Aiming Power Side-Channel Protection. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 197–208. IEEE, Dec. 2020. .

[38] M. Brosch, M. Probst, M. Glaser, and G. Sigl. A Masked Hardware Accelerator for Feed-Forward

Neural Networks With Fixed-Point Arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 32(2):231–244, 2024. .

[39] Q. Fang, L. Lin, H. Zhang, T. Wang, and M. Alioto. Voltage Scaling-Agnostic Counteraction of Side-Channel Neural Net Reverse Engineering via Machine Learning Compensation and Multi-Level Shuffling. In *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, June 2023. .

[40] X. Yan, C. H. Chang, and T. Zhang. Defense Against ML-based Power Side-Channel Attacks on DNN Accelerators with Adversarial Attacks. *arXiv preprint*, Dec. 2023. https://arxiv.org/abs/2312.04035.

[41] N. Lungu, B. B. Dash, M. R. Mishra, L. Barik, A. Tripathy, and S. S. Patra. GPUSecBench: Evaluating the Cache Side-Channel Resilience of a GPU Security Execution Pipeline. In *2024 Second International Conference on Intelligent Cyber Physical Systems and Internet of Things (ICoICI)*, pages 564–571. IEEE, 2024. .

[42] T. Joshi, A. Kawalay, A. Jamkhande, and A. Joshi. Hybrid Deep Learning Model for Multiple Cache Side Channel Attacks Detection: A Comparative Analysis. *arXiv preprint*, Jan. 2025. https://arxiv.org/abs/2501.17123.

[43] H. Wang, H. Sayadi, S. Rafatirad, A. Sasan, and H. Homayoun. SCARF: Detecting Side-Channel Attacks at Real-time using Low-level Hardware Features. In *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, July 2020. .

[44] H. Kim, C. Hahn, H. J. Kim, Y. Shin, and J. Hur. Deep Learning-Based Detection for Multiple Cache Side-Channel Attacks. *IEEE Transactions on Information Forensics and Security*, 19:1672–1686, 2024. .

Zahra Mohammadi is currently a Ph.D. candidate in Computer Engineering at the University of Tehran, specializing in Computer Architecture. Her research primarily focuses on detecting sleep disorders through advanced analysis of biomedical signals, utilizing machine learning and deep learning techniques. In addition, she actively investigates hardware side-channel attacks on deep learning models, with an emphasis on understanding and mitigating hardware-level vulnerabilities. She received her M.S. in Computer Engineering from the University of Tehran in 2023, where she ranked first among all graduates, and her B.S. in Computer Engineering from Shahid Beheshti University in 2020.

Mona Hashemi completed her Ph.D. at the University of Tehran, Iran, in 2025. She received her M.S. and B.S. degrees in Computer Engineering from Sharif University of Technology and K. N. Toosi University of Technology in 2016 and 2012, respectively. During her Ph.D., she was a visiting student at the National University of Singapore (NUS), where she is currently a postdoctoral researcher. Her research interests include Very Large Scale Integration Systems (VLSI), Hardware Security and Trust, as well as Efficient and Dependable Computing.

Siamak Mohammadi received his BSc, MSc and PhD degrees from the University of Paris Sud Orsay, France in 1990, 1992 and 1996, respectively, all in electrical engineering. From 1997 to 1999 he was a Research Associate with the Department of Computer Science, University of Manchester, England. In 1999 he moved to Canada and worked in Semiconductor industry in Toronto until 2005. Currently he is an Associate Professor in School of Electrical and Computer engineering, at the University of Tehran, Iran. He has over 30 years of experience in Low Power Design , Verification, and Hardware Security of Embedded Systems.