Peering Inside the Black-Box: Long-Range and Scalable Model Architecture Snooping via GPU Electromagnetic Side-Channel

Rui Xiao[§], Sibo Feng[§], Soundarya Ramesh[¶], Jun Han[‡*], and Jinsong Han[§*]
[§]Zhejiang University, [¶]National University of Singapore, [‡]KAIST
{ruixiao24, sibofeng}@zju.edu.cn, sramesh@comp.nus.edu.sg, junhan@cyphy.kaist.ac.kr, hanjinsong@zju.edu.cn
*Corresponding authors

***Abstract*—As deep neural networks (DNNs) are increasingly adopted in safety-critical applications such as autonomous driving and face recognition, they have also become targets for adversarial attacks. However, confidential information of DNNs – including *model architecture* – is typically hidden from attackers. As a result, adversarial attacks are often launched in black-box settings, which limits their effectiveness. In this paper, we propose *ModelSpy*, a stealthy DNN architecture snooping attack based on GPU electromagnetic (EM) leakage. *ModelSpy* is capable of extracting complete architecture from several meters away, even through walls. *ModelSpy* is based on the key observation that GPU emanates far-field EM signals that exhibit architecture-specific amplitude modulation during DNN inference. We develop a hierarchical reconstruction model to recover fine-grained architectural details from the noisy EM signals. To enhance scalability across diverse and evolving architectures, we design a transfer-learning scheme by exploiting the correlation between external EM leakage and internal GPU activity. We design and implement a proof-of-concept system to demonstrate *ModelSpy*'s feasibility. Our evaluation on five high-end consumer GPUs shows *ModelSpy*'s high accuracy in architecture reconstruction, including 97.6% in layer segmentation and 94.0% in hyperparameter estimation, with a working distance of up to 6 m. Furthermore, *ModelSpy*'s reconstructed DNN shows comparable performance to victim architecture, and can effectively enhance black-box adversarial attacks.**

I. Introduction

Deep neural networks (DNNs) are revolutionizing numerous critical applications, including face recognition, autonomous driving, and medical diagnosis [1], [2], [3]. However, their growing adoption also makes them prime targets for adversarial attacks, such as evasion attacks that deceive the system with adversarial examples [4]. To mitigate these risks, DNN-based systems and services avoid disclosing implementation details about their DNN models. Consequently, attackers are often left with limited prior knowledge and have to launch their attacks in *black-box* settings with reduced attack effectiveness [5], [6].

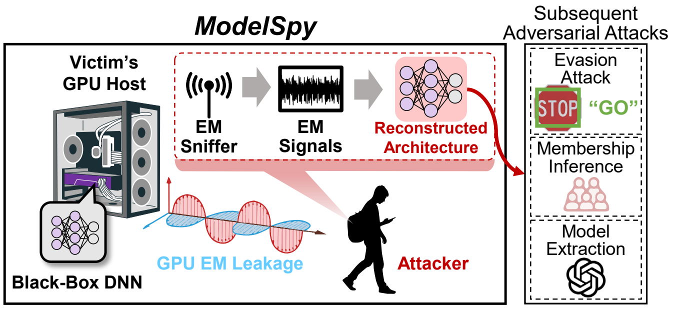


Fig. 1: Figure depicts an example attack scenario of *ModelSpy*. An attacker, posing as maintenance staff, walks along the corridor outside an AI company's office and reconstructs the victim's DNN architecture by sniffing EM signals leaked through the wall from the victim's GPU to facilitate subsequent black-box adversarial attacks.

To improve the effectiveness of black-box attacks, compromising the confidentiality of DNN models, particularly their *architectures*, has garnered significant attention [7], [8], [9], [10]. The architecture of a DNN, determined by its *layer structure* and *hyperparameters*, defines how data flows through the network and is a critical secret within the DNN system. Attackers with knowledge of the architecture can execute more targeted and effective attacks [7], [11]. For example, attackers can tailor their adversarial examples to deceive autonomous vehicles or bypass security checks like face recognition with a higher success rate [12]. Besides evasion attacks, architecture knowledge also plays an important role in enhancing other severe attacks, including model extraction attacks [13], [14] and membership inferencing attacks [15] (detailed in §II-A). As a result, the DNN architectures are considered a highly confidential asset in DNN-powered systems and demand robust protection against potential adversaries.

While model architecture file is typically encrypted and beyond the reach of adversaries [16], [17], [18], [19], prior works have shown the feasibility of extracting DNN architectures by analyzing *side-channel* with cache analysis, bus snooping, and performance interface monitoring [20], [21], [22], [9]. They typically exploit shared hardware resources and assume co-locating a malware with the victim's DNN process. Researchers have proposed various mitigation strategies to close such vulnerabilities [23], [24], [25]. In scenarios where

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230141
www.ndss-symposium.org

co-location is infeasible or effectively defended, physical side channels have emerged as a promising alternative [8], [26], [27], [13]. Recent studies show that an attacker within the vicinity of the DNN inference hardware could collect their physical signals, such as electromagnetic (EM) leakage, to extract DNN architecture. However, the practical adoption of these attacks is severely constrained by their limited range, e.g., requiring sniffers to be directly attached to GPU's power cable inside the victim host [8] or invasive procedures like chip decapsulation [26], thereby lacking stealth.

In light of these limitations, we pose the following research question: *Is it possible to design a novel physical attack that can fully extract a victim's DNN architecture, while maintaining attack stealthiness?* To this end, we introduce *ModelSpy*, a long-range DNN architecture snooping attack based on far-field EM signals that can reconstruct the ***complete*** architecture of ***unseen*** victim DNNs, including *number of layers*, *layer types*, and *layer-wise hyperparameters*. The key insight of *ModelSpy* lies in the observation that GPUs emit far-field radiative EM leakage, generated by digital components such as memory clocks and voltage regulators. These leakage signals are architecture-specifically modulated during DNN inferences (§III-B). By intercepting these modulated signals, ModelSpy achieves a high level of stealth, effectively operating from several meters away with the sniffer concealed in a backpack, even with obstacles. Figure 1 illustrates an example attack scenario: an attacker, posing as maintenance staff, walks along the corridor outside an AI company's office and reconstructs the complete DNN architecture by sniffing their GPU's EM signals leaked through the wall. With access to the victim architecture, attackers can launch various subsequent adversarial attacks. For example, an **evasion attack** in the context of an autonomous driving company occurs when an attacker uses the architecture information to create adversarial examples that mislead the DNN into making faulty decisions, potentially putting vehicles at risk. Additionally, a **membership inference attack** can be carried out to uncover sensitive information about the data used in training, or a **model extraction attack** may be performed to replicate the model's functionality (detailed in §II-A).

Designing *ModelSpy* involves two primary challenges. The first challenge lies in performing fine-grained reconstruction using noisy and attenuated EM signals. Although radiative EM signals can propagate over long distances, they suffer from low signal-to-noise ratios (SNR) due to ambient interference and propagation loss. Existing methods, designed for clean near-field signals, exhibit significant performance degradation [8]. To address this, we employ a specialized noise removal pipeline that isolates relevant EM components (§IV-E) and develop a hierarchical neural reconstruction engine that leverages global signal contexts for fine-grained reconstruction (§IV-C). This two-fold approach is demonstrated to be effective even in harsh signal conditions.

The second challenge is to enhance the attack scalability across the vast architecture space with implementation variations. Modern DNNs often consist of deeply stacked layers with diverse hyperparameters. Besides, runtime implementation variations result in distinct EM signatures even for the same logical layer. Since our reconstruction engine relies on supervised training, scaling the training data to capture this architectural and implementation diversity becomes a significant bottleneck. To address this, we propose a transfer learning scheme that leverages the temporal correlation between external EM leakage and internal GPU activity – particularly DRAM read and write – which are efficient to collect and label (§IV-B & §IV-D). *ModelSpy* is first pretrained on DRAM traces, i.e., time-series sequences of DRAM access activity, and then fine-tuned on a small set of EM signals, enabling scalable and robust architecture reconstruction.

We implement a proof-of-concept system leveraging a 5 GHz antenna for capturing the GPU leakage signals (discussed in §III-B). *ModelSpy* is evaluated through real-world experiments across five GPUs, collecting a substantial dataset from 141,000 DNN inference trials. We comprehensively evaluate *ModelSpy*'s performance under various conditions, including different distances, wall obstructions, and ambient EM interference. Additionally, we conduct an end-to-end adversarial attack using the reconstructed architecture obtained by *ModelSpy*. Overall, *ModelSpy* achieves 97.6% accuracy in layer segmentation (identifying layer numbers and types) and 94.0% accuracy in hyperparameter estimation, remaining effective even at a distance of five meters, significantly outperforming the state-of-the-art systems.

Our main contributions are summarized as follows:

- We propose *ModelSpy*, the first long-range DNN architecture snooping attack capable of reconstructing complete architectures of unseen black-box DNNs from up to 6 m away, demonstrating superior stealthiness and practicality compared to prior millimeter-range attacks.
- Our cross-layer analysis shows that DNN inference induces architecture-specific DRAM access, which in turn produces specific amplitude-modulated EM traces. We further establish a scalable pipeline for fine-grained architecture reconstruction from noisy EM traces.
- We evaluate *ModelSpy* through extensive real-world experiments, demonstrating high reconstruction accuracy and its effectiveness in enhancing subsequent black-box adversarial attacks.

Through this work, we hope to highlight that securing advanced AI systems will require an evolution in infrastructure security, especially in light of rapid AI advancements and the growing deployment of DNN on edge and mobile devices, which are more physically accessible and thus more susceptible to physical attacks like *ModelSpy* (detailed in §VI-B).

II. Motivation and Threat Model

We present the attacker's motivation, goal, and capabilities.

A. *Motivation of ModelSpy*

Attackers may steal a DNN architecture as a **precursor to subsequent black-box adversarial attacks on DNN-based systems**, including evasion attacks, membership inference

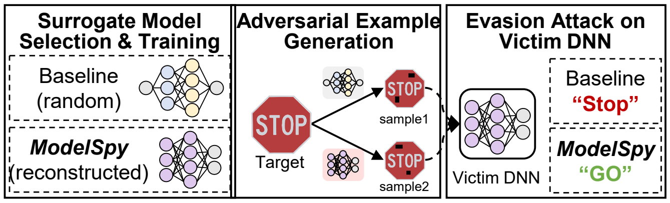


Fig. 2: Figure depicts how attackers can leverage the architecture extracted by *ModelSpy* to enhance the success of adversarial attacks on autonomous vehicles.

attacks, and model extraction attacks [15], [28], [29]. In black-box settings, attackers have no access to the victim model's internals and thus rely on a *surrogate model* – a locally constructed substitute used to craft or simulate attacks [29], [30]. The closer the surrogate matches the victim's architecture, the more effective the attack is when applied to the victim model. This is where *ModelSpy* comes in: by sniffing EM signals, *ModelSpy* extracts fine-grained DNN architecture, providing valuable prior knowledge to build high-fidelity surrogates and boost the success rate of adversarial attacks. Below, we present example attack scenarios enhanced by *ModelSpy*, with experimental validation in §V-D.

A typical adversarial attack that can be enhanced by *ModelSpy* is **evasion attack with adversarial examples** [28]. As shown in Figure 2, the attackers first train a surrogate model to generate the adversarial examples, which are then applied to the victim model. Instead of selecting a surrogate architecture at random, the attacker can use the high-fidelity architecture extracted by *ModelSpy* to significantly increase the attack success rate. In a real-world scenario, a malicious employee at an autonomous vehicle company could use *ModelSpy* to steal the model architecture, then generate subtle adversarial perturbation to road signs (e.g., stickers or markings) to mislead the vehicle's perception system into dangerous behavior, such as accelerating when it should stop [31], [32].

Similarly, by increasing the fidelity of the surrogate model, *ModelSpy* can enhance **membership inference attacks**. In a real-world scenario, a malicious hospital employee could use *ModelSpy* to steal the architecture of a diagnostic model and build a more accurate surrogate, improving their ability to determine whether a specific individual was included in the training data, potentially exposing sensitive health information [15]. *ModelSpy* can also enhance **model extraction attacks**. For instance, a malicious insider at an AI startup could use *ModelSpy* to extract the architecture of a proprietary DNN deployed on a production server to replicate the model's functionality with higher effectiveness, facilitating unauthorized misuse or commercialization [14].

B. Threat Model

Attacker's Goal. The *goal* of the attacker is to reconstruct the victim's DNN architecture by exploiting EM leakage signals emitted from the victim's GPU. A DNN consists of two main components: its architecture and weights[1]. The architecture defines the overall structure and computational flow of the network and is a key confidential asset that governs model behavior. Specifically, architecture attributes include:

- **Number of Layers.** The target DNN may contain an arbitrary number of layers with no upper-bound assumption.
- **Types of Each Layer.** The layer types include convolutional, recurrent, transformer, and linear layers, along with interspersed normalization (e.g., BatchNorm, LayerNorm), activation (e.g., ReLU, Sigmoid), and pooling layers (e.g., MaxPool, AvgPool).
- **Layer-Wise Hyperparameters.** Hyperparameters specific to each layer, such as filter sizes in convolutional layers or hidden sizes in transformer layers.

ModelSpy attacker aims to extract ***all*** the above architectural attributes. These components represent the core building blocks of modern DNNs widely deployed across various machine learning applications [1], [10].

Attacker's Capabilities. The attacker can intercept EM leakage signals from the victim's GPU during DNN inference in a non-intrusive, black-box manner, without interacting with or tampering with the victim's hardware or software. The DNN's code, memory, and design remain inaccessible, and only the inference stage runs on the victim's host. *ModelSpy* assumes no prior knowledge of the DNN, model family, layers, input size, or batch size. While the attack does not require knowing the victim's GPU model, optimal results are achieved when this information is available, as shown in §V-C5. Finally, the primary scope of *ModelSpy* is to extract the DNN architecture, which does not require additional information such as the victim model's training dataset and weights. Such information may be needed for subsequent adversarial attacks, but it is not essential for *ModelSpy*'s architecture snooping process.

III. Background

We present the background of GPU EM side-channel and the feasibility study of *ModelSpy*.

A. GPU EM Side-Channel

Electronic circuits within computers generate EM emanations as a side-effect of current flows. Since current flows in the systems vary with program activity, these EM emanations often convey sensitive information. As a typical electronic component, GPUs emit – ❶ *non-radiative magnetic signals*, as well as ❷ *radiative EM signals*. Previous works have leveraged non-radiative magnetic signals to infer DNN architectures, as these signals are clean and strongly correlated with GPU activities [33], [8]. However, these signals attenuate rapidly with distance and require **close proximity**, such as placing the sniffer within millimeters of the GPU power line, significantly **lacking stealth**. To enable a long-range attack, *ModelSpy* exploits radiative EM signals, which can propagate

[1] Weights are the learnable parameters optimized during training. While both components are critical, this work focuses exclusively on recovering the architecture, which is also a prerequisite for weight extraction [13].

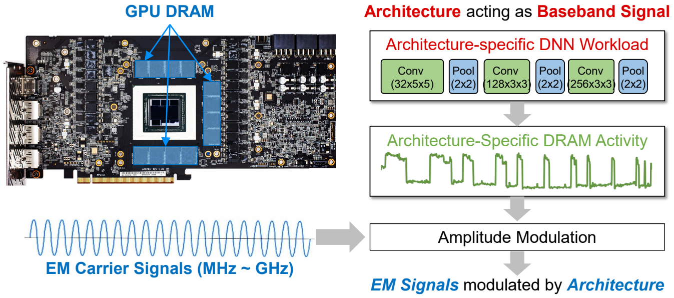

GPU DRAM
Architecture acting as Baseband Signal
Architecture-specific DNN Workload
Conv (32x5x5)
Pool (2x2)
Conv (128x3x3)
Pool (2x2)
Conv (256x3x3)
Pool (2x2)
Architecture-Specific DRAM Activity
Amplitude Modulation
EM Carrier Signals (MHz ~ GHz)
EM Signals modulated by Architecture

Fig. 3: Figure depicts the mechanism behind architecture leakage in EM side-channel. Digital components on the GPU continuously emit carrier signals, which are amplitude-modulated by architecture-specific activity such as DRAM access.

longer distances despite being significantly noisier than non-radiative leakages.

B. Characterizing Long-Range EM Radiation

During DNN inference, the radiative EM signals exhibit an **amplitude-modulation** scheme, with DNN architecture as the baseband signals. Figure 3 illustrates this process. GPUs continuously emit radiative EM signals from their digital components such as voltage regulators, memory clocks, and memory refresh circuits [34], forming the *carrier signal* $s_{\text{carrier}}(t)$. When DNN inference is executed, it induces architecture-specific hardware activity (e.g., DRAM access), which serves as the *baseband signal* $s_{\text{DNN}}(t)$. The resulting modulated signal can be simplified as $s_{\text{EM}}(t) = s_{\text{carrier}}(t) \cdot s_{\text{DNN}}(t)$. We further validate this phenomenon in §III-C. Although GPU manufacturers, like NVIDIA, apply techniques such as spread-spectrum clocking (SSC) to reduce EM interference at specific frequencies by broadening the frequency range, these signals remain detectable from several meters away.

The resulting EM signals are **wideband**, due to combined contributions of both carrier and baseband components [34], [35], [36]. To model this process, we use SSC as a representative carrier source [37], expressed as:

$$s_{\text{carrier}}(t) = \sum_{n=0}^{N} A_{ssc,n} e^{j2\pi f_{ssc,n} t}. \tag{1}$$

Here, $A_{ssc,n}$ and $f_{ssc,n} = f_{clk} - n f_m$ represent the amplitude and frequency of the n-th sub-clock, where f_{clk} is the base clock frequency, and f_m represents the SSC modulation frequency. N denotes the total number of sub-clocks. The baseband signals caused by DRAM accesses during DNN inference are modeled as:

$$s_{\text{DNN}}(t) = \sum_{m=1}^{M} A_m \cdot u(t - t_m), \tag{2}$$

where A_m and t_m denote the amplitude and the timestamp of the m-th DRAM access, with $u(t - t_m)$ representing step-like DRAM transition via Heaviside function. Then, the frequency domain representation $s_{\text{EM}}(f) = \mathcal{F}[s_{EM}(t)] = \mathcal{F}[s_{\text{carrier}}(t) \cdot s_{\text{DNN}}(t)]$ can be mathematically derived as:

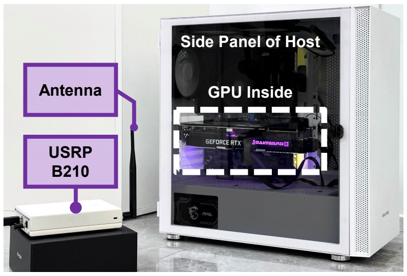


Fig. 4: Figure depicts *ModelSpy*'s feasibility setup composed of a test GPU host and a USRP B210 with an omnidirectional antenna for EM signal acquisition.

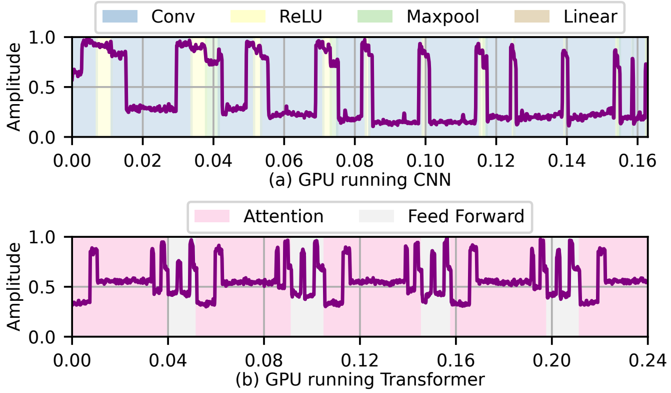


Fig. 5: Figure depicts the denoised EM signals from GPU when the host is performing inference on (a) a 12-layer CNN model and (b) a four-layer Transformer model.

$$\begin{aligned} s_{\text{EM}}(f) = & \sum_{m=1}^{M} \sum_{n=0}^{N} A_m A_{ssc,n} \, e^{-i2\pi(f - f_{ssc,n})t_m} \\ & \times \left(\frac{1}{2}\delta(f - f_{ssc,n}) + \frac{1}{i2\pi(f - f_{ssc,n})} \right). \end{aligned} \tag{3}$$

This formulation illustrates that the GPU EM signal $s_{\text{EM}}(f)$ disperses across a wide frequency band, with energy spreading around each sub-carrier frequency $f_{ssc,n}$ due to the intrinsic hardware execution. This phenomenon is then validated experimentally. We observe that although the memory clocks of all five evaluated GPUs exceed 6 GHz, i.e., the upper-frequency limit of the USRP B210, the same DNN inference is detectable across center frequencies from 5 to 6 GHz, indicating that $s_{\text{EM}}(f)$ spans a wide frequency band. We specifically selected 5 GHz as the center frequency because it lies outside the 5.15 to 5.85 GHz range typically utilized by WiFi, thereby minimizing potential interference with ambient wireless communication signals. We provide detailed modeling and discussion in Appendix A.

C. Feasibility Study

We verify the correlation of EM leakage with architecture.

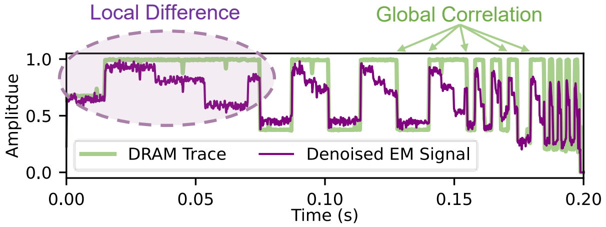


Fig. 6: Figure depicts the global temporal correlation between the DRAM trace and EM signals despite local differences.

Feasibility Setup. Figure 4 depicts our setup consisting of a test host with an RTX 3060 Ti GPU and a USRP B210 with an omnidirectional antenna for EM signal acquisition [38]. EM signals are captured using GQRX, an SDR receiver application, at a 5 GHz center frequency and 8 MHz sampling rate, as discussed in §III-B. The USRP downconverts the received signal s_{EM} with a 5 GHz local oscillator and outputs complex I/Q baseband samples representing the time-series signal s_{DNN}.

Correlation between EM Signals and DNN Architectures. We capture EM signals during GPU inference on a 12-layer CNN and a four-layer Transformer model. After applying a specialized denoising workflow (§IV-E), the signals, shown in Figure 5(a) and (b), exhibit amplitude variations that align with transitions between layer executions, suggesting that EM signals can capture architecture-specific characteristics. As discussed in §III-B, this is due to the correlation between EM signals with internal GPU activity, e.g., the GPU DRAM access pattern, referred to as DRAM traces. As shown in Figure 6, the DRAM trace and denoised EM signal show a globally aligned stepped pattern, with simultaneous rises and drops. There exist some local differences as EM signals are also influenced by other hardware activities such as computation and control logic. In short, this feasibility study supports our key observation: **DNN architecture acts as a baseband signal that amplitude-modulates the EM signals, enabling these signals as a proxy for architecture reconstruction.**

IV. Design and Implementation

We now present *ModelSpy*'s design and implementation.

A. Design Overview

ModelSpy's design enables the reconstruction of victim DNN architecture via the EM leakages of GPUs. It comprises two key phases: the *Bootstrapping Phase* and the *Attack Phase*, as shown in Figure 7(a) and (b). The goal of *Bootstrapping Phase* is to train a reconstruction engine (§IV-C) capable of inferring the DNN architecture from denoised EM signals. This is achieved by training on a large, labeled dataset of randomly generated DNN architectures. To minimize training data collection overhead to enhance scalability, *ModelSpy* employs transfer learning (§IV-D) with a hybrid training dataset (§IV-B): the reconstruction engine is initially pre-trained on labeled DRAM traces collected during DNN inference and then fine-tuned with a smaller set of labeled EM signals.

During the *Attack Phase*, the trained reconstruction engine is used by the attacker to reconstruct the victim's DNN architecture. This includes determining the number of layers, the type of each layer, and the layer-wise hyperparameters, all by intercepting EM signals from the victim's GPU running the target DNN. The collected EM traces in both phases are first preprocessed in the Noise Removal module (§IV-E) to filter out ambient and internal noises before being fed into the reconstruction engine, which finally outputs the victim architecture with high accuracy.

Designing *ModelSpy* involves two main challenges –

Challenge 1: Fine-grained Reconstruction with Low-SNR EM Signals. Although radiative EM signals can propagate over long distances, the received signals exhibit low SNR due to ambient interference and significant attenuation during propagation. The challenge is further intensified by the short inference time on high-end GPUs (e.g., ResNet-101 completes in around 0.1 seconds). Existing approaches are limited to coarse-grained recognition of **previously seen** architectures via classification-based pattern matching [39], [40], [26], or can only handle clean near-field signals but not effective for noisy far-field signals [8] (shown in §V-B). To address this challenge, we propose a specialized Noise Removal Module (§IV-E) and a Hierarchical Reconstruction Engine (§IV-C).

Challenge 2: Scalable Reconstruction Across Vast Architecture Space with Implementation Variations. Fine-grained reconstruction requires exploring a huge search space. For instance, even with just five layer types, a 100-layer network yields approximately 10^{70} possible configurations, which is further expanded when including hyperparameter variations. Moreover, runtime optimizations introduce **implementation variations** that are unknown to the attacker, resulting in distinct EM signatures even for the same logical layer. For example, a convolution layer can be implemented using GEMM, FFT, or Winograd [41]. As shown in Figure 8, the 12-layer CNN example mixes GEMM and FFT-based convolutions, with layers 5 to 8 using FFT and the rest using GEMM. Meanwhile, training the reconstruction engine requires fine-grained supervision, i.e., with *synchronized* labels for each EM signal segment indicating the corresponding layer type and hyperparameters. Labor-intensive manual collection and annotation can hardly scale to such architectural and implementation diversity. We address this through our Hybrid Training Set Collection (§IV-B) and DRAM-Assisted Transfer Learning Module (§IV-D).

B. Hybrid Training Dataset Collection

This module generates a hybrid, labeled dataset to train *ModelSpy*'s reconstruction engine, addressing the scalability challenge (i.e., Challenge 2). *Hybrid* refers to the combination of a smaller dataset of EM signals and a larger dataset of internal GPU activities, specifically DRAM read and write operations. As detailed in §IV-D, the DRAM traces are used for pre-training, while the EM signals are used only for fine-tuning. This approach leverages the following two key insights.

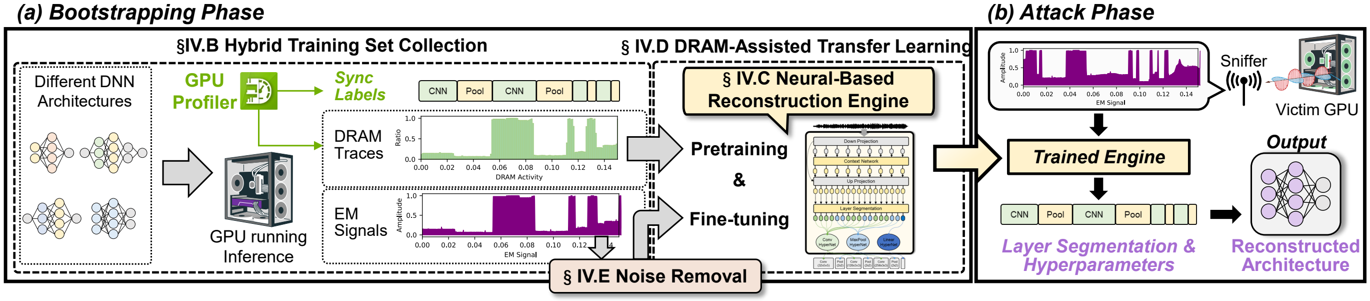


Fig. 7: Figure depicts *ModelSpy*'s design overview. (a) depicts *Bootstrapping Phase*, where the reconstruction engine is pre-trained with DRAM traces from random DNN architectures and fine-tuned with EM signals. (b) depicts *Attack Phase*, where the trained engine reconstructs the complete architecture of black-box DNN from the intercepted GPU EM signals.

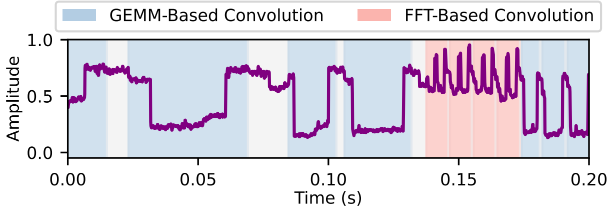


Fig. 8: Figure depicts that EM behavior of convolution layers varies with different implementations.

Key Insight 1: Temporal Correlations between DRAM Trace with EM signals. As discussed in §III-C with Figure 6, DRAM traces exhibit strong global temporal correlation with denoised EM signals. While EM signals may show local fluctuations due to other hardware activities, the overall correlation makes DRAM traces ***well-suited for pre-training***.

Key Insight 2: Scalable DRAM Collection and Annotation. We use DRAM for pre-training primarily due to its scalability in data collection and annotation. Unlike externally collected EM signals, which lack synchronization with the GPU host, DRAM traces can be collected internally using a GPU profiler[2]. This enables synchronized tracing of GPU workloads and generation of **time-step-level layer segmentation labels**. Since DRAM collection does not require external sensing hardware or manual alignment, it can be fully automated using scripted collection workflows. As a result, we can ***efficiently assemble a large and diverse training dataset with minimal effort***, ensuring the scalability of *ModelSpy*. Experimental results in §V-C6 further validate its effectiveness.

C. Neural-Based Reconstruction Engine

Our reconstruction engine is a neural network that takes EM signals as input and outputs the fine-grained DNN architecture, including all layers and their hyperparameters (i.e., addressing Challenge 1). At its core, the engine is designed to leverage global contextual information to improve robustness and accuracy. As illustrated in Figure 9, it is organized into three stages. The first two stages follow a sequence-to-sequence (Seq2Seq) framework: *Stage I* extracts time-step-level contextual features from the EM signals, and *Stage II* assigns layer type labels to each segment. These labeled contextual features are then passed to *Stage III* for hyperparameter estimation.

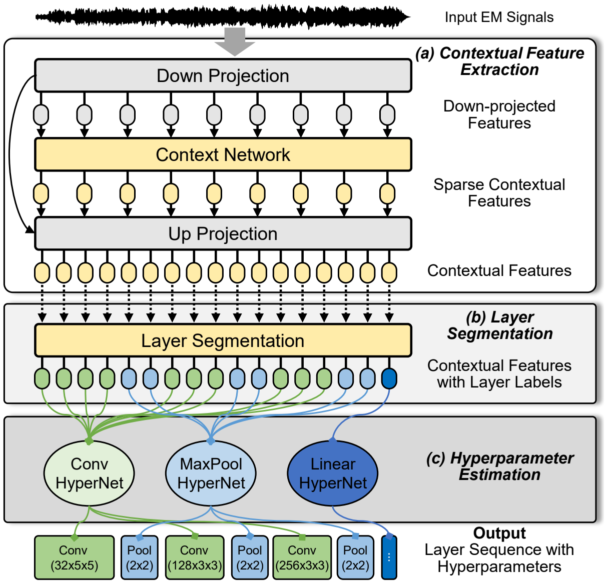


Fig. 9: Figure depicts the design of *ModelSpy*'s reconstruction engine, where the time-step level contextual features are extracted from the EM signals (Stage I) and assigned layer labels (Stage II). Then, the contextual features with layer labels are fed into Stage III for hyperparameter estimation.

1) Stage I: Contextual Feature Extraction: This stage aims to transform the input signal sequence $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_T)$ into contextual features $\mathbf{C} = (\mathbf{c}_1, \cdots, \mathbf{c}_T)$ that capture sequence-level contextual semantics, as shown in Figure 10.

• *[Key Design 1]: Transformer-based Context Network.* The core of Stage I is a Transformer-based Context Network, which is known for its capability to capture contextual dependencies from the entire sequence [42], [43]. It begins with learnable positional embeddings, followed by 12 Transformer blocks, each with a model dimension of 128, an inner dimension of 512, and 8 attention heads.

[2] The profiler is only run on an attacker-controlled host in *Bootstrapping Phase* for training data collection. It is not required in the *Attack Phase*.

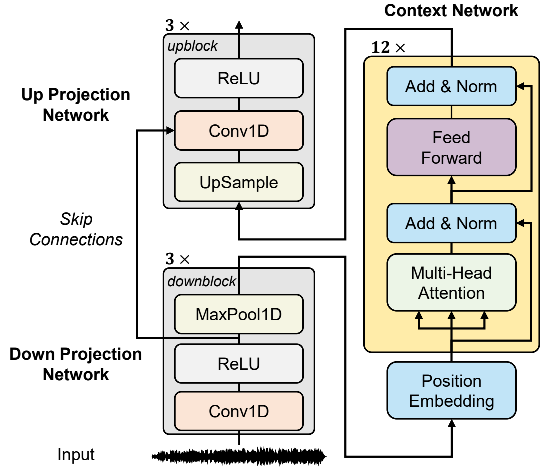


Fig. 10: Figure depicts the Contextual Feature Extraction (Stage I) of the reconstruction engine, consisting of a Down Projection Network, a Context Network, and an Up Projection Network. Skip connections link each downblock to its corresponding upblock.

- *[Key Design 2]: Down and Up Projection.* Transformer-based networks have memory and computational requirements that grow *quadratically* with sequence length, making it infeasible to process *long sequences* like EM signals [44], [45]. Hence, we first shorten the sequence using a Down Projection Network before the Context Network, and up-project it back using an Up Projection Network. As shown in Figure 10, the Down Projection Network consists of three *downblocks*. The convolutions in each downblock have (32, 64, 128) channels with MaxPool strides of (5, 2, 2) and kernel widths of (10, 3, 3). With an *input frequency of 10 KHz*, this configuration results in an *down projected frequency of 500 Hz*, making it suitable for the Context Network. The Up Projection Network, with three *upblocks* using the same hyperparameters as the downblocks, generates the dense contextual features $\mathbf{C} = (\mathbf{c}_1, \cdots, \mathbf{c}_T)$ that align with the time steps of the input.
- *[Key Design 3]: Skip Connections.* To minimize the loss of details due to Down Projection, we introduce skip connections between the k-th (k=1,2,3) downblocks and their corresponding upblocks, as shown in Figure 10, to enhance the resolution of contextual feature extraction.

2) Stage II: Layer Segmentation: Utilizing the contextual features $\mathbf{C} = (\mathbf{c}_1 \cdots \mathbf{c}_T) \in \mathbb{R}^{T \times d}$, where d is the feature dimension, the Layer Segmentation Stage segments the time series by associating each time step of the raw EM signal with a specific layer label, such as Conv, MaxPool, LSTM, Transformer, Linear, etc. Our approach involves learning a linear classifier $h_{class} \in \mathbb{R}^{d \times N_L}$ that maps the contextual features $\mathbf{C}$ to the corresponding layer label map $\hat{\mathbf{Y}}$ by minimizing the cross-entropy (CE) loss on labeled data, where N_L is the total number of layer types [46]. Given the ground-truth label map $\mathbf{Y}$, the loss is defined as: $\mathcal{L}_{layer} = \sum_{m=1}^{M} \mathbf{CE}(\mathbf{Y}_m, \hat{\mathbf{Y}}_m)$, where M is the total number of labeled signals in the dataset. This loss function measures the discrepancy between the predicted and ground truth labels for each time step, ensuring precise layer segmentation of the EM signals.

3) Stage III: Hyperparameter Estimation: ModelSpy estimates the hyperparameters of each layer using a set of specialized *HyperNets* tailored for individual hyperparameter types. Some layers, e.g., activation functions, do not require parameters, and thus we have 11 HyperNets in total. Depending on the type, HyperNets are implemented as either linear classifiers or regressors. For example, the kernel size of a convolution layer is one of a few discrete values, and hence is treated as a classification problem using cross-entropy loss. For hyperparameters with a larger discrete range, we perform regression with mean squared error loss. Since a layer spans many sampling points, we determine the final hyperparameter for a layer by taking the mode of the predicted values from all sampling points. We optimize the total loss, $\mathcal{L}_{hyp}$, which is the sum of individual HyperNet losses, for accurate estimation of all hyperparameters. Finally, we train the three stages in an **end-to-end** manner [47], with joint optimization of the sum of the loss functions $\mathcal{L} = \mathcal{L}_{layer} + \mathcal{L}_{hyp}$. We provide further training details below.

D. DRAM-Assisted Transfer Learning

We now detail how *ModelSpy*'s reconstruction engine is trained via transfer learning with the hybrid training dataset (§IV-B) to further enhance scalability (i.e., Challenge 2).

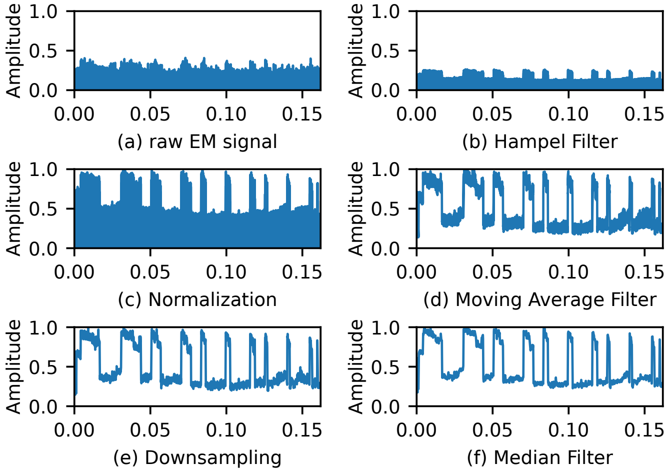
Pre-training with DRAM Traces. *ModelSpy*'s reconstruction engine is first pre-trained end-to-end on a large DRAM dataset $\mathcal{D}_D$, which contains M_D DRAM samples and their corresponding label maps. Given that DRAM traces are generally smoother than EM signals, we introduce **random Gaussian noise** at a 30 dB level during pre-training to bridge this gap. This augmentation helps the model adapt better when fine-tuning with the more complex and noisier EM signals. We use the Adam optimizer with a learning rate of 1.6e-04 [48], training with a batch size of 64 for 400K updates.

Fine-tuning with EM Signals. The model is then fine-tuned on a small EM dataset $\mathcal{D}_E$. We compare two popular fine-tune methods: full fine-tuning (updating all parameters) versus linear probing (updating only the last layer) [49]. We find that full fine-tuning yields better performance on *ModelSpy*. We also use the Adam optimizer with a learning rate of 1.6e-04 and a batch size of 64 for 2K updates. The transfer learning approach significantly enhances the scalability of *ModelSpy*, with its effectiveness thoroughly evaluated in §V-C6.

E. Noise Removal on EM signals

This module processes the noisy EM signals through noise removal steps, ultimately producing relatively clean signals (i.e., Challenge I). Simultaneously, we downsample EM signals from 8 MHz to 10 KHz to reduce computation overhead for the reconstruction engine. Figure 11(a) to (f) describes the noise removal pipeline.

Pulse Noise Removal. EM signals consist of impulsive noise due to other electronic components on the host and ambient interference. We resolve this by applying the *Hampel*



(a) raw EM signal (b) Hampel Filter (c) Normalization (d) Moving Average Filter (e) Downsampling (f) Median Filter

Fig. 11: Figure depicts *ModelSpy*'s noise removal pipeline consisting of five steps.

No.	GPU Model	Architecture	# Cores	Memory	Year
1	RTX 3060	Ampere	3584	12GB	2021
2	RTX 3060Ti	Ampere	4864	8GB	2021
3	RTX 3070	Ampere	5888	8GB	2020
4	RTX 4060	Ada Lovelace	3072	8GB	2023
5	RTX 4060Ti	Ada Lovelace	4352	16GB	2023

TABLE I: Table lists the tested GPUs, their architecture, number of CUDA cores, memory size, and release year.

filter [50], which detects and removes pulse-like noises. The filter computes the median m and standard deviation σ for each 0.1-ms sliding window. Any sample deviating more than 3σ from m is identified as an outlier and replaced with the median, as shown in Figure 11(b).

Normalization. To mitigate the impact of sniffing distance on signal strength, we perform *z-normalization* on the EM signal, $x(t)$, and obtain the normalized signal, $\tilde{x}(t)$. The denoised signal $\tilde{x}(t)$ is computed as $\frac{(x(t)-\mu)}{\sigma}$, where μ and σ are the mean and standard deviation of the overall signals [51].

Moving Average Filter. Before downsampling, we apply a moving average filter over 800 consecutive samples (0.1 ms) to smooth the signal, reducing high-frequency noise and preventing aliasing. This filter is aligned with the downsampling factor for consistency in the processing pipeline.

Downsampling. For computational efficiency in the reconstruction engine, we downsample the EM signal from 8 MHz to 10 KHz using a factor of 800. This reduces the sampling rate, lowering computational complexity while preserving the essential features of the signal for further analysis.

Median Filter. To suppress residual noise in the downsampled EM signal while preserving edges, we apply a median filter with a window size of 3 samples. Our overall noise removal design provides cleaner and downsampled EM signals for DNN reconstruction.

V. Evaluation

We present the evaluation of *ModelSpy* through comprehensive real-world experiments, demonstrating its feasibility.

Dataset	# DNN Archs	# Inference Trials	# Layers
Train (DRAM)	2,300	23,000×5 GPUs	314,170×5 GPUs
Train (EM)	260	2,600×5 GPUs	41,110×5 GPUs
Test (EM)	260	2,600×5 GPUs	39,110×5 GPUs

TABLE II: Table enumerates the number of distinct DNN architectures (Archs), DNN inferences, and distinct layers in the training and test datasets.

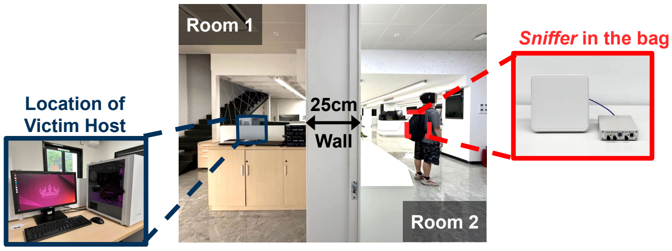


Fig. 12: Figure depicts the cross-wall experiment setup with the victim host and attacker separated by a wall; the sniffer is hidden in the attacker's backpack.

A. Experimental Setup

We first introduce *ModelSpy*'s experimental setup.

1) Platforms: Our setup (Figure 4) consists of a *target device* – a GPU host running DNN inference with Ubuntu 24.04 OS, as well as our *EM sniffer*, consisting of a USRP B210 [38] with a radio antenna. We employ two types of antennas: an omnidirectional antenna and a 20×20 cm panel antenna, both with a center frequency of 5 GHz and a sampling rate of 8 MHz. The EM sniffer is positioned 1 meter from the target device by default. We also evaluated *ModelSpy* under cross-wall scenario (Figure 12) in §V-C2. The EM signal captured is then denoised and downsampled to 10 KHz using our Noise Removal module (§IV-E). The entire EM sniffer (antenna + USRP) is compact and can be concealed within a 20L backpack.

2) Dataset Collection: We evaluate *ModelSpy* on a total of five GPU models released in the past four years with varying architectures and numbers of cores as depicted in Table I.

The collected dataset is shown in Table II. The training and test architectures are generated from two sources. First, we select popular and widely used model families, including AlexNet, VGG, ResNet, BERT, and their variants [52], [53], [54], [55]. We build the variant models by randomly altering network layers (e.g., adding or removing layers) and adjusting hyperparameters while adhering to the design principles of their respective model families. Second, we generate random models, consisting of CNNs (with random combinations of convolutional layers, activation layers, pooling layers, and possibly normalization layers), LSTMs, Transformers, and random combinations of linear layers. In total, we create **2,820 diverse DNN architectures**. These models vary in network depth, ranging from the **shallowest with 2 layers** (e.g., 1 convolutional layer + 1 fully connected layer) to the **deepest with 152 layers** (e.g., ResNet-152). We allow the host to select the optimized layer implementations, leading to *variations in* *the same layers*. For each model, we consider five commonly used input sizes (299×299, 224×224, 192×192, 162×162, 128×128), each with three channels. We select the batch size randomly from 64, 128, 256, 512, and 784. The training and test datasets are collected over all five GPUs.

Hybrid Training Dataset Collection (DRAM + EM). Out of the 2,820 architectures, we randomly selected 2,560 for training. To reduce data collection overhead, we employ our hybrid training set collection approach (§IV-B). We collect DRAM traces from 2,300 of these architectures for pre-training, and EM signals from the other 260 architectures for fine-tuning. DRAM traces and ground truth layer labels were collected using NVIDIA Nsight at a 10 KHz sampling rate to match the downsampled EM signal [56].

Test Set Collection (EM). The remaining 260 random DNN architectures are used for testing, and EM signals are collected during their inference. As all these models are distinct, the test architectures are ***unseen*** by the trained reconstruction engine. On average, EM acquisition takes 0.12 seconds with a standard deviation of 0.09 seconds, enabling fast side channel collection and preserving the stealth of the attack. We evaluate *ModelSpy*'s robustness under varying distances (§V-C1), wall conditions (§V-C2), DNN libraries (§V-C3), EM interference (§V-C4), and cross-GPU scenarios (§V-C5).

3) Performance Metrics: We define two metrics to evaluate layer segmentation performance and one metric to assess hyperparameter estimation accuracy. **(1) Layer Segmentation Accuracy:** The percentage of time-steps in the EM signal that are correctly classified according to their ground truth layer labels. **(2) Normalized Levenshtein Distance (NLD):** The Levenshtein edit distance measures the dissimilarity between the predicted and ground truth layer sequences. NLD normalizes this distance by the length of the ground truth sequence [57], [58]. A lower NLD indicates better alignment[3]. **(3) Hyperparameter Accuracy:** The percentage of estimated hyperparameters that match their ground truth values.

B. ModelSpy Overall Performance

We evaluate *ModelSpy* on five GPUs and 260 unseen DNN architectures, comparing it to the state-of-the-art baseline.

1) Baseline Selection: Most existing works rely on the classification of seen architectures, rendering them inapplicable to *unseen* DNN architectures and unsuitable for direct comparison with *ModelSpy* [40]. Therefore, we select the state-of-the-art *near-field EM leakage-based* architecture snooping attack that can identify unseen architectures as our baseline [8]. As the original baseline operates on clean, millimeter-range signals, we apply *ModelSpy*'s noise removal module to adapt it to far-field signals. To ensure a fair comparison, we train and test the baseline using the same sets of EM signals that were used to evaluate *ModelSpy*.

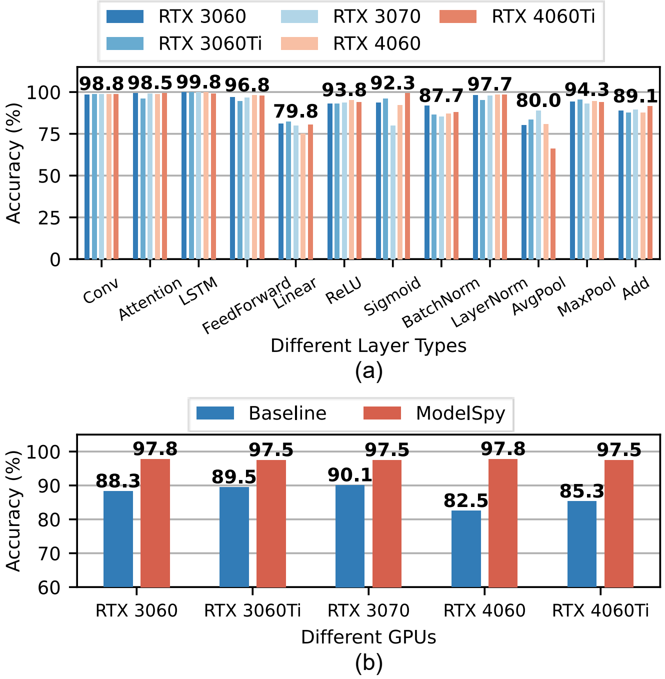


Fig. 13: Figure depicts (a) *ModelSpy*'s overall layer segmentation accuracy and (b) the comparison with baseline.

2) Layer Segmentation: We first present *ModelSpy*'s overall layer segmentation performance. As shown in Figure 13(a), the individual layer accuracy for each GPU is displayed, along with the average accuracy across GPUs for each layer type. *ModelSpy* achieves accuracy exceeding 92.3% for most layer types, except four layers: Linear, BatchNorm, AvgPool, and Add. These lower accuracies can be attributed to two primary factors. First, there is an imbalance in the dataset, such as the fewer instances of AvgPool compared to MaxPool. Second, some layers, like Linear and Add, have shorter execution durations (tens of milliseconds), making them harder to distinguish. Overall, *ModelSpy* achieves an average accuracy of 97.6% across the entire dataset, highlighting ***ModelSpy's effectiveness across all five GPUs on unseen architectures***. We further compare *ModelSpy* to the baseline in Figure 13(b), where the baseline achieves only 87.1% accuracy, i.e., *ModelSpy* outperforms baseline by 10.5%. The ***superior performance of ModelSpy over baseline*** is attributed to two primary factors. First, *ModelSpy*'s reconstruction engine outperforms the baseline in handling noisy EM sequences by leveraging global contextual information, thereby enhancing performance. Second, the scalability of the baseline is limited by its laborious training data collection, which fails to assemble a sufficiently large and diverse dataset to capture real-world implementation variations.

3) Layer Topology Reconstruction: We investigate the similarity of layer topology reconstructed based on the *ModelSpy*'s layer segmentation results compared to the ground truth layer topology. We use NLD (defined in §V-A3) as the metric, where a smaller value indicates better performance. We present the

[3] The edits include insertions, deletions, and substitutions. For example, if the ground truth layer sequence of a two-layer CNN is (Conv, BN, ReLU, MaxPool, Conv, BN, ReLU, FC) and the predicted sequence is (Conv, BN, ReLU, MaxPool, Conv, BN, ~~ReLU~~, FC), then NLD = 1/8 = 0.125.

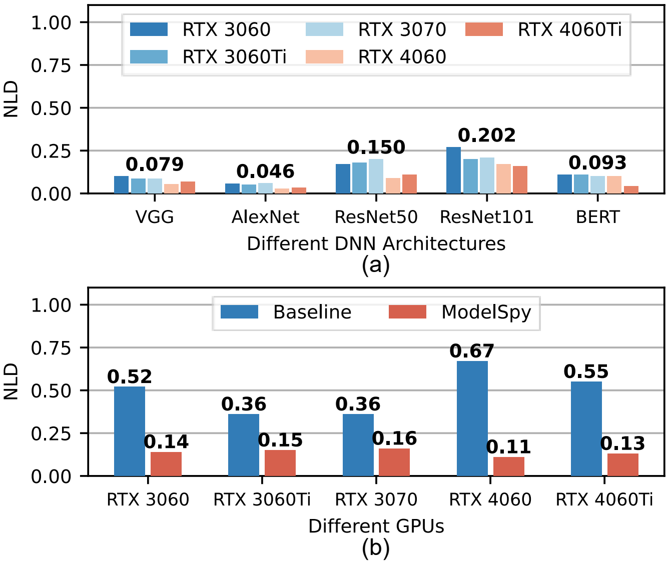


Fig. 14: Figure depicts (a) *ModelSpy*'s NLD performance for different popular DNN architectures.

NLDs for variants of five widely used network architectures in Figure 14(a). On AlexNet, VGG, and BERT variants, the average NLDs are 0.046, 0.079, and 0.093, respectively – all below 0.1. ResNet poses more challenges, particularly with its higher number of layers and complex topology (e.g., skip connections). However, *ModelSpy* achieves NLDs of 0.15 and 0.20 for ResNet-50 and ResNet-101, respectively, indicating that *ModelSpy* can reconstruct these layer topologies with over 80% similarity even in these challenging cases. In Figure 14(b), we further compare the NLD of *ModelSpy* against the state-of-the-art baseline. *ModelSpy* achieves an average NLD of 0.14, reducing the baseline's NLD of 0.49 by 71%, demonstrating significantly superior performance.

4) Hyperparameter Estimation: We now assess the hyperparameter estimation performance of *ModelSpy*. Different network types have distinct hyperparameter sets. For example, in CNNs, the Conv and Pool layers depend on kernel size, while Transformer layers require an accurate estimation of the number of attention heads and hidden size. The results presented in Figure 15(a) indicate that *ModelSpy* achieves hyperparameter estimation accuracy above 89.6% for CNN and Transformer layers. The accuracy for Linear layers is slightly lower at 84.2%, likely due to the short duration of these layers when the output dimensions are small, complicating hyperparameter estimation. The LSTM model exhibits the lowest accuracy at 78.3%, primarily because each LSTM layer is composed of many *small kernel* operations, with each operation being brief. In contrast, CNNs and Transformers use *larger kernels* that process data in parallel, with each kernel spanning a longer duration, making the feature more distinct. This phenomenon is exacerbated on certain GPUs that produce noisier signals, further reducing accuracy. Overall, *ModelSpy* achieves an average accuracy of 94.0% across the entire dataset. Finally, Figure 15(b) shows that *ModelSpy* outperforms baseline by 14% on average, demonstrating its superior effectiveness in fine-grained hyperparameter estimation.

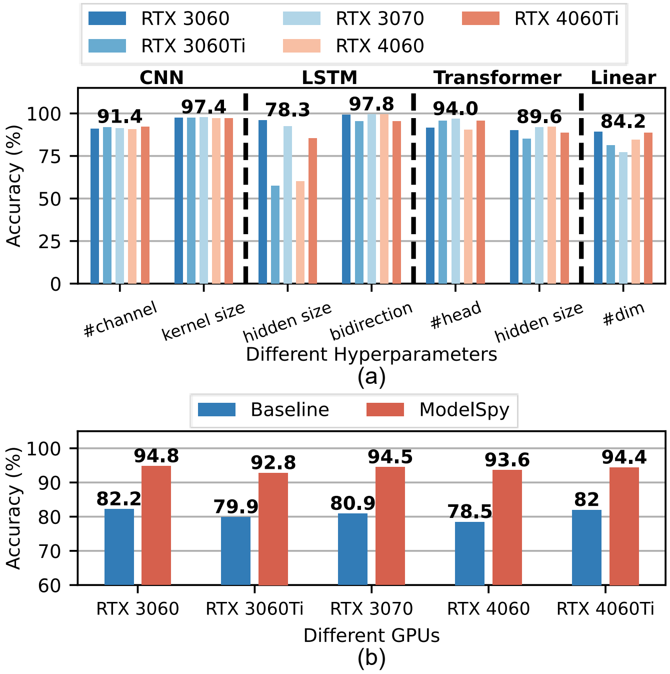


Fig. 15: Figure depicts *ModelSpy*'s overall performance on hyperparameter estimation.

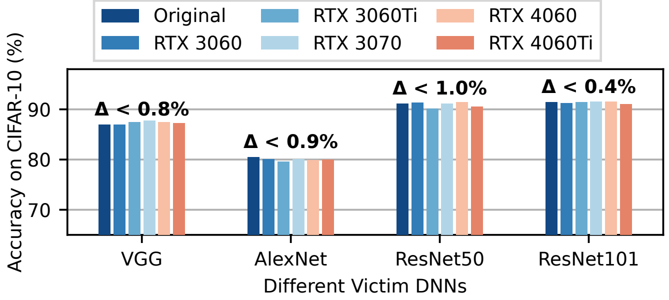


Fig. 16: Figure depicts performance comparison between *ModelSpy*'s reconstructed architecture and original victim's architecture, with Δ indicating the performance gap.

5) Effectiveness of Reconstructed Architecture: Using the sniffed layer and hyperparameter information, attackers can reconstruct the full DNN architecture of the victim. To evaluate the quality of the reconstructed architectures, we assess their performance on the same task as the victim DNN. We tested four networks–VGG, AlexNet, ResNet-50, and ResNet-101–all for CIFAR-10 image classification [59]. These networks were treated as black-box models and reconstructed from their GPU EM signals. For a fair architecture comparison, we used the same training dataset, number of training epochs, and training hyperparameters like optimizer, as the original models. As shown in Figure 16, the reconstructed networks closely match the performance of the original models, even for deep networks like ResNet-101.

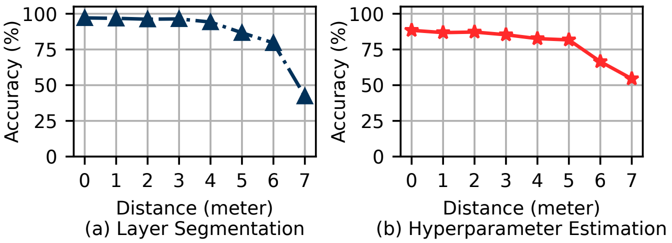


Fig. 17: Figure depicts *ModelSpy*'s performance under different distances.

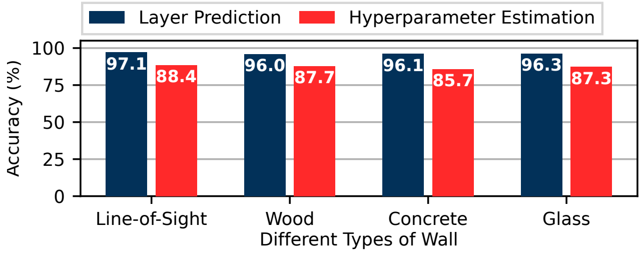


Fig. 18: Figure depicts *ModelSpy*'s performance under different types of walls.

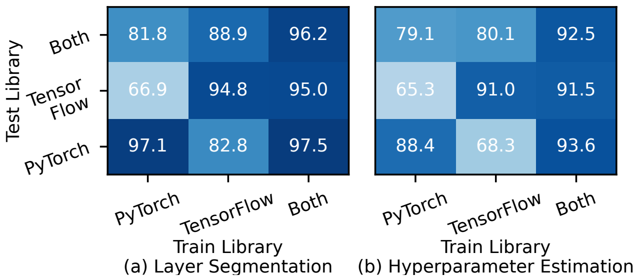


Fig. 19: Figure depicts *ModelSpy*'s performance under different DNN libraries.

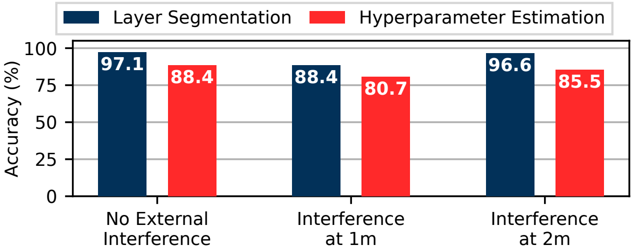


Fig. 20: Figure depicts *ModelSpy*'s performance under ambient EM interference.

C. *Differing Experimental Conditions*

We evaluate *ModelSpy*'s performance across several factors using a representative GPU, the NVIDIA RTX 3060 Ti. All experiments use the same trained model based on data from the RTX 3060 Ti. For each condition, *ModelSpy* is tested on 100 randomly sampled unseen network architectures.

1) Varying Distances: We assess *ModelSpy*'s performance at varying distances up to seven meters from the victim host's exterior side panel, where the EM sniffer is concealed in a backpack. As the distance increases, the SNR of EM signals decreases. As shown in Figure 17(a) and (b), *ModelSpy* achieves a layer segmentation accuracy above 86.7% at five meters, decreasing to 42.3% at seven meters. For hyperparameter estimation, the accuracy at five meters is 81.7%, reflecting only a 6.7% decrease compared to the zero-meter scenario. Overall, *ModelSpy* remains effective up to six meters. ***ModelSpy's six-meter working distance represents a significant improvement over the state-of-the-art baseline [8]***, which requires the sniffer to be positioned within millimeters of the GPU power cable. *ModelSpy*'s long-range capability stems from its ability to perform fine-grained analysis on noisy EM radiations and overcome scalability challenges, making *ModelSpy* a substantially more practical and stealthy attack.

2) Cross-Wall Conditions: ModelSpy is capable of operating as a through-wall side-channel attack. We assess three types of walls: glass, wood, and concrete. The experimental setup for the concrete wall is shown in Figure 12, where the attacker is positioned in an **adjacent room** with the EM sniffer hidden in a backpack. The attack results are presented in Figure 18. Layer segmentation accuracy remains above 90.1% across all conditions, while hyperparameter estimation accuracy is lowest behind the concrete wall at 85.7%, compared to 88.4% with no wall. These results demonstrate that *ModelSpy* can effectively perform even through walls, making the attack highly stealthy.

3) Impact of Different DNN libraries: We evaluate the scalability of *ModelSpy* across two widely used DNN libraries, PyTorch and TensorFlow. The results, shown in Figure 19(a) and (b), reveal that cross-library testing (e.g., training on PyTorch and testing on TensorFlow) results in reduced accuracy. This is due to differences in library-specific implementations and runtime optimizations (i.e., Challenge 2), causing a domain shift. However, we show that ***ModelSpy* can effectively mitigate the domain shift** by training on both libraries, achieving comparable results. Notably, this strategy incurs minimal overhead because of *ModelSpy*'s scalable training scheme with automated data collection.

4) Ambient EM Interference: All previous evaluations were conducted in the presence of typical EM interference from electronic components like power supply units, memory, and CPU. We now assess the impact of external ambient noise on *ModelSpy*, specifically from a WiFi access point (AP) whose frequency band is close to *ModelSpy*'s monitoring 5 GHz band. We introduced a Xiaomi AX6000 AP operating on channel 36 with a center frequency of 5.18 GHz, the closest available to 5 GHz on the AP, and varied its distance from the victim host. During testing, a nearby node connected to the AP was downloading at approximately 240 Mbps. As shown in Figure 20, placing the AP one meter away reduced layer segmentation accuracy by 8.7% and hyperparameter estimation accuracy by 7.7%, indicating notable interference when the AP is in close proximity. However, under conditions of weaker interference, such as when the AP was positioned two

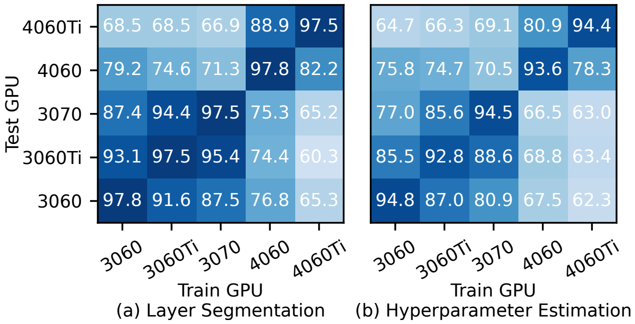


Fig. 21: Figure depicts *ModelSpy*'s cross-GPU results.

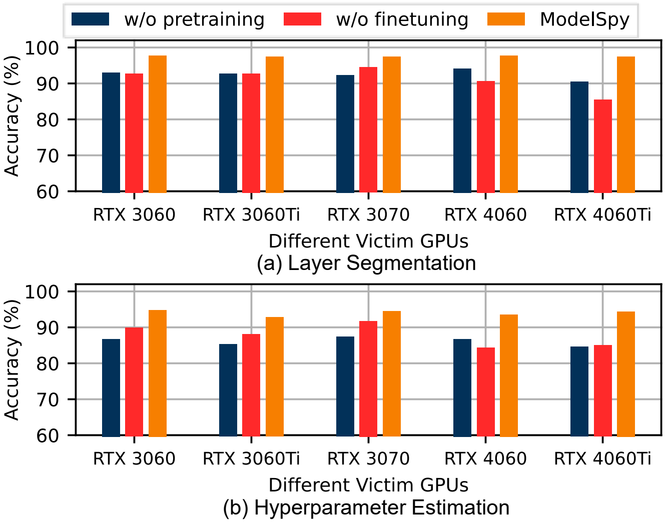


Fig. 22: Figure depicts module evaluation results.

meters away, the attenuated interference signal was effectively mitigated by *ModelSpy*'s noise removal module, resulting in only minimal performance degradation.

5) Cross-GPU Performance: To assess the transferability of *ModelSpy* on unseen GPUs, we conducted cross-GPU experiments where the reconstruction engine was trained on one GPU and tested on four different GPUs, as shown in Figure 21(a) and (b). We observe that *ModelSpy* **performs better when training and testing occur on same-generation GPUs** (e.g., training on RTX 3060 Ti and testing on RTX 3070) compared to cross-generation scenarios. For layer segmentation, the average cross-generation accuracy is 70.5%, while within the same generation (30 and 40 series), the accuracy increases to 85.5% and 91.6%, respectively. Hyperparameter estimation shows similar trends. This finding demonstrates attack feasibility even when the training and testing GPUs are not identical, thereby lowering the attack barrier.

6) Ablation Study on Transfer-Learning Module: We evaluate the impact of pre-training with DRAM by comparing *ModelSpy* with and without this module. Specifically, we assess the performance when the model is only fine-tuned with EM signals, i.e., trained directly from scratch using EM signals. The results, shown in Figure 22(a) and (b), indicate that the pre-training is effective across all five GPUs. The average accuracy of layer segmentation across GPUs increases from 92.5% to 97.6%, while hyperparameter estimation performance shows a significant increase from 86.2% to 94.2%, thereby demonstrating the overall effectiveness of pre-training with DRAM to enhance scalability for huge architecture space.

TABLE III: Table shows attack success rates using different surrogate architectures. *ModelSpy*'s reconstructed architecture can effectively enhance subsequent adversarial attacks.

D	Surrogate Architecture	Victim Architecture				
		AlexNet	VGG	Resnet34	Resnet101	ViT
CIFAR-10	AlexNet	0.47	0.26	0.25	0.29	0.20
	VGG	0.17	0.49	0.49	0.49	0.20
	Resnet34	0.11	0.36	0.68	0.56	0.13
	Resnet101	0.07	0.40	0.60	0.58	0.12
	ViT	0.07	0.06	0.10	0.14	0.59
	Ensemble	0.17	0.31	0.42	0.41	0.24
	ModelSpy	**0.44**	**0.49**	**0.64**	**0.56**	**0.59**
CIFAR-100	AlexNet	0.52	0.48	0.47	0.42	0.38
	VGG	0.27	0.49	0.57	0.49	0.22
	Resnet34	0.23	0.48	0.74	0.63	0.18
	Resnet101	0.21	0.38	0.68	0.67	0.22
	ViT	0.15	0.15	0.18	0.16	0.56
	Ensemble	0.27	0.39	0.52	0.47	0.31
	ModelSpy	**0.53**	**0.51**	**0.74**	**0.66**	**0.53**

D. Case Study: Adversarial Attacks on Image Classification with ModelSpy

As discussed in §II-A, access to the DNN architecture significantly improves the effectiveness of black-box adversarial attacks. In this case study, we demonstrate how *ModelSpy*'s reconstructed architectures can enhance evasion attacks on image classification tasks.

We consider a typical black-box setting where the attacker has no access to the victim model's internals and must rely on surrogate models. While conventional approaches use randomly selected or ensemble-based surrogates, an attacker equipped with *ModelSpy* can instead use the reconstructed architecture to train a high-fidelity surrogate, leading to more effective adversarial examples. We evaluate five widely used victim architectures – AlexNet, VGG, ResNet-34, ResNet-101, and Vision Transformer (ViT) – on two benchmark datasets: CIFAR-10 and CIFAR-100 [59], each with 10,000 randomly selected test images spanning 10 and 100 classes, respectively.

Adversarial examples are crafted using the Fast Gradient Sign Method (FGSM) [60] with perturbation magnitude $\epsilon = 0.03$. Table III reports the resulting attack success rates, where higher values indicate more effective attacks. Results show that using *ModelSpy*'s reconstructed architectures improves attack success rates by an average of 22.8% on CIFAR-10 and 19.8% on CIFAR-100 compared to ensemble-based surrogates. Notably, these success rates are within 4% of those achieved using the exact architecture, demonstrating ***ModelSpy*'s effectiveness in enhancing black-box adversarial attacks**.

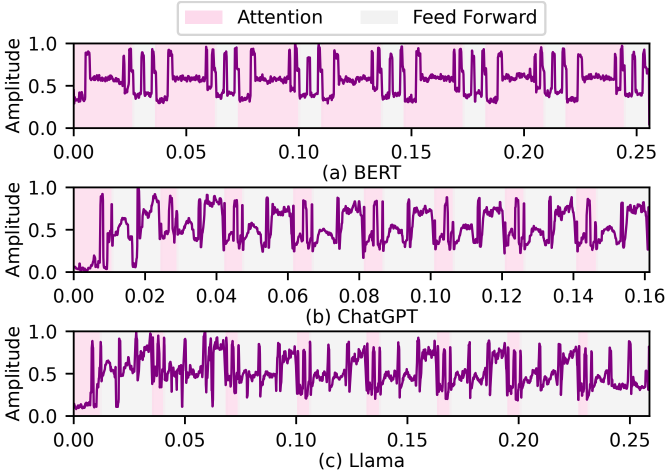


Fig. 23: Figure depicts EM signals from the GPU when running mainstream LLMs. *ModelSpy* can potentially extend to extracting LLM architectures used in black-box applications.

VI. Discussion

We discuss potential limitations and future directions for *ModelSpy*'s attack, as well as attack mitigation strategies.

A. Deployment Considerations

Limitation on Non-conventional Architecture Designs. Currently, *ModelSpy* can accurately identify previously *unseen* architectures as long as their layer types and hyperparameters are present in the training data. However, non-conventional architecture designs (e.g., layers with even-sized kernels) that are not included in *ModelSpy*'s training dataset are difficult to extract. This limitation is also noted in prior work [61]. Nevertheless, *ModelSpy* adopts a scalable design, specifically through its hybrid training set collection, which allows efficient incorporation of additional data to mitigate this limitation.

Impact of Operator Fusion. Some DNN frameworks support fusing consecutive operators for optimization during model inference, such as combining addition and ReLU into a single kernel. To address this, we can introduce combined layer classes (e.g., "Conv+ReLU") during layer segmentation, enabling *ModelSpy* to work seamlessly with operator fusion. Our experiments show minimal performance degradation, with occasional misidentification of the BatchNorm layer due to subtle differences between fused and non-fused versions.

B. Extension of ModelSpy

Attacking Large Language Models. *ModelSpy* is the **first** architecture snooping attack capable of targeting Transformer-based architectures, which are backbones of large language models (LLMs). As demonstrated in Figure 14, *ModelSpy* successfully attacks BERT, a typical LLM using stacked Transformer encoders [55]. ***ModelSpy* can also naturally extend to other LLMs** such as ChatGPT and Llama, which employ Transformer decoders. Their components like attention and feedforward layers are detectable by *ModelSpy*, as evidenced in Figure 23. A potential challenge arises in multi-GPU setups, often used for large-scale LLM inference, where pipeline parallelism is employed to distribute the model across multiple GPUs. This division generates distinct EM signals across GPUs corresponding to different layers or operations. To address this, *ModelSpy* could incorporate multiple receiving antennas, enabling the isolation of EM signals from each individual GPU. We leave this exploration for future work.

Attacking Edge DNN infrastructures. As edge DNN inference evolves, it introduces a **new attack surface** for *ModelSpy*. Recent studies suggest that DNN inference could be offloaded to smart infrastructure, such as lamp posts [62], [63]. Such roadside infrastructures may easily fall within *ModelSpy*'s attack range. Specifically, in open environments with less surveillance, attacks from *longer distances* become possible with the use of larger antennas. For example, we conducted a preliminary experiment with a parabolic antenna, which successfully operated from *over 10 m*.

C. Countermeasures

We suggest two solutions to mitigate *ModelSpy*'s attack. First, one could leverage *EM jamming* techniques that introduce noise to interfere with the target GPU's EM signals. As discussed in §V-C4, a WiFi AP placed with proximate distance can reduce *ModelSpy*'s reconstruction accuracy, indicating its effectiveness for jamming. However, the GPU's EM leakage signal is *wide-band* – covering 5-6 GHz range (§III-B), hence jamming this frequency band can disrupt communication, e.g., WiFi transmissions. An alternate mitigation strategy is *obfuscation* where other dummy DNN inferences concurrently run on the GPU, thereby masking the leakage from the inference, executing the DNN architecture of interest. This method has been shown to be effective in curbing EM side channels in a prior work [33]. However, such strategies can significantly increase the DNN inference time, leading to financial losses for AI companies.

VII. Related Work

We now present closely related work with *ModelSpy*.

Side-Channel-Based DNN Architecture Snooping. Prior work falls into two categories: co-location-based and physical leakage-based methods. Co-location-based attacks require the attacker to share system resources with the victim, leveraging cache analysis, bus snooping, or performance API monitoring [20], [21], [22], [7], [9], [58], [64], [65]. Such logical access may raise suspicion and can be effectively mitigated [23], [24], [25]. In contrast, physical-leakage attacks exploit power or EM leakages [27], [26], [13], [40], [39], [61], [66], [67]. *ModelSpy*'s unique contribution is the first to achieve fine-grained reconstruction on unseen architecture with *long-range* and *through-wall* capabilities. Our cross-layer analysis further ensures *ModelSpy*'s scalability to real-world DNN instances.

EM Side-Channels. Prior work has demonstrated the use of EM leakages to recover a wide range of sensitive information, including screen content [68], passwords [69], [70], fingerprints [71], cryptographic keys [72], audio [73], [74] and application activity [75], [76]. EM signals have also been used to detect hidden devices such as spy cameras, microphones, and GPS tracker [77], [78], [79] Among these, one prior study leveraged GPU dynamic voltage and frequency scaling (DVFS) patterns for long-range keystroke inference [80]. In contrast, *ModelSpy* bypasses *binary-like* DVFS patterns and instead leverages the *amplitude modulation* of GPU EM leakage, allowing fine-grained extraction of DNN architectures.

VIII. Conclusion

We propose *ModelSpy*, a novel DNN architecture snooping attack, based on the far-field EM leakage signals emanating from GPUs while conducting DNN inference. We design and implement *ModelSpy*, as well as perform a real-world evaluation on five GPUs over diverse unseen DNN architectures. *ModelSpy* achieves high layer segmentation and hyperparameter estimation accuracy, and can effectively enhance the subsequent black-box adversarial attacks. Furthermore, *ModelSpy* is both stealthy and scalable, as it can operate from a distance and through walls. We hope this work encourages the research community and GPU vendors to consider this potential threat and explore appropriate countermeasures.

Acknowledgment

We sincerely thank the anonymous reviewers for their valuable feedback, and Songfan Li (HKUST) and Yinghui He (NTU) for helpful discussions on GPU EM signal origin and collection. This paper is supported by the National Natural Science Foundation of China under grants U21A20462 and 62372400, and Samsung Electronics (Grant No. IO240424-09655-01).

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in IEEE Conference on Computer Vision and Pattern Recognition, CVPR'16. IEEE Computer Society, 2016, pp. 770–778.

[2] C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, "State-of-the-art speech recognition with sequence-to-sequence models," in 2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'18. IEEE, 2018, pp. 4774–4778.

[3] Y. Chen, X. Yuan, J. Zhang, Y. Zhao, S. Zhang, K. Chen, and X. Wang, "Devil's whisper: A general approach for physical adversarial attacks against commercial black-box speech recognition devices," in 29th USENIX Security Symposium, USENIX Security'20. USENIX Association, 2020, pp. 2667–2684.

[4] S. Hussain, P. Neekhara, S. Dubnov, J. J. McAuley, and F. Koushanfar, "Waveguard: Understanding and mitigating audio adversarial examples," in 30th USENIX Security Symposium, USENIX Security'21. USENIX Association, 2021, pp. 2273–2290.

[5] G. Tao, S. An, S. Cheng, G. Shen, and X. Zhang, "Hard-label black-box universal adversarial patch attack," in 32nd USENIX Security Symposium, USENIX Security'23. USENIX Association, 2023, pp. 697–714.

[6] H. Guo, G. Wang, Y. Wang, B. Chen, Q. Yan, and L. Xiao, "Phantomsound: Black-box, query-efficient audio adversarial attack via split-second phoneme injection," in Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses, RAID'23. ACM, 2023, pp. 366–380.

[7] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures," in 29th USENIX Security Symposium, USENIX Security'20, 2020. USENIX Association, 2020, pp. 2003–2020.

[8] H. T. Maia, C. Xiao, D. Li, E. Grinspun, and C. Zheng, "Can one hear the shape of a neural network?: Snooping the GPU via magnetic side channel," in 31st USENIX Security Symposium, USENIX Security'22. USENIX Association, 2022, pp. 4383–4400.

[9] 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 IEEE Symposium on Security and Privacy, SP'24. IEEE, 2024, pp. 3311–3326.

[10] T. Nayan, Q. Guo, M. Alduniawi, M. Botacin, A. S. Uluagac, and R. Sun, "Sok: All you need to know about on-device ML model extraction - the gap between research and practice," in 33rd USENIX Security Symposium, USENIX Security'24. USENIX Association, 2024.

[11] Y. Ge, P. Chen, Q. Wang, L. Zhao, N. Mou, P. Jiang, C. Wang, Q. Li, and C. Shen, "More simplicity for trainers, more opportunity for attackers: Black-box attacks on speaker recognition systems by inferring feature extractor," in 33rd USENIX Security Symposium, USENIX Security'24. USENIX Association, 2024.

[12] Y. Cao, S. H. Bhupathiraju, P. Naghavi, T. Sugawara, Z. M. Mao, and S. Rampazzi, "You can't see me: Physical removal attacks on lidar-based autonomous vehicles driving frameworks," in 32nd USENIX Security Symposium, USENIX Security'23. USENIX Association, 2023, pp. 2993–3010.

[13] P. Horváth, L. Chmielewski, L. Weissbart, L. Batina, and Y. Yarom, "BarraCUDA: GPUs do leak DNN weights," in 34th USENIX Security Symposium, USENIX Security'25. USENIX Association, 2025.

[14] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in 25th USENIX Security Symposium, USENIX Security'16. USENIX Association, 2016, pp. 601–618.

[15] X. Yuan and L. Zhang, "Membership inference attacks and defenses in neural network pruning," in 31st USENIX Security Symposium, USENIX Security'22. USENIX Association, 2022, pp. 4561–4578.

[16] Z. Sun, R. Sun, C. Liu, A. R. Chowdhury, L. Lu, and S. Jha, "Shadownet: A secure and efficient on-device model inference system for convolutional neural networks," in 44th IEEE Symposium on Security and Privacy, SP'23. IEEE, 2023, pp. 1596–1612.

[17] X. Jiang, M. Kim, K. E. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS'18. ACM, 2018, pp. 1209–1222.

[18] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS'16. ACM, 2016, pp. 308–318.

[19] H. Hu, Y. Huang, Q. Chen, T. Y. Zhuo, and C. Chen, "A first look at on-device models in ios apps," ACM Trans. Softw. Eng. Methodol., vol. 33, no. 1, pp. 26:1–26:30, 2024.

[20] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu, "Hermes attack: Steal DNN models with lossless inference accuracy," in 30th USENIX Security Symposium, USENIX Security'21. USENIX Association, 2021, pp. 1973–1988.

[21] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. A. Faruque, "Leaky DNN: stealing deep-learning model secret with GPU context-switching side-channel," in 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN'20. IEEE, 2020, pp. 125–137.

[22] S. Potluri and A. Aysu, "Stealing neural network models through the scan chain: A new threat for ML hardware," in IEEE/ACM International Conference On Computer Aided Design, ICCAD'21. IEEE, 2021, pp. 1–8.

[23] M. Yan, B. Gopireddy, T. Shull, and J. Torrellas, "Secure hierarchy-aware cache replacement policy (SHARP): defending against cache-based side channel attacks," in Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA'17. ACM, 2017, pp. 347–360.

[24] M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. W. Fletcher, and J. Torrellas, "Invisispec: Making speculative execution invisible in the cache hierarchy," in 51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO'18. IEEE, 2018, pp. 428–441.

[25] P. W. Deutsch, W. T. Na, T. Bourgeat, J. S. Emer, and M. Yan, "Metior: A comprehensive model to evaluate obfuscating side-channel defense schemes," in Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA'23. ACM, 2023, pp. 38:1–38:16.

[26] L. Chmielewski and L. Weissbart, "On reverse engineering neural network implementation on GPU," in Applied Cryptography and Network Security Workshops - ACNS'21 Satellite Workshops, vol. 12809. Springer, 2021, pp. 96–113.

[27] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN: reverse engineering of neural network architectures through electromagnetic side channel," in 28th USENIX Security Symposium, USENIX Security'19. USENIX Association, 2019, pp. 515–532.

[28] R. Song, M. O. Ozmen, H. Kim, R. Muller, Z. B. Celik, and A. Bianchi, "Discovering adversarial driving maneuvers against autonomous vehicles," in 32nd USENIX Security Symposium, USENIX Security'23. USENIX Association, 2023, pp. 2957–2974.

[29] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in 28th USENIX Security Symposium, USENIX Security'19. USENIX Association, 2019, pp. 321–338.

[30] H. Guo, X. Chen, J. Guo, L. Xiao, and Q. Yan, "MASTERKEY: practical backdoor attack against speaker verification systems," in Proceedings of the 29th Annual International Conference on Mobile Computing and Networking, MobiCom'23. ACM, 2023, pp. 48:1–48:15.

[31] D. Song, K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, F. Tramèr, A. Prakash, and T. Kohno, "Physical adversarial examples for object detectors," in 12th USENIX Workshop on Offensive Technologies, WOOT'18. USENIX Association, 2018.

[32] G. Lovisotto, H. Turner, I. Sluganovic, M. Strohmeier, and I. Martinovic, "SLAP: improving physical adversarial examples with short-lived adversarial perturbations," in 30th USENIX Security Symposium, USENIX Security'21. USENIX Association, 2021, pp. 1865–1882.

[33] R. Xiao, T. Li, S. Ramesh, J. Han, and J. Han, "Magtracer: Detecting GPU cryptojacking attacks via magnetic leakage signals," in Proceedings of the 29th Annual International Conference on Mobile Computing and Networking, MobiCom'23. ACM, 2023, pp. 68:1–68:15.

[34] R. L. Callan, A. G. Zajic, and M. Prvulovic, "FASE: finding amplitude-modulated side-channel emanations," in Proceedings of the 42nd Annual International Symposium on Computer Architecture, ISCA'15. ACM, 2015, pp. 592–603.

[35] ——, "A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events," in 47th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO'14. IEEE Computer Society, 2014, pp. 242–254.

[36] N. Sehatbakhsh, A. Nazari, H. A. Khan, A. G. Zajic, and M. Prvulovic, "EMMA: hardware/software attestation framework for embedded systems using electromagnetic signals," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO'19. ACM, 2019, pp. 983–995.

[37] C. Shen, T. Liu, J. Huang, and R. Tan, "When lora meets EMR: electromagnetic covert channels can be super resilient," in 42nd IEEE Symposium on Security and Privacy, SP'2021. IEEE, 2021, pp. 1304–1317.

[38] E. Research, "Usrp b210 usb software defined radio (sdr)," https://www.ettus.com/all-products/ub210-kit/, 2024.

[39] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin, "Deepem: Deep neural networks model recovery through EM side-channel information leakage," in 2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST'20. IEEE, 2020, pp. 209–218.

[40] S. Liang, Z. Zhan, F. Yao, L. Cheng, and Z. Zhang, "Clairvoyance: Exploiting far-field EM emanations of GPU to see your DNN models through obstacles at a distance," in 43rd IEEE Security and Privacy, SP Workshops 2022. IEEE, 2022, pp. 312–322.

[41] Y. Zhou, M. Yang, C. Guo, J. Leng, Y. Liang, Q. Chen, M. Guo, and Y. Zhu, "Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators," in IEEE International Symposium on Workload Characterization, IISWC'21. IEEE, 2021, pp. 214–225.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in Annual Conference on Neural Information Processing Systems 2017, NIPS'17, 2017, pp. 5998–6008.

[43] R. Xiao, J. Liu, J. Han, and K. Ren, "Onefi: One-shot recognition for unseen gesture via COTS wifi," in Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems, SenSys'21. ACM, 2021, pp. 206–219.

[44] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," CoRR, vol. abs/2004.05150, 2020. [Online]. Available: https://arxiv.org/abs/2004.05150

[45] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," in Annual Conference on Neural Information Processing Systems 2020, NeurIPS'20, 2020.

[46] A. Szabó, H. J. Rad, and S. Mannava, "Tilted cross-entropy (TCE): promoting fairness in semantic segmentation," in IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2021. Computer Vision Foundation / IEEE, 2021, pp. 2305–2310.

[47] R. Li, S. Zhang, and X. He, "SGTR: end-to-end scene graph generation with transformer," in IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR'22. IEEE, 2022, pp. 19 464–19 474.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in 3rd International Conference on Learning Representations, ICLR'15, 2015.

[49] K. You, Z. Kou, M. Long, and J. Wang, "Co-tuning for transfer learning," in Annual Conference on Neural Information Processing Systems 2020, NeurIPS'20, 2020.

[50] R. K. Pearson, Y. Neuvo, J. Astola, and M. Gabbouj, "Generalized hampel filters," EURASIP Journal on Advances in Signal Processing, vol. 2016, no. 1, pp. 1–18, 2016.

[51] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD'12. ACM, 2012, pp. 262–270.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in 26th Annual Conference on Neural Information Processing Systems, NIPS'12, 2012, pp. 1106–1114.

[53] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in 3rd International Conference on Learning Representations, ICLR'15, 2015.

[54] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in 14th European Conference on Computer Vision, ECCV'16, vol. 9908. Springer, 2016, pp. 630–645.

[55] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT'19. Association for Computational Linguistics, 2019, pp. 4171–4186.

[56] NVIDIA, "Nvidia nsight systems," https://developer.nvidia.com/nsight-systems, 2024.

[57] A. Graves, S. Fernández, F. J. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in Machine Learning, Proceedings of the Twenty-Third International Conference ICML'06, ser. ACM, vol. 148. ACM, 2006, pp. 369–376.

[58] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood, and Y. Xie, "Deepsniffer: A DNN model extraction framework based on learning architectural hints," in Architectural Support for Programming Languages and Operating Systems, ASPLOS'20. ACM, 2020, pp. 385–399.

[59] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[60] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in 3rd International Conference on Learning Representations, ICLR'15, 2015.

[61] P. Horváth, D. Lauret, Z. Liu, and L. Batina, "Sok: Neural network extraction through physical side channels," in 33rd USENIX Security Symposium, USENIX Security'24. USENIX Association, 2024.

[62] S. Shi, N. Ling, Z. Jiang, X. Huang, Y. He, X. Zhao, B. Yang, C. Bian, J. Xia, Z. Yan, R. W. Yeung, and G. Xing, "Soar: Design and deployment of A smart roadside infrastructure system for autonomous driving," in Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, MobiCom'24. ACM, 2024, pp. 139–154.

[63] Y. He, L. Ma, Z. Jiang, Y. Tang, and G. Xing, "Vi-eye: semantic-based 3d point cloud registration for infrastructure-assisted autonomous driving," in The 27th Annual International Conference on Mobile Computing and Networking, MobiCom'21. ACM, 2021, pp. 573–586.

[64] W. Hua, Z. Zhang, and G. E. Suh, “Reverse engineering convolutional neural networks through side-channel information leaks,” in Proceedings of the 55th Annual Design Automation Conference, DAC’18. ACM, 2018, pp. 4:1–4:6.

[65] S. Hong, M. Davinroy, Y. Kaya, S. N. Locke, I. Rackow, K. Kulda, D. Dachman-Soled, and T. Dumitras, “Security analysis of deep neural networks operating in the presence of cache side-channel attacks,” CoRR, vol. abs/1810.03487, 2018.

[66] F. Regazzoni, S. Bhasin, A. Alipour, I. Alshaer, F. Aydin, A. Aysu, V. Beroulle, G. D. Natale, P. D. Franzon, D. Hély, and et al., “Machine learning and hardware security: Challenges and opportunities -invited talk-,” in IEEE/ACM International Conference On Computer Aided Design, ICCAD’20. IEEE, 2020, pp. 141:1–141:6.

[67] V. Yli-Mäyry, A. Ito, N. Homma, S. Bhasin, and D. Jap, “Extraction of binarized neural network architecture and secret parameters using side-channel information,” in IEEE International Symposium on Circuits and Systems, ISCAS’21. IEEE, 2021, pp. 1–5.

[68] Z. Liu, N. Samwel, L. Weissbart, Z. Zhao, D. Lauret, L. Batina, and M. A. Larson, “Screen gleaning: A screen reading TEMPEST attack on mobile devices exploiting an electromagnetic side channel,” in 28th Annual Network and Distributed System Security Symposium, NDSS’21. The Internet Society, 2021.

[69] W. Jin, S. Murali, H. Zhu, and M. Li, “Periscope: A keystroke inference attack using human coupled electromagnetic emanations,” in 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS. ACM, 2021, pp. 700–714.

[70] M. Choi, S. Oh, I. Kim, and H. Kim, “Magsnoop: listening to sounds induced by magnetic field fluctuations to infer mobile payment tokens,” in MobiSys ’22: The 20th Annual International Conference on Mobile Systems, Applications and Services. ACM, 2022, pp. 409–421.

[71] T. Ni, X. Zhang, and Q. Zhao, “Recovering fingerprints from in-display fingerprint sensors via electromagnetic side channel,” in Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS’23, 2023, pp. 253–267.

[72] G. Camurati, S. Poeplau, M. Muench, T. Hayes, and A. Francillon, “Screaming channels: When electromagnetic side channels meet radio transceivers,” in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS. ACM, 2018, pp. 163–177.

[73] H. Chen, W. Jin, Y. Hu, Z. Ning, K. Li, Z. Qin, M. Duan, Y. Xie, D. Liu, and M. Li, “Eavesdropping on black-box mobile devices via audio amplifier’s EMR,” in 31st Annual Network and Distributed System Security Symposium, NDSS’24. The Internet Society, 2024.

[74] J. Choi, H. Yang, and D. Cho, “TEMPEST comeback: A realistic audio eavesdropping threat on mixed-signal socs,” in 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS. ACM, 2020, pp. 1085–1101.

[75] T. Ni, X. Zhang, C. Zuo, J. Li, Z. Yan, W. Wang, W. Xu, X. Luo, and Q. Zhao, “Uncovering user interactions on smartphones via contactless wireless charging side channels,” in 2023 IEEE Symposium on Security and Privacy, SP’23). IEEE, 2023, pp. 3399–3415.

[76] T. Ni, G. Lan, J. Wang, Q. Zhao, and W. Xu, “Eavesdropping mobile app activity via radio-frequency energy harvesting,” in 32nd USENIX Security Symposium, USENIX Security’23. USENIX Association, 2023.

[77] Y. Li, Z. Yan, W. Jin, Z. Ning, D. Liu, Z. Qin, Y. Liu, H. Zhu, and M. Li, “Gpsbuster: Busting out hidden GPS trackers via msoc electromagnetic radiations,” in Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS’24. ACM, 2024, pp. 3302–3316.

[78] Q. Zhang, D. Liu, X. Zhang, Z. Cao, F. Zeng, H. Jiang, and W. Jin, “Eye of sauron: Long-range hidden spy camera detection and positioning with inbuilt memory EM radiation,” in 33rd USENIX Security Symposium, USENIX Security’24. USENIX Association, 2024.

[79] S. Ramesh, G. S. Hadi, S. Yang, M. C. Chan, and J. Han, “Ticktock: Detecting microphone status in laptops leveraging electromagnetic leakage of clock signals,” in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS. ACM, 2022, pp. 2475–2489.

[80] Z. Zhan, Z. Zhang, S. Liang, F. Yao, and X. D. Koutsoukos, “Graphics peeping unit: Exploiting EM side-channel information of gpus to eavesdrop on your neighbors,” in 43rd IEEE Symposium on Security and Privacy, SP’22. IEEE, 2022, pp. 1440–1457.

Appendix A
Modeling Radiative GPU EM Emanation

GPUs unintentionally emit EM signals that can be used to infer the DNN architecture. These emanations consist of **carrier signals**, generated by digital components including voltage regulators, memory clocks, and memory refresh signals [34], and **baseband signals**, which are modulations resulting from GPU computing and memory activity.

The resulting EM leakage is **wideband** in nature, owing to both the carrier and the baseband signals contributing to its frequency spectrum [34], [35], [36]. First, it is common for the carrier signals to spread out across a broad frequency range. For instance, many periodic activities, such as the switching of voltage regulators, do not require precise timing and are typically generated by *lower-cost but less stable oscillators*. Moreover, techniques like spread-spectrum clocking (SSC) are applied to **deliberately widen** the clock signal’s frequency range to reduce electromagnetic interference. On the other hand, the baseband signal also exhibits a wideband characteristic, driven by step-function-like transitions in GPU memory access and computational activities. The combined wideband nature of both the carrier and baseband signals results in complex EM leakage patterns that can be used to reconstruct the DNN architectures.

While multiple sources contribute to GPU EM leakage, we use SSC as a representative wideband EM source to model the amplitude modulation process in DNN inference [37]. The carrier signal generated by SSC can be expressed as:

$$s_{\text{carrier}}(t) = \cos\left(2\pi f_{clk} t + \frac{\Delta f}{f_m}\sin(2\pi f_m t)\right), \tag{4}$$

where f_{clk} is the frequency of the base clock, and f_m and Δf represent the SSC modulation frequency and peak frequency deviation, respectively. The spectrum of the SSC signal can be mathematically derived as:

$$s_{\text{carrier}}(f) = \sum_n J_n\left(\frac{\Delta f}{f_m}\right)\Big[\delta(f - f_{clk} + n f_m) - \delta(f - f_{clk} - n f_m)\Big], \tag{5}$$

where $J_n(\cdot)$ is the Bessel function of the first kind, and $\delta(\cdot)$ denotes the Dirac delta function. Theoretically, the energy of an SSC signal is non-zero only at the frequencies $f_{clk} \pm n f_m$, forming a series of discrete *sub-clock* signals. However, in practice, a continuous spectral response is observed between these discrete frequencies, with the sub-clock frequencies appearing as peaks. We observe that all significant spectral peaks are confined within a 10 MHz frequency range on the lower sideband of f_{clk}, due to the band-pass filtering properties of the clock generator. Consequently, the SSC signal can be re-expressed as:

$$s_{\text{carrier}}(t) = \sum_{n=0}^{N} A_{ssc,n} e^{j2\pi f_{ssc,n} t}, \tag{6}$$

where $A_{ssc,n}$ and $f_{ssc,n} = f_{clk} - n f_m$ represent the amplitude and frequency of the n-th sub-clock, respectively. Here, N denotes the total number of sub-clocks presented.

We now consider the baseband signals $s_{\text{DNN}}(t)$ caused by computing and memory accesses in DNN inference, which can be represented as:

$$s_{\text{DNN}}(t) = \sum_{m=1}^{M} A_m \cdot u(t - t_m), \tag{7}$$

where A_m denotes the amplitude of the m-th memory access step, t_m represents the time at which the m-th access occurs, and $u(t - t_m)$ is the Heaviside step function that models the abrupt changes caused by the memory access. Therefore, the modulated signal $s_{\text{EM}}(t)$ can be written as:

$$\begin{aligned} s_{\text{EM}}(t) &= s_{\text{carrier}}(t) \cdot s_{\text{DNN}}(t) \\ &= \sum_{m=1}^{M} \sum_{n=0}^{N} A_m A_{ssc,n} \, u(t - t_m) \cdot e^{j2\pi f_{ssc,n} t}. \end{aligned} \tag{8}$$

The frequency domain signal can be mathematically derived as:

$$\begin{aligned} s_{\text{EM}}(f) &= \mathcal{F}[s_{\text{EM}}(t)] \\ &= \sum_{m=1}^{M} \sum_{n=0}^{N} A_m A_{ssc,n} \int_{-\infty}^{\infty} u(t - t_m) e^{i2\pi f_{ssc,n} t} e^{-i2\pi f t} \mathrm{d}t \\ &= \sum_{m=1}^{M} \sum_{n=0}^{N} A_m A_{ssc,n} \int_{-\infty}^{\infty} u(t - t_m) e^{-i2\pi (f - f_{ssc,n}) t} \mathrm{d}t \\ &= \sum_{m=1}^{M} \sum_{n=0}^{N} A_m A_{ssc,n} \mathcal{F}[u(t - t_m)](f - f_{ssc,n}) \\ &= \sum_{m=1}^{M} \sum_{n=0}^{N} A_m A_{ssc,n} e^{-i2\pi (f - f_{ssc,n}) t_m} \mathcal{F}[u(t)](f - f_{ssc,n}) \\ &= \sum_{m=1}^{M} \sum_{n=0}^{N} A_m A_{ssc,n} \, e^{-i2\pi (f - f_{ssc,n}) t_m} \\ &\qquad \cdot \left(\frac{1}{2} \delta(f - f_{ssc,n}) + \frac{1}{i2\pi (f - f_{ssc,n})} \right). \end{aligned} \tag{9}$$

This formulation illustrates how the GPU EM signal, $s_{\text{EM}}(f)$, disperses across the wideband spectrum due to the intrinsic hardware execution. This phenomenon is then validated experimentally. We observe that although all five of our GPUs' memory clocks exceed 6 GHz—the upper frequency limit of the USRP B210—the same DNN inference is detectable across center frequencies ranging from 5 to 6 GHz, with $s_{\text{EM}}(f)$ spanning a wide frequency band. We specifically selected 5 GHz as the center frequency because it lies outside the 5.15 to 5.85 GHz range typically utilized by Wi-Fi, thereby minimizing potential interference with ambient wireless communication signals.

Appendix B
Artifact Appendix

A. Description & Requirements

This artifact provides the implementation, dataset, and evaluation scripts for the paper. It allows reproduction of the main experimental results, including Figures 13, 14, 15, 17, and 18 in the paper.

1) How to access: The artifact is available at https://doi.org/10.5281/zenodo.17080118. It contains:

- Source code for the (under `dnn-prediction/`)
- Dataset (in `data/`)
- Conda environment file (`environment.yml`)
- Automation script for evaluation (`auto-eval.sh`)

2) Hardware dependencies:

- NVIDIA GPU with larger than 4GB memory

3) Software dependencies:

- Linux OS (tested on Ubuntu 20.04)
- Conda and Python

4) Benchmarks:

- EM signal samples with labels in `data/`

B. Artifact Installation & Configuration

1. Unzip `data.zip` and move the `data/` folder into the project root:

```
unzip data.zip
mv data modelspy-artifact/
```

2. Create the conda environment:

```
conda env create -f environment.yml
                 -p /home/user/anaconda3/envs/
                    env_name
```

3. Edit `dnn-prediction/configs/meta_data.yaml`:

```
root_data_path: /path/to/modelspy-artifact
```

C. Experiment Workflow

From the `dnn-prediction/` directory, run:

```
bash auto-eval.sh
```

This script will run all experiments and write the results to:

- `eval-results/model-eval-results.txt`
- `eval-results/comparison.txt`

D. Major Claims

- (C1): *ModelSpy*'s effectiveness on *Layer Segmentation* across all five GPUs (corresponding to Section V-B2 and Figure 13). Proven by (E1).
- (C2): *ModelSpy*'s effectiveness on *Layer Topology Reconstruction* (corresponding to Section V-B3 and Figure 14). Proven by (E2).
- (C3): *ModelSpy*'s effectiveness on *Hyperparameter Estimation* (corresponding to Section V-B4 and Figure 15). Proven by (E3).
- (C4): *ModelSpy*'s *long-range* effectiveness (corresponding to Section V-C1 and Figure 17). Proven by (E4).
- (C5): *ModelSpy*'s *cross-wall* effectiveness (corresponding to Section V-C2 and Figure 18). Proven by (E5).

E. Evaluation

1) Experiment (E0): [Produce all the results] [20 min human-time + 1.5 GPU hour]

[Preparation]: Complete environment setup as above.

[Execution]: Run *auto-eval.sh*, which produces all results for E1 to E5. After execution, all results are logged in *eval-results/model-eval-results.txt*. We provide the **expected results** in *eval-results/comparison.txt*, which corresponds to the results shown in our paper.

2) Experiment (E1): [5 min human-time]

[Results]: Lines 1–134 ("Layer Average Acc" and "test_layer_acc_*" items) match those in *comparison.txt*. **All line numbers refer to entries in the file *eval-results/model-eval-results.txt***.

3) Experiment (E2): [5 min human-time]

[Results]: Lines 171–215 match those in *comparison.txt*.

4) Experiment (E3): [5 min human-time]

[Results]: Lines 1–134 ("Hyperparam Average Acc" and "test_hyperparam_acc_*" items) match those in *comparison.txt*.

5) Experiment (E4): [5 min human-time]

[Results]: Lines 217–272 match those in *comparison.txt*.

6) Experiment (E5): [5 min human-time]

[Results]: Lines 273–294 match those in *comparison.txt*.

F. Notes

Execution time may vary depending on the GPU model used. Please ensure that all configuration files use absolute paths to prevent file resolution errors. For further details, refer to the *README.md* provided in the repository.