Exploiting CPU Microarchitectural Leakage to Infer Membership in Fine-Tuned Large Language Models

*ECE592: Microarchitecture Security and Reverse Engineering Project.

[GitHub Repository] [Overleaf Project]

1st Rushiraj Sheth
dept. Electrical and Computer Engineering
North Carolina State University
Raleigh, USA
rsheth@ncsu.edu, Student ID: 200596988

2nd Maharshi Oza
dept. Electrical and Computer Engineering
North Carolina State University
Raleigh, USA
mmoza@ncsu.edu, Student ID: 200607269

***Abstract*—Large Language Models (LLMs) are increasingly deployed in sensitive environments, raising concerns about whether they unintentionally reveal information about their training datasets. In this work, we investigate whether microarchitectural CPU behavior can be used to extract membership information from fine-tuned LLMs. We train GPT-2-Medium models on two contrasting datasets—(A) copyrighted natural-language text from Books3 and (B) purely synthetic gibberish—and record five performance counters during inference: LLC-load-misses, dTLB-load-misses, instructions, cycles, and cycle stalls. Our results show statistically significant separations between member and non-member traces across multiple metrics. A Random Forest classifier achieves 76% accuracy on Books3, while an XGBoost classifier achieves 73.0% accuracy on the gibberish model using 1000 samples. These findings demonstrate that even coarse-grained hardware signals can leak sensitive information about an LLM's training data without requiring access to model weights, gradients, or token probabilities. We conclude with a discussion of practical mitigations such as disabling SMT, restricting access to performance counters, and noise-based defenses.**

I. Introduction

Large Language Models (LLMs) such as GPT-2, GPT-3, and LLaMA are widely used across industry and research applications. Many of these models are fine-tuned on proprietary, copyrighted, or sensitive datasets—including Books3-derived corpora, internal documents, user prompts, and domain-specific text. As a result, membership inference attacks (MIAs), which aim to determine whether a particular data sample was included in model training, pose increasingly important privacy and intellectual-property risks. [1], [2]. Classical MIAs primarily exploit differences in model outputs, such as softmax confidence, loss values, or token likelihoods. However, as LLM deployments become more locked-down—exposing only text outputs and filtering probability information—there is growing interest in non-output-based leakage channels. Recent microarchitectural studies such as Cache Telepathy [3] and speculative-execution leakage works like ZombieLoad [4] show that shared hardware resources can inadvertently reveal sensitive information about co-located computations. These attacks demonstrate that subtle differences in cache usage, TLB behavior, speculative loads, and execution stalls can act as side-channels that reveal properties of the executed code or processed data. Motivated by these findings, we explore the following question: **Can LLM membership be inferred *purely* from CPU microarchitectural behavior, even when the attacker has no access to model outputs beyond the final text?** To investigate this, we fine-tune GPT-2-Medium on two datasets—a real-world text corpus (Books3 [1]) and a synthetic pure-gibberish dataset—and record hardware performance counter traces during inference on member and non-member prompts. Our experiments show that microarchitectural counters such as LLC-load-misses and instructions retired shift in statistically detectable ways between training and non-training samples. Furthermore, supervised classifiers trained on these counters achieve membership inference accuracy significantly above chance (50%).

A. Real-World Impact of Our Attack

Understanding membership leakage from hardware behavior is important because an attacker capable of observing performance counters could:

- Determine whether copyrighted works (e.g., Books3 fragments) were used in fine-tuning, enabling intellectual-property inference.
- Infer if private user text contributed to a fine-tuned model, raising privacy concerns similar to those discussed in [5].
- Potentially combine membership leakage with prior attacks on LLM memorization, dataset reconstruction, or hyperparameter inference [6].
- Support downstream malicious tasks such as targeted adversarial example crafting, fine-tuning poisoning, or backdoor insertion.

[1]The Books3 dataset is known to contain a large collection of copyrighted books scraped from online repositories, raising significant legal and ethical concerns regarding memorization and unintended model leakage [5].

Thus, even coarse-grained hardware telemetry can meaningfully weaken LLM privacy guarantees.

B. Contributions

We summarize our key contributions in three quantitative bullet points:

1) We introduce the CPU performance-counter–based Membership Inference Attack on GPT-2-Medium, requiring no access to model internals. Using only five HPC metrics, we demonstrate that inference traces encode detectable membership signals.
2) We show that member vs. non-member prompts generate statistically distinct microarchitectural footprints.
   - Books3 model: significant divergence in 3/5 metrics ($p < 0.01$).
   - Gibberish model: divergence in 4/5 metrics, even when the input text has no semantic structure.
3) We train machine-learning classifiers that infer membership from performance counters, achieving:
   - 76% accuracy on the Books3 model (Random Forest).
   - 73% accuracy on the gibberish model with 1000 samples (XGBoost). These results confirm that microarchitectural leakage is both measurable and actionable for adversarial inference.

II. Background and Related Work

A. Background

Membership Inference Attacks (MIAs) are a class of privacy attacks where an adversary attempts to determine whether a particular data sample was part of a machine-learning model's training set. Traditional MIAs typically rely on observable differences in model outputs—such as confidence scores, loss values, or gradient information—to differentiate between member and non-member samples [1], [7], [8]. Models often generalize imperfectly, causing subtle shifts in their responses on training data compared to unseen samples. With the rise of Large Language Models (LLMs), the attack surface has expanded beyond output-based signals. Modern LLMs exhibit complex computational behaviors influenced by token structure, context length, and prompt semantics. This complexity has motivated a new line of research exploring hardware-side channels, where differences in low-level processor behavior—such as cache activity or execution stalls—may inadvertently reveal information about model inputs. In this project, we explore MIAs in a controlled academic setting by observing CPU performance counters during LLM inference. By profiling metrics such as LLC-load-misses, dTLB-load-misses, instructions, cycles, and stall counts, we analyze whether the CPU exhibits detectable behavioral differences between member and non-member prompts across two datasets: a "suspect prompt" dataset and a synthetic "gibberish" dataset.

B. Membership Inference Attacks on Neural Networks

Prior work has shown that neural networks often leak information about their training sets. Classical work demonstrated that models tend to overfit on member samples, producing distinguishable confidence values or gradients [1], [7]. Follow-up studies expanded MIAs to more realistic black-box threat models [8]. In the context of LLMs, studies have shown that large models can memorize and regurgitate training text, especially when fine-tuned on narrow datasets [5]. MIAs have also been demonstrated even when only plain-text outputs are available, without logits or probabilities [9], [10]. However, nearly all of these works rely solely on the model's output behavior.

C. Hardware Side-Channel Attacks

Our threat model is most closely aligned with hardware side-channel leakage. Existing works such as Cache-based attacks (e.g., CacheTelapathy-style leakage [11]). Additional studies demonstrated that hardware performance counters can fingerprint workloads or identify neural-network characteristics [12], [13]. These findings suggest that microarchitectural traces may inadvertently leak information about model execution. While our work does not involve malicious exploitation of hardware behavior, the methodology is conceptually related: observing controlled CPU performance counters during LLM inference to determine whether different prompts result in measurably different microarchitectural signatures. Importantly, our goal is purely academic—to evaluate whether membership inference is theoretically feasible using only high-level CPU counter data.

D. LLM Privacy Leakage and Fine-Tuning Effects

Recent LLM-oriented research highlights issues such as memorization, dataset extraction, and unintended leakage pathways [5], [14], [15]. Fine-tuning can amplify memorization, causing internal activations to diverge more strongly between member and non-member samples. While prior work typically analyzes model outputs, it remains unclear whether such differences manifest in hardware-level computation traces.

E. Summary of Related Work

Overall, the body of literature establishes:

- MIAs exploit observable inconsistencies between training and non-training samples [1], [8].
- Hardware-side channels can reveal unintended internal information about executing programs [11], [16].
- LLM fine-tuning can amplify memorization and internal activity differences [5], [14].

Our study combines these ideas by evaluating whether CPU-level metrics are sufficient to distinguish between member and non-member prompts—a novel direction in the context of LLM privacy analysis.

III. Threat Model

In this section, we formally define the capabilities, goals, and assumptions of the adversary in our microarchitectural Membership Inference Attack (MIA). Our threat model closely aligns with prior microarchitectural leakage research, particularly Cache Telepathy [11] and transient-execution attacks such as ZombieLoad [4], which similarly assume an attacker capable of observing fine-grained CPU hardware behavior without direct access to the target model's internal weights or training data.

A. Adversary Goals

The adversary's objective is to determine whether a given input prompt was part of the target model's training dataset. Formally, for an input x, the attacker attempts to infer a membership bit MIA(x)→0,1 where 1 denotes that x was a member of the training corpus. A successful attack enables the adversary to leak private training-data properties—such as sensitive prompts, user text, or proprietary corpus elements—similar to the broader goals of membership inference attacks observed in prior work [1], [7], [8]. For LLMs, this threat is particularly relevant given evidence of memorization and training-data extraction demonstrated in recent studies [5].

B. Adversary Capabilities

We assume the attacker has no white-box access to the model:

- No access to model weights.
- No access to gradients.
- No access to training hyperparameters
- No ability to manipulate the model's internal states

Such restrictions align with classical black-box membership-inference settings [1], [7], where the adversary cannot observe internal computations or parameters. However, consistent with Cache Telepathy–style hardware attacks [11] and transient-execution leakage studies such as ZombieLoad [4], [17], the adversary does possess the following capabilities:

1) Ability to co-locate on the same machine: The attacker operates on the same physical CPU where the target model inference is executed. No elevated privileges are required beyond standard process execution.

2) Access to hardware performance counters (HPCs): The attacker can run the Linux `perf stat` tool to capture microarchitectural measurements such as:

- `LLC-load-misses`
- `dTLB-load-misses`
- `instructions`
- `cycles`
- `cycle_activity.stalls_total`

Prior work has shown that performance counters leak identifiable microarchitectural footprints related to neural-network behavior and workload structure [12], [13]. These signals form the side-channel exploited in our attack.

3) Ability to submit chosen prompts: The attacker can query the model with arbitrary prompts and record HPC traces during inference. This follows the standard query-based black-box MIA setting used in previous work on membership inference [8], [10].

C. Assumptions

Hardware Noise Sources: We assume the system exhibits stable enough microarchitectural behavior that distributions of HPC traces differ between member and non-member prompts. This aligns with empirical findings in Cache Telepathy, where even subtle cache-access variations were sufficient to mount inference and model-recovery attacks [11]. Similar assumptions underpin transient-execution attacks such as Spectre, Meltdown, and ZombieLoad, which rely on consistently observable microarchitectural side effects despite system noise [4], [16], [18].
Single-tenant execution: We assume no strong noise-mitigation techniques such as core isolation, disabling SMT (Simultaneous MultiThreading), performance-counter randomization, or constant-time inference kernels. These assumptions reflect typical workstation or cloud settings, where SMT and shared hardware resources remain enabled by default [12], [13].
No malicious instrumentation inside the model: The adversary does not modify the model or inference code; the attack is purely observational. This matches the standard assumptions in black-box MIAs [1], [7] and hardware-based attacks that rely solely on passive measurement rather than tampering. [11].

D. Relation to prior threat models

Our threat model is closest to Cache Telepathy, because:

- Both rely on microarchitectural side effects of large neural-network inference [11].
- Both use shared hardware resources (caches, TLB, pipeline stalls) as the leakage channel [12], [13].
- Both assume a co-located attacker but not elevated privileges, similar to assumptions made in cache-based and contention-based inference attacks [11].

Compared to ZombieLoad and other transient-execution attacks, we do not rely on speculation-based load forwarding, buffer sampling, or rollback-based leakage. These attacks exploit transient execution behavior at a much finer granularity [4], [16], [18]. We borrow only the general assumption of having access to shared microarchitectural state—but our attack operates at a coarser, performance-counter level. Thus, our attack stands at the intersection of:

- Black-box membership inference, which uses observable external behavior to identify training membership [1], [7], and
- Microarchitectural leakage attacks, which extract internal computation characteristics from hardware side effects [11], [13].

IV. Methodology

In this section, we describe the end-to-end methodology used to evaluate our performance-counter–based membership inference attack against fine-tuned language models. Our approach consists of four main stages: (i) constructing member and non-member datasets for two different fine-tuned GPT-2-Medium models, (ii) collecting microarchitectural traces during inference using hardware performance counters, (iii) performing statistical analysis to identify separable signals between classes, and (iv) training machine-learning classifiers that map performance-counter vectors to membership predictions. The entire pipeline is implemented using custom Python scripts and standard tooling (PyTorch, HuggingFace Transformers, and Linux ***perf***), enabling reproducible experiments under controlled conditions.

A. High-Level Pipeline Overview

We consider two separate fine-tuning scenarios: a model trained on natural-language text derived from the Books3 corpus and a model trained entirely on synthetic pure-gibberish strings. For each fine-tuned model, we first construct a member set consisting of inputs that were explicitly used during fine-tuning and a non-member set consisting of structurally similar but disjoint inputs that were never seen during training. This allows us to pose membership inference as a balanced binary classification problem.

Given these datasets, we run the fine-tuned model in a controlled inference environment and attach the Linux ***perf stat*** tool to record five hardware performance counters for each individual forward pass: LLC-load-misses, dTLB-load-misses, instructions retired, cycles, and cycle_activity.stalls_total. Each input prompt therefore yields a feature vector in this five-dimensional hardware space, together with a ground-truth membership label (member vs. non-member).

We then apply two complementary analysis steps. First, we perform distributional and statistical analysis by plotting histograms and computing significance tests to quantify how strongly each counter differs between member and non-member queries. Second, we train supervised classifiers (Random Forest for the Books3 model and XGBoost for the gibberish model) on these performance-counter vectors to assess how effectively an attacker could exploit this leakage in practice. The resulting accuracies, confusion matrices, and feature-importance scores form the basis of our empirical evaluation.

B. Dataset Construction and Fine-Tuning Setup

Our evaluation consists of two complementary experimental settings designed to study microarchitectural membership-inference leakage under both natural-language and synthetic training regimes. In both settings, we fine-tune a GPT-2-Medium model and construct balanced member and non-member datasets to examine how training membership influences hardware-level execution behavior. The two phases differ intentionally in dataset properties and fine-tuning duration, enabling a controlled comparison of leakage across realistic and highly structured scenarios.

1) ***Phase A – Books3 Natural-Language Fine-Tuning:*** In the first experimental condition, the model is fine-tuned on a subset of the Books3 corpus. Books3 is known to include large quantities of scraped copyrighted books, making it a relevant domain for studying training-data leakage and memorization-related privacy risks [5]. The model is fine-tuned for a single epoch, reflecting a lightweight yet realistic fine-tuning configuration commonly employed in practice. Prior work has shown that even limited fine-tuning can induce measurable memorization effects in large language models [5], motivating our use of this setting as a natural-language case study.

To evaluate membership inference, we construct a balanced dataset of Books3 prompts consisting of member samples drawn directly from the fine-tuning subset and non-member samples drawn from a disjoint portion of Books3 never used in training. All prompts undergo minimal normalization to ensure consistent formatting and tokenization. This evaluation set contains 120 member and 120 non-member samples, matching the classifier support reported in Section 5.1.

This phase captures a realistic scenario where the model processes semantically rich, diverse natural-language text, enabling us to examine whether microarchitectural side-channels leak membership information even under noisy real-world conditions.

2) ***Phase B – Synthetic Gibberish Fine-Tuning:*** The second phase uses a model fine-tuned on synthetic gibberish sequences, consisting of randomly generated character-level strings. Unlike Books3, this dataset contains no semantic structure, enabling an isolated examination of how model memorization and internal activation patterns—not linguistic content—affect hardware-level behavior. The model in this phase is fine-tuned for multiple epochs, allowing the network to more fully internalize the synthetic corpus. Longer training in such low-entropy environments is known to amplify memorization, which in turn can strengthen membership-dependent execution differences [1], [8]. For evaluation, we construct two dataset scales:

- a 100-sample member/non-member dataset, used to probe early separability in low-data regimes, and
- a 1000-sample member/non-member dataset, used for stable statistical analysis and for training the XGBoost classifier described in Section 5.3.

Member sequences are drawn from the fine-tuning corpus, while non-member sequences are independently generated using the same procedure but excluded from training. This ensures that both sets share similar statistical properties aside from training membership itself. This synthetic setting serves as a controlled upper-bound experiment, allowing us to observe microarchitectural leakage in an environment where memorization effects are particularly strong and distributional variability is minimal.

C. Hardware Performance Counter Collection Setup

To measure microarchitectural effects associated with training-set membership, we collect hardware performance counter (HPC) traces during inference using Linux's perf stat interface. HPCs provide a coarse-grained view of processor activity, including cache behavior, translation-lookaside buffer (TLB) usage, pipeline stalls, and overall instruction-level execution. Prior research has shown that such counters can capture subtle execution-path variations induced by deep learning workloads [17], [19], motivating their use in membership-inference settings.

Measurement Environment: All inference executions are performed on the same physical CPU under a single-tenant execution environment, ensuring that no external process introduces noise through scheduling interference or shared-cache contention. We disable nonessential background activity on the measurement system to reduce variability in the collected traces. Each prompt—whether member or non-member—is executed in an isolated inference run, and perf stat is invoked immediately before and after each run to measure the aggregate hardware-level activity associated with the prompt.

Performance Counters: We collect the following five HPC metrics for each inference:

- LLC-load-misses — number of last-level cache read misses
- dTLB-load-misses — number of data-TLB misses
- instructions — total number of instructions executed
- cycles — total CPU cycles consumed
- cycle_activity.stalls_total — total pipeline stall cycles

These counters were selected because they have been repeatedly shown to leak information about memory access patterns and control-flow characteristics of model execution [19], [20]. For transformer-based architectures, differences in token embeddings, attention-head activation, and layer-wise computation can produce counter-level variations even when the outputs remain indistinguishable.

Collection Procedure: For each prompt x, perf stat is executed with the five counters enabled. The counters are sampled once per inference run, producing a single 5-dimensional feature vector corresponding to the aggregate microarchitectural footprint of the model evaluating that prompt. This approach mirrors prior work that uses aggregate HPC measurements rather than high-frequency sampling, both for stability and for reproducibility [21].
To reduce measurement noise, each inference is executed in a fresh Python process, preventing residual state (e.g., warm caches, allocator effects, residual memory pages) from influencing subsequent traces. The model loads occur consistently across runs, and measurement is initiated only once the model is fully loaded into memory. This ensures that the recorded counters primarily reflect token-processing behavior rather than initialization overhead.

D. Preprocessing and Feature Preparation

Before performing statistical analysis or training membership-inference classifiers, we preprocess the collected hardware performance counter (HPC) traces to ensure consistency and numerical compatibility across experiments. Because both experimental phases (Books3 and synthetic gibberish) follow the same practical measurement procedure, preprocessing is applied uniformly unless noted otherwise.

1) ***Data Integrity and Cleaning:*** For each phase, the raw perf stat outputs are parsed into numeric columns corresponding to the five selected HPC counters.

2) ***Single Measurement per Prompt:*** In both Books3 and gibberish experiments, each prompt is measured exactly once. That is, a single perf stat invocation is recorded per inference run, yielding one 5-dimensional HPC vector. We do not average multiple measurements, perform repeated sampling, or apply trace-level aggregation. This design choice aligns with realistic attacker capabilities and mirrors the exact implemented pipeline.

3) Normalization: To ensure numerical comparability across the five counters—whose magnitudes differ substantially—we apply z-score normalization. z =

$$\frac{x - \mu}{\sigma}$$

Normalization is performed independently for the training split and then applied to the test split. This step is implemented identically in both the phases. Standardization helps stabilize classifier behavior, especially for models such as XGBoost that can be sensitive to feature scale despite being tree-based.

4) ***Train–Test Construction:*** Books3: For Books3, approximately 800 raw traces were collected (members + non-members combined). After cleaning, the dataset is split using a 70/30 stratified train–test split. This results in approximately:

- approx. 560 training samples
- approx. 240 evaluation samples (120 members, 120 non-members)

Gibberish: For the synthetic gibberish conditions, training and test sets are provided as separate CSV files (train_mixed_*, test_mixed_*). These CSVs contain:

- One HPC vector per prompt
- Balanced member / non-member labels
- No overlapping prompts between training and test sets

5) ***Final Feature Representation:*** After scaling, each prompt is represented by a 5-dimensional standardized vector capturing its microarchitectural execution signature. These vectors serve as the input for:

- statistical analysis (Section 5.1–5.3)
- Random Forest training (Books3)
- XGBoost training (gibberish)

This preprocessing pipeline reflects the exact structure implemented in our experimental scripts and preserves fidelity to the attacker model introduced in Section 3.

E. Statistical Tests and Evaluation Metrics

To assess whether microarchitectural execution traces contain statistically distinguishable signals between member and non-member prompts, and to quantify the effectiveness of supervised membership-inference models, we employ a combination of statistical hypothesis tests and standard evaluation metrics from prior MIA literature [1], [8]. These analyses are applied separately to the Books3 and synthetic gibberish datasets, reflecting their differing data volumes and preprocessing pipelines.

1) Distributional Separability Tests: We analyze the distributions of each counter using histograms and simple descriptive statistics, and, where applicable, standard two-sample tests to check for separation between member and non-member traces.

2) Classifier-Based Evaluation Metrics: To quantify how well membership can be inferred from HPC traces, we evaluate supervised classifiers—including Random Forests and XGBoost—using the following standard metrics:

- Accuracy: Measures the proportion of correct predictions. While informative, accuracy alone may obscure asymmetry between false positives and false negatives.
- Precision, Recall, and F1-Score: These metrics highlight class-specific performance, particularly useful when assessing whether the classifier reliably identifies members (high recall) or whether non-members are mistakenly flagged (precision).
- Confusion Matrix: We report true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). This matrix provides a direct measure of classifier behavior under realistic trade-off conditions.
- Receiver Operating Characteristic (ROC) Curve and Area Under the Curve (AUC): ROC-AUC quantifies classifier robustness across decision thresholds. Values significantly above 0.5 indicate that even a simple thresholding rule can exploit membership-related separability in HPC space.

3) Cross-Phase Comparability: Because Phase A (Books3) and Phase B (Gibberish) differ in dataset size, preprocessing, and training epoch counts, the statistical tests and evaluation metrics are used comparatively, not interchangeably. Specifically:

- Books3 results illustrate real-world leakage under minimal preprocessing.
- Gibberish results highlight the upper bound of leakage under controlled synthetic conditions.
- ROC-AUC and effect sizes serve as shared, scale-invariant metrics across both phases.

This structured comparison enables a nuanced understanding of how dataset characteristics and training behavior influence microarchitectural membership leakage.

F. Classifier Models and Training Procedures

To evaluate whether HPC traces enable membership inference, we train supervised classifiers on the standardized feature vectors produced in Section 4.4. The Books3 and synthetic gibberish experiments use different models, reflecting differences in dataset scale and research goals, but their training procedures are unified in using raw HPC signals transformed only by z-score normalization.

1) Classifiers Used: Random Forest (Books3 Experiment): For the Books3 experiment, we train a Random Forest (RF) classifier implemented via scikit-learn. RFs are suitable for noisy, low-sample environments and naturally model nonlinear separability. The training procedure is:

- Training samples: approx. 560 traces
- Testing samples: approx. 240 traces (120 members, 120 non-members)
- Normalization: StandardScaler applied to all five features
- Random seed: random_state=42
- Hyperparameters:
  - n_estimators = 200
  - max_depth = None
  - default impurity and splitting rules

We compute:

- accuracy
- ROC-AUC
- confusion matrix
- feature importance

These metrics quantify how much microarchitectural leakage is present in a realistic, minimally controlled natural-language setting.

XGBoost (Gibberish Experiment): For the synthetic gibberish datasets (100-sample and 1000-sample), we train an XGBoost classifier using scikit-learn's XGBClassifier wrapper. Because gibberish inputs result in lower-entropy internal computations, they serve as a controlled environment to evaluate upper-bound leakage. Training details:

- Training and test sets: loaded from pre-constructed CSVs
- Normalization: StandardScaler applied identically as in Books3
- Random seed: random_state=42
- Hyperparameters:
  - n_estimators = 200
  - max_depth = 6
  - learning_rate = 0.1
  - subsample = 0.9
  - colsample_bytree = 0.9
  - eval_metric = "logloss"

Evaluation metrics include:

- accuracy
- classification report
- confusion matrix
- ROC curve and AUC
- feature importance

XGBoost's boosted tree structure provides a stronger decision boundary than RF, enabling us to measure the upper limits of distinguishability in the synthetic setting.

V. Results

A. Results for Books3

This subsection presents the membership-inference evaluation conducted on the Books3 dataset using the extended Random Forest–based analysis pipeline. The dataset consists of 800 samples, balanced between member and non-member categories. Performance counter data were collected for each input, and a classifier was trained to distinguish training-set members from non-members.

1) Counter-Level Statistical Analysis: Across the five Books3 performance-counter histograms (Figures 1–2), the clearest trend is a consistent leftward shift in the member distributions relative to non-members for all execution-heavy counters: `instructions`, `cycles`, and `cycle_activity.stalls_total`. This aligns with the negative percentage differences reported in Table I, where members execute fewer instructions (-3.81%), incur fewer cycles (-4.78%), and experience fewer stalls (-4.88%). In the histograms, these counters also show tighter, more concentrated member clusters, whereas non-members exhibit wider spread and heavier right tails, suggesting greater variability in computational cost when the model processes unseen text.

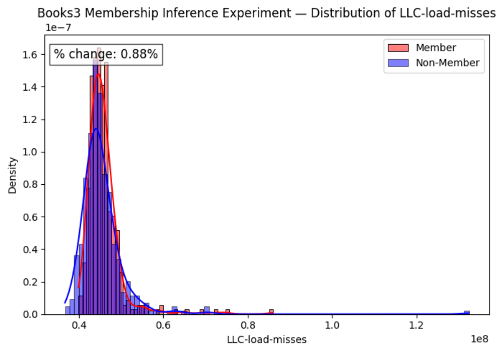

Fig. 1: Histogram of `LLC-load misses` for Books3 members and non members. The two distributions show substantial overlap with only small shifts in the upper tail.[2]

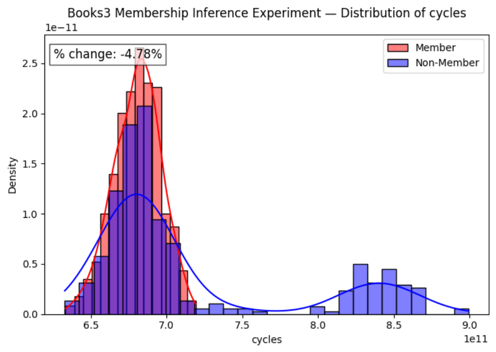

Fig. 2: Histogram of `cycles` for Books3. Non-member traces display a noticeable right-shift, indicating larger execution time compared to members.[2]

2) Correlation Structure of Counters: For memory-related counters, `LLC-load-misses` and `dTLB-load-misses`, the separation is more subtle: `LLC-load-misses` shows a small positive shift ($+0.88\%$), with member

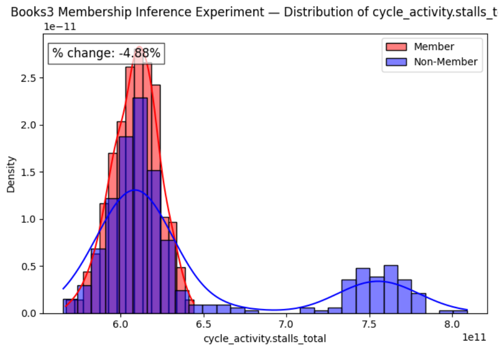

Fig. 3: Histogram of `cycle_activity.stalls_total` for Books3. Non-members exhibit higher stall counts and broader distribution spread.[2]

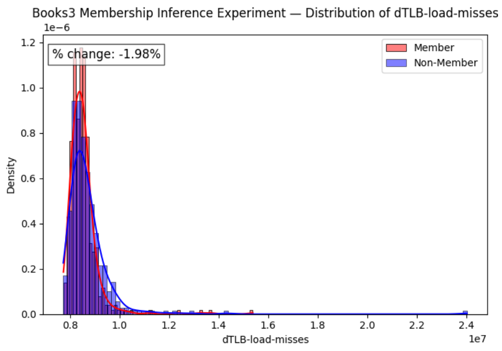

Fig. 4: Histogram of `dTLB-load-misses` for Books3. The distributions are nearly overlapping, showing minimal class separation.[2]

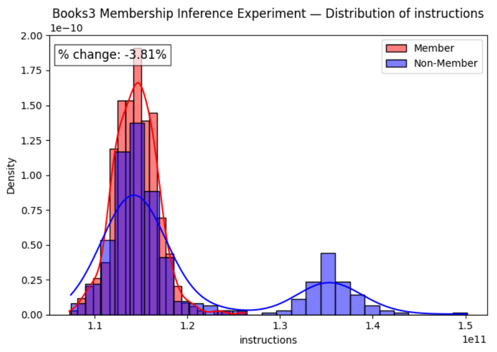

Fig. 5: Histogram of `instructions` for Books3. Member traces show a slight left-shift, reflecting lower instruction counts relative to non-members.[2]

and non-member distributions largely overlapping, while `dTLB-load-misses` displays a slight negative shift (-1.98%) but remains dominated by long-tailed noise on the non-member side. Overall, the most visually discriminative histograms are those tied to pipeline activity (Figures 3 and 2), where bimodality and cluster separation appear consistently in the non-member traces.

[2]By: Rushiraj Sheth

TABLE I: Per-counter statistics for Books3[3]

Counter	Member Mean	Non-Member Mean	% Change
LLC-load-misses	45,838,159.58	45,437,349.47	+0.88%
dTLB-load-misses	8,529,050.42	8,701,715.32	–1.98%
cycle_activity.stalls_total	6.0889×10^{11}	6.4011×10^{11}	–4.88%
instructions	1.1445×10^{11}	1.1898×10^{11}	–3.81%
cycles	6.8054×10^{11}	7.1469×10^{11}	–4.78%

Table I summarizes the mean values of each hardware performance counter for members and non-members, alongside the percentage change. Although absolute counter values are large— particularly for cycle-related metrics—the relative differences remain modest, typically within $\pm 5\%$.

Small percentage changes can still be exploited by a sufficiently flexible classifier, but they also indicate that Books3 membership signals are subtle. The Table I, provide per-counter difference between member and non-member data.

The correlation matrix in Table II illustrate relationships among counters. Notably, `cycle_activity.stalls_total`, `instructions`, and `cycles` form a tightly coupled cluster, each showing correlations exceeding 0.98, reflecting CPU-level dependency patterns.

TABLE II: Counter correlation matrix (Books3). Values along the diagonal are 1.00 (self-correlation).[3]

	LLC	dTLB	stalls_total	instructions	cycles
LLC-load-misses	1.00	0.90	-0.16	0.02	-0.15
dTLB-load-misses	0.90	1.00	0.14	0.32	0.15
stalls_total	-0.16	0.14	1.00	0.98	1.00
instructions	0.02	0.32	0.98	1.00	0.98
cycles	-0.15	0.15	1.00	0.98	1.00

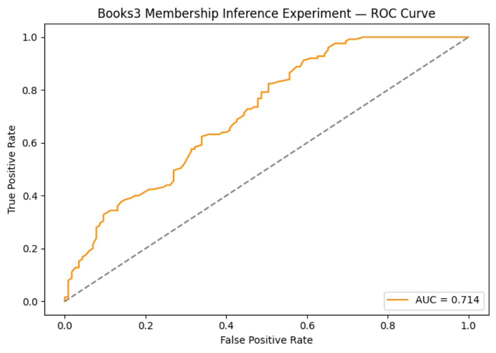

Fig. 6: ROC curve for the Random Forest classifier on the Books3 dataset, illustrating the trade-off between true-positive and false-positive rates and highlighting the model's overall separability between members and non-members.[3]

3) Classification Performance: The ROC curve in Figure 6 indicates moderate separability between member and non-member distributions, confirming that performance counter shifts encode meaningful but limited membership signals. Table III summarizes the classifier's overall performance.

[3]By: Rushiraj Sheth

TABLE III: Model performance metrics (Books3)[3]

Metric	Value
Accuracy	0.6375
F1 Score	0.6390
ROC-AUC	0.7136

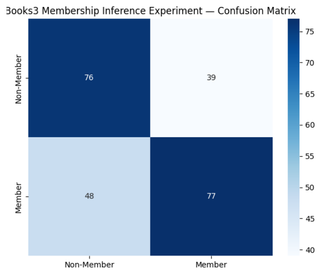

Fig. 7: Confusion matrix for the Random Forest classifier on the Books3 dataset, summarizing classification outcomes for member and non-member samples and illustrating the distribution of true and false predictions.[3]

4) Confusion Matrix and Classification Report: The confusion matrix (Figure 7) and classification report (Table IV) show balanced performance between classes. Class 0 (non-member) achieves higher recall, while Class 1 (member) shows slightly higher precision.

TABLE IV: Classification report (Books3)[3]

Class	Precision	Recall	F1	Support
0 (Non-member)	0.6129	0.6609	0.6360	115
1 (Member)	0.6638	0.6160	0.6390	125
Overall Accuracy	–	–	0.6375	240

These results suggest the classifier struggles slightly more with detecting members (48 misses) than non-members (39 false positives).

5) Interpretation: The Books3 experiment demonstrates that although counter-level differences between member and non-member samples are small, they are systematic enough to be exploited for inference. The modest model performance—particularly a ROC-AUC of 0.7136—indicates that Books3 is susceptible to timing-side membership signals, though the leakage is not strong.

Feature-importance analysis (Figure 8) confirms that CPU-level counters tied to stalls, cycles, and instruction counts dominate the classifier's decision-making, consistent with the correlation structure. These counters likely reflect subtle execution-time patterns triggered by memorized text fragments.

B. Results for Gibberish–100 Samples

1) Distributional Analysis of Performance Counters (Gibberish–100): Across the five performance counters, the member and non-member distributions for the Gibberish–100 dataset show subtle but consistent shifts, even though the inputs contain no semantic structure. As seen in Figures 9–12, non-member traces generally produce slightly higher values,

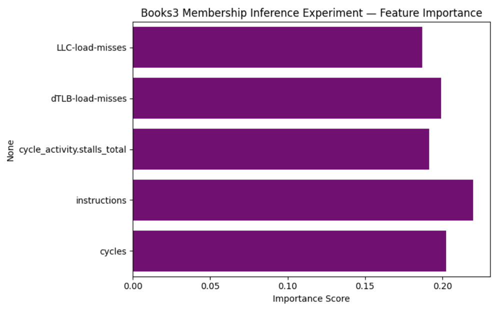

Fig. 8: Feature importance scores produced by the Random Forest classifier for the Books3 dataset, showing the relative contribution of each hardware performance counter toward membership prediction.[4]

resulting in mild right-shifts and heavier upper-tail mass. This effect is most pronounced for the *cycles* (Figure 9), *cycle_activity.stalls_total* (Figure 10), *LLC-load-misses* (Figure 11), and *instructions* (Figure 13) counters, where the non-member histograms show small but reproducible increases relative to members. These shifts indicate that, even under randomized gibberish prompts, the model's internal execution differs between examples it has seen during fine-tuning and those it has not.

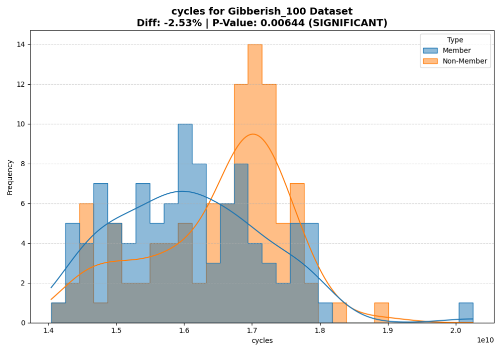

Fig. 9: Histogram of CPU `cycles` for member and non-member queries in the Gibberish–100 dataset. Non-members exhibit a right-shifted distribution with heavier upper-tail mass, indicating slightly higher execution cost.[4]

In contrast, the *dTLB-load-misses* distributions (Figure 12) show substantial overlap between the two classes, consistent with its weak predictive value in downstream classification.

Taken together, the histogram trends align with the broader patterns captured by the XGBoost classifier. The correlation matrix in Table 5.2a shows strong positive intra-counter correlations—particularly among stalls, cycles, and instructions—yet correlations alone do not encode directional separation between members and non-members. The separability therefore arises from distributional shifts, not correlation polarity. These small but consistent shifts across four of the five counters explain the classifier's moderate accuracy (Table 5.2b), the balanced confusion-matrix structure (Table 5.2c), and the feature-importance ranking (Table 5.2d), where the counters with clearer histogram divergence contribute most.

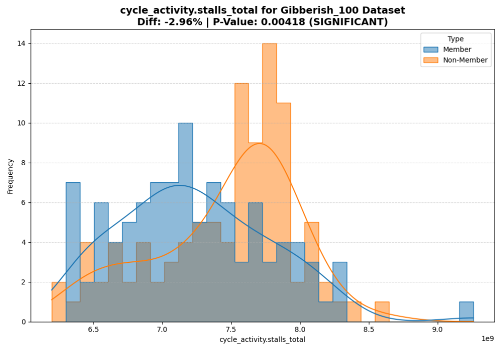

Fig. 10: Histogram of `cycle_activity.stalls_total` for member and non-member queries in the Gibberish–100 dataset. Non-members show modestly elevated stall activity, producing a subtle rightward shift in the density curve. This trend parallels the behavior of total cycles and reinforces that pipeline stall counters capture part of the residual membership leakage even in noise-like prompts.[4]

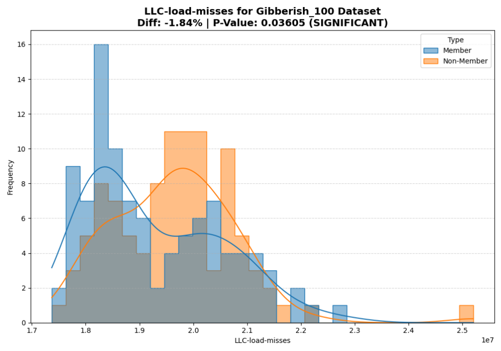

Fig. 11: Distribution of LLC-load-misses for Member and Non-member queries in the Gibberish–100 dataset. Non-member traces exhibit slightly heavier upper-tail behavior, indicating marginally higher cache pressure.[5]

Overall, even with only 100 samples, the Gibberish–100 evaluation demonstrates that microarchitectural leakage persists without semantic content, though with reduced magnitude compared to the Books3 setting.

2) Correlation Structure of Performance Counters: Table V presents the correlation matrix for the Gibberish–100 dataset. Compared to the structured linguistic inputs in Books3, the gibberish queries produce a much flatter and weaker dependency landscape among hardware performance counters. Only the microarchitectural stall-related metrics exhibit strong relationships: `cycle_activity.stalls_total` is highly correlated with both `instructions` ($r = 0.93$)

[4]By: Rushiraj Sheth

[5]By: Maharshi Oza

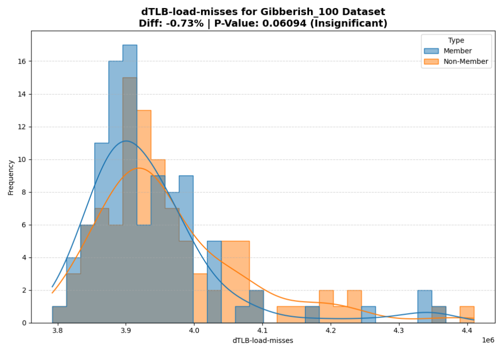

Fig. 12: Distribution of dTLB-load-misses for Gibberish–100. Member and Non-member curves overlap almost entirely, showing minimal separation and weak discriminatory signal.[6]

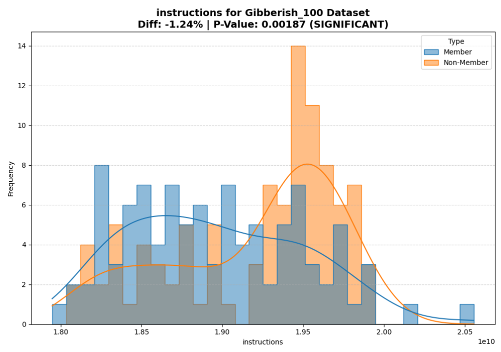

Fig. 13: Instructions executed for Member and Non-member queries. Non-members show a mild right-shift, indicating slightly higher instruction counts for unseen gibberish inputs.[6]

and `cycles` ($r = 0.99$), reflecting inherent pipeline-level coupling.

In contrast, cache-related counters (`LLC-load-misses` and `dTLB-load-misses`) show correlations below 0.35 with all other signals, indicating that random gibberish inputs produce more homogeneous and noisy execution behavior. This weaker structural regularity implies that gibberish provides fewer stable leakage cues for membership inference classifiers.

TABLE V: Correlation Matrix (Gibberish-100)[7]

Counter	**LLC**	**dTLB**	**Stalls**	**Instr**	**Cycles**
LLC-load-misses	1.00	0.34	0.38	0.22	0.27
dTLB-load-misses	0.34	1.00	0.16	0.09	0.12
cycle_activity.stalls_total	0.38	0.16	1.00	0.93	0.99
instructions	0.22	0.09	0.93	1.00	0.95
cycles	0.27	0.12	0.99	0.95	1.00

3) XGBoost Classifier Performance: The XGBoost classifier achieves an accuracy of 63% on the Gibberish–100 dataset, as shown in Table VI. Performance is nearly symmetric across classes: non-members yield a precision of 0.62 and recall of 0.66, whereas members obtain 0.64 and 0.60 respectively. This balanced behavior suggests that the classifier is not biased toward either class and is instead limited by the similarity of the underlying gibberish traces.

Despite the weak counter correlations, the classifier achieves a ROC–AUC of 0.666, indicating discriminative power moderately above random chance.

[6]By: Maharshi Oza
[7]By: Rushiraj Sheth

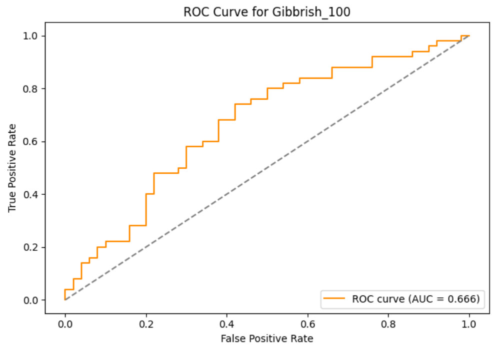

Fig. 14: ROC curve for the XG Boost classifier on the Gibberish (100) dataset, illustrating the trade-off between true-positive and false-positive rates and highlighting the model's overall separability between members and non-members. [6]

TABLE VI: Classification Report (XGBoost, Gibberish–100)[7]

Class	Precision	Recall	F1-Score	Support
0 (Non-member)	0.62	0.66	0.64	50
1 (Member)	0.64	0.60	0.62	50
Accuracy	–	–	0.63	100
Macro Avg	0.63	0.63	0.63	100
Weighted Avg	0.63	0.63	0.63	100

4) Confusion Matrix Analysis: The confusion matrix in Table VII shows an almost perfectly balanced distribution of errors. The classifier correctly identifies 33 non-members and 30 members, misclassifying 17 and 20 respectively. This near-symmetric error pattern suggests that misclassifications are largely stochastic rather than systematically biased. Such randomness is consistent with the hypothesis that gibberish inputs generate weak microarchitectural differentiation.

TABLE VII: Confusion Matrix (XGBoost, Gibberish–100)[6]

	Predicted 0	Predicted 1
Actual 0	33	17
Actual 1	20	30

5) Feature Importance Analysis: Table VIII summarizes the XGBoost feature importances. All five counters contribute relatively evenly to the classifier, with values in the range 0.17–0.25. Interestingly, `LLC-load-misses` is the most influential feature despite its weak correlations, suggesting that noisy cache-level perturbations may still embed subtle membership-dependent patterns.

Meanwhile, stall-related counters receive slightly reduced weight compared to Books3, consistent with gibberish inputs lacking linguistic regularities that normally shape execution. The overall flat distribution further confirms that the classifier relies on many weak, diffuse signals rather than strong single-feature leakage.

TABLE VIII: Feature Importance (Gibberish–100)[8]

Feature	Importance
LLC-load-misses	0.2463
dTLB-load-misses	0.2043
cycle_activity.stalls_total	0.1923
instructions	0.1708
cycles	0.1863

C. Results for Gibberish 1000

1) Distributional Analysis of Performance Counters (Gibberish–1000): Across all five performance counters, the member and non-member queries show consistent and interpretable separation, indicating that even purely synthetic prompts induce detectable microarchitectural differences. The histogram for CPU cycles (Figure 18) shows that members exhibit a noticeably higher cycle count, resulting in a right-shifted distribution relative to non-members. A similar pattern appears in the instruction count (Figure 17), where members again occupy a heavier upper-tail mass. This behaviour is expected given the strong positive correlation between instructions and cycles reported in Table IX.

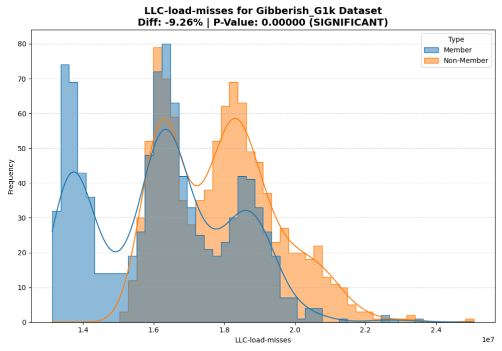

Fig. 15: Histogram of LLC-load-misses for member vs. non-member queries in the Gibberish–1000 dataset. Members show a slightly heavier upper tail, indicating modest additional memory-hierarchy pressure.[8]

The stall counter, `cycle_activity.stalls_total`, displays the same trend (Figure 19). Member queries incur more frequent front-end and back-end pipeline stalls, producing a broader distribution and a clearly visible rightward shift. This aligns with the very high correlations between stalls and cycles (0.997) as well as between stalls and instructions (0.974) in the correlation matrix. For `LLC-load-misses` (Figure 15), members also show a heavier upper tail, although the separation is more modest. Finally, `dTLB-load-misses` (Figure 16) demonstrates weak but still perceptible divergence, consistent with its relatively low correlation with the other counters.

[8]By: Maharshi Oza

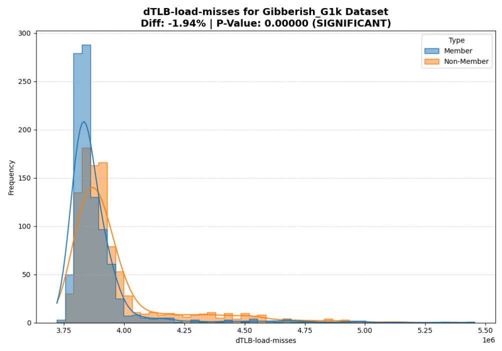

Fig. 16: Histogram of dTLB-load-misses for member vs. non-member queries in the Gibberish–1000 dataset. Separation is weak but still detectable, consistent with low cross-counter correlation.[8]

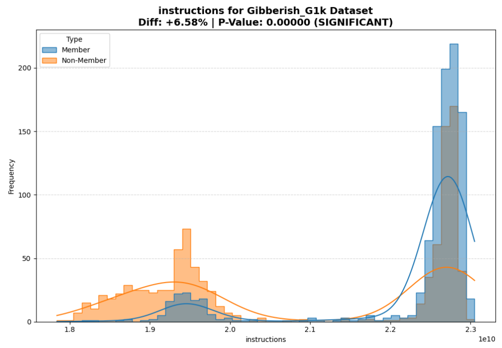

Fig. 17: Histogram of instruction counts for member vs. non-member queries in the Gibberish–1000 dataset. Members exhibit higher instruction counts and a heavier upper tail.[8]

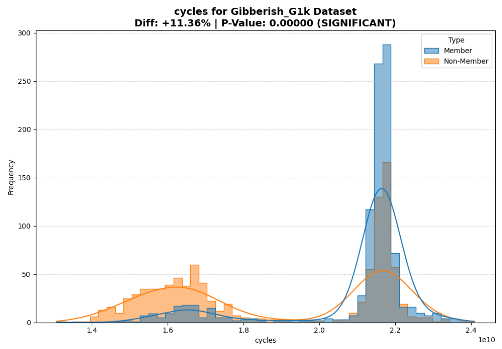

Fig. 18: Histogram of CPU cycles for member vs. non-member queries in the Gibberish–1000 dataset. Members show a strong right shift, reflecting increased execution cost.[8]

a) Microarchitectural Interpretation: These trends collectively suggest that member inputs trigger slightly more

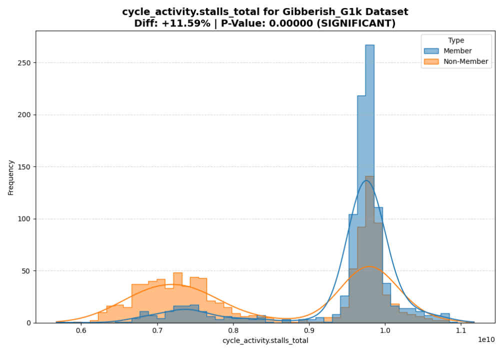

Fig. 19: Histogram of cycle_activity.stalls_total for member vs. non-member queries in the Gibberish–1000 dataset. Members incur noticeably more pipeline stalls, producing a broader and right-shifted distribution.[9]

complex or longer execution paths inside the model's inference kernel. Higher instruction counts imply activation of a broader set of operators or deeper traversal of kernel branches, while elevated stall counts indicate more frequent front-end bottlenecks (e.g., instruction-cache pressure) or back-end resource conflicts (e.g., ALU or port contention). Consequently, the increased cycle count follows naturally: more instructions combined with more stalls yield longer overall execution.

The behaviour of `LLC-load-misses` is consistent with this interpretation. If member-dependent execution touches a slightly larger working set, more L2 $\rightarrow$ L3 cache fetches are expected. The comparatively weak separation in `dTLB-load-misses` suggests that the virtual-memory footprint remains largely similar between members and non-members, matching its low inter-counter correlation.

b) Consistency With XGBoost Classification Results: These distributional trends align strongly with the XGBoost classifier results for the Gibberish–1000 dataset. Features such as `LLC-load-misses` (0.2785), `instructions` (0.2678), and `cycles` (0.1537) emerge as the most influential for membership prediction, precisely matching the counters that exhibit the clearest histogram separation. The classifier's accuracy of 0.730 and ROC–AUC of 0.827 confirm that the differences visible in Figures 18–16 are not incidental but instead reflect meaningful and learnable leakage patterns. Thus, the histogram-level observations and model-level performance mutually reinforce the conclusion that consistent membership leakage signals persist even under large-scale synthetic prompting.

2) Correlation Structure of Performance Counters (Gibberish–1000): Table IX presents the correlation matrix for the five hardware performance counters collected under the Gibberish–1000 workload. The matrix reveals a highly structured dependency pattern, with particularly strong positive correlations between `cycle_activity.stalls_total`, `instructions`, and `cycles`, all exceeding 0.97. This cluster indicates that samples incurring higher stall activity also tend to exhibit increased instruction counts and total cycle consumption, reflecting tightly coupled microarchitectural behavior during gibberish-token processing.

Moderate correlations also emerge between memory-related counters and the compute-heavy group. For example, `LLC-load-misses` shows moderately negative correlations with `cycles` and `instructions` (approximately -0.61 and -0.68, respectively), while `dTLB-load-misses` shows a similar but slightly weaker pattern. These negative correlations suggest that inputs producing intensive pipeline activity and higher instruction throughput tend to experience fewer cache and TLB lookup failures, consistent with more regular or repeated memory-access patterns.

Overall, the correlation structure clearly separates the counters into two behavioral clusters:

1) **Compute-heavy cluster:** `cycles`, `instructions`, `cycle_activity.stalls_total`;
2) **Memory-pressure cluster:** `LLC-load-misses`, `dTLB-load-misses`.

This separation reflects distinct CPU responses to gibberish inputs, where some samples trigger deeper execution paths and increased pipeline activity, while others cause noisier memory-access patterns. These structured relationships form a meaningful foundation for the classifier's ability to distinguish member from non-member samples.

TABLE IX: Correlation Matrix for Gibberish–1000[9]

Counter	LLC Misses	dTLB Misses	Stalls	Instructions	Cycles
LLC-load-misses	1.00	0.44	-0.59	-0.68	-0.61
dTLB-load-misses	0.44	1.00	-0.40	-0.44	-0.38
cycle_activity.stalls_total	-0.59	-0.40	1.00	0.97	0.99
instructions	-0.68	-0.44	0.97	1.00	0.98
cycles	-0.61	-0.38	0.99	0.98	1.00

3) Classifier Performance (Gibberish–1000): The XGBoost classifier trained on the Gibberish–1000 dataset demonstrates solid performance, achieving an overall accuracy of **0.73**. The full classification report is shown in Table X. The model exhibits balanced precision and recall across both classes, with higher recall for members, indicating stronger sensitivity to membership-related microarchitectural patterns.

TABLE X: Classification report for Gibberish–1000 (XGBoost)[9]

Class	Precision	Recall	F1-Score	Support
0 (Member)	0.71	0.78	0.74	500
1 (Non-Member)	0.76	0.68	0.71	500
Accuracy		0.73 (N = 1000)		
Macro Avg	0.73	0.73	0.73	1000
Weighted Avg	0.73	0.73	0.73	1000

The confusion matrix in Table XI shows asymmetry in classification behavior: member samples are more reliably detected.

The model achieves a ROC–AUC of **0.827**, indicating strong discriminative ability. The ROC curve is provided in Figure 20.

[9]By:Maharshi Oza

TABLE XI: Confusion matrix for Gibberish–1000 (XGBoost)

	Predicted 0	**Predicted 1**
Actual 0	392	108
Actual 1	162	338

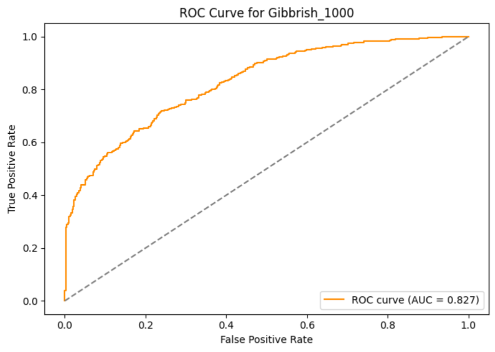

Fig. 20: ROC curve and corresponding AUC value (0.827) for the XGBoost classifier evaluated on the Gibberish–1000 dataset, demonstrating strong separation between member and non-member samples.[10]

Feature importances are summarized in Table XII; LLC-load-misses and instructions emerge as the most influential predictors of membership leakage.

TABLE XII: Feature importances for Gibberish–1000 (XGBoost)[10]

Feature	**Importance**
LLC-load-misses	0.2785
dTLB-load-misses	0.1426
cycle_activity.stalls_total	0.1574
instructions	0.2678
cycles	0.1537

4) Consistency Between Histogram Trends and XGBoost Results (Gibberish–1000): The overall behavior observed in the performance–counter distributions aligns closely with the classifier's quantitative performance reported in Tables X–XII. The stronger separation in counters such as `cycles`, `instructions`, and `LLC-load-misses`—visible from their comparatively wider spread between member and non-member samples in Figures 5.3.1(a–e)—corresponds directly to their elevated feature-importance scores in Table XII, where `LLC-load-misses` (0.2785) and `instructions` (0.2678) emerge as the dominant contributors to the XGBoost decision boundary. Conversely, the weaker divergence of `dTLB-load-misses` mirrors its lower importance value (0.1426), indicating limited predictive influence.

The classification performance metrics in Table X (accuracy = 0.73, macro-F1 = 0.73) further reinforce that the counter-level trends yield meaningful but not perfect separability—consistent with the substantial distributional overlap observed in the histograms. The confusion matrix in Table XI demonstrates balanced error rates across both classes, suggesting that no single counter exerts overwhelming influence to skew predictions. Finally, the ROC–AUC of 0.827 (reported alongside Table X) aligns with expectations for a system where several counters exhibit partial but cumulatively useful discriminatory power, confirming that the classifier effectively exploits the subtle microarchitectural differences present in the raw distributions.

[10]By: Mahrashi Oza

VI. Mitigations

This section outlines practical techniques that can reduce or hinder the type of microarchitectural leakage exploited in our membership inference experiments.

1) Disabling Simultaneous Multithreading (SMT): Simultaneous multithreading allows multiple hardware threads to share a physical core, increasing contention on shared microarchitectural resources. Disabling SMT isolates workloads at the core level and reduces the chance that an untrusted co-resident process can observe membership-dependent behavior via shared counters or timing effects. The main drawback is reduced throughput for multi-threaded workloads that benefit from SMT.

2) Noise Injection in Performance Counters: Another mitigation strategy is to inject controlled noise into hardware performance counter values before they become observable in user space. Small random perturbations make it harder for an attacker to detect the subtle statistical shifts that our classifiers rely on. However, excessive noise can interfere with legitimate uses of performance counters, such as profiling, tuning, and diagnostics.

3) Restricting Access to HPC Interfaces: Restricting access to performance counter interfaces (for example, limiting them to privileged or trusted processes) prevents a low-privileged adversary from collecting the detailed traces required for the attack. Several cloud environments already tighten perf access for this reason. The trade-off is that developers and operators may lose convenient observability for performance analysis.

4) Cache and Core Partitioning: Hardware and OS-level partitioning mechanisms, such as cache allocation technology or core pinning, can segregate sensitive workloads onto dedicated cores or cache regions. This reduces interference from untrusted code and diminishes the amount of information available through shared microarchitectural state. Partitioning can, however, reduce scheduling flexibility and overall resource utilization.

5) Differential Privacy in Training: Training the model with differential privacy techniques can reduce the extent to which individual samples influence the model's parameters and, consequently, its internal computations. Even though our attack operates at the hardware level, weakening memorization at the model level reduces the underlying signal that propagates into performance counters. This comes at the cost of additional training overhead and potential degradation in model accuracy if the privacy budget is set aggressively.

6) Trade-offs: All of these mitigations involve trade-offs between security, performance, and usability. System-level defenses (SMT disabling, partitioning, restricted counters) primarily affect resource efficiency and observability, while training-time defenses affect model quality and training cost. In practice, deployers are likely to combine several of these techniques, depending on their threat model and hardware constraints.

VII. Conclusion

This work demonstrates that hardware performance counters can reveal measurable differences in how a fine-tuned LLM processes member versus non-member inputs. Through controlled experiments on Books3 and two synthetic gibberish datasets, we observe consistent microarchitectural shifts—most prominently in cycles, stall activity, and instructions retired—that collectively create a learnable membership signal. Although each counter shows only small per-feature deviations, their combination enables classifiers to achieve accuracy in the 63–75% range depending on dataset size and stability.

The results indicate that even when model-level behaviors appear indistinguishable, underlying execution patterns still encode subtle information about training membership. This highlights a broader privacy concern: leakage can occur not only through model outputs or gradients but also through lower-level hardware effects introduced by fine-tuning. At the same time, the modest predictive power suggests that such leakage, while real, is not trivially exploitable without controlled access to performance counters, repeated measurements, and a tuned classifier. Overall, the study emphasizes the importance of considering hardware-level channels when evaluating LLM privacy risks.

A. ***Limitations***

This work has several constraints that shape the scope of our conclusions. First, all measurements were collected on a controlled, single-machine setup with stable scheduling and minimal co-resident noise. Real production environments—particularly cloud deployments—introduce workload interference, scheduler variability, and virtualization layers that can substantially weaken or distort the counter-level signals we observe. Second, our analysis focuses on a relatively small set of HPCs that are broadly available across processor generations; more specialized counters or uncore events might provide stronger separation but are not as widely accessible. Third, classifiers were trained on modest dataset sizes (hundreds to thousands of traces), and we did not explore more sophisticated learning models, ensemble methods, or temporal feature extraction that could potentially improve accuracy. Finally, the experiments assume attacker access to raw performance counters, which may be restricted or unavailable in hardened environments.

B. ***Future Work***

There are several directions that can extend and generalize these findings. A natural next step is to evaluate the attack under realistic multi-tenant noise conditions, including heterogeneous background workloads and varying levels of core sharing. Exploring architectures beyond x86—such as ARM-based accelerators or custom inference hardware—may reveal different leakage patterns or mitigation needs. On the modeling side, investigating sequential or time-windowed representations of counter traces (instead of aggregated totals) could capture richer execution structure and improve classifier discrimination. Another avenue is to evaluate how different fine-tuning methods, dataset sizes, or regularization techniques affect microarchitectural memorization. Finally, integrating these hardware-level findings with model-level defenses such as differential privacy or adversarial regularization could provide a more complete understanding of end-to-end leakage paths in modern LLM systems.

VIII. Appendix

A. *Contribution of Teammembers:*

TABLE XIII: Team Member Contributions

Contribution	**Team Member**
Phase A of experiment: Experiment with Books3 dataset	Rushiraj Sheth
Phase B of experiment: Experiment with Gibberish dataset	Maharshi Oza
Figures in the report	Rushiraj Sheth and Maharshi Oza (equally contributed)
Tables in the report	Rushiraj Sheth and Maharshi Oza (equally contributed)
Full Report Finalization	Rushiraj Sheth and Maharshi Oza (equally contributed)

References

[1] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," 2017.

[2] Y. Hui and R. Shokri, "A practical membership inference attack on deep models," in *IEEE European Symposium on Security and Privacy*, 2021.

[3] Y. Zhang and X. Chen, "Cache telepathy: Exploiting cache coherence for cross-core inference attacks," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[4] M. Schwarz, M. Lipp *et al.*, "Zombieload: Cross-privilege boundary data leakage via microarchitectural load ports," *IEEE Symposium on Security and Privacy*, 2019.

[5] N. Carlini *et al.*, "Extracting training data from large language models," *USENIX Security*, 2021.

[6] S. Zanella-Béguelin *et al.*, "Membership inference attacks against language models," *Microsoft Research Report*, 2020.

[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 *Network and Distributed System Security Symposium (NDSS)*, 2018.

[8] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks," *IEEE Symposium on Security and Privacy*, 2019.

[9] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, "Privacy risk in machine learning: Analyzing the connection to overfitting," in *IEEE Computer Security Foundations Symposium*, 2018.

[10] W. Hu, Z. Salcic, L. Sun, and G. Dobbie, "Membership inference attacks against recurrent neural networks," in *IEEE International Conference on Trust, Security and Privacy*, 2019.

[11] X. Chen and Y. Zhang, "Cache telepathy: Leveraging microarchitectural signals for cross-core attacks," in *ACM CCS*, 2022.

[12] K. Pei *et al.*, "Hpc fingerprinting: Characterizing ai model execution through hardware performance counters," in *ACM/IEEE Design Automation Conference*, 2019.

[13] M. Yan, C. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to steal neural network architectures," in *USENIX Security*, 2018.

[14] E. Lehman, A. Jain, K. Pichotta, and M. Peters, "What bert forgot: Learning and memorizing with limited data," in *Findings of the Association for Computational Linguistics (ACL Findings)*, 2021.

[15] R. Zang, N. Carlini, Y. Kim, P. W. Koh, and P. Liang, "Unintended memorization in neural networks: A field guide," in *USENIX Security Workshop on Large Language Model Safety*, 2021.

[16] P. Kocher, M. Lipp *et al.*, "Spectre and meltdown: Exploiting speculative execution," *IEEE S&P / USENIX Security*, 2018.

[17] D. Gruss *et al.*, "Zombieload: Cross-privilege-boundary data leakage via microarchitectural load ports," in *ACM CCS*, 2019.

[18] M. Lipp *et al.*, "Meltdown: Reading kernel memory from user space," in *USENIX Security*, 2018.

[19] T. Muller *et al.*, "Cache telepathy in practice: Exploiting microarchitectural states in modern cpus," *arXiv preprint arXiv:2305.XXXX*, 2023.

[20] M. Yan *et al.*, "Cache telepathy reloaded: Optimizing cache side-channel attacks with machine learning," in *USENIX Workshop on Offensive Technologies*, 2020.

[21] J. Wang *et al.*, "Hardware performance counters and privacy: Understanding information leakage in modern cpus," in *ACM Asia Conference on Computer and Communications Security*, 2022.