InferNet: Exploiting Aggregate GPU Profiles as Side-Channel for DNN Architecture Inference

RAJA HASNAIN ANWAR, University of Massachusetts Amherst, USA
JONAH O'BRIEN WEISS, University of Massachusetts Amherst, USA
TIAGO ALVES, Rio de Janeiro State University, Brazil
SANDIP KUNDU, University of Massachusetts Amherst, USA
MUHAMMAD TAQI RAZA, University of Massachusetts Amherst, USA

Deep Neural Networks (DNNs) have become ubiquitous for their ability to solve problems across various domains, including computer vision, natural language processing, and speech recognition. However, as their adoption grows, they face a range of security threats, such as model stealing, architecture extraction, and manipulation, which can compromise their integrity, privacy, and functionality. Past works have relied on complex, fine-grained, and time-series analysis to launch DNN model extraction attacks. These approaches require extensive amounts of data, which are often challenging to acquire and analyze effectively. This paper introduces InferNet, an attack method that leverages simple, non-intrusive, and coarse-grained system-level information to identify the underlying DNN architecture of a victim's application. By analyzing GPU kernel calls, memory events, and system-level metrics, InferNet fingerprints the DNN and infers its architecture with very high accuracy. It can predict the architecture family (*e.g.,* Inception vs. BERT), as well as the architecture variant (*e.g.,* InceptionV1 vs. InceptionV3). The evaluation results demonstrate the effectiveness of InferNet across AI/ML frameworks (TensorFlow, PyTorch), different DNN types (vision, LLMs), and hardware platforms (`NVIDIA Tesla T4`, `NVIDIA Quadro RTX 8000`). The results show that InferNet achieves 100% model extraction accuracy using only a partial GPU profile under various attack settings.

CCS Concepts: • **Security and privacy** → **Information flow control**; **Side-channel analysis and countermeasures**; *Embedded systems security*; • **Computing methodologies** → Machine learning; Artificial intelligence.

Additional Key Words and Phrases: large language models, vision-language models, side-channel attacks, model extraction, privacy

ACM Reference Format:
Raja Hasnain Anwar, Jonah O'Brien Weiss, Tiago Alves, Sandip Kundu, and Muhammad Taqi Raza. 2026. InferNet: Exploiting Aggregate GPU Profiles as Side-Channel for DNN Architecture Inference. 1, 1 (April 2026), 27 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The last decade has witnessed Deep Neural Networks (DNNs) become a cornerstone of AI and machine learning, especially when tasks demand high accuracy and mimic human-like cognitive processes. This proliferation is increasingly facilitated by Machine-Learning-as-a-Service (MLaaS) platforms, which have democratized access to AI capabilities.: image processing [33, 60], speech recognition [74], natural language processing [7, 11, 68], and automotive driving [3,

Authors' Contact Information: Raja Hasnain Anwar, ranwar@mass.edu, University of Massachusetts Amherst, USA; Jonah O'Brien Weiss, University of Massachusetts Amherst, USA; Tiago Alves, Rio de Janeiro State University, Brazil; Sandip Kundu, University of Massachusetts Amherst, USA; Muhammad Taqi Raza, University of Massachusetts Amherst, USA.

Manuscript submitted to ACM

28, 30, 71], and many more[21, 55, 58, 59, 61, 75]. The economic scale of this transformation is immense; the global MLaaS market is projected to expand from approximately USD 44.2 billion in 2024 to over USD 1.2 trillion by 2034 [56]. Trained DNN models are no longer mere software artifacts but are now considered invaluable intellectual property (IP) and core business assets. With training compute costs estimated to be in millions of dollars, the high barrier to entry and the immense value embodied in the trained model make them a highly attractive target for theft and replication [63].

The architectural paradigm of AI development is undergoing a significant evolution, moving from bespoke, task-specific models toward large-scale, pre-trained "foundation models" [62]. These models are subsequently fine-tuned to perform a multitude of downstream tasks, a strategy that has become dominant across the industry. This centralization of value creates a corresponding centralization of risk. A successful attack that compromises a single proprietary foundation model no longer affects just one application; it can undermine an entire ecosystem of services built upon it. Consequently, the security of foundation models has transcended being a mere operational concern to become a strategic imperative for the entire AI industry.

The concentration of high-value IP within MLaaS platforms has naturally led to the development of model extraction attacks, where an adversary queries a model to reverse-engineer its functionality without direct access to its weights or architecture [36]. Historically, these attacks have relied on privileged access to the host system to monitor fine-grained, low-level resources such as memory, cache, or PCIe traffic [18]. However, such methods are often impractical in real-world public cloud environments for several reasons: first, they require access to system resources that are typically unavailable to a co-located user (such as in a public cloud environment); second, the extraction process is prone to overfitting on limited query data, diminishing the stolen model's ability to generalize; and third, the reliance on fine-grained observations makes these attacks highly susceptible to data distortion and noise, leading to inaccurate model reconstruction.

The paradigm shift toward fine-tuning pre-trained foundation models fundamentally alters the adversary's objective. The focus has shifted to leveraging standard pre-trained models, trained on large, publicly available datasets, and then fine-tuning them for specific tasks (*e.g.,* by adding a few layers and training on smaller, task-specific datasets) [9, 20, 54]. This shift in development practices reduces the need to extract model parameters [8, 53, 64], as the core features of these pre-trained models are already well-established. Motivated by this 2-phase AI training approach, we propose stealthy and non-intrusive attacks that can easily infer the underlying DNN *architecture* (*e.g.,* GPT, BERT, or ResNet) used by the target applications. Our primary goals are: (1) to accurately identify the DNN architecture without requiring physical access or root privileges in deployments that expose profiling interfaces, (2) to ensure robust architecture prediction with sparse and noisy data, (3) to achieve high accuracy prediction even when the model is compressed into binary executable file, and (4) to evaluate transferability across hardware/framework settings.

Our empirical analysis demonstrates that aggregate GPU profiles effectively distinguish different DNN architectures, such as convolution-heavy vision models and transformer-based language models. This is primarily due to the specific kernel calls and system-level workloads that vary across DNN architectures, and their variants. From our study, we identify two key insights: first, each combination of DNN layer type and size corresponds to a specific GPU kernel call; second, the type of kernel calls, their execution times, and resource utilization form a unique fingerprint for DNN architectures.

This paper introduces *InferNet*, an attack method that exploits statistical GPU performance metrics and workload behavior during the execution of a DNN model to identify its underlying architecture. InferNet identifies key GPU kernel calls and memory management events, along with their execution times, to create unique fingerprints for individual DNN layers. These metrics, combined with system-level data (such as memory usage, clock speed, and power

Table 1. Distribution of DNN architectures across PyTorch and TensorFlow for each architecture family used in the InferNet attack. VT = Vision Transformer.

Model Class	DNN Family	Number of Variants	
		PyTorch	TensorFlow
Vision	AlexNet	1	–
	ConvNeXt	3	3
	DenseNet	4	3
	EfficientNet	11	8
	Inception	2	2
	MnasNet	4	–
	MobileNet	3	3
	ResNet	9	3
	SqueezeNet	2	–
	ShuffleNet	4	–
	VGG	8	2
VT	MaxVit	1	–
	ViT	4	–
LLMs	ALBERT	–	1
	BART	–	1
	BERT	–	1
	Gemma	–	1
	GPT	–	1
	RoBERTa	2	–
	XLM	2	–
	XLM-RoBERTa	–	1

consumption), are aggregated to profile the DNN model, and serve as a side-channel. By correlating the GPU profiles with known DNN architectures, InferNet accurately identifies both the DNN family (*e.g.,* Inception, Transformer) and its variant (*e.g.,* InceptionV1, InceptionV3).

InferNet adopts a temporally coarse but semantically rich profiling approach. Unlike prior work that depends on physical or low-level side-channels (*e.g.,* PCIe sniffing or EM traces), our attack directly leverages GPU kernel names and system metrics captured via `nvprof` aggregate mode. While aggregate mode lacks kernel execution order (compared to time-series), it suppresses profile noises from GPU scheduling and internal resource contention, and therefore, provides precise kernel identities and summary statistics that form a strong architectural fingerprint. This challenges the prevailing assumption that fine-grained traces are essential for DNN architecture inference, and highlights the attack surface exposed by aggregate GPU profiling available to the attacker.

We evaluate the practicality and generalizability of InferNet across 90 diverse DNN architectures, including vision models, large language models (LLMs), and vision transformers. The architecture pool encompasses 21 DNN model families, each with multiple variants (see Table 1). These models are pre-trained and implemented in two major AI/ML frameworks: TensorFlow [1] and PyTorch [52]. Experiments are conducted on two hardware platforms: the `NVIDIA Tesla T4` GPU on Google Colaboratory and the `NVIDIA Quadro RTX 8000` GPU on an Ubuntu server.

Our evaluation results show that InferNet is effective across multiple DNN types, AI/ML frameworks, and GPU hardware configurations. Key results includes DNN architecture extraction accuracy of: (i) 100% using complete GPU

profile, (ii) nearly 100% using only the top 3 kernel features, (iii) 100% on modified DNN architectures, and (iv) 75% across GPUs. Compared to similar methods, InferNet achieves 100% architecture extraction accuracy over a candidate set that is over four (4) times larger than the next best method (refer to Table 2 in the Related Work, §2).

Contributions. To summarize, our work makes the following important contributions:

- We identify aggregate GPU profiles—comprising kernel calls and system metrics—as a novel side-channel to fingerprint DNN architectures.
- We develop InferNet, a machine-learning-based attack method that uses GPU profiles to extract a wide range of DNN architectures, *e.g.*, vision, and large language models (LLMs).
- Our method can fully recover the model architecture even when the model has undergone extensive optimizations.
- Empirical evaluation shows that the complete GPU aggregate profile is not required for the prediction; instead, a few key features are sufficient for accurate DNN architecture extraction.
- We characterize the transferability of InferNet across GPU platforms and AI/ML frameworks, and show that heterogeneous training data can substantially improve cross-environment prediction.

Organization. The rest of the paper is organized as follows. Section 2 reviews related work on DNN architecture leakage. Section 3 provides a technical overview of deep neural networks (DNNs) and their acceleration in GPU-enabled systems. In Section 4, we present empirical observations on the mapping of DNN layers to low-level GPU kernel calls, which form the foundation for the design of InferNet. Section 5 outlines the threat model and explains the methodology for extracting DNN architectures using GPU profiles. Sections 6 and 7 evaluate the effectiveness of our attack across different challenging scenarios. In Section 8, we discuss potential countermeasures to mitigate model extraction attacks. Finally, Section 9 summarizes our findings.

2 Review of Related Works

The high-value, centralized landscape of modern AI development has given rise to a diverse spectrum of security threats against DNNs. These threats include adversarial attacks, which manipulate model inputs to cause misclassification; data poisoning and backdoor attacks, which corrupt the training process to embed malicious behavior; and privacy-violating attacks like membership inference, which aim to determine if a specific data point was used in the model's training set [23]. Among these threats, a particularly insidious category is the Model Extraction Attack (MEA), where an adversary seeks to steal the model's architecture and weights [36, 69]. This section provides a comprehensive review of the literature on model extraction, with a particular focus on side-channel attacks that exploit unintentional information leakages from the underlying hardware and software systems to infer a model's confidential architecture.

2.1 Escalated Attack Privileges

The landscape of Deep Neural Network (DNN) model extraction is defined by a fundamental trade-off between an attack's capabilities and its required level of privilege. At one end of the spectrum lie attacks that achieve near-perfect model reconstruction but demand elevated, often impractical, access. For instance, the Hermes [78] attack accomplishes lossless model recovery by snooping unencrypted PCIe traffic between the CPU and GPU. Similarly, DeepSniffer [26] reconstructs architectural layouts by analyzing the volume of memory read/write operations from bus snooping. Other methods have demonstrated success by analyzing physical emanations; the CSI NN attack, for example, uses electromagnetic (EM) traces to reverse engineer the architecture and weights of networks on embedded devices [4].

Table 2. Comparison with related works, which also formulate architecture extraction as a multiclass classification problem.

Related Work	Side-Channel	Candidate Size	Accuracy	Aggregate Data
Yu et al. [77]	Electromagnetic Trace	15	100%	✔
Hong et al. [24]	Cache Timing Trace	13	100%	✗
Patwari et al. [53]	Memory, CPU, and GPU Usage Trace	20	99%	✗
Kumar Jha et al. [29]	Memory, Timing, Power, and GPU Kernels	15	100%	✗
Liu et al. [37]	Adversarial GPU Kernel Execution Latency	5	100%	✗
Xiang et al. [73]	Power Trace	6	96.5%	✔
InferNet	**GPU Kernel Profile**	**90**	**100%**	✔

While formidable, the reliance of these techniques on direct hardware interaction or physical proximity renders them infeasible in typical cloud or Machine-Learning-as-a-Service (MLaaS) environments [70]. Moreover, attacks like DeepPeep [29] and MoSConS [72] require access to the victim DNN during model training. The impracticality of these high-privilege methods in common cloud settings highlights the need for an attack vector that does not rely on physical access or strong adversary capabilities.

2.2 Microarchitectural and Time-Series Side-Channels

A more practical threat model involves exploiting microarchitectural side-channels in shared computing environments. These attacks typically rely on capturing a fine-grained, time-series trace of events to infer the sequence of a DNN's operations. Cache-timing attacks [15, 24, 41], using underlying mechanisms like Prime+Probe or Flush+Reload, are a prominent example. Some, like Cache Telepathy [76], monitor cache hits on specific, shared DNN library functions to reconstruct the layer sequence.

A more novel approach, GANRED [39], captures the overall cache timing signature and uses a Generative Adversarial Network (GAN) to train a substitute model that mimics this signature, thereby avoiding a dependency on shared libraries. Other time-series attacks have targeted the GPU, such as Leaky DNN [72], which exploits the timing penalties of GPU context-switching to infer layer composition and hyperparameters during the training phase. The critical vulnerability of these time-series methods is their fragility; they are susceptible to system noise and can be defeated by countermeasures that disrupt the temporal order of operations, such as randomized kernel scheduling and precision limiting [44, 72]. The fragility of the time-series methods and unreliability of microarchitectural approaches create a clear need for a more robust side-channel that is resilient to system noise and temporal countermeasures.

Recent work further broadens the landscape of architecture leakage across side-channel modalities. Arefin and Serwadda [2] show that architecture-level information remains exposed even without direct physical access; Batina et al. [5] synthesize reverse-engineering attacks on neural networks through side channels; and Malan et al. [43] demonstrate DVFS-based fingerprinting in edge inference services. These studies reinforce that architecture information can leak through diverse hardware/software observables. InferNet differs by focusing on aggregate GPU profiling and multiclass prediction over a substantially larger candidate pool.

2.3 Binary-based Attacks

In scenarios where the adversary has access to the model's binary or executable file (*e.g.,* compiled CUDA code, ONNX, or LiteRT), several binary-based model extraction techniques become feasible. Prior works have demonstrated architecture

recovery through function call and cache tracing [40], control-flow analysis [27], or side-channel observations of compiled code [4, 31]. Tools such as angr, Ghidra, and IDA Pro have also been used to disassemble binaries and recover model components when access to the model artifact is assumed. These approaches, however, typically require stronger adversarial capabilities, such as file system access, reverse engineering expertise, or dynamic instrumentation. In contrast, InferNet targets a weaker and more realistic threat model, where the attacker relies solely on aggregate GPU profiling data to infer the DNN architecture.

Our proposed attach method, InferNet, introduces a paradigm that circumvents the limitations of existing approaches by strategically sacrificing temporal data for statistical robustness. Attacks like Hermes [78], DeepSniffer [26], and DeepPeep [29] assume identical conditions in both training and deployment phases of the DNN lifecycle. Although identical settings maximize performance for all attacks, InferNet retains significant efficacy under varied conditions, demonstrating its practical versatility. It exhibits broader applicability, encompassing a wider range of deep neural network architectures, including vision models, vision transformers, and large language models (LLMs). As shown in Table 2, InferNet achieves 100% architecture extraction accuracy across a diverse candidate set of 90 models, encompassing vision models, vision transformers, and large language models (LLMs) across both PyTorch and TensorFlow—a pool more than four times larger than the next-best classification-based method.

3 Background

This section provides the technical background on deep neural networks (DNNs) and their use of multithreading technology on modern GPUs. We begin by describing the structure of DNNs and their key building blocks. Next, we introduce the CUDA toolkit [49] and explain how it accelerates data processing on GPUs. Finally, we discuss how profilers can help monitor various GPU workloads involved in a DNN execution.

3.1 Deep Neural Networks

Deep neural networks (DNNs) comprise a hierarchical organization of connected layers to model the relationship between the input x and output y by approximating an unknown function $f^{\theta}(x) = y$. Here, θ denotes the *learned* parameters, *i.e.,* weights, biases, and other model-specific parameters of the layers in the architecture f; and f^{θ} is the resultant model. Computationally, the input x passes through the DNN in a series of feed-forward matrix operations. Each layer processes the input to extract specific features, progressively transforming it into higher-level representations. The final output y is typically a probability distribution over multiple categories.

DNNs operate in two main phases: training and inference. In the *training* phase, the model processes input data to produce an output, which is compared to the ground truth using a loss function to quantify the error in the model's predictions. The model's parameters θ are adjusted iteratively through backpropagation to minimize this error [35]. *Hyper-parameters*, such as learning rate, epochs, and batch size that control the training process, are independent of the data [35]. In the *inference* phase, the trained model makes predictions on new, unseen data. When a DNN is deployed in a real-world setting, it operates in the inference phase.

3.1.1 DNN Architecture. DNN architecture is defined by its layers, their types or dimensions, and the topology of connections, which form a computational graph. Layers can be connected either sequentially or non-sequentially. In a sequential connection, each layer directly takes the output of the previous layer as input. In contrast, non-sequential connections include more complex structures such as skip connections, branches, or shared layers.

Modern DNN models can be broadly categorized into two types based on their application domains: *vision models*, designed for processing image data, and *large language models* (LLMs), specialized for understanding and generating text. Vision models typically employ architectures that process spatial data, utilizing layers such as convolutional and pooling layers to capture local patterns and hierarchies in images. These models rely on grid-like structures to preserve spatial relationships between pixels. In contrast, LLMs are designed to process sequential data, leveraging architectures like transformers [68] with self-attention to capture dependencies across text tokens. While vision models focus on spatial feature extraction, LLMs emphasize understanding contextual relationships in sequential data. Therefore, the architecture and layer configuration of a DNN must be tailored to the specific characteristics of the data it is intended to process.

3.1.2 DNN Optimization. Modern 'deep' neural networks (DNN) tend to have a large number of layers. It is, therefore, essential to reduce the resource requirements for such architectures without affecting their performance. DNN optimization techniques, like **pruning** [19], reduce the computation overhead by removing the least important parameters $\in \theta$. This creates a sparser architecture and has been shown to significantly decrease the size of the model with minimal impact on accuracy [19]. Similarly, **neural architecture search (NAS)** [13] automates discovering optimized architectures tailored to specific constraints, *e.g.,* hardware and cost. Depending on specific constraints, DNN architectures can be adapted into optimized variants, such as MobileNetV2 [57] and MobileNetV3 [25], designed explicitly for resource-constrained hardware platforms.

3.2 GPU-Accelerated DNNs

Modern AI/ML frameworks leverage the parallel processing capabilities of GPUs [51] for DNN workloads, such as matrix multiplications and convolutions. GPUs come with advanced development toolkits, such as NVIDIA's CUDA [49], which offer high-level APIs and libraries for general-purpose use.

3.2.1 CUDA. CUDA is a parallel computing platform equipped with a suite of tools to create and optimize programs to run on NVIDIA GPUs. It acts as an interface between AI/ML frameworks, such as TensorFlow [1] and PyTorch [52], and GPU hardware. The AI/ML frameworks define the DNN architecture and computational workflows, while CUDA translates high-level mathematical operations (*e.g.,* convolutions) into low-level GPU functions, called **kernels**[1]. Figure 1 depicts the key elements involved in the GPU system stack to run a DNN using the CUDA toolkit.

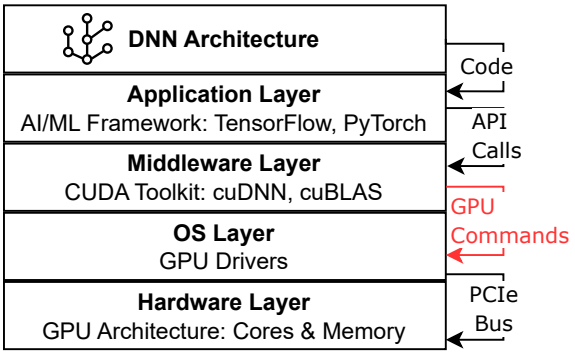


Fig. 1. The architecture stack that runs a DNN on an NVIDIA GPU using CUDA. Red color indicates data collected for profiling.

CUDA maps DNN operations, represented as a computation graph, onto the GPU hardware using specialized libraries. In CUDA libraries, each *kernel* is designed to break down complex operations into smaller, unit-level tasks that can be executed concurrently across multiple GPU cores. For instance, matrix multiplication operations in fully connected layers use the cuBLAS (CUDA Basic Linear Algebra Subprograms) [48] library, while convolutional operations

[1]A CUDA kernel, not to be confused with an OS kernel, refers to a function executed concurrently by multiple GPU threads.

Table 3. Partial view of aggregated profiles acquired by running 10 inferences of GoogleNet [65] and RoBERTa [38] models on an NVIDIA Tesla T4 GPU using PyTorch framework. Full profile data would include ~15 GPU kernels, ~30 API calls, and 5 system signals.

Model	Type	Name	Time(%)	Time(ms)	# Calls	Avg(μs)	Min(μs)	Max(ms)
GoogLeNet	Kernel	cudnn_infer_volta_scudnn_winog...	31.87	13.94	190	73.36	20.74	0.26
		volta_sgemm_32x32_sliced1x4_nn	14.43	6.31	190	33.22	19.36	0.05
		[CUDA memcpy HtoD]	8.87	3.88	354	10.96	0.64	0.75
	API call	cudaDeviceGetStreamPriorityRange	39.73	232.00	1141	203.33	0.39	231.07
		cudaLaunchKernel	33.93	198.11	2370	83.59	4.71	50.05
		cudaGetDevice	3.78	22.09	28307	0.78	0.29	6.15
RoBERTa	Kernel	[CUDA memcpy HtoD]	49.92	285.88	302	946.61	0.54	42.87
		volta_sgemm_128x64_tn	29.67	169.89	720	235.96	81.28	0.44
		volta_sgemm_128x128_tn	15.11	86.52	240	360.52	262.05	0.49
	API call	cudaLaunchKernel	53.20	942.32	3250	289.94	6.93	783.65
		cudaMemcpyAsync	17.10	302.90	302	1002.99	5.12	43.07
		cudaDeviceGetStreamPriorityRange	11.65	206.25	1	206249.62	206249.62	206.25

are optimized using the cuDNN (CUDA Deep Neural Network) [50] library. Activation functions like ReLU and sigmoid are executed as element-wise operations using highly parallelized CUDA kernel calls.

3.2.2 NVIDIA Profiler. `nvprof` [47] is a command-line tool for monitoring CUDA activities during the execution of GPU-accelerated applications. `nvprof` captures details like the type of kernel executed and time spent on execution. It also tracks memory-related activities, such as host-to-device and device-to-host data transfers, as well as memory allocation and deallocation events. Additionally. It monitors system-level performance metrics, including GPU clock speed, power consumption, and thermal data, to provide a comprehensive view of the application's performance. All the collected data is compiled into a detailed report, referred to as a *profile*.

`nvprof` can be configured to generate profiles in either time series or aggregate mode. In time-series mode, it records detailed temporal data, including the execution sequence of GPU kernels, memory operations, and performance metrics associated with each kernel invocation. It is, therefore, useful for analyzing fine-grained sequences of operations.

Unlike time-series, the aggregate provides a high-level view of resource usage and performance distribution. Here, `nvprof` collects metrics including total time spent, number of calls, and average, minimum, and maximum execution times for each type of kernel. Figure 2 illustrates the process of collecting and summarizing various metrics to create an aggregate GPU profile. Table 3 shows partial aggregated GPU profiles generated by `nvprof` for 10 inferences on GoogleNet [65] and RoBERTa [38]. Similarly, it collects average, minimum, and maximum values for various system-level metrics (see Table 4). In this paper, we will only focus on the profiles collected in the aggregate mode of `nvprof`.

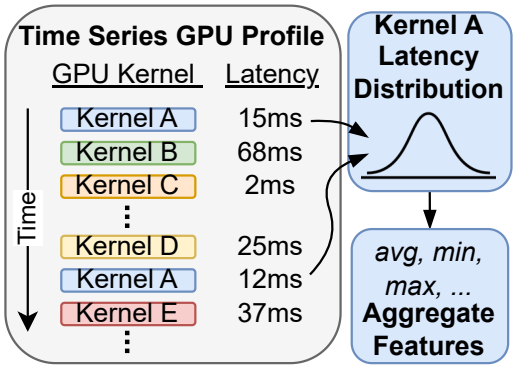


Fig. 2. Collection of aggregate GPU profiles without preserving the kernel execution order.

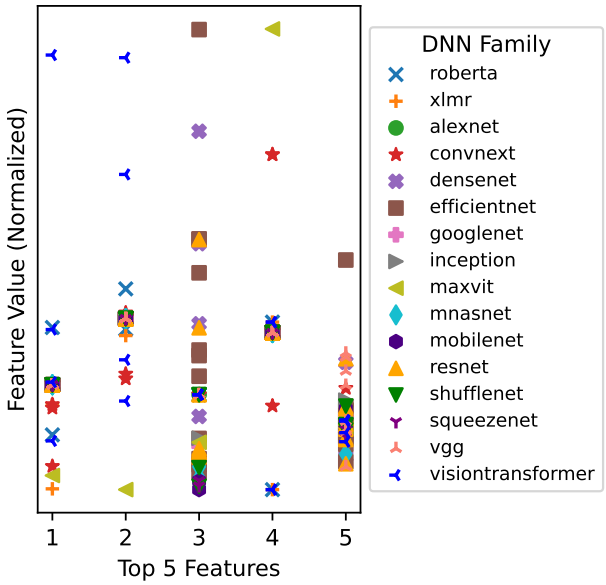
4 Aggregate GPU Profiling

We demonstrate that adversaries can infer DNN architectures by leveraging *aggregate GPU profiles* as a side-channel. While these profiles are simple, non-intrusive, and coarse-grained (in contrast to time-series), they are powerful enough to reveal DNN architectures. Our work stands apart from previous works [26, 29, 78] that rely on a complex analysis of fine-grained time-series data to reconstruct DNN architectures layer-by-layer with a certain degree of uncertainty. This section highlights key insights into the *aggregate* CUDA kernel calls that reveal unique characteristics of DNN architectures.

4.1 Mapping CUDA Kernels to DNN Layers

By mapping CUDA kernel functions to DNN layers, it is possible to identify key characteristics of DNN architectures. For example, matrix multiplication in fully connected layers is executed by dividing the input matrix into smaller blocks and assigning them to GPU threads for parallel computation. Similarly, convolutional layers use kernels that distribute the workload across threads to compute multiple filters simultaneously. NVIDIA's cuDNN library provides tailored functions to optimize operations like convolutions, where the computation strategy may vary based on filter size, *e.g.,* 3×3 vs. 5×5 convolutions. As a result, convolution operations with different filter sizes invoke different kernels.

Specifically, kernels like `volta_sgemm_AxB_tn`, which handle matrix multiplications, can indicate the presence of fully connected layers and their relative sizes. Similarly, kernels such as `void splitKreduce_kernel<int=A, int=B, ...>`[2] provide insights into convolutional operations, including the size and the number of filters. Analyzing low-level kernel calls across multiple iterations can reveal and fingerprint architectural nuances specific to different DNN families. Figure 3 highlights the most common kernels for various DNN families. These kernels, as we ranked them by relevance, reveal the architectural characteristics that distinguish vision models from LLMs, as well as their types.



(1) Time (ms) volta_sgemm_128x64_tn
(2) Avg (us) volta_sgemm_128x64_tn
(3) Time (ms) void splitKreduce_kernel<int=32, int=16, ...
(4) #Calls _ZN2at6native29vectorized_elementwise_kernel...
(5) Min (us) cudaDeviceGetAttribute

Fig. 3. GPU kernel metrics for all 56 TorchVision and 4 TorchText models grouped by their architecture families. The features are ranked from the set of non-memory related GPU kernels using Recursive Feature Elimination on a Random Forest [6] model.

4.2 Profiling CUDA Activities with nvprof

Aggregate profiling offers high-level summary metrics for kernel activities and resource utilization that can fingerprint DNN architectures. Table 4 compares aggregated kernel and API metrics for two popular DNN models: GoogLeNet [65] and RoBERTa [38]. A closer examination of the metrics shows that GoogLeNet exhibits higher usage of

[2]Truncated long kernel name.

Table 4. System-level signals acquired by running 10 inferences of GoogleNet [65] and RoBERTa [38] models on an NVIDIA Tesla T4 GPU using PyTorch framework.

Model	Signal	Count	Avg	Min	Max
GoogLeNet	SM Clock (MHz)	13	563.1	300	585
	Mem Clock (GHz)	13	4.6	0.4	5.0
	Temperature (C)	26	51.5	51	52
	Power (W)	26	27.9	10.7	31.8
RoBERTa	SM Clock (MHz)	31	751.9	300	1125
	Mem Clock (GHz)	31	4.8	0.4	5.0
	Temperature (C)	61	62.1	61	63
	Power (W)	60	34.5	11.8	56.6

convolutional kernels like `cudnn_infer_volta_scudnn_winograd_128...`[3], accumulating over 46.3% of all operations. The higher usage of convolution kernels aligns with the architectural characteristics of the vision models. In contrast, RoBERTa relies more heavily on specialized fully connected operations with kernels like `volta_sgemm_XXXXX_tn`, reflecting the matrix-heavy computations needed for large transformer blocks in language models [68].

Additionally, system-level signals from Table 4 further emphasize the architectural differences between LLMs and vision models. GoogLeNet's lower average SM clock speeds and power consumption reflect its smaller size and fewer trainable parameters, *i.e.,* ~6M [65]. Whereas, RoBERTa exhibits significantly higher power usage and clock rates, corresponding to its larger model size and a much higher number of trainable parameters, *i.e.,* ~355M [14].

Within the same DNN family, variations in activity patterns emerge due to differences in layer depth and applied optimization strategies. For example, GoogLeNet [65] (InceptionV1) and InceptionV3 [66] are both vision models that employ branched convolutions but differ in kernel configurations and architectural complexity. In GoogLeNet, a large portion of GPU time is spent on Winograd convolution [34] kernels, with `cudnn_infer_volta_scudnn_winograd_128x...`[4] accounting for 31.87% of total execution time. Other prominent operations include matrix multiplications using `volta_sgemm_32x32_sliced1x4_nn` (14.43%) and host-to-device memory transfers (8.87%). In contrast, the InceptionV3 profile reveals a significant portion of execution time spent in FFT-based operations like `fft2d_r2c_32x32`, which contributes 39.36% of execution time, and `gemv2T_kernel_val`, a matrix-vector multiplication kernel, accounting for 22.24%. Figure 4 shows a scatter plot comparing kernel runtime ranks for both models. The ranking differences indicate that InceptionV3 employs more specialized and optimized kernels in place of the more generic kernels used in GoogLeNet. These distinctions in kernel usage and time distribution yield unique execution profiles, enabling fingerprinting of model variants even within the same architecture family.

4.3 Key Takeaways

Our analysis demonstrates that aggregate profiling effectively captures distinct execution characteristics of DNN families, such as convolution-heavy vision models or transformer-based language models. For instance, the divergence in kernel usage and system-level metrics between GoogLeNet and RoBERTa underscores the utility of aggregate GPU profiles in identifying architectures across diverse families. Similarly, the higher runtime contribution of FFT-based kernels in InceptionV3 compared to GoogLeNet highlights differences across variants of the same DNN family captured

[3]Truncated long kernel name.
[4]Truncated long kernel name.

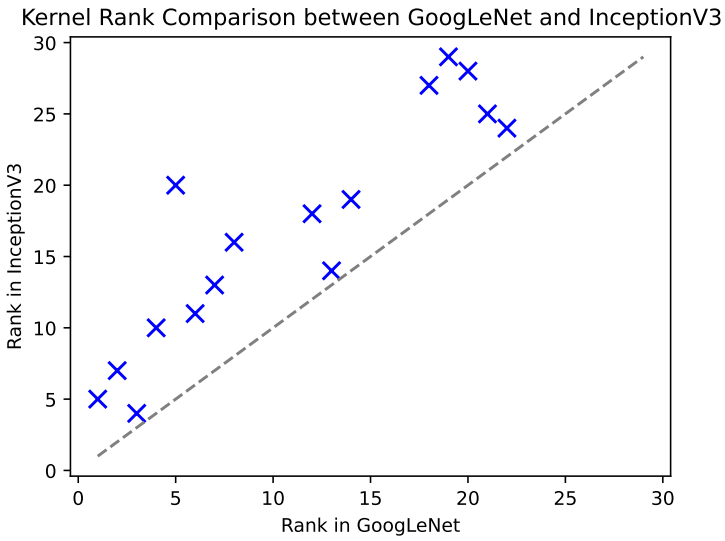


Fig. 4. Rank comparison of kernels used in both GoogLeNet and InceptionV3 models in PyTorch. Points further from the diagonal indicate greater differences in kernel usage between the models.

by GPU profiles. We argue that aggregate GPU profiles can be used as a side-channel for extracting DNN architectures. Our argument is based on two key observations:

(1) Each unique combination of DNN layer type and size maps to a specific CUDA kernel call.
(2) Patterns of kernel calls, along with their execution times and resource utilization, provide unique fingerprinting for DNN architectures.

5 InferNet Design

Using the aforementioned insights, we present InferNet: an attack paradigm to extract the DNN architectures using aggregate GPU profiles as a side-channel (illustrated in Figure 5).

5.1 Threat Model

The adversary aims to conduct an accurate and cost-effective model extraction attack using a coarse-grained and non-intrusive side-channel. We assume that the adversary has black-box access to standard DNN architectures, including state-of-the-art vision and large language models, through an executable file that they can run on a GPU-enabled system. They have no prior knowledge of the target DNN architecture and its parameters, and may only query the model by providing an input x and observing the output $y = f^{\theta}(x)$.

The adversary is assumed to have non-root access to the GPU-enabled system running the target model and access to profiling tools such as `nvprof`. While NVIDIA introduced an administrator-only access flag in later versions of `nvprof` to address security concerns [44], this restriction is not uniformly enforced across cloud platforms. Cloud environments may deploy different versions of the CUDA Toolkit—*e.g.,* CUDA 10.2 uses `nvprof` 10.2, CUDA 11.0 uses `nvprof` 11.0, and CUDA 11.1 onward migrates to Nsight Systems—with the specific profiler version depending on the provider's deployment choices. In some deployments, these environments allow profiling by default for non-root users.

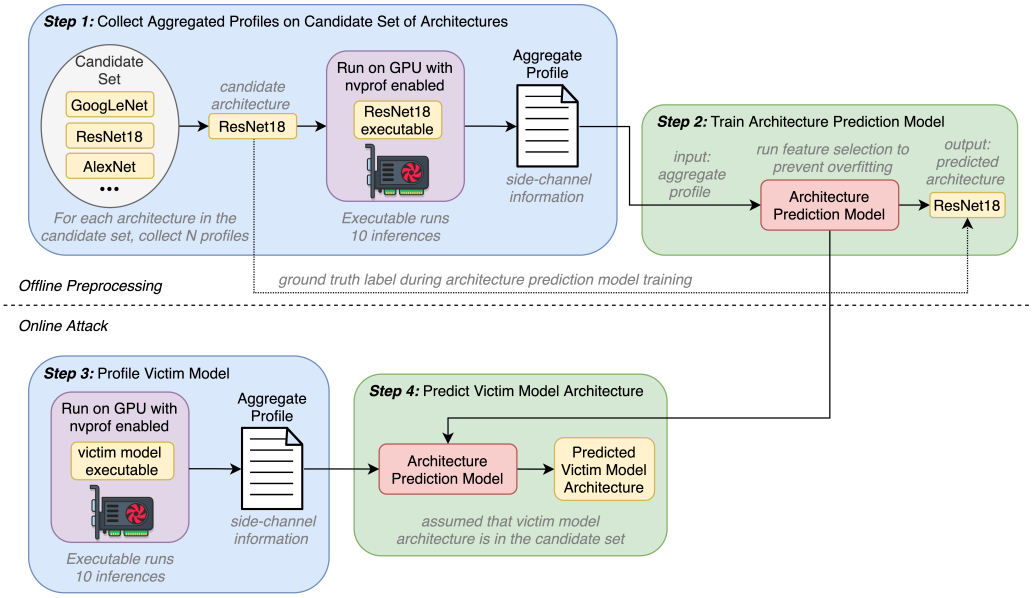


Fig. 5. InferNet Attack Overview. InferNet involves an *offline preprocessing* phase that includes collecting GPU profiles, cleaning the dataset, and training the architecture prediction model, followed by an *online attack* phase. For instance, ResNet18 model is selected from the candidate set C and executed on a GPU for 10 inferences to collect its aggregate GPU profile $\mathcal{H}$. The architecture prediction model $\mathcal{A}$ uses `ResNet18` architecture name as the ground truth to learn the correlations between GPU profile and the architecture.

Additionally, even when newer profilers enforce access restrictions, legacy CUDA toolchains may remain usable in containerized user-space environments [72]. Accordingly, InferNet applies to deployments in which aggregate GPU profiling remains available to the attacker due to platform configuration, legacy software stacks, or equivalent profiling exposure; we do not claim that this access model is universal across all MLaaS platforms.

We assume that the adversary does not tamper with system assets: software or hardware. The attack does not require access to the PCIe bus, operating system (OS), or any internal system components. This makes our attack non-intrusive, which relies solely on profiling data generated during the execution of the target model.

Why aggregate profiles? Aggregate profiles are less sensitive to instrumentation differences across profiler versions. Aggregation suppresses noise arising from non-deterministic GPU scheduling across multiple processes and kernels—GPU's internal scheduling policies may vary between runs, affecting timing and execution order. Aggregate profiles also mitigate variability introduced by thermal throttling and power management, where dynamic changes in performance states (P-states) alter runtime characteristics across inferences. Additionally, aggregate metrics average out fluctuations caused by internal resource contention, such as shared memory and SM-level warp scheduling. These effects introduce instability in fine-grained (time series) profiles, whereas aggregate data provide a more stable and representative fingerprint. Moreover, our feature selection process further reduces the influence of residual noise by retaining only the most predictive components of the profile, enabling high inference accuracy (see section 6) with minimal complexity.

5.2 Architecture Extraction Attack

We formulate a model extraction attack where an adversary infers the architecture of a *victim model* f_v^θ with architecture f_v using aggregate GPU profiles $\mathcal{H}(f_v^\theta)$ as a side-channel. Our attack methodology includes an *offline preprocessing* phase that includes collecting GPU profiles, cleaning the dataset, and training the architecture prediction model, followed by an *online attack* phase (see Figure 5).

Experiment Setup. We take a diverse set of 90 candidate DNN architectures, including widely used LLMs (*e.g.,* GPT [7], BERT [11]), vision models (*e.g.,* ResNet [22], GoogLeNet [65]), and vision transformer models (*e.g.,* MaxViT [67], ViT [12]). The candidate models are *pretrained* and sourced from two major AI/ML frameworks: TensorFlow (v2.17.1) [1], and PyTorch (v2.3.0) [52], where 24 models are implemented in both of the frameworks. We import TensorFlow vision models from `tf.keras.applications`, and LLMs from HuggingFace's AutoModels [14]. Similarly, for PyTorch, we import models from TorchVision (v0.18.0) and TorchText (v0.18.0). The complete list of the models used in this paper is provided in Appendix A.

To further enhance the robustness of our approach, we conduct our experiments on two different NVIDIA GPUs: `Tesla T4` [46] on Google Colaboratory [16], and `Quadro RTX 8000` [45] on a local server (running Ubuntu 24.04). By using two separate experiment setups, we demonstrate that our approach is generalizable to different hardware and software platforms.

A complete summary of the experimental settings used for architecture prediction, including profile collection, feature engineering, classifier configuration, and evaluation metrics, is provided in Appendix B.

Collecting Aggregate Profiles. We define a candidate architectures set $C = \{f_{c_1}, f_{c_2}, ...f_{c_n}\}$ comprising popular large language models (LLMs) and vision models. For each candidate $f_{c_i} \in C$, we collect $N = 20$ aggregate profiles $\mathcal{H}(f_{c_i}^{\theta_i})$ using `nvprof` in *aggregate* mode to construct a data sample $S_i = \{\mathcal{H}(f_{c_i}^{\theta_i}), f_{c_i}\}^N$. Every profile contains combined metrics, such as kernel runtimes, memory operations, and system-level resource utilization, for ten (10) inferences with `zero` input. We avoid batch execution to reflect realistic attack conditions where the batch sizes are unknown to the adversary.

We consolidate all the profiles from the candidate architectures into a comprehensive training dataset $D = \bigcup_i S_i$. With 90 candidates and 2 GPU platforms, our dataset accumulates $|D| = 90 \times 2 \times 20 = 3600$ GPU profiles representing 36,000 inference runs. This dataset represents the computational characteristics of the architectures in the candidate set C, where each kernel name serves as a distinct *feature* for the architecture's prediction model.

Data Cleaning. In the combined dataset D, some GPU kernels and memory operations appear only for specific architectures; therefore, creating a sparse dataset. We address this by adding binary *indicator* columns for each kernel that denote the kernel's presence or absence in a given profile. Then, we fill any missing values with the mean of the corresponding feature across the dataset. Finally, we account for the different magnitudes of metrics by normalizing and scaling all the features in the dataset.

We perform *feature selection* to improve the signal-to-noise ratio and reduce the risk of overfitting. In this regard, we use Recursive Feature Elimination (RFE) to identify and retain the most relevant features. The resulting trained prediction model creates a more reliable mapping between aggregate profiles and the target architecture f_v.

Training an Architecture Prediction Model. We train an architecture prediction model $\mathcal{A}$ on the clean dataset D to learn the mapping between aggregate GPU profiles $\mathcal{H}$ and their corresponding architectures f_{c_i}. Formally, $\mathcal{A}$ attempts to solve the problem:

$$\mathcal{A}(\mathcal{H}(f_v^\theta)) = \mathbb{E}[f_v|\mathcal{H}(f_v^\theta)] = \hat{f_v} \tag{1}$$

where $\hat{f_v}$ is the predicted architecture of the victim model, and $\mathcal{A}$ is typically a machine learning model. The training process uses simple supervised classifier models, *i.e.,* Logistic Regression, Neural Network, KNN, Nearest Centroid, Naive Bayes, Random Forest, and AdaBoost. Using multiple models allows us to optimize the attack for different architectures.

Online Attack. The online attack phase focuses on extracting the architecture of an unknown victim model. We use a binary executable to run the victim model f_v^θ on a GPU-enabled system, and use `nvprof` to generate an *aggregate* profile $\mathcal{H}(f_v^\theta)$. This profile is fed into the trained prediction model $\mathcal{A}$, which outputs a predicted architecture $\hat{f_v} = \mathcal{A}(\mathcal{H}(f_v^\theta))$.

6 InferNet Attack Evaluation

In this section, we assess the accuracy of InferNet across various attack scenarios. We begin by evaluating a practical attack scenario using vanilla settings. Next, we optimize the attack, enhancing its resilience across diverse use cases. Finally, we demonstrate the scalability of the attack with respect to future DNN architectures. For model training, unless otherwise stated, we used the default hyperparameters of the corresponding scikit-learn implementation, with the exceptions listed in Appendix B.

6.1 Practical Attack

We begin by evaluating whether the aggregate GPU profiles can reveal DNN architectures in a *practical* scenario with standard settings. In this setting, the victim model f_v^θ and candidate model $f_{c_i}^{\theta_i}$ have the same number of output classes. The profiles are split into two datasets: D_1 for training (75%) and D_2 for testing (25%). Dataset D_1 represents profiles collected during the *offline preprocessing* phase of the attack, while D_2 simulates victim profiles collected for the *online attack*. We stratify the data by the DNN architecture names to ensure that each DNN architecture is represented in both the training and test datasets. We train seven (7) architecture prediction models $\mathcal{A}$ on D_1 and evaluate their performance on D_2. Each model is trained using different subsets of profile features (*e.g.,* GPU kernel features, API call features) as shown in Table 5.

Key Findings. Our key findings are (refer to Table 5):
Effectiveness of architecture prediction models: The results confirm that aggregated profiles can effectively reveal victim architectures in a common scenario. Most models, including Neural Network, K Nearest Neighbors, Naive Bayes, and Random Forest, achieve near-perfect accuracy ($\geq$ 96%) on D_2 when trained on all 849 features. Models like Random Forest, Naive Bayes, and K Nearest Neighbors exhibit robustness even when trained on reduced feature sets.
Impact of feature selection: GPU kernel features are the most critical for architecture prediction. Models trained only on GPU kernel features achieve accuracy comparable to those trained on all features. API call features and system-level metrics contribute less, as training exclusively on these features results in significant accuracy drops.
Overfitting on system features: We observe spurious correlations in system-level metrics that do not generalize across datasets. Models trained only on the system features overfit on the training data D_1. For example, K Nearest Neighbors

Table 5. Top1 Train (D_1) / Test (D_2) accuracy of architecture prediction models $\mathcal{A}$ by the subset of profile features used to train $\mathcal{A}$. System features, GPU kernels, and API calls correspond to the system signals, GPU kernels, and API calls from Table 3 and Table 4, respectively; whereas, Indicator features denote the presence or absence of a profile feature. Train and test datasets D_1 and D_2 contain profiling data for all 56 TorchVision and 4 TorchText models.

Architecture Prediction Model $\mathcal{A}$	**Subset of Profile Features Used to Train Architecture Prediction Model**						
	All (849)	**System (27)**	**No System (822)**	**GPU Kernel (447)**	**API Calls (375)**	**Indicator (68)**	**No Indicator (781)**
Logistic Regression	99.9/99.1	36.3/31.5	99.9/99.2	99.3/99.4	94.2/83.4	57.4/52.8	99.9/99.0
Neural Network	100/99.1	65.9/58.7	100/98.7	100/99.9	99.4/84.3	57.1/52.3	100/98.8
K Nearest Neighbors	100/96.0	100/47.3	100/95.8	100/98.8	100/66.4	56.2/55.5	100/95.3
Nearest Centroid	99.3/97.8	34.6/32.8	99.1/97.7	98.4/98.5	81.3/71.2	55.5/57.4	99.0/97.8
Naive Bayes	100/99.7	47.4/43.6	100/99.9	100/99.8	100/99.6	57.2/52.0	100/99.7
Random Forest	100/100	100/61.3	100/100	100/100	100/100	56.7/52.4	100/100
AdaBoost	42.5/41.8	6.11/4.02	45.8/44.9	46.7/48.6	24.7/20.8	15.4/12.3	37.5/36.6

achieves 100% accuracy on D_1 but drops to 47.3% on D_2. Similarly, the accuracy for Random Forest, the best model overall, drops from 100% to 61.3%. These results suggest that system-level metrics, may reflect specific runtime conditions, hardware configurations, or workload distributions that are unique to the input data and not the DNN architecture.

Impact of memory management kernels: Memory-related GPU kernels, such as `CUDA_memcpy_HtoD`, are less informative for architecture prediction. Removing memory management features from the GPU kernel subset causes minimal ($\leq$ 1%) accuracy loss for all models except AdaBoost. This highlights that memory operations contribute little to distinguishing architectures.

6.2 Attack Optimization for Sparse Data

Since not all features strongly indicate the DNN architecture, we investigate how small a subset of features is sufficient for high prediction accuracy. We focus primarily on the 450 *non-memory* GPU kernel features (see *Finding 2* in § 6.1). Using Recursive Feature Elimination (RFE) [17], we rank the kernel features based on their importance and train the architecture prediction models $\mathcal{A}$ on D_1 from the baseline attack (subsection 6.1) using only the top features.

Key Findings. Our results, illustrated in Figure 6, validate the hypothesis that a small number of features can achieve comparable accuracy to models trained on the full feature set. These findings highlight the importance of feature selection in choosing a minimal feature subset, mitigating overfitting, and reducing computational overhead.

The top features are highly effective: Models like K Nearest Neighbors, Nearest Centroid, Naive Bayes, and Random Forest achieve the same high accuracy (~100%) using only the top 3 features as they do with over 450 features. The top three features, shown in Figure 3, include `Time (ms) volta_sgemm_128x64_tn`, `Avg (us) volta_sgemm_128x64_tn`, and `Time (ms) void splitKreduce_kernel<int=A, int=B, ...>`[5].

Feature requirements vary by prediction model $\mathcal{A}$: Neural Networks and Logistic Regression require more features, *i.e.,* 17 and 22, respectively, to achieve peak accuracy. This suggests that not all models capture the same activity patterns and therefore prioritize different feature sets for predictions.

[5]Truncated long kernel name.

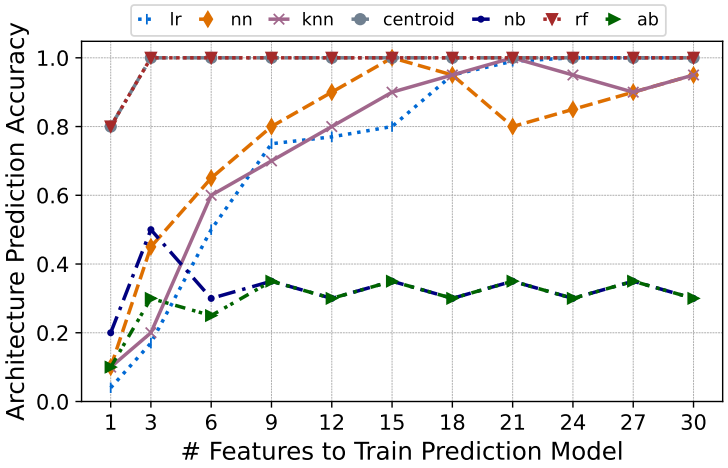


Fig. 6. Top1 accuracy of architecture prediction models $\mathcal{A}$ on profiles in D_2 by number of GPU kernel features used to train $\mathcal{A}$.

The top-kernel features are informative because the paper's mapping analysis ties kernels to layer structure and variant-specific execution behavior; therefore, feature selection suppresses environment-specific noise rather than discovering arbitrary correlations.

6.3 Attack Resilience Against Modified Architectures

We extend our attack to a more challenging scenario where the victim model f_v^θ is modified by additional final layer(s) with a different number of output classes than the original base model. We simulate this scenario by taking a subset of $D_1\prime$ from subsection 6.1, containing only the 56 TorchVision models. The $D_1\prime$ dataset represents candidate models *pretrained* on ImageNet [10] with 1000 output classes. However, we construct a new dataset D_3 with vision models trained in CIFAR10 [32] with only 10 output classes.

Key Findings. For evaluation, we train the prediction models $\mathcal{A}$ on $D_1\prime$, and test them on D_3. The features used for training include the top 3, top 25, and all 450 non-memory GPU kernel features identified in § 6.2. Our findings, from Table 6, are as follows:

High correlation with top features: The top 3 features: (i) `Time (ms) volta_sgemm_128x64_tn`, (ii) `Avg (us) volta_sgemm_128x64_tn`, and (iii) `Time (ms) void splitKreduce_kernel<int=A, int=B, ...>`[6], continue to show strong predictive power. This suggests that these kernel features are highly correlated with the architectural characteristics of DNNs and are robust to variations in the number of output classes.

Performance gain with top 25 features: Expanding the feature set to the top 25 features improves the performance of Logistic Regression and Neural Network models, with accuracy on the test set D_3 of 100%, and 95.2%, respectively. Therefore, with additional features available, a broader range of models can be used for architecture prediction with high confidence.

[6]Truncated long kernel name.

Table 6. Top1 Train (D_1′) / Test (D_3) architecture prediction accuracy for architecture prediction models $\mathcal{A}$ by the top 3, 25, and 450 GPU kernel features (without memory data) used to train $\mathcal{A}$. D_1′ consists of profiles generated by models trained on ImageNet, and D_2 consists of profiles generated by models trained on CIFAR10.

Architecture Prediction Model $\mathcal{A}$	**Top 3 Features**	**Top 25 Features**	**All 450 Features**
Logistic Regression	38.2/40.5	99.9/100	97.6/58.7
Neural Network	20.3/18.9	100/95.2	100/61.4
K Nearest Neighbors	100/100	100/100	100/85.9
Nearest Centroid	100/100	100/100	96.8/81.4
Naive Bayes	98.7/94.3	100/100	100/63.5
Random Forest	100/100	100/100	100/100
AdaBoost	15.4/17.6	37.9/35.4	28.6/26.3

Overfitting with all features: Using all 450 non-memory kernel features introduces overfitting for models that prioritize memory as well as non-memory kernels. In particular, the accuracies for Logistic Regression and Neural Networks drop to 58.7% and 61.4%, respectively, on D_3. This performance is far lower than their predictions using all GPU kernels (see Table 5), with accuracies of 99.4%, and 99.9%, respectively. Random Forest, however, remains robust and consistently achieves 100% accuracy.

The results confirm that our attack method remains effective even when the DNN is fine-tuned or modified. This demonstrates that GPU kernel features serve as a reliable side channel for extracting DNN architectures, regardless of changes to the model architecture.

The high predictive value of the top-ranked features is consistent with the kernel-to-layer mapping discussed in Section 4: these kernels are not arbitrary correlates, but compact summaries of architecture-dependent operations. At the same time, our results show that indiscriminate inclusion of many low-value features can reduce transfer performance, which is why we treat full-feature results as an ablation rather than as a primary attack configuration.

6.4 Scaling Attack to Future Architectures

Adding new candidate DNN architectures to the attack's scope requires additional profiles and significant computation for retraining the prediction model $\mathcal{A}$. For reference, generating a single profile for DNN execution takes approximately 45-120 seconds, and generating the complete dataset D_1 requires 20+ hours. In contrast, the *online attack* phase is computationally inexpensive, with less than one minute needed to profile the victim model and predict its architecture. Reducing the time spent in the *offline preprocessing* phase is critical for enhancing the practicality of the attack.

We analyze the computational efficiency of the InferNet attack by reducing the training dataset size – and therefore, the time for collecting profiles – without compromising the accuracy. We hypothesize that fewer profiles per architecture are sufficient for training accurate architecture prediction models. To test this, we train the prediction models $\mathcal{A}$ on subsets of D_1′ with a varying number of profiles per candidate architecture. We limit the training to the top 25 GPU kernel features identified in subsection 6.3. Finally, we test $\mathcal{A}$ on D_3 (profiles of victim models trained on CIFAR10) to evaluate the impact of dataset size reduction.

Key Findings. Figure 7 confirms that architecture prediction models can generalize effectively with minimal training data. For all models except Logistic Regression and AdaBoost, a single profile per candidate architecture is sufficient to

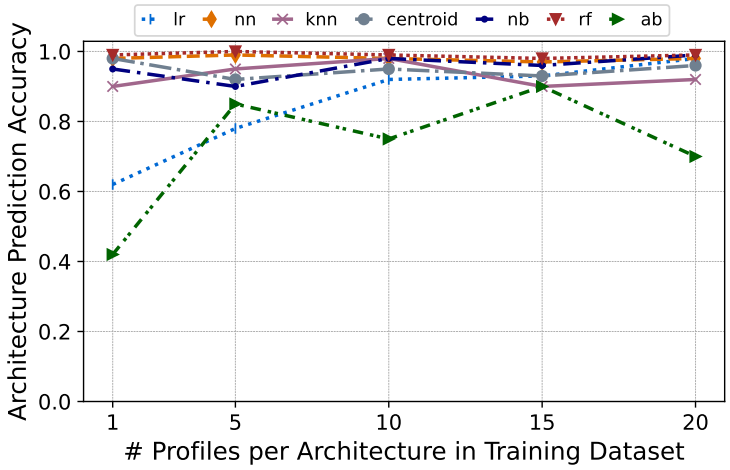


Fig. 7. Top1 accuracy of architecture prediction models $\mathcal{A}$ on profiles in dataset D_3 by number of profiles per candidate architecture from D_1 used to train $\mathcal{A}$.

achieve the same accuracy as training on the full dataset of 20 profiles. Similarly, these models show very low variance for the top 25 features which indicates that repeated executions of DNN models often result in identical GPU profiles.

Reducing the training data requirements offers significant practical advantages. Collecting just one profile per architecture reduces the *offline preprocessing* phase to under 30 minutes (*i.e.,* 97.5% time reduction). Similarly, adding a new architecture to the candidate set requires only one additional profile or just over one minute to fine-tune the attack. These findings highlight the scalability of InferNet and its adaptability to expanding candidate model pools.

7 Generalizability Evaluation of InferNet

While our evaluation results in Section 6 indicate that InferNet is reliably architecture-agnostic; it is strongest when the attacker can match or approximate the victim's software and hardware stack. This section evaluates the generalizability of InferNet across various deployment scenarios, including pruned DNNs, different GPU hardware, and multiple AI/ML frameworks.

7.1 Evaluation Over Pruned Models

DNNs are often pruned to optimize inference performance and reduce computational overhead. To evaluate whether InferNet is effective against such models, we create a dataset D_p using profiles of 56 TorchVision models trained on CIFAR10 [32] and pruned to 50% capacity. Architecture prediction models $\mathcal{A}$ are trained on the top 25 non-memory GPU kernel features from the unpruned dataset D_1 (from § 6.1) and tested on D_p. This setup also simulates the scenario where the DNN architecture is modified and fine-tuned with a different number of output classes (see §6.3).

Key Findings. The results, as shown in Table 7, demonstrate that InferNet generalizes well to pruned models with minimal performance degradation. Random Forest and Naive Bayes models achieve 100% architecture prediction accuracy. However, the rest of the models show a slight performance decline from the non-pruned test set (see Table 6). Performance for some of the models, like Logistic Regression and Neural Network, improves significantly in Top5

Table 7. Top1/Top5 test accuracy of architecture prediction models trained on profiles from unpruned (ImageNet) models and tested on profiles of pruned (CIFAR10) models.

Architecture Prediction Model $\mathcal{A}$	**Top1 %**	**Top5 %**	**Family %**
Logistic Regression	94.2	99.7	100
Neural Network	81.3	90.9	87.5
KNN	96.8	97.5	100
Nearest Centroid	97.5	96.9	100
Naive Bayes	100	100	100
Random Forest	100	99.9	100
AdaBoost	30.4	41.3	46.1

Table 8. Top1/Top5 and Architecture Family test accuracy of architecture prediction models trained on profiles from one GPU and tested on profiles from another GPU. Training data consists of profiles of candidate models trained on ImageNet, while test data consists of profiles of candidate models trained on CIFAR10.

Architecture Prediction Model $\mathcal{A}$	**$\text{Quadro}_{\text{train}}$ $\text{Tesla}_{\text{test}}$**		**$\text{Tesla}_{\text{train}}$ $\text{Quadro}_{\text{test}}$**	
	Top1/Top5	*Fam.*	*Top1/Top5*	*Fam.*
Logistic Regression	34.1/74.2	78.5	28.5/56.8	48.1
Neural Network	10.7/28.3	32.5	12.0/25.5	31.1
KNN	35.2/42.5	74.1	28.1/26.8	51.2
Nearest Centroid	35.6/42.0	76.3	28.3/28.5	50.5
Naive Bayes	71.4/73.5	96.7	74.6/74.1	96.3
Random Forest	42.5/74.1	81.4	45.3/78.6	59.7
AdaBoost	6.2/28.5	26.1	34.1/62.8	65.2

evaluation, indicating that the true architecture is often among the top five predictions. We also include DNN Family prediction to analyze if the prediction models $\mathcal{A}$ recognize the DNN family, if they fail to recognize the architecture itself. Most models, except AdaBoost, deliver reliable performance, with many achieving 100% accuracy in predicting the DNN family.

7.2 Evaluation Across GPUs

In practice, a diverse range of GPUs is available for training and running DNN models. Therefore, it is crucial to establish the effectiveness of the attack when the adversary does not have easy access to the same GPU for both offline and online phases.

We collect two datasets $D_2\prime$ and $D_3\prime$ using profiles `NVIDIA Quadro RTX 8000` GPU; whereby, $D_2\prime$ contains profiles of candidate models trained on ImageNet [10] and $D_3\prime$ contains profiles of models trained on CIFAR10 [32]. Additionally, we re-use the datasets $D_1\prime$, D_2, and D_3 collected from `NVIDIA Tesla T4` GPU in § 6.1 and § 6.3. Both datasets use the top 25 non-memory GPU kernel features for training and testing. For cross evaluation, training data consists of profiles from one GPU, while the test data comes from the other GPU, and vice versa.

Key Findings. The results from Table 8 show that InferNet achieves varying levels of accuracy across GPUs. For example, Logistic Regression achieves only 34.1% Top1 accuracy when trained on Quadro and tested on Tesla, and

Table 9. Top1/Top5 and Family accuracy of architecture prediction models trained on profiles from one ML framework (PyTorch) and tested on profiles from another (TensorFlow), and vice versa.

Architecture Prediction Model $\mathcal{A}$	**$\text{PyTorch}_{\text{train}}$ $\text{TensorFlow}_{\text{test}}$**		**$\text{TensorFlow}_{\text{train}}$ $\text{PyTorch}_{\text{test}}$**	
	Top1/Top5	*Fam.*	*Top1/Top5*	*Fam.*
Logistic Regression	26.1/50.3	67.4	20.8/51.7	41.3
Neural Network	9.3/22.8	27.6	11.2/22.1	29.4
KNN	43.5/52.2	82.7	36.7/36.3	59.9
Nearest Centroid	32.8/41.7	70.1	26.8/32.4	48.9
Naive Bayes	58.2/60.3	80.0	64.1/65.4	78.3
Random Forest	51.6/63.2	91.0	41.1/54.3	66.9
AdaBoost	14.1/27.3	32.6	17.9/29.5	51.7

even lower accuracy when reversed. Neural Network accuracy is similarly low, at 10.7% and 12% Top1 accuracy in both cross-GPU scenarios, respectively. However, models like Naive Bayes and Random Forest demonstrate stronger cross-GPU transfer across the two evaluated NVIDIA GPUs, with Naive Bayes achieving 71.4% Top1 accuracy when trained on Quadro and tested on Tesla, and 74.6% when trained on Tesla and tested on Quadro. Interestingly, even when architecture prediction accuracy drops, family prediction accuracy remains relatively high for most models.

Our hardware selection of Tesla T4 and Quadro RTX 8000 represents two widely used deployment tiers within NVIDIA's Turing generation: a data-center accelerator and a high-end workstation GPU. The evaluation reinforces that InferNet exploits fundamental architectural behaviors rather than specific hardware instance noise. Successfully transferring the attack between two vastly different operational tiers of the Turing architecture, *i.e.,* the T4 and the RTX 8000, indicates that the aggregate profile acts as a robust, hardware-agnostic side-channel within an architectural generation.

Analysis of the combined data shows that the GPU kernels used by DNN architectures are largely consistent across GPUs. This consistency enables some generalizability. However, the observed performance drop highlights the need for further tuning or additional training data to improve the cross-GPU transferability of InferNet.

7.3 Evaluation Across ML Frameworks

Given the effectiveness of InferNet in various scenarios, the next important question is whether the adversary must know the framework used to implement the victim model or if architecture prediction models can be generalized across frameworks. To evaluate this, we create a dataset of GPU aggregate profiles for 24 vision DNNs implemented in TensorFlow v2.17.1 (see Appendix A). We choose those DNNs that are also available in PyTorch, and they are part of our $D_1\prime$ profiles dataset. To collect 20 profiles each for the TensorFlow models to construct a dataset D_t. For evaluation, we train using profiles from one framework and test with profiles from the other framework, and vice versa. The results indicate that TensorFlow DNNs utilize a largely *disjoint* set of GPU kernels compared to PyTorch. This results in a significant accuracy drop when using PyTorch-trained models to predict TensorFlow architectures (see Table 9).

In cross-framework evaluations (PyTorch to TensorFlow), while raw Top-1 accuracy drops, the security implication remains severe. The ability to identify the model family (e.g., VGG) reduces the architectural search space for an adversary from thousands of possibilities to a handful of known variants. This facilitates targeted model-stealing, as the

Table 10. Top1/Top5 test accuracy of architecture prediction models trained on combined PyTorch and TensorFlow profiles.

Architecture Prediction Model $\mathcal{A}$	Top1 %	Top5 %	Family %
Logistic Regression	93.1	98.6	100
Neural Network	84.4	91.7	92.3
KNN	96.5	96.9	100
Nearest Centroid	95.7	95.8	100
Naive Bayes	100	100	100
Random Forest	100	100	100
AdaBoost	32.8	43.2	48.7

adversary can focus their resources on a narrow set of probable architectures once the family is confirmed.

Optimization. To address the poor performance, we create a heterogeneous dataset by appending PyTorch profiles with the TensorFlow profiles. Since these two frameworks are the most prevalent, this dataset generation step is typically required only once, as it is unlikely that a new framework will significantly challenge their importance. With heterogeneous training data, Random Forest achieves 100% Top1 accuracy, while Naive Bayes and K Nearest Neighbors exceed 95% (see Table 10). This confirms that adding profiles from multiple frameworks can effectively remove the dependency on a specific framework and, therefore, ensure generalization across frameworks.

8 Proposed Countermeasures

To defend against side-channel architecture extraction attacks like InferNet, our recommendations to disrupt or mitigate the attack surface include:

Dummy Computations: Adding dummy GPU workload—such as no-op kernels or randomized matrix operations—can disrupt the signal-to-noise ratio in the side-channel data [24, 26, 29, 42, 53, 78]. Such computations should vary across inference runs to prevent adversaries from filtering them out via differential analysis. This defense impacts various GPU kernel features, such as average runtime, total runtime, and kernel invocation counts. Therefore, it requires careful design, as perturbing some features (*e.g.,* average time spent per kernel invocation) can incur significant performance overhead.

Randomized Kernel Scheduling: Introducing randomized scheduling between dependent or independent GPU kernels can distort execution time distributions without altering model behavior. Such noise makes it harder for adversaries to associate specific runtime patterns with architectural components. Timing jitter and minor delay injection, implemented at the framework or driver level, can increase feature variance and reduce prediction accuracy for attacks relying on stable kernel profiles.

Limited Precision in GPU Profiling Tools: Restricting the precision of profiling information (*e.g.,* truncating runtime metrics or kernel counts) can reduce the value of GPU profiles to adversaries [44, 72]. This approach is especially effective against statistical learning methods, which rely on feature importance derived from fine-grained runtime behavior. However, tuning the defense requires careful assessment of the impact on the legitimate use of profiling tools for debugging and performance optimization.

Monitor and Restrict Access to Profiling Tools: Access to tools like `nvprof` should be restricted to trusted users with administrative privileges [44, 72]. Enabling administrator-only flags in profiling tools can prevent adversaries from

leveraging these tools for side-channel attacks. Similarly, inference services should audit profiler invocations and implement rate-limiting or sandboxing to detect repeated or automated profiling attempts.

9 Conclusion

This paper presents InferNet, a novel attack method that uses simple, non-intrusive GPU performance metrics to predict the underlying DNN architecture of a victim application. Unlike traditional model extraction techniques, which rely on fine-grained data or full access to the model's parameters, InferNet only requires high-level system metrics such as GPU kernel calls, memory usage, and clock speed, making it both practical and stealthy. Our empirical evaluation demonstrates that InferNet achieves high accuracy across a diverse set of DNN architectures, including vision models, large language models, and transformers, with minimal data and even under model compression or optimization. The method is robust across multiple AI/ML frameworks and GPU platforms, providing a powerful tool for identifying DNN architectures without requiring root access or detailed model parameters.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/.

[2] Sayed Erfan Arefin and Abdul Serwadda. 2021. Deep neural exposure: You can run, but not hide your neural network architecture!. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*. 75–80.

[3] Baidu. 2024. Apollo Open Platform. https://developer.apollo.auto/developer.html.

[4] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. {CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*. 515–532.

[5] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2021. Sca strikes back: Reverse engineering neural network architectures using side channels. *IEEE Design and Test* (2021).

[6] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[8] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. 2020. Cryptanalytic Extraction of Neural Network Models. In *Advances in Cryptology - CRYPTO (Lecture Notes in Computer Science, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, 189–218. doi:10.1007/978-3-030-56877-1_7 https://doi.org/10.1007/978-3-030-56877-1_7.

[9] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. 2020. Generative pretraining from pixels. In *International conference on machine learning*. PMLR, 1691–1703.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR*. IEEE Computer Society, 248–255. doi:10.1109/CVPR.2009.5206848 https://doi.org/10.1109/CVPR.2009.5206848.

[11] Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[12] Alexey Dosovitskiy. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[13] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *Journal of Machine Learning Research* 20, 55 (2019), 1–21.

[14] Hugging Face. 2019. Pretrained models — transformers 2.4.0. https://huggingface.co/transformers/v2.4.0/pretrained_models.html.

[15] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. 2018. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *Journal of Cryptographic Engineering* 8, 1 (2018), 1–27.

[16] Google. 2024. Google Colaboratory. https://colab.research.google.com/.

[17] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. 2002. Gene Selection for Cancer Classification using Support Vector Machines. *Mach. Learn.* 46, 1-3 (2002), 389–422. doi:10.1023/A:1012487302797 https://doi.org/10.1023/A:1012487302797.

[18] William Hackett, Stefan Trawicki, Zhengxin Yu, Neeraj Suri, and Peter Garraghan. 2023. PINCH: An Adversarial Extraction Attack Framework for Deep Learning Models. arXiv:2209.06300 [cs.CR] https://arxiv.org/abs/2209.06300

[19] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. *CoRR* abs/1506.02626 (2015). arXiv:1506.02626 http://arxiv.org/abs/1506.02626.

[20] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. 2021. Pre-trained models: Past, present and future. *AI Open* 2 (2021), 225–250.

[21] Janmanchi Harika, Palavadi Baleeshwar, Kummari Navya, and Hariharan Shanmugasundaram. 2022. A review on artificial intelligence with deep human reasoning. In *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*. IEEE, 81–84.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[23] Yingzhe He, Guozhu Meng, Kai Chen, Xingbo Hu, and Jinwen He. 2020. Towards security threats of deep learning systems: A survey. *IEEE Transactions on Software Engineering* 48, 5 (2020), 1743–1770.

[24] Sanghyun Hong, Michael Davinroy, Yigitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitras. 2018. Security Analysis of Deep Neural Networks Operating in the Presence of Cache Side-Channel Attacks. *CoRR* abs/1810.03487 (2018). arXiv:1810.03487 http://arxiv.org/abs/1810.03487.

[25] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*. 1314–1324.

[26] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. 2020. DeepSniffer: A DNN Model Extraction Framework Based on Learning Architectural Hints. In *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne*, James R. Larus, Luis Ceze, and Karin Strauss (Eds.). ACM, 385–399. doi:10.1145/3373376.3378460 https://doi.org/10.1145/3373376.3378460.

[27] Weizhe Hua, Zhiru Zhang, and G Edward Suh. 2018. Reverse engineering convolutional neural networks through side-channel information leaks. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.

[28] IEEE. 2022. The Radical Scope of Tesla's Data Hoard. https://spectrum.ieee.org/tesla-autopilot-data-scope.

[29] Nandan Kumar Jha, Sparsh Mittal, Binod Kumar, and Govardhan Mattela. 2020. DeepPeep: Exploiting Design Ramifications to Decipher the Architecture of Compact DNNs. *ACM* 17, 1 (2020), 5:1–5:25. doi:10.1145/3414552 https://doi.org/10.1145/3414552.

[30] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. 2015. An open approach to autonomous vehicles. *IEEE Micro* 35, 6 (2015), 60–68.

[31] Kalpesh Krishna, Gaurav Singh Tomar, Ankur P Parikh, Nicolas Papernot, and Mohit Iyyer. 2019. Thieves on sesame street! model extraction of bert-based apis. *arXiv preprint arXiv:1910.12366* (2019).

[32] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009). https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems*, Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger (Eds.). 1106–1114. https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[34] Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4013–4021.

[35] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[36] Jiacheng Liang, Ren Pang, Changjiang Li, and Ting Wang. 2025. Model Extraction Attacks Revisited. arXiv:2312.05386 [cs.LG] https://arxiv.org/abs/2312.05386

[37] Sihang Liu, Yizhou Wei, Jianfeng Chi, Faysal Hossain Shezan, and Yuan Tian. 2019. Side Channel Attacks in Computation Offloading Systems with GPU Virtualization. In *2019 IEEE Security and Privacy Workshops, SP*. IEEE, 156–161. doi:10.1109/SPW.2019.00037 https://doi.org/10.1109/SPW.2019.00037.

[38] Yinhan Liu. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* 364 (2019).

[39] Yuntao Liu and Ankur Srivastava. 2020. GANRED: GAN-based Reverse Engineering of DNNs via Cache Side-Channel. In *CCSW'20, Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, Yinqian Zhang and Radu Sion (Eds.). ACM, 41–52. doi:10.1145/3411495.3421356 https://doi.org/10.1145/3411495.3421356.

[40] Yuntao Liu and Ankur Srivastava. 2020. Ganred: Gan-based reverse engineering of dnns via cache side-channel. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*. 41–52.

[41] Yangdi Lyu and Prabhat Mishra. 2018. A survey of side-channel attacks on caches and countermeasures. *Journal of Hardware and Systems Security* 2, 1 (2018), 33–50.

[42] Henrique Teles Maia, Chang Xiao, Dingzeyu Li, Eitan Grinspun, and Changxi Zheng. 2022. Can one hear the shape of a neural network?: Snooping the GPU via Magnetic Side Channel. In *31st USENIX Security Symposium, USENIX*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 4383–4400. https://www.usenix.org/conference/usenixsecurity22/presentation/maia.

[43] Erich Malan, Valentino Peluso, Andrea Calimera, and Enrico Macii. 2023. Enabling DVFS Side-Channel Attacks for Neural Network Fingerprinting in Edge Inference Services. In *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 1–6.

[44] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael B. Abu-Ghazaleh. 2018. Rendered Insecure: GPU Side Channel Attacks are Practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 2139–2153. doi:10.1145/3243734.3243831 https://doi.org/10.1145/3243734.3243831.

[45] Nvidia. 2019. Nvidia Quadro RTX 8000 GPU Datasheet. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/quadro-rtx-8000-us-nvidia-946977-r1-web.pdf.

[46] Nvidia. 2019. Nvidia Tesla T4 GPU Datasheet. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf.
[47] Nvidia. 2019. Profiler user's guide. https://docs.nvidia.com/cuda/archive/10.2/profiler-users-guide/index.html.
[48] Nvidia. 2024. cuBLAS. https://developer.nvidia.com/cublas.
[49] Nvidia. 2024. CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit.
[50] Nvidia. 2024. cuDNN: CUDA Deep Neural Network. https://developer.nvidia.com/cudnn.
[51] Kyoung-Su Oh and Keechul Jung. 2004. GPU implementation of neural networks. *Pattern Recognition* 37, 6 (2004), 1311–1314.
[52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019). https://pytorch.org/.
[53] Kartik Patwari, Syed Mahbub Hafiz, Han Wang, Houman Homayoun, Zubair Shafiq, and Chen-Nee Chuah. 2022. DNN Model Architecture Fingerprinting Attack on CPU-GPU Edge Devices. In *7th IEEE European Symposium on Security and Privacy, EuroS&P*. IEEE, 337–355. doi:10.1109/EuroSP53844.2022.00029 https://doi.org/10.1109/EuroSP53844.2022.00029.
[54] Alec Radford. 2018. Improving language understanding by generative pre-training. (2018).
[55] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *International conference on machine learning*. Pmlr, 8821–8831.
[56] Precedence Research. 2024. Machine Learning as a Service Market Size, Share, and Trends 2024 to 2034. https://www.precedenceresearch.com/machine-learning-as-a-service-market.
[57] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.
[58] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.
[59] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).
[60] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.1556.
[61] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. 2022. Make-a-video: Text-to-video generation without text-video data. *arXiv preprint arXiv:2209.14792* (2022).
[62] Stanford Institute for Human-Centered Artificial Intelligence. 2024. *AI Index: State of AI in 13 Charts*. https://hai.stanford.edu/news/ai-index-state-ai-13-charts Accessed: 2025-10-01.
[63] Yuchen Sun, Tianpeng Liu, Panhe Hu, Qing Liao, Shaojing Fu, Nenghai Yu, Deke Guo, Yongxiang Liu, and Li Liu. 2023. Deep Intellectual Property Protection: A Survey. arXiv:2304.14613 [cs.AI] https://arxiv.org/abs/2304.14613
[64] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. 2021. Mind Your Weight(s): A Large-scale Study on Insufficient Machine Learning Model Protection in Mobile Apps. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 1955–1972. https://www.usenix.org/conference/usenixsecurity21/presentation/sun-zhichuang.
[65] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.
[66] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2818–2826.
[67] Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan Bovik, and Yinxiao Li. 2022. Maxvit: Multi-axis vision transformer. In *European conference on computer vision*. Springer, 459–479.
[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).
[69] Tianyang Wang, Ziqian Bi, Yichao Zhang, Ming Liu, Weiche Hsieh, Pohsun Feng, Lawrence KQ Yan, Yizhu Wen, Benji Peng, Junyu Liu, et al. 2024. Deep Learning Model Security: Threats and Defenses. *arXiv preprint arXiv:2412.08969* (2024).
[70] Zhendong Wang, Xiaoming Zeng, Xulong Tang, Danfeng Zhang, Xing Hu, and Yang Hu. 2022. Demystifying arch-hints for model extraction: An attack in unified memory system. *arXiv preprint arXiv:2208.13720* (2022).
[71] Waymo. 2024. Waymo: Self-Driving Car Technology for a Reliable Ride. https://waymo.com/waymo-driver/.
[72] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. 2020. 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*. IEEE, 125–137. doi:10.1109/DSN48063.2020.00031 https://doi.org/10.1109/DSN48063.2020.00031.
[73] Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, Zhefu Wu, Qi Xuan, and Xiaoniu Yang. 2020. Open DNN Box by Power Side-Channel Attack. *IEEE Trans. Circuits Syst.* 67-II, 11 (2020), 2717–2721. doi:10.1109/TCSII.2020.2973007 https://doi.org/10.1109/TCSII.2020.2973007.

[74] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. 2017. The microsoft 2016 conversational speech recognition system. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*. IEEE, 5255–5259. doi:10.1109/ICASSP.2017.7953159 https://doi.org/10.1109/ICASSP.2017.7953159.

[75] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. 2019. Explainable AI: A brief survey on history, research areas, approaches and challenges. In *Natural language processing and Chinese computing: 8th cCF international conference, NLPCC 2019, dunhuang, China, October 9–14, 2019, proceedings, part II 8*. Springer, 563–574.

[76] Mengjia Yan, Christopher W Fletcher, and Josep Torrellas. 2020. Cache telepathy: Leveraging shared resource attacks to learn {DNN} architectures. In *29th USENIX Security Symposium (USENIX Security 20)*. 2003–2020.

[77] Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. 2020. DeepEM: Deep Neural Networks Model Recovery through EM Side-Channel Information Leakage. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST*. IEEE, 209–218. doi:10.1109/HOST45689.2020.9300274 https://doi.org/10.1109/HOST45689.2020.9300274.

[78] Yuankun Zhu, Yueqiang Cheng, Husheng Zhou, and Yantao Lu. 2021. Hermes Attack: Steal DNN Models with Lossless Inference Accuracy. In *30th USENIX Security Symposium, USENIX*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 1973–1988. https://www.usenix.org/conference/usenixsecurity21/presentation/zhu.

A Model List

Table 11. The 56 TorchVision and 4 TorchText architectures that are part of our InferNet attack.

AlexNet	Inception_v3	GoogleNet
ConvNeXt_Tiny	ShuffleNet_v2_x0_5	EfficientNet_B0
ConvNeXt_Small	ShuffleNet_v2_x1_0	EfficientNet_B1
ConvNeXt_Base	ShuffleNet_v2_x1_5	EfficientNet_B2
SqueezeNet1_0	ShuffleNet_v2_x2_0	EfficientNet_B3
SqueezeNet1_1	EfficientNetV2-S	EfficientNet_B4
ViT_b_16	EfficientNetV2-M	EfficientNet_B5
ViT_b_32	EfficientNetV2-L	EfficientNet_B6
ViT_l_16	VGG11	EfficientNet_B7
ViT_l_32	VGG11_bn	MobileNet_v2
ResNet18	VGG13	MobileNet_v3_large
ResNet34	VGG13_bn	MobileNet_v3_small
ResNet50	VGG16	MnasNet0_5
ResNet101	VGG16_bn	MnasNet0_75
ResNet152	VGG19	MnasNet1_0
ResNext50_32x4d	VGG19_bn	MnasNet1_3
ResNext101_32x8d	DenseNet121	MaxVit_t
Wide_ResNet50_2	DenseNet169	DenseNet161
Wide_ResNet101_2	DenseNet201	XLMR_BASE
RoBERTa_BASE	RoBERTa_LARGE	XLMR_LARGE

B Implementation Details and Hyperparameters

We used a common preprocessing pipeline across all prediction models: feature-wise mean imputation, binary indicator augmentation for sparse kernels/events, z-score normalization, and stratified train/test splitting by architecture label. For sparse-feature experiments, features were ranked with RFE and truncated to the top-3, top-25, or full 450 non-memory GPU-kernel features, depending on the experiment. Tables 13–19 summarize the experimental settings used for the architecture-prediction component of InferNet.

Note. We report only the settings relevant to the architecture-prediction results in the main paper. Broader end-to-end extraction settings not directly used in the reported architecture-inference evaluation are omitted for clarity.

Table 12. The 24 vision and 6 LLM architectures supported by TensorFlow that are part of our InferNet attack.

ConvNeXtTiny	DenseNet121	EfficientNet_B0
ConvNeXtSmall	DenseNet169	EfficientNet_B1
ConvNeXtBase	DenseNet201	EfficientNet_B2
ResNet50	MobileNetV2	EfficientNet_B3
ResNet101	MobileNetV3Large	EfficientNet_B4
ResNet152	MobileNetV3Small	EfficientNet_B5
VGG16	InceptionResNetV2	EfficientNet_B6
VGG19	InceptionV3	EfficientNet_B7
XLM-RoBERTa	GPT2	Gemma
BERT	BART	ALBERT

Table 13. Candidate models and execution platforms.

Parameter	**Value**
Candidate architectures	90 total architectures across PyTorch and TensorFlow
PyTorch models	56 vision models + 4 NLP models
TensorFlow models	24 vision models + 6 LLMs
AI/ML frameworks	PyTorch 2.3.0, TorchVision 0.18.0, TorchText 0.18.0, TensorFlow 2.17.1
Profiling GPUs	NVIDIA Tesla T4, NVIDIA Quadro RTX 8000
Runtime environment	Linux, Python 3.9, CUDA with `nvprof` in aggregate mode

Table 14. GPU profile collection settings.

Parameter	**Value**
Inferences per profile	10
Profiles per architecture	20
Input configuration	Single-input inference; no batch execution
Per-kernel statistics	`min_us`, `max_ms`, `avg_us`, `num_calls`, `time_ms`, `time_percent`
System signals	SM clock, memory clock, temperature, power, fan speed

Table 15. Preprocessing and feature-engineering settings.

Parameter	**Value**
Train/test split	Stratified 75% / 25% by architecture label
Indicator features	Binary presence/absence indicators for GPU kernels and events
Missing-value handling	Column-wise mean imputation
Feature normalization	Standardization followed by min-max normalization
Feature selection	Recursive Feature Elimination (RFE)
Baseline feature subsets	All (849), System (27), No-system (822), GPU-kernel (447), API-call (375), Indicator (68), No-indicator (781)
Reduced kernel subsets	Top-3, Top-25, and all 450 non-memory GPU-kernel features
Transfer-study subset	Top-25 non-memory GPU-kernel features

Table 16. Preprocessing and feature-engineering settings.

Parameter	Value
Train/test split	Stratified 75% / 25% by architecture label
Indicator features	Binary presence/absence indicators for GPU kernels and events
Missing-value handling	Column-wise mean imputation
Feature normalization	Standardization followed by min-max normalization
Feature selection	Recursive Feature Elimination (RFE)
Baseline feature subsets	All (849), System (27), No-system (822), GPU-kernel (447), API-call (375), Indicator (68), No-indicator (781)
Reduced kernel subsets	Top-3, Top-25, and all 450 non-memory GPU-kernel features
Transfer-study subset	Top-25 non-memory GPU-kernel features

Table 17. Victim-model training settings used for modified and pruned evaluations.

Parameter	Value
Dataset	CIFAR-10 for modified/pruned victim-model generation
Base models	ImageNet-pretrained models where applicable
Optimizer	SGD
Default learning rate	0.1
Momentum	0.9
Nesterov momentum	Enabled
Weight decay	10^{-4}
Loss function	Cross-entropy loss
LR scheduler	ReduceLROnPlateau
Scheduler patience	10
Epochs	150
Batch size	128
Workers	8
Overrides	AlexNet, ResNeXt50_32x4d, ResNeXt101_32x8d, and all VGG variants use LR = 0.01; all MNASNet variants use LR = 0.001; SqueezeNet1_0 and SqueezeNet1_1 use Adam with LR = 10^{-4}

Table 18. Pruning and modified-model evaluation settings.

Parameter	Value
Pruning method	Unstructured L1 pruning
Pruning ratio	0.5
Fine-tuning epochs	20
Modified-model setting	Base ImageNet models (1000 classes) adapted to CIFAR-10 (10 classes)

Table 19. Reported evaluation metrics.

Parameter	Value
Primary metrics	Train accuracy, test accuracy, Top-1 accuracy
Additional metrics	Top-5 accuracy, architecture-family accuracy
Ranking outputs	Top-k accuracy and Top-k confidence rankings