Exploiting Attention Kernel Leakage on GPUs for Side-channel-based Membership Inference in Vision Transformers

Arunava Chaudhuri*, Shubhi Shukla†, Sarani Bhattacharya*, and Debdeep Mukhopadhyay*
*Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, Kharagpur, India
†Centre for Computational and Data Sciences, Indian Institute of Technology, Kharagpur, Kharagpur, India
{arunavachaudhuri392, shuklashubhi6, bhattacharya.sarani.iitkgp, debdeep.mukhopadhyay}@gmail.com

***Abstract*—We report a privacy vulnerability in PyTorch's fused multi-head attention (FMHA) implementation on NVIDIA GPUs that enables a privileged cloud operator to perform membership inference attacks during transformer inference. Under an MLaaS threat model, where the service provider has no access to training data but can monitor GPU hardware counters, we find that FMHA kernels exhibit input-dependent control flow that leak information through performance-counter telemetry. By collecting these traces, an attacker can reliably infer whether a query sample was part of the model's training set. Our experiments on multiple ViT architectures achieve up to 90% membership-inference accuracy without modifying the model or accessing any training data, revealing a hardware-level privacy risk in current transformer inference pipelines.**

1. Introduction

Transformer models, including ViTs, run on GPUs as sequences of optimized kernel calls for operations like matrix multiplication, softmax, and attention. Frameworks such as PyTorch, TensorFlow, and JAX compile these high-level operations into GPU kernels for efficient execution. While crucial for fast inference, these optimized kernels still exhibit data-dependent control flow and memory access patterns, which propagate to microarchitectural signals observable via performance counters. This creates a side-channel risk when a cloud provider can passively monitor runtime telemetry. Within this execution framework, our exploration reveals a clear and measurable leakage pathway arising from specific kernel-level behaviors during transformer inference. Among the various GPU kernels invoked, we identify the PyTorch-optimized fused attention kernel, *fmha_cutlassF_f32_aligned_64x64_rf_sm80* (FMHA), as a primary contributor to input-dependent variability. This kernel, responsible for efficient multi-head attention computation, encapsulates multiple operations, including softmax and matrix multiplications, within a single fused routine to minimize memory overhead and latency.

Profiling this kernel reveals that performance counters, such as shared memory transactions, warp divergence, and instruction activity, show statistically significant differences between training and non-training samples. We also observe distinct execution patterns across input classes, indicating class leakage as noted in [1]. These variations show that the model's execution footprint encodes semantic information, enabling both membership and class inference attacks. The leakage stems from data-dependent control flow in the *iterative_softmax* routine, where branching and divergence vary with the input distribution. These effects are amplified in massively parallel GPU execution, making them observable through standard profiling interfaces without modifying the model or runtime.

We consider a MLaaS deployment scenario, where pre-trained models are uploaded to cloud platforms (e.g., Amazon AWS , Google Cloud) and executed on shared GPU infrastructure, while the training data remains entirely within the model owner's local environment. Prior to this work, it was widely assumed that because training data never reaches the cloud host, and therefore it is not possible for the host to reveal training-related information. We challenge this assumption for the first time in the context of GPU-deployed ViT models. We show that a cloud provider, despite having no access to the training data itself, can still mount effective membership inference attacks (MIAs) using privileged visibility into the execution environment. While prior research focuses on client-side adversaries, our threat model considers a server-side adversary who cannot retrieve the training data but can observe low-level runtime signals. We demonstrate that subtle, input-dependent variations in GPU performance counters, driven by the model's memorization behavior, expose a viable and previously unexplored attack surface.

To evaluate this threat, we conduct experiments on two widely used Vision Transformer families (Google [2] and Facebook [3]), each trained on *CIFAR-10* [4] and *GTSRB* [5]. We adopt a realistic threat model in which the adversary (the cloud provider) has access only to model outputs and passive telemetry (e.g., GPU performance counters), but no training data. The adversary trains a shadow model on a surrogate dataset and collects GPU performance-counter traces during inference on both training and non-training inputs. These traces, gathered using tools such as NVIDIA Nsight Compute, are used to train a lightweight classifier that predicts membership from execution-level telemetry. Our experiments show that this attack achieves an average of 90% MIA accuracy across both ViT families,

reliably distinguishing training from non-training examples using only side-channel data. These observations highlight a hardware-level privacy risk that need to revisit by the PyTorch developers to come up with an efficient implementation of attention kernel which effectively reduce the overall attack surface in inference pipeline.

2. Membership and Class Leakage via GPU Hardware Metrics in ViTs

In this section, we demonstrate for the first time that transformer-based deep neural networks, particularly Vision Transformers (ViTs), exhibit data-dependent, inference-time leakage that can be systematically captured through GPU hardware performance counters. We show that such telemetry exposes membership and class-level information, enabling membership inference attacks (MIAs) and class prediction directly from low-level GPU execution traces. We begin by outlining our motivation and experimental setup, followed by a kernel-level analysis uncovering the root causes of this leakage. We then present empirical findings on membership leakage and class inference, showing how GPU hardware metrics alone reveal sensitive model–data relationships in ViT inference.

2.1. Motivation

MIAs aim to determine whether a data point was used during model training. Such attacks succeed largely due to overfitting, where models learn training-specific patterns, producing lower loss and distinct activations for member samples [6], [7]. These differences propagate through the computational graph and influence GPU kernel execution during inference. Because GPU kernels are sensitive to tensor structure and numerical patterns, they exhibit measurable hardware-level variations in metrics such as warp divergence, instruction throughput, and memory access. Although MIAs have been extensively studied in black-box and white-box settings, relatively little work explores GPU side-channel leakage, particularly in transformer architectures. This work addresses that gap by characterizing kernel-level side-channel behavior using NVIDIA's Nsight Compute (NCU) profiler [8].

2.2. Experimental Setup

We study two widely used pre-trained Vision Transformers: *vit-base-patch16-224* from Google and *deit-base-distilled-patch16-224* from Meta, both fine-tuned on CIFAR-10 and GTSRB. These models share a similar architecture with 12 encoder layers, 12 attention heads, and a 768-dimensional embedding for 224×224 input images. During inference, batches of 128 identical images per class are used, and each experiment is repeated three times for consistency. For each workload, we collect over a thousand GPU performance metrics across all launched CUDA kernels using NCU. ViT models, like all DNNs, execute on GPUs via a collection of low-level kernels. While some kernels are common across architectures (e.g., GEMM, elementwise ops, normalization), others are architecture-specific. For transformers, we observe repeated invocation of *fmha_cutlassF_f32_aligned_64x64_rf_sm80*, a fused multi-head attention kernel optimized using CUDA [9]. Table 1 lists all observed kernels and their roles.

TABLE 1: List of GPU kernels executed during ViT model inference, along with their frequency and description.

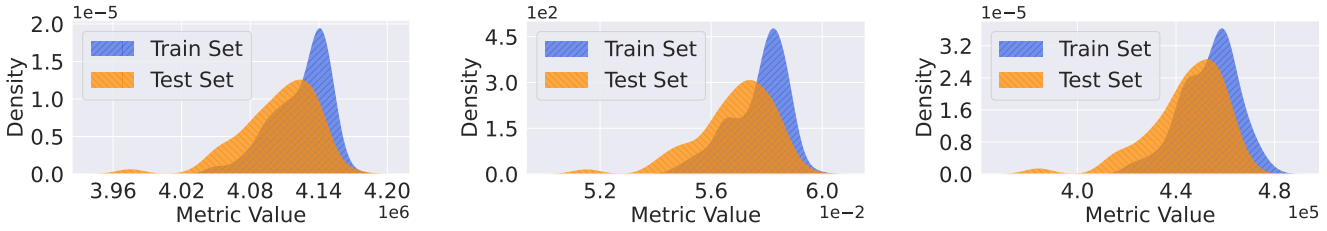
Kernel Name (Abbreviated)	Count	Description
ampere_sgemm_128x64_tn	48	GEMM on Ampere for 128×64 tiles
vectorized_elementwise_kernel	36	Fused elementwise ops
vectorized_layer_norm_kernel	25	Layer normalization with vectorized memory access
ampere_sgemm_128x128_tn	24	GEMM on Ampere for 128×128 tiles
fmha_cutlassF_f32_aligned ..	12	Fused multi-head attention using FP32
elementwise_kernel	2	Elementwise math: add or multiply
nchwToNhwcKernel	1	Layout transform: NCHW to NHWC
_xmma_fprop_implicit_gemm ..	1	Forward propagation using implicit GEMM
CatArrayBatchedCopy	1	Concatenates tensors by copying batched arrays.
ampere_sgemm_128x32_tn	1	GEMM on Ampere for 128×33 tiles
reduce_kernel	1	Parallel reduction

Note: Refer to App. D, Table 9 for full description of kernels.

2.3. Kernel-Level Membership Inference Analysis

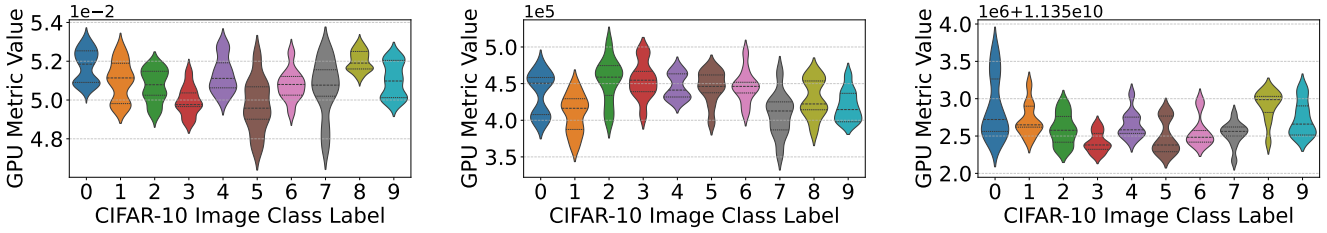
To investigate the presence of membership leakage, we analyze GPU performance metrics collected from all major transformer kernels, with a particular focus on the fused multi-head attention (FMHA) kernel. Among the twelve instances of this kernel launched per inference, several metrics consistently show strong variation between training (member) and test (non-member) samples. Notably, metrics related to control flow and shared memory operations, such as *smsp__inst_executed_op_branch*, *smsp__inst_executed_op_shared_atom*, *thread_inst_executed*, and *smsp__sass_thread_inst_executed_op_fadd_pred_on.min*, demonstrate clear and reproducible distributional shifts across different data categories.

In addition to FMHA, we observed some distinctive patterns in kernels such as *ampere_sgemm_128x64_tn*, *elementwise_kernel*, and *reduce_kernel*, where memory and compute-related metrics (e.g., instruction counts, execution divergence) varied across data inputs. However, FMHA remained the most sensitive to input-dependent behaviour. In particular, kernel instance indices 6 and 9 exhibited pronounced separation in several key metrics when comparing member and non-member inputs. We quantify this separability using two-sample t-tests and find that many metric-kernel instance pairs yield statistically significant differences ($|t| > 4.5$). In Figure 1, the Kernel Density Estimation (KDE) plots visually confirm the distinction,



(a) smsp__inst_executed_op_branch **(b)** smsp__inst_executed_op_shared **(c)** smsp__sass_branch_targets

Figure 1: GPU metric distribution gap between members and non-members on *fmha_cutlassF_f32_aligned* kernel. Shown for the CIFAR-10 dataset using the *vit-base-patch16-224* model.



(a) smsp__inst_executed_op_shared **(b)** smsp__branch_targets_threads **(c)** thread_inst_executed

Figure 2: Plot image of three different metrics for the fourth instance of *fmha_cutlassF_f32_aligned_64x64_rf_sm80* kernel, launched 12 times during image classification workload using the *vit-base-patch16-224* model.

showing clearly distinct distributions between member and non-member traces.

These observations suggest that the FMHA kernel, as optimized in PyTorch for modern NVIDIA architectures, encodes enough data-dependent behavior in its performance counter footprint to leak membership information. The divergence likely originates from variation in softmax attention scores and numerical sparsity across attention heads, which influences the kernel's internal execution path (e.g., shared memory usage and thread divergence). Later in Section 4, we demonstrate a complete MIA, exploiting this input dependent leakage.

Key Takeaway: *We identify the fused multi-head attention (FMHA) kernel as a key source of membership leakage, with specific GPU performance counters revealing statistically significant, input-dependent execution patterns exploitable for MIA.*

2.4. Class Inference via Kernel Metrics

While our primary goal is to evaluate membership leakage, our profiling shows that GPU metrics also exhibit consistent patterns indicative of *class leakage*. We analyzed metrics reflecting memory access, control flow, and shared-resource usage, such as *smsp__inst_executed_op_branch*, *smsp__inst_executed_op_shared_atom*, and *smsp__sass_branch_targets_threads_divergent*. As shown in Figure 2, violin plots across CIFAR-10 classes form clearly separable clusters. We validated this using ANOVA tests on ten images per class for CIFAR-10 and GTSRB, finding that 40 of 45

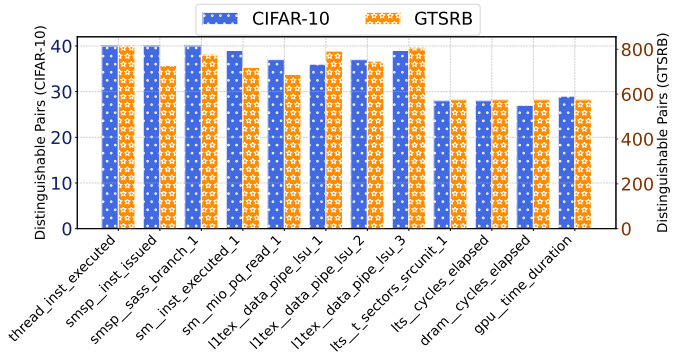


Figure 3: Number of distinguishable class pairs using different GPU metrics in *vit-base-patch16-224* model on CIFAR10 (out of 45) and GTSRB (out of 903) dataset.

CIFAR-10 class pairs and 811 of 903 GTSRB pairs exhibit statistically distinguishable GPU-metric profiles (Figure 3).

To quantify exploitability, we trained a CNN classifier using only the most discriminative FMHA metric. The model achieves 100% classification accuracy on GTSRB and 97.9% average accuracy on CIFAR-10, as summarized in Table 2. These results confirm that GPU performance metrics can be reliably used for class inference, revealing yet another dimension of side-channel leakage in ViT workloads.

Prior work demonstrated class leakage in CPU-based CNN inference via non-constant-time branching in PyTorch's max-pool operation [1], but did not mount a full MIA. In contrast, our study both characterizes GPU performance-metric class signals in transformer kernels and

TABLE 2: Fine-tuned ViT's Class Leakage Success Rate

Family	Transformer Name	Dataset	Max. Success Rate (%)	Avg. Success Rate (%)
Google	vit-base-patch16-224	CIFAR-10	100	97.9
		GTSRB	100	100
META	deit-base-distilled-patch16-224	CIFAR-10	100	98.5
		GTSRB	93.37	92.09

leverages those same side-channel traces to build a complete MIA (Section 4). Combined with membership leakage, class-level inference poses a significant privacy risk when ViTs run on shared or untrusted GPUs. We next present a root-cause analysis of this leakage, focusing on how the FMHA kernel's design amplifies data-dependent variations.

Key Takeaway: *GPU performance metrics leak not only membership but also strong class-specific signals in ViT workloads, enabling near-perfect classification accuracy purely from hardware traces and exposing a broader surface of side-channel leakage.*

3. Root Cause Analysis of FMHA Leakage

Building on our class-leakage attack using GPU metrics, we identify metrics that consistently exhibit high t-test values and clear distributional differences between member and non-member inputs. Table 3 summarizes these metrics and their hypothesized causes, ranging from control-flow divergence to shared-memory and instruction-level variations. These observations form the basis of our root-cause analysis of data-dependent behaviour in the FMHA kernel.

On the software side, we compile PyTorch from source to obtain fine-grained visibility into the operations triggering FMHA kernel launches. Using the PyTorch profiler, we trace the call hierarchy preceding each invocation. The functions *scaled_dot_product_attention*, *_scaled_dot_product_efficient_attention*, and *_efficient_attention* each appear twelve times, precisely matching the twelve FMHA kernel launches observed during execution. This one-to-one correspondence confirms a direct mapping between attention calls and kernel executions, enabling deeper analysis of how data-dependent computation arises.

TABLE 3: List of GPU metrics with their probable cause for train/test data-centric behaviour.

ID	Metric Name (Abbreviated)	Cause 1	Cause 2	Cause 3	Cause 4
1	smsp__thread_inst_executed_pred	✔	✔	✘	✔
2	smsp__inst_executed_op_shared_atom	✘	✘	✔	✘
3	smsp__sass_inst_executed_op_shared	✘	✔	✔	✔
4	smsp__inst_executed.sum	✔	✔	✔	✔
5	smsp__branch_targets_threads_divergent	✔	✔	✘	✔
6	smsp__sass_branch_targets_threads_divergent	✔	✔	✘	✔
7	smsp__inst_executed_op_branch	✔	✔	✘	✔
8	sm__inst_executed_pipe_cbu.avg	✔	✔	✘	✔
9	sm__mio_pq_read_cycles_active	✘	✔	✔	✘
10	l1tex__data_pipe_lsu_wavefront_mem_shared	✘	✔	✔	✔
11	thread_inst_executed	✔	✔	✔	✔

Note: ✔ : Metric impacted by the corresponding cause. ✘ : Metric not impacted at all. Refer to Table 10 of App. C for full description.

Furthermore, after mapping the relevant PyTorch functions to the FMHA kernel invocations, we delve deeper into the underlying implementation by debugging our ViT inference script and tracing the execution path back to the `kernel_forward.h`[1] file inside PyTorch's `transformer` module. This file contains the templated definition of the fused attention kernel, including the iterative softmax computation characteristic of transformer encoder modules. As illustrated in Figure 6b, the fused multi-head attention (FMHA) operation begins by receiving input in the form of query (Q), key (K), and value (V) matrices derived from word embeddings. The first matrix multiplication ($Q \times K^T$) computes raw attention scores, followed by an online softmax applied to this result. A second matrix multiplication is then performed between the softmax-normalized scores and the value matrix (V). This fused implementation leverages GPU register files and shared memory for storing intermediate results, thereby minimizing off-chip memory accesses and improving both performance and scalability across diverse workloads.

3.1. Cause 1: Divergence with Attention Max

Table 3 lists several GPU performance metrics, *smsp__branch_targets_threads_divergent*, *smsp__inst_executed_op_shared_atom*, and *thread_inst_executed*, that show the largest t-test scores and distributional gaps between member and non-member inputs. To investigate the software-level source of these differences, we examine the fused softmax routine, specifically the `iterative_softmax` function defined in `kernel_forward.h`.

```
[&](int accum_m, int accum_n, int idx) {
  if (accum_n < max_col) {
    max = cutlass::fast_max(max, frag[idx]);}},
```

Listing 1: Row-wise max in the FMHA kernel

The above code performs a row-wise maximum computation over attention logits `frag[idx]`. The condition `accum_n < max_col` acts as a mask to ignore padded or future tokens and is a key contributor to warp-level divergence. Threads in a warp may follow different paths depending on their column index, triggering divergence observable via control-flow and memory-related GPU metrics. As shown in Table 3, metrics 1, 5, 6, and 11, tracking predicate usage, thread divergence, and branches, are notably affected. Metrics 4, 7, and 8 also vary based on whether conditional branches are executed. Crucially, the `cutlass::fast_max` operation introduces a hard nonlinearity, even minor differences in input logits can alter the row-wise maximum, resulting in divergent execution traces for the following:

- **Training vs. test samples:** Overfitted models produce attention logits that are more peaked for training samples, concentrating probability mass in specific positions. These peaked logits influence the location of the row-wise maximum relative to `max_col`, causing variations in the number of times `fast_max` is executed and which threads take the branch.

1. https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/native/transformers/cuda/mem_eff_attention/kernel_forward.h

- **Different classes:** Each class induces distinct embedding patterns and QK-dot product distributions. As a result, the row maxima occur at class-specific indices, which in turn alter the branching pattern and shared memory accesses.

Moreover, variations in the `max_col` masking, especially near boundaries, can lead to mismatched attention behavior between members and non-members. For example, a training sample with a large logit just inside the `max_col` boundary may dominate the `fast_max` operation, while a similar non-member without it triggers a different maximum and hardware trace. These divergence patterns reveal both *membership* (train vs. test) and *class identity* (via class-specific QK patterns), making the fused attention kernel a key source of side-channel leakage.

3.2. Cause 2: Exponentiation in Softmax

Beyond the control flow, the softmax routine contains exponentiation steps that can also leak information. In particular, the fused attention kernel employs the `exp2f` function to compute exponentials after subtracting the row-wise maximum value `mi_row`, as shown below:

```
[&](int accum_m, int accum_n, int idx) {
  frag[idx] =
    (accum_n < max_col) ? exp2f(frag[idx] -
   mi_row) : accum_t(0.0);},
```

Listing 2: Exponentiation step in softmax

This subtraction stabilizes numerical computation but introduces dependence on the original logits. Training examples often produce more extreme logits, yielding larger `frag[idx] - mi_row` in some positions. Exponentiating these creates disproportionately large attention scores and high-confidence softmax outputs. Non-members, in contrast, lead to flatter distributions. The `exp2f` operation thus amplifies sharpness differences, contributing to distinct runtime behavior which directly impacts *sm__inst_executed_pipe_cbu* metric that records the number of warp-level convergence, barrier, and branch instructions executed on SM. The masking condition `accum_n < max_col` zeroes out invalid positions, yet also opens a side-channel. Inputs crafted near the `max_col` boundary can cause varied masking behaviour based on membership or class, which alters execution and impacts performance counters like *smsp__inst_executed* and *smsp__inst_executed_op_branch*. Subtle numerical artifacts from floating-point subtraction and exponentiation may also differ between members and non-members, especially when the former produce extreme values. These differences may momentarily affect shared memory (Metric 3, 9 and 10 from Table 3) or instruction execution (Metric 1, 5, 6, and 11 in Table 3), even if the results are eventually masked.

In summary, this exponentiation stage heightens sensitivity to input confidence, causing sharper attention distributions on training samples and producing distinct execution patterns that enable membership or class inference.

3.3. Cause 3: Atomic Logit Max

The `atomicMaxFloat` function (Listing 3) updates `mi[accum_m]`, which stores the maximum logit for each attention query row. Because multiple threads process different columns of a row, this atomic operation ensures a synchronized and accurate row-wise maximum without race conditions:

```
[&](int accum_m) {
  atomicMaxFloat(&mi[accum_m], max);}
```

Listing 3: Atomic update for row-wise max logit

Training examples often contain extreme logits, especially for memorized tokens. The atomic update guarantees that these outliers are preserved as maxima, enabling the subsequent `exp2f` computation to produce sharply peaked attention. This consistent behaviour makes it easier for MIAs to detect subtle input changes, since even perturbed versions of member samples yield similar high-confidence patterns.

Incorrect maxima (e.g., due to imprecise reduction) would smooth these patterns and reduce leakage. However, atomic operations preserve these effects, which appear as variations in GPU counters such as *smsp__inst_executed_op_shared_atom* and *smsp_sass_inst_executed_op_shared_*, highlighting a leakage channel tied to memorization and alignment of the maximum value. Moreover, metric *sm__mio_pq_read_cycles_active* and *l1tex__data_pipe_lsu_wavefront_mem_shared*, which measure the read cycles for single shared-memory access and the total shared-memory accesses performed by the Load/Store unit (LSU), also contribute to the side-channel leakage due to the variable number of atomic operations performed for a given input.

3.4. Cause 4: Row Summation and Buffer Store

The final stage of the softmax routine involves computing the row-wise sum of exponentiated logits, stored in a temporary buffer for normalization. Listing 4 shows the code segment where each thread contributes its local value to the cumulative `total_row` if the lambda function `reduceSameRow(...)` returns true:

```
[&](int accum_m) {
    if (LambdaIterator::reduceSameRow(lane_id,
    total_row, [](accum_t a, accum_t b) {
        return a + b;
    })) {
       // Faster and deterministic reduction
       addition_storage[accum_m + kQueriesPerBlock
     * tile_offset.column()] = total_row;}}
```

Listing 4: Code snippet for row summation and buffer store

Here, `total_row` represents the unnormalized softmax denominator for each attention row, while `addition_storage` serves as an intermediate buffer. Accurate computation of `total_row` is essential to ensure that the final attention weights reflect the true probability mass distribution. For memorized training samples, the

softmax numerator, the exponentiated logit for a dominant token, is typically very large. An accurate `total_row` ensures this high numerator translates into a sharply peaked softmax distribution.

This repeatability significantly aids MIAs: repeated queries to a model with member samples yield consistent softmax outputs, whereas test-time inputs often produce less stable behaviour. This behavior influences GPU-side metrics such as *smsp__branch_targets_threads_divergent* and *smsp__sass_branch_targets_threads_divergent*, which captures data-dependent control flow divergence. Moreover, writes to `addition_storage` occur only when `reduceSameRow` is satisfied, resulting in input-sensitive memory access patterns. Because member samples often induce consistent reduction outcomes (due to fewer masked values), their write patterns differ measurably from non-member samples. These differences manifest in metrics like *smsp__sass_inst_executed_op_shared_*, and *sm__mio_pq_read_cycles_active*, providing another leakage vector tied to the softmax normalization stage. Additionally, similar to Cause 1, metrics 1, 4, 7, and 11 from Table 3 which count the number of instructions executed with active predicates or conditional branches at the thread level, and metric 8, which tracks the number of warp instructions executed by Convergence Barrier Unit (CBU), are also significantly impacted during this operation.

Key Takeaway: *We identified four key sources of FMHA leakage that create data-dependent execution traces: (1) Masked row-wise max induces warp divergence from sharp training-time logits. (2) Exponentiation amplifies logit differences, altering softmax behavior across inputs. (3) Atomic max preserves extreme values, reinforcing member-specific kernel paths. (4) Row summation and conditional stores lead to input-sensitive memory access patterns.*

4. Membership Inference Attack

Building upon the GPU-level leakages identified in Section 3, we now demonstrate how these side-channel signals can be exploited to perform a MIA on Vision Transformer (ViT) models. Our attack leverages input-dependent GPU performance metrics to infer whether a given data sample was part of the model's training set. Specifically, we focus on ViT models executed within a realistic black-box MLaaS setting. We begin by formally defining our threat model.

4.1. Threat Model

We consider a trusted cloud environment where users, especially small businesses, deploy proprietary deep neural network (DNN) models to offer inference-based services under the Machine Learning as a Service (MLaaS) paradigm. The cloud service provider (CSP), acting as system administrator, manages inference workloads using its own

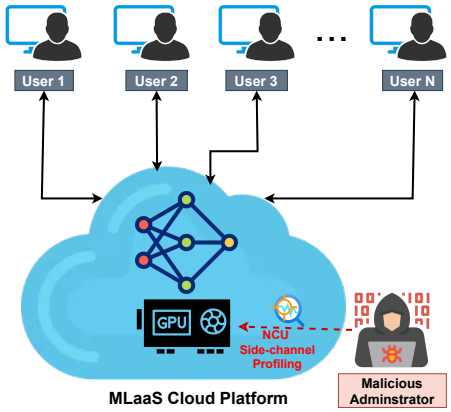


Figure 4: Threat model.

GPU infrastructure. While the CSP lacks direct access to internal details like model architecture or weights, prior work [10] shows some properties can be inferred via side-channel analysis. However, the CSP has no access to the model's training data, which is never uploaded or stored, so even privileged access cannot retrieve it. Still, a curious or malicious administrator may infer sensitive training information through indirect signals. We now formally define the adversary's capabilities and objectives. Figure 4 provides an overview of the threat model.

Adversary's capabilities: The adversary (CSP) is assumed to have administrator-level access to the MLaaS platform hosting the user's private DNN model. With this privileged access, the adversary can enable NVIDIA's Nsight Compute profiler to record fine-grained, input-dependent GPU performance metrics during inference. As the cloud service provider, the adversary also has access to user inputs, such as queries or uploaded content, and can observe the model's hard-label outputs (i.e., predicted class index), a standard assumption in MLaaS settings.

Adversary's objective: With the aforementioned capabilities, the adversary's objective is to distinguish the training data of the victim DNN model from a pool of input samples provided during inference. To achieve this, the adversary can launch a profiling-based membership inference attack by leveraging GPU performance metric leakage. This leakage enables a binary classifier to learn the distinguishing patterns between member (training) and non-member (non-training) samples from the input set.

Note: *While we assume administrative privileges for the adversary, we stress that the training data is never present on the server. Thus, despite elevated access, the adversary cannot directly retrieve it. Our attack instead exploits the memorization behavior of DNNs, using subtle differences in GPU performance counters to infer membership via side-channel analysis.*

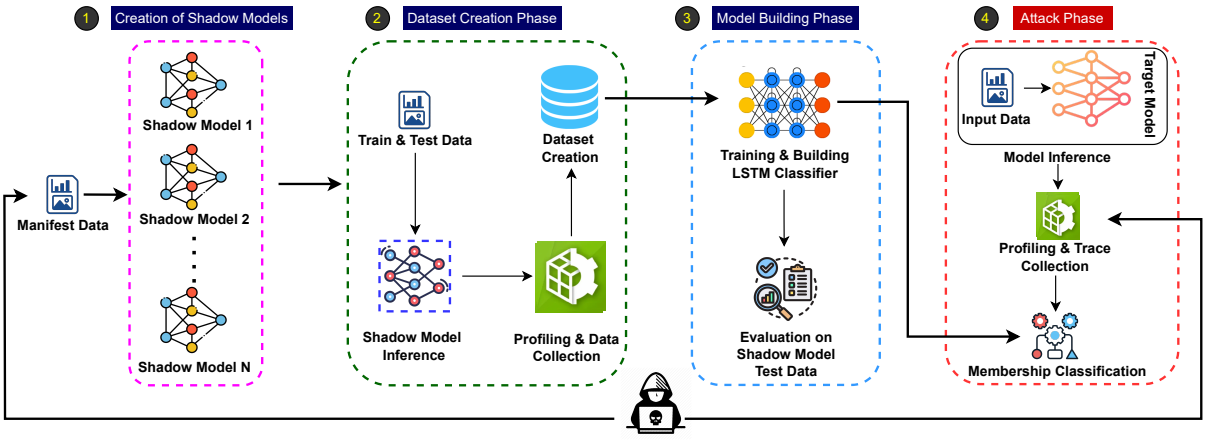


Figure 5: Membership inference attack pipeline with four phases: (1) Shadow model training, (2) Trace collection and dataset creation, (3) LSTM classifier training, and (4) Membership prediction on the target model using side-channel traces.

4.2. Experimental Setup

We evaluate our attack in a black-box setting using a setup consistent with our earlier GPU leakage analysis.

Datasets. We use three widely used image classification benchmarks: *CIFAR-10* [4], *CIFAR-100* [4], and *GTSRB* [5]. These datasets are commonly used in prior MIA studies, allowing direct comparison with existing methods.

ViT Models. We evaluate on two encoder-only Vision Transformers (ViTs) aligned with our GPU side-channel analysis (Section 2): *ViT-B/16* (Google) and *DeiT-B Distilled* (Meta).

GPU Specification. We conducted experiments on two NVIDIA GPUs, the A40 (Ampere) and RTX A4500 (Ada Lovelace), to assess the effectiveness of the attack across generations. The A40 is server-grade and commonly used in cloud settings, while the RTX A4500 targets workstation deployments. Key architectural differences are summarized in Table 6 of App. B.

Software Tools. We use NVIDIA's Nsight Compute (NCU) to collect kernel-level GPU performance metrics in all our experiments. All profiling is performed on Ubuntu 22.04 systems. The A40 GPU system runs CUDA 12.2 with driver version 550.130.6 and Nsight Compute 2023.2.2.0. The A4500 setup uses CUDA 12.6 with driver version 555.42.6 and Nsight Compute 2024.3.2.0.

Evaluation Metrics. To comprehensively assess the effectiveness of our proposed attack, we adopt standard metrics commonly used in the membership inference literature [11], [12], [13]. Specifically, we report the following:

- *Attack Success Rate (ASR):* Measures the proportion of correctly identified member and non-member samples.
- *Area Under the ROC Curve (AUC):* Captures the model's ability to discriminate between member and non-member samples across thresholds.
- *F1 Score:* Represents harmonic mean of precision and recall, indicating the accuracy of positive predictions.

All experiments across Sections 4 are evaluated under a balanced setting where the member and non-member sets are of equal size, and use the same experimental setup.

Model Training and Finetuning. We opted to use Kaggle's ViT model fine-tuning notebook based on HuggingFace's trainer package, as our initial model training script, and modified it depending on the datasets we have used in our study. We present the default fine-tuning and attack settings in Table 7 of App. B.

4.3. Attack Methodology

Based on these settings, we now present our GPU side-channel MIA methodology in detail. As illustrated in Figure 5, the attack proceeds in four sequential phases.

1. Shadow Model Creation. we fine-tune multiple shadow models that are trained on a small subset of data (10%), referred to as *Manifest Data*, which is drawn from the same distribution as the victim model's training data. These shadow models may or may not share the exact architecture of the target model. The purpose of training shadow models is to simulate the behavior of the victim model in a controllable environment, allowing the adversary to observe and learn how GPU performance counters respond to member versus non-member inputs. Since the actual training data and model parameters of the target model are inaccessible, these shadow models act as proxies to generate labeled trace data (with known membership status) that is essential for training the downstream membership inference classifier.

2. Dataset Creation. We perform inference using each shadow model while simultaneously collecting fine-grained GPU performance metrics via the Nsight Compute (NCU) command-line interface. To minimize noise and localize the leakage source, we restrict metric collection to the fused multi-head attention (FMHA) kernel. Each input sample triggers twelve FMHA kernel invocations, producing twelve rows of performance metrics that capture the temporal dynamics of GPU execution. These rows are then preprocessed (e.g., normalization, alignment) and aggregated into a single

vector per sample. To enhance the effectiveness of our MIA classifier, we exploit the observation that the leakage is not only membership-dependent but also class-dependent, i.e., different classes induce distinct patterns in GPU execution. Therefore, we include the true class label of each sample as an additional input feature, resulting in a 13-dimensional input vector: 12 dimensions from the FMHA kernels and one from the class label. This enriched dataset, labeled with membership status (member or non-member), serves as the training and evaluation foundation for our final classifier.

3. Model Building Phase. This phase involves training a binary classifier to distinguish between training (member) and non-training (non-member) samples based on the collected GPU traces. We employ a Bidirectional Long Short-Term Memory (Bi-LSTM) network that processes a 3D input tensor of shape `(samples, sequence_length, feature_dim)`. The model architecture includes two Bi-LSTM layers with hidden sizes of 64 and 32, respectively, each followed by dropout layers to reduce overfitting. These are followed by a fully connected layer, another dropout, and a sigmoid-activated output node for binary classification. The model is trained using the binary cross-entropy loss and optimized using the Adam optimizer with a learning rate of $1e{-}5$. Evaluation is performed on the test split derived from the shadow model's dataset.

4. Attack Phase. During attack we collect GPU performance traces from the actual target model during inference, using the same methodology as applied to the shadow models, capturing only the FMHA kernel metrics for each input. These traces, which are unlabeled from the adversary's perspective, are then fed into the previously trained Bi-LSTM classifier. The classifier predicts whether each input sample is a *member* (i.e., part of the target model's original training data) or a *non-member*, based solely on the observed GPU-level execution patterns. This completes the MIA, enabling the adversary to exploit input-dependent leakage solely via GPU performance counters, without needing model parameters, or training data.

4.4. Attack Results

Given that our target models are ViT and DeiT classifiers, each inference involves twelve FMHA kernel invocations. We begin by evaluating all twelve instances as potential inputs to the classifier. As noted in Section 2.3, only a subset of these exhibit meaningful differences between training and test samples. Among them, the sixth instance shows the highest t-test score and the most distinct distributional shift (Figure 1). However, using only this instance fails to capture sufficient cumulative patterns for effective classification.

Through empirical search, we identify that combining instances 1 through 10 yields the highest attack accuracy. This trend holds consistently across both model families (Google and Facebook) and datasets (CIFAR-10, CIFAR-100 and GTSRB). Based on this selection, we evaluate the performance of the LSTM-based classifier for membership inference. Table 4 shows results for both Ampere and Ada Lovelace architecture. The classifier achieves an average attack success rate of over 85% across all experiments. Using a model from the Google family, our proposed attack identifies member and non-member samples with up to 93.67% accuracy and an AUC of 0.97 on the CIFAR-10 dataset, while achieving an average of 87% accuracy and 0.92 AUC on the GTSRB dataset. For models from the Facebook family, the classifier reaches a maximum attack accuracy of 90% with an AUC of 0.93 on CIFAR-10, and 86.67% accuracy with 0.91 AUC on GTSRB. Similar trend is also observed on the CIFAR-100 dataset using those ViT models experimented in both GPU architectures. Across all datasets, our attack achieves an average F1 score of 0.91, indicating that the selected combination of GPU metrics and kernel instances can reliably identify training set samples of a target model. Additionally, we evaluated our attack against a noisy environment by varying the number of concurrent processes in App. C.

TABLE 4: Fine-tuned Vision Transformer's MIA Results

GPU Architecture	Transformer Name	Dataset	Shadow Model ASR (%)	Target Model ASR (%)	AUC	F1 Score
Ampere	google/vit-base-patch16-224	GTSRB	100	100.00	1.0	1.0
		CIFAR-10		86.00	0.91	0.86
		CIFAR-100		93.00	0.97	0.92
Ada Lovelace		GTSRB		87.00	0.92	0.88
		CIFAR-10		93.67	0.97	0.94
		CIFAR-100		97.00	0.99	0.97
Ampere	META/deit-base-patch16-224	GTSRB	100	86.33	91.60	0.86
		CIFAR-10		93.00	0.98	0.93
		CIFAR-100		91.33	0.94	0.91
Ada Lovelace		GTSRB		86.67	0.91	0.86
		CIFAR-10		90.00	0.93	0.89
		CIFAR-100		93.00	0.96	0.92

Comparison With Traditional MIA. Furthermore, we have compared our attack methodology against traditional MIA techniques to evaluate the effectiveness of our proposed approach. Specifically, we employed a model output probability–based MIA method, in which the shadow model's output probabilities corresponding to each input image were collected and used to train a classification model that distinguishes between member and non-member samples of a target model. Using both the ViT models, our approach shows noticeable improvements (refer to Table 5) in the attack success rate as well as the F1 score on the benchmark dataset (CIFAR-10). This is due to the fine grained distinguishable information collected in between attention computation captured using kernel metrics. Alongside, we have also identified impaction GPU metrics which has been summarize in Appendix D.

TABLE 5: MIA Attack Success Rate Comparison Against Traditional Approach

Transformer Name	Method	Target Model ASR (%)	AUC	F1 Score
vit-base-patch16-224	Output Probabilities	77.00	0.69	0.70
	GPU Kernel Metrics (Ours)	**93.67**	**0.97**	**0.94**
deit-base-patch16-224	Output Probabilities	89.00	**0.97**	0.88
	GPU Kernel Metrics (Ours)	**90.00**	0.93	**0.89**

5. Conclusion

In this work, we revisit the assumption that deploying only a pre-trained model to the cloud preserves training-data

privacy, and show that this boundary does not hold for GPU-based transformer inference. We uncover a privacy vulnerability in PyTorch's transformer library, demonstrating that GPU performance counters can be exploited to perform an effective side-channel membership inference attack on Vision Transformers (ViTs) in MLaaS settings. By targeting the fused multi-head attention (FMHA) kernel, specifically the *iterative_softmax* routine, we show that input-dependent behaviors such as warp divergence, exponentiation sharpness, and atomic memory operations yield distinguishable execution traces. Our attack achieves an average of 90% inference accuracy using only GPU performance counters and remains effective even under differential privacy. To mitigate this leakage, we propose kernel-level defenses, including temperature scaling and controlled noise injection, that reduce attack success while preserving utility. These results highlight the need to consider hardware-level behavior in deep learning frameworks like PyTorch to ensure side-channel resilience in cloud environments.

References

[1] S. Shukla, M. Alam, S. Bhattacharya, P. Mitra, and D. Mukhopadhyay, ""whispering mlaas" exploiting timing channels to compromise user privacy in deep neural networks," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2023, no. 2, pp. 587–613, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i2.587-613

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[3] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 9630–9640. [Online]. Available: https://doi.org/10.1109/ICCV48922.2021.00951

[4] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[5] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The german traffic sign detection benchmark," in *The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013*. IEEE, 2013, pp. 1–8. [Online]. Available: https://doi.org/10.1109/IJCNN.2013.6706807

[6] C. Li, A. Kumar, Z. Guo, J. Hou, and R. Tourani, "Unveiling the unseen: Exploring whitebox membership inference through the lens of explainability," *CoRR*, vol. abs/2407.01306, 2024.

[7] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/ml-leaks-model-and-data-independent-membership-inference-attacks-and-defenses-on-machine-learning-models/

[8] NVIDIA Corporation, *NVIDIA Nsight Compute Documentation*, https://docs.nvidia.com/nsight-compute/, NVIDIA Corporation, 2024, available at: https://docs.nvidia.com/nsight-compute/.

[9] ——, *CUDA C++ Programming Guide*, https://docs.nvidia.com/cuda/cuda-c-programming-guide/, NVIDIA Corporation, 2024, available at: https://docs.nvidia.com/cuda/cuda-c-programming-guide/.

[10] A. Chaudhuri, S. Shukla, S. Bhattacharya, and D. Mukhopadhyay, ""energon": Unveiling transformers from gpu power and thermal side-channels," 2025. [Online]. Available: https://arxiv.org/abs/2508.01768

[11] Z. Chen and K. Pattabiraman, "A method to facilitate membership inference attacks in deep learning models," in *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/a-method-to-facilitate-membership-inference-attacks-in-deep-learning-models/

[12] M. Ko, M. Jin, C. Wang, and R. Jia, "Practical membership inference attacks against large-scale multi-modal models: A pilot study," in *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*. IEEE, 2023, pp. 4848–4858. [Online]. Available: https://doi.org/10.1109/ICCV51070.2023.00449

[13] J. Ye, A. Maddi, S. K. Murakonda, V. Bindschaedler, and R. Shokri, "Enhanced membership inference attacks against machine learning models," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 3093–3106. [Online]. Available: https://doi.org/10.1145/3548606.3560675

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[15] PyTorch Core Team, *PyTorch Documentation*, https://pytorch.org/docs/, Meta AI, 2024, available at: https://pytorch.org/docs/.

[16] T. Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," in *International Conference on Learning Representations (ICLR)*, 2024.

[17] M. Milakov and N. Gimelshein, "Online normalizer calculation for softmax," *CoRR*, vol. abs/1805.02867, 2018. [Online]. Available: http://arxiv.org/abs/1805.02867

[18] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 3–18. [Online]. Available: https://doi.org/10.1109/SP.2017.41

[19] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, "Privacy risk in machine learning: Analyzing the connection to overfitting," in *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 2018, pp. 268–282. [Online]. Available: https://doi.org/10.1109/CSF.2018.00027

[20] H. Hu, Z. Salcic, L. Sun, G. Dobbie, P. S. Yu, and X. Zhang, "Membership inference attacks on machine learning: A survey," *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 235:1–235:37, 2022. [Online]. Available: https://doi.org/10.1145/3523273

[21] A. Sablayrolles, M. Douze, C. Schmid, Y. Ollivier, and H. Jégou, "White-box vs black-box: Bayes optimal strategies for membership inference," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 5558–5567. [Online]. Available: http://proceedings.mlr.press/v97/sablayrolles19a.html

[22] A. Rajabi, R. Pimple, A. Janardhanan, S. Asokraj, B. Ramasubramanian, and R. Poovendran, "POSTER: double-dip: Thwarting label-only membership inference attacks with transfer learning and randomization," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2024, Singapore, July 1-5, 2024*, J. Zhou, T. Q. S. Quek, D. Gao, and A. A. Cárdenas, Eds. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3634737.3659429

[23] L. Song and P. Mittal, "Systematic evaluation of privacy risks of machine learning models," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. D. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 2615–2632. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/song

[24] M. Fredrikson, E. Lantz, S. Jha, S. M. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, K. Fu and J. Jung, Eds. USENIX Association, 2014, pp. 17–32. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/fredrikson_matthew

[25] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015. [Online]. Available: https://arxiv.org/abs/1412.6572

[26] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang, "When machine unlearning jeopardizes privacy," *CoRR*, vol. abs/2005.02205, 2020. [Online]. Available: https://arxiv.org/abs/2005.02205

[27] G. Liu, C. Wang, K. Peng, H. Huang, Y. Li, and W. Cheng, "Socinf: Membership inference attacks on social media health data with machine learning," *IEEE Trans. Comput. Soc. Syst.*, vol. 6, no. 5, pp. 907–921, 2019. [Online]. Available: https://doi.org/10.1109/TCSS.2019.2916086

[28] Y. Long, L. Wang, D. Bu, V. Bindschaedler, X. Wang, H. Tang, C. A. Gunter, and K. Chen, "A pragmatic approach to membership inferences on machine learning models," in *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 2020, pp. 521–534. [Online]. Available: https://doi.org/10.1109/EuroSP48549.2020.00040

[29] R. Shokri, M. Strobel, and Y. Zick, "On the privacy risks of model explanations," in *AIES '21: AAAI/ACM Conference on AI, Ethics, and Society, Virtual Event, USA, May 19-21, 2021*, M. Fourcade, B. Kuipers, S. Lazar, and D. K. Mulligan, Eds. ACM, 2021, pp. 231–241. [Online]. Available: https://doi.org/10.1145/3461702.3462533

[30] S. Truex, L. Liu, M. E. Gursoy, L. Yu, and W. Wei, "Demystifying membership inference attacks in machine learning as a service," *IEEE Trans. Serv. Comput.*, vol. 14, no. 6, pp. 2073–2089, 2021. [Online]. Available: https://doi.org/10.1109/TSC.2019.2897554

Appendix A. Additional Background

A.1. Transformer Models

Transformer models [14] revolutionized the field of natural language processing by introducing a fully attention-based architecture that surpasses traditional models in both performance and scalability. Their design enables parallel computation across entire sequences, eliminating the sequential bottlenecks of earlier recurrent models. At the heart of the architecture lie two key components: the encoder and the decoder, both leveraging self-attention mechanisms.

Encoder: The encoder comprises a stack of identical layers, each designed to encode contextual relationships between input tokens. Every layer includes a multi-head self-attention mechanism followed by a feed-forward neural network, with residual connections and layer normalization applied to both. In the attention mechanism, query, key, and value vectors are derived from token embeddings, allowing the model to compute attention scores and aggregate contextual information. The final encoder output serves as input to the decoder.

Decoder: The decoder generates the output sequence by attending to both the encoder's representation and previously generated tokens. Although structurally similar to encoder layers, each decoder layer incorporates masked self-attention to prevent future token information from influencing the current prediction. The decoder progressively generates output tokens, feeding each prediction back into the next decoding step, until an end-of-sequence token is produced.

Self-attention: The self-attention mechanism (ref. Figure 6a) allows each token to dynamically weigh the relevance of all other tokens in the sequence, enabling the model to capture long-range dependencies. Multi-head attention further enriches this process by learning multiple representation subspaces in parallel, offering diverse perspectives on token relationships.

A.2. Attention Methodology

Algorithm 1 describes the basic operations of self-attention mechanism inside transformer's encoder and decoder module. It computes the attention score in a particular layer by starting with multiplying query (Q) and key (K) matrix followed online *softmax* operation. After that, the softmax output is multiplied by value (V) matrix to generate attention score which is further transfer to the next encoder or decoder layer.

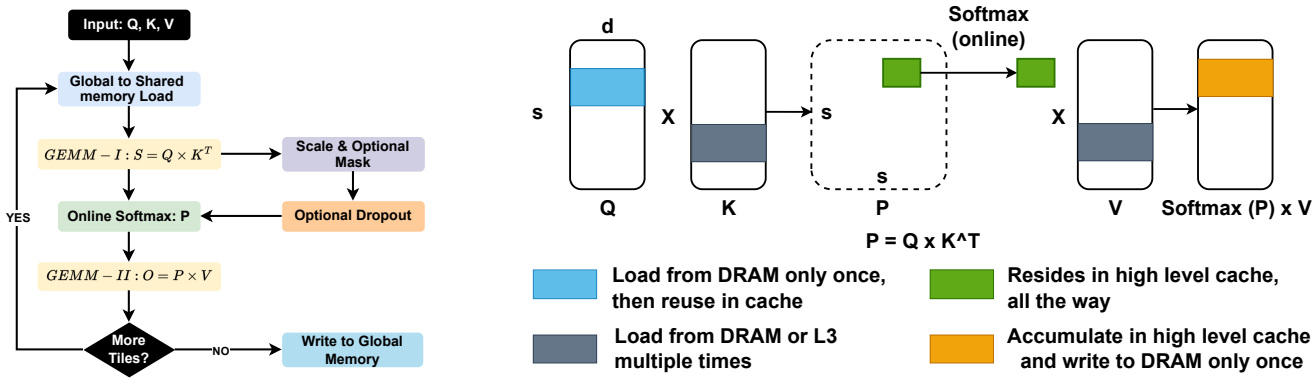
Algorithm 1 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

A.3. Vision Transformers

In this work, we focus on Vision Transformer (ViT) models, which use only the encoder module of the original Transformer architecture (detailed in App. A) alongside the multi-head attention technique. Compared to NLP applications, where attention score is often described as the relationship between words (tokens) in a sentence, in computer vision application, attention looks at the relationships between patches (tokens) in an image. This function is calculated by launching GPU-accelerated FMHA kernel which is described in the following.



(a) Generalized Attention execution Flow

(b) Diagram of Fused Multi-head Attention technique.

Figure 6: Overview of basic Attention operation with its fused implementation using *fmha_cutlassF_f32_aligned* kernel.

Fused Multi-head Attention Kernel: The fused multi-head attention (FMHA) kernel, *fmha_cutlassF_f32_aligned_64x64_rf_sm80* (Figure 6b) , is a core operation repeatedly invoked during inference in transformer-based models, as seen in our ViT workloads. Introduced in recent PyTorch versions [15] and supported from CUDA 12.0 onward [9], it is optimized for modern NVIDIA GPUs (Ampere+), accelerating attention by fusing multiple steps. Traditional attention involves two matrix multiplications and a softmax, with intermediate results stored in high-bandwidth memory (HBM), causing latency and memory overhead. FMHA, inspired by FlashAttention [16], avoids this by using a fused CUTLASS-based kernel that performs all steps in a single call. FlashAttention2's key innovation is the online-softmax algorithm [17], which computes partial softmax in registers/shared memory after the first GEMM, preventing memory spills and enabling the second GEMM from local memory. This reduces bandwidth use, improves performance, but, crucially for our work, introduces *data-dependent variation* due to branching, shared memory reuse, and instruction divergence. To the best of our knowledge, this observation is not reported in published literature, and our work is the first to establish the corresponding leakage w.r.t. membership inference attack.

Later in Section 2, we demonstrate that this kernel exhibits consistent metric variations across training membership and input class. Its control flow and shared-memory metrics are particularly sensitive to input shifts, making it a key source of side-channel leakage in ViT models.

A.4. Membership Inference Attack on ML Models

Membership Inference Attack (MIA) [18] targets a core privacy risk: determining whether a specific data point was part of a model's training set. This is critical, as training data often contains sensitive personal information [19], [20]. MIAs are broadly categorized into metric-based and shadow model-based approaches.

In *metric-based MIAs*, adversaries compute metrics over prediction vectors [7], [19], [21], [22], [23] to distinguish members from non-members using predefined thresholds. *Shadow model-based attacks* assume the adversary knows the architecture or training method of the target model and builds shadow models to mimic its behavior [24], [25]. With access to both training and test sets for these models, the adversary labels data with membership status and trains a binary classifier. This classifier can operate under black-box (query access to prediction vectors only) or white-box settings (access to internal computations) [26], [27], [28], [29], [30].

In this work, we adopt a shadow model-based MIA in the black-box setting to infer membership for ViT models. Unlike prior work relying on prediction vectors, our approach leverages fine-grained GPU side-channel traces. To the best of our knowledge, this is the first MIA against ViTs using only GPU-level side-channel signals, extending beyond the capabilities of prior methods.

Appendix B. Additional Information for Experimental Setup

Table 6 contains the hardware specification of the GPUs on which we have conducted our experiments to evaluate the performance of our proposed attack.

TABLE 6: NVIDIA GPU Basic Specification

Model	Architecture	Compute Capability	#CUDA Core	#Tensor Core	Storage Capacity (GB)
A4500	Ada Lovelece	8.9	7168	224	20
A40	Ampere4	8.6	9216	336	48

Table 7 shows the detailed parameter settings that we have followed to fine-tune shadow ViT models on a particular dataset.

TABLE 7: Default parameters used in model fine-tuning

Parameters	Training settings for our work
Training data size	5000
#Epochs	50
Image resolution	224×224
Batch size	10
Learning rate	2×10^{-5}
Weight decay	0.01
Inference batch size	4
Image feature extractor	DeiT/ViT
Attack type	Classifier-based

Appendix C. Additional Results

We have evaluated our attack against a complex noisy environment where several CNN processes running in parallel. Experimental result shows that (refer Table 8), our attack consistently achieved over 80% accuracy across all experiments, even as background process interference increased. Furthermore, the model demonstrated strong performance with F1 scores exceeding 0.85 in every case, highlighting both the effectiveness and robustness of our attack strategy under complex and realistic MLaaS conditions.

TABLE 8: Fine-tuned Vision Transformer's MIA attack success rate on CIFAR-10 dataset in noisy environment

# Background Process	Transformer Name	Shadow Model ASR (%)	Target Model ASR (%)	AUC	F1 Score
2	vit-base-patch16-224	100	93.67	0.965	0.9369
	deit-base-patch16-224		85.00	0.8784	0.8580
4	vit-base-patch16-224		88.67	0.9504	0.8924
	deit-base-patch16-224		84.00	0.8858	0.8462

Appendix D. GPU Kernel & Metric Details

In Table 9, we demonstrate the functionality of each of the kernel launched while executing inference workload using ViT models. While most of the kernels are used multiple times to execute basic operation of DNN model, we particularly focus on *fmha_cutlassF_f32_aligned_64x64_rf_sm80* kernel which is the source of privacy leakage.

Impactful GPU Metrics Alongside the reported attack success rates, we examined which metrics appeared most consistently across our experiments. We started by assembling a candidate set of metrics that showed reliably distinguishable patterns between input classes across multiple inference runs. This set comprised metrics from l1tex, lts, sm, smsp, and thread-group counters, including several aggregated statistics. Our objective was to identify a minimal subset of metrics that maximized attack success on the target dataset. For example, we initially focused on metrics related to the total number of instructions executed per warp in an active SM (Metric ID 9 of Table 10), since differences here could reveal whether member and non-member inputs trigger different instruction counts. These set of metrics had produced an attack success rate of 83% on CIFAR-100 dataset using vit-base model. We also considered branch-related metrics (Metric ID 10), which measure branch instructions executed by the Convergence Barrier Unit (CBU), because the divergence of execution paths during inference can vary between training and test data and thus affect label-deciding control flow. Combiniing these metrics collectively increased the attack success rate by 2%.

Moreover, metrics such as 11, 12 and 13 are also included to identify the total number of warp instructions executed for shared memory load and store operations during inference. Only these set of metrics produces 84.55% of attack success rate on the same dataset. Furthermore, to enhance the result, we also consider metric 8, which calculates the number of divergent branch targets generated during model execution. In short those above mentioned set of metrics (Metric ID 8, 9, 11, 12, 13 of Table 10) can generate cumulative attack success rate of 93% on CIFAR-100. Afterwards, depending on the dataset characteristic we have chosen a few other metrics, which is detailed in Table 10. Another important observation is that all twelve instances of the FMHA kernel do not always contribute to our attack model; only the kernel instances till eight are mostly common in our experiments. Based on that, we have selected a different set of kernel instances for each of our target datasets and model combinations.

TABLE 9: List of GPU kernels executed during ViT model inference, along with their frequency and description.

Kernel Name	Count	Description
ampere_sgemm_128x64_tn	48	Tensor core GEMM kernel optimized for 128×64 tiles (transpose-normal layout).
vectorized_elementwise_kernel	36	Performs elementwise operations with memory vectorization for efficiency.
vectorized_layer_norm_kernel	25	Applies layer normalization with vectorized memory access.
ampere_sgemm_128x128_tn	24	High-throughput tensor core GEMM kernel for Apmere 128×128 tiles.
fmha_cutlassF_f32_aligned_64x64_rf_sm80	12	CUTLASS-based fused multi-head attention kernel using FP32 on SM80.
elementwise_kernel	2	Elementwise math: add or multiply
nchwToNhwcKernel	1	Converts tensors from NCHW to NHWC layout.
sm86_xmma_fprop_implicit_gemm_indexed_tf32f32_tf32f32_f32 _nhwckrsc_nchw_tilesize128x64x32_stage4_warpsize2x2x1_g1_ tensor16x8x8_alignc4_execute_kernel__5x_cudnn	1	cuDNN kernel for forward propagation using implicit GEMM with TF32 precision on SM86.
CatArrayBatchedCopy	1	Concatenates tensors by copying batched arrays
ampere_sgemm_128x32_tn	1	Tensor core GEMM on Ampere for 128×32 tiles
reduce_kernel	1	Parallel reduction (e.g., sum or max) on elements

TABLE 10: List of GPU metrics used for MIA attack, along with their description.

ID	GPU Metric Name	Description
1	l1tex__data_bank_conflicts_pipe_lsu_mem_shared	No. of shared memory data bank conflicts generated by ATOMS, LDS, LD, STS, ST
2	l1tex__data_pipe_lsu_wavefronts_mem_shared	No. of shared memory wavefronts processed by Data-Stage
3	l1tex__data_pipe_lsu_wavefronts_mem_shared_op_atom	No. of shared memory wavefronts processed by Data-Stage for only ATOMS
4	l1tex__data_pipe_lsu_wavefronts_mem_shared_op_ld	No. of shared memory wavefronts processed by Data-Stage for only LDS, LD
5	l1tex__data_pipe_lsu_wavefronts_mem_shared_op_st	No. of shared memory wavefronts processed by Data-Stage for only STS, ST
5	lts__t_sector_op_read_hit_rate	No. of sectors hit for _op_read per sector access for _op_read
6	sm__inst_executed_pipe_cbu_pred_on_any	No. of warp instruction executed by CBU pipe with at least one thread predicated on
7	sm__mio_pq_read_cycles_active	No. of cycles where MIOP PQ sent register operands to a pipeline
8	smsp__branch_targets_threads_divergent	Number of divergent branch targets, including fallthrough
9	smsp__inst_executed	No. of warp instruction executed
10	smsp__inst_executed_op_branch	No. of warp instruction executed: By CBU pipe, except BMOV, BSSY
11	smsp__inst_executed_op_shared_atom	No. of warp instruction executed: ATOMS*
12	smsp__sass_inst_executed_op_shared_ld	No. of warp instruction executed: LDS, LD
13	smsp__sass_inst_executed_op_shared_st	No. of warp instruction executed: STS, ST
14	smsp__sass_thread_inst_executed_op_fmul_pred_on	No. of FMUL thread instruction executed where all predicates were true
15	smsp__thread_inst_executed_per_inst_executed	Average no. of active threads per instruction executed
16	smsp__thread_inst_executed_pred_on_per_inst_executed	Average no. of predicated-on threads per instruction executed
17	thread_inst_executed	Number of thread-level instructions executed
18	thread_inst_executed_true	Number of thread-level instructions executed where predicate evaluated as true