LLMProbe: Microarchitectural Side-Channel Fingerprinting of LLM Inference Parameters

ECE-592 Micro-architecture Reverse Engineering and Security
Department of Electrical and Computer Engineering
Devesh Jani - 200597856, dhjani2@ncsu.edu
Misha Joshi - 200599998, mjoshi7@ncsu.edu
NC State University
Raleigh, NC, USA

***Abstract*—Microarchitectural side channels become more exploitable at the extremes. We demonstrate this principle by examining opposing operating conditions in LLM inference: comparing minimum vs. maximum context windows (128 vs 2048 tokens), deterministic vs. stochastic decoding (greedy vs. sampling), and small vs. large models (1.1B vs. 8B parameters). These limit studies reveal that the exploitable "channel width"—the measurable divergence in microarchitectural behavior—amplifies dramatically with model scale.**

We present LLMProbe, an SMT-based attack leveraging cache, TLB, and branch predictor contention on sibling hyperthreads to fingerprint CPU-based LLM inference parameters. Evaluating against TinyLLaMA-1.1B and DeepSeek-8B running under `llama.cpp`, we achieve 80.5% context classification accuracy (47.2% above random baseline), 90.0% decoding accuracy (40.0% advantage), and 61.4% semantic classification (36.4% advantage). Critically, the 512-token context creates the strongest signal with 93% recall due to L3 cache capacity effects, while greedy decoding shows 98% recall from predictable control flow.

Our cross-model analysis reveals systematic leakage amplification: DeepSeek-8B exhibits 2.2$\times$ higher kurtosis (1498 vs 666), 1.8$\times$ higher skewness (38.7 vs 21.7), and 1.6$\times$ larger cycle count range compared to TinyLLaMA. This amplification stems from larger KV-cache working sets exhausting LLC capacity (256MB vs 32MB), deeper call stacks creating more branch mispredictions, and extended execution windows providing richer temporal signatures. Extrapolating to GPT-4-class models (100B+ parameters), we project 5–10$\times$ signal amplification, making production deployments critically vulnerable.

We discuss mitigation trade-offs: SMT isolation eliminates the attack but halves throughput, while context padding and decoding normalization impose 1.5–8$\times$ computational overhead.

***Index Terms*—Side-channel attacks, LLM security, microarchitecture, SMT contention, inference fingerprinting**

I. Introduction

Large language models have become foundational components of modern computing infrastructure, deployed in applications ranging from code generation to customer service. These models are increasingly offered as black-box services where clients interact only through text-based APIs, while the underlying implementation details—model architecture, quantization scheme, context limits, and decoding parameters—remain opaque by design. This opacity serves multiple purposes: protecting service providers' intellectual property, preventing adversarial model extraction, and maintaining competitive advantages in a rapidly evolving market.

In shared computing environments, particularly cloud infrastructure and multi-tenant systems, LLM inference services commonly execute on general-purpose CPUs where multiple user processes time-share cores through Simultaneous Multi-Threading (SMT). While this improves hardware utilization and reduces operational costs, it creates opportunities for microarchitectural side-channel attacks where malicious co-resident processes can observe victim behavior through contention on shared hardware resources.

A. Motivation and Problem Statement

Microarchitectural side channels have been extensively studied in cryptographic contexts, where attackers exploit cache timing to extract secret keys [1], [2]. More recent work has explored neural network inference [3]. However, the specific characteristics of modern LLM inference pipelines—particularly those implemented in highly optimized C/C++ frameworks like `llama.cpp`—remain relatively unexplored from a side-channel perspective.

This gap is significant because LLM inference exhibits several properties that may create exploitable microarchitectural signatures:

- **Context-dependent working sets:** Different context lengths (e.g., 128 vs. 2048 tokens) create substantially different memory access patterns in the KV-cache, potentially visible through cache and TLB contention.
- **Decoding strategy control flow:** Greedy decoding follows deterministic token selection, while sampling involves random number generation and branching, potentially creating distinct branch predictor signatures.
- **Semantic computation patterns:** Different types of content (mathematical expressions, code, natural language) may induce varied execution paths and memory access patterns during token generation.

B. Research Questions

This work investigates the following central question: *What can an unprivileged co-located attacker learn about LLM inference runs using only microarchitectural timing measurements from a sibling hyperthread?*

Specifically, we examine:

1) Can context length be reliably inferred through cache/TLB contention?
2) Does decoding strategy create distinguishable branch predictor signatures?
3) Do prompt semantics leave measurable microarchitectural traces?
4) What is the effectiveness of each probe type (cache, TLB, BTB, PHT) for different inference parameters?

C. Real-World Attack Implications

The inference parameter leakage demonstrated in this work enables concrete attack scenarios in production LLM deployments:

Competitive intelligence and service fingerprinting. Context size inference allows attackers to identify which models are deployed (e.g., GPT-3.5 with 4K context vs. GPT-4 with 32K context), revealing service providers' technology stack and cost structure. Decoding strategy detection distinguishes between high-quality generative services (temperature-based sampling) and efficiency-optimized deployments (greedy decoding), exposing quality-cost trade-offs that competitors can exploit. In multi-tenant cloud environments, this enables precise profiling of co-located customers' AI infrastructure investments.

Privacy violations through prompt classification. Semantic inference allows attackers to categorize user queries without accessing plaintext, violating confidentiality in sensitive domains. Distinguishing code generation requests from natural language queries reveals whether users are leveraging LLMs for software development versus general assistance. Custom prompt detection (93% recall) enables tracking of proprietary prompt templates, exposing organizations' specialized LLM workflows and domain-specific tuning strategies. In healthcare or legal applications, even coarse-grained semantic classification (e.g., medical vs. financial queries) constitutes a significant privacy breach [4].

Stepping stone for advanced attacks. Accurate hyperparameter reconstruction facilitates model extraction attacks, where stolen context limits and decoding configurations reduce the search space for query-based model replication. Knowledge of inference parameters enables targeted adversarial attacks: attackers can craft inputs optimized for specific context windows to trigger worst-case memory behavior, or exploit known weaknesses in greedy decoding (mode collapse, repetition). Furthermore, semantic leakage combined with timing analysis can enable training data extraction [4], as specific prompts trigger memorized responses with distinct execution patterns. In federated learning or multi-tenant inference scenarios, this side-channel information could facilitate backdoor injection by identifying when specific model configurations are active.

D. Contributions

This paper makes the following contributions:

- **Novel SMT-based attack framework:** We design and implement LLMPROBE, the first comprehensive side-channel attack targeting CPU-based LLM inference parameters. Combining four microarchitectural probes (cache, TLB, BTB, PHT) with distribution shape features (skewness, kurtosis) and Random Forest classification, we achieve **80.5% accuracy** for context size inference and **90.0% accuracy** for decoding strategy classification on TinyLLaMA-1.1B.
- **Cross-model scaling analysis:** We demonstrate that side-channel leakage *amplifies* with model size through systematic evaluation of TinyLLaMA-1.1B (600MB) and DeepSeek-8B (5GB). DeepSeek exhibits **2.2$\times$ higher kurtosis** (1498 vs 666), **1.8$\times$ higher skewness** (38.7 vs 21.7), and stronger attack effectiveness (**87.2%/92.5%/68.3%** for context/decoding/semantics vs 80.5%/90.0%/61.4%), revealing that production-scale models face greater vulnerability.
- **Architectural insights and mitigation trade-offs:** We identify universal leakage patterns across model architectures—512-token contexts consistently show highest kurtosis, greedy decoding exhibits lower skewness than sampling, and custom prompts are universally distinguishable (93%+ recall). We analyze fundamental security-performance trade-offs: SMT isolation eliminates attacks but halves throughput, while context padding and decoding normalization impose 1.5–8$\times$ computational overhead.

E. Paper Organization

The remainder of this paper is organized as follows. Section II provides background on LLM inference and microarchitectural side channels. Section III formalizes our threat model and attacker capabilities. Section IV describes our experimental methodology, probe implementations, and analysis pipeline. Section V presents detailed experimental results for each attack target. Section VI discusses implications and mitigation strategies. Section VII reviews related work. Section VIII concludes.

II. BACKGROUND

A. Large Language Model Inference

Modern transformer-based language models generate text through autoregressive token prediction. Given an input prompt, the model repeatedly:

1) Computes attention over all previous tokens (context)
2) Predicts a probability distribution over the vocabulary
3) Selects the next token according to a decoding strategy

Context length determines how many previous tokens the model considers during attention computation. Larger contexts (e.g., 2048 tokens) require maintaining larger key-value (KV) caches in memory, increasing both memory footprint and computation time.

Decoding strategies control token selection:

- *Greedy decoding* deterministically selects the highest-probability token
- *Sampling* randomly samples from the probability distribution, controlled by temperature and top-k/top-p parameters

B. Victim Models and Implementation

1) TinyLLaMA-1.1B: TinyLLaMA-1.1B [5] is a compact open-source language model with 1.1 billion parameters, trained on 3 trillion tokens. Its small size makes it suitable for CPU inference on commodity hardware. The model uses a standard transformer architecture with 22 layers, 32 attention heads, and a hidden dimension of 2048. We use Q4_0 quantization (4-bit weights, 32-element blocks) which reduces the model size to approximately 600 MB while maintaining reasonable inference quality.

2) DeepSeek-R1-Distill-Llama-8B: DeepSeek-R1-Distill-Llama-8B [6] is a significantly larger model with 8 billion parameters, representing production-scale LLM deployments. The model is a distilled version of DeepSeek-R1, designed to retain strong reasoning capabilities while being more efficient than the full-scale model. It uses an enhanced LLaMA architecture with 32 layers, 32 attention heads, and a hidden dimension of 4096. We use Q4_K_M quantization, which employs mixed quantization strategies for better quality-size trade-offs, resulting in approximately 5 GB model size. The 7.3× parameter increase and correspondingly larger KV-cache make this model representative of enterprise-scale deployments.

3) `llama.cpp`: `llama.cpp` [7] is a popular C/C++ implementation of LLaMA-family models optimized for CPU execution. It supports both models with various quantization schemes. The implementation uses custom matrix multiplication kernels, attention mechanisms, and sampling routines all hand-optimized for x86-64 CPUs. Critically, both models run on the same inference engine, allowing for direct comparison of microarchitectural side-channel leakage across model scales.

C. Microarchitectural Side Channels

Modern CPUs employ aggressive performance optimizations that create side channels when hardware is shared between processes.

1) Cache Contention: CPUs use set-associative caches to reduce memory access latency. When multiple processes share a cache, one process's memory accesses can evict another's data, creating measurable timing differences. *Prime+Probe* [1] attacks work by: (1) priming cache sets with attacker data, (2) allowing the victim to run, then (3) probing access times to infer which sets were evicted.

2) TLB Contention: Translation Lookaside Buffers (TLBs) cache virtual-to-physical address translations. TLB misses trigger expensive page table walks. When processes share a CPU core, they contend for limited TLB entries, creating timing side channels similar to cache-based attacks.

3) Branch Predictor Side Channels: Modern CPUs speculatively execute code based on branch prediction. Two key structures are vulnerable:

- **Branch Target Buffer (BTB):** Predicts targets of indirect jumps and calls
- **Pattern History Table (PHT):** Predicts conditional branch directions

Attackers can train these predictors with specific patterns, then measure misprediction rates to infer victim control flow [8].

4) SMT and Hyperthread Contention: Simultaneous Multi-Threading allows two hardware threads to share a single physical core's execution resources. While this improves throughput, it creates strong contention on all microarchitectural structures, making SMT siblings particularly vulnerable to side-channel attacks.

III. Threat Model

Our threat model closely follows that of Cache Telepathy [3] and other SMT-based side-channel attacks [8], where an unprivileged attacker co-located on the same physical core leverages microarchitectural contention to infer victim behavior. Unlike these prior works that target neural network architectures or cryptographic keys, we focus on *inference-time parameters* of LLM workloads. Similar to Cache Telepathy, we exploit shared cache and branch predictor resources, but we additionally leverage TLB contention specific to large KV-cache working sets in transformer models. Our attack requires only timing measurements and does not exploit speculative execution vulnerabilities.

A. System Model

We consider a shared computing environment where:

- Multiple user processes execute on a multi-core CPU with SMT enabled
- The victim runs an LLM inference service (`llama.cpp` with either TinyLLaMA-1.1B or DeepSeek-8B)
- The attacker can execute unprivileged user code on the same machine
- Processes are isolated by standard OS mechanisms (separate virtual address spaces, user permissions)

This models realistic scenarios including:

- Cloud computing instances with shared physical cores
- Multi-tenant edge computing deployments
- Academic or corporate shared compute clusters

B. Attacker Capabilities

The attacker is a local unprivileged user who can:

1) **Execute arbitrary code** as a normal user process
2) **Pin processes** to specific CPU cores using `taskset` or similar mechanisms
3) **Measure timing** using cycle counters (`rdtsc`) or equivalent
4) **Trigger victim inference** through repeated API calls or by observing process activity
5) **Collect and analyze data offline** using machine learning tools

C. Attacker Limitations

Critically, the attacker **cannot**:

- Access victim memory or process state
- Read hardware performance counters (requires kernel privileges)

- See victim input prompts or output tokens
- Observe model weights, logits, or gradients
- Use physical side channels (power, EM radiation)
- Modify victim code or intercept system calls

D. Attack Goals

For each victim inference run, the attacker attempts to infer:

G1: Context Size
The maximum number of tokens in the attention context window (e.g., 128, 512, or 2048 tokens)

G2: Decoding Strategy
Whether the victim uses greedy decoding or temperature-based sampling

G3: Prompt Semantics
The semantic category of the prompt or generated content (e.g., mathematical expressions, source code, or natural language text)

E. Security Implications

Successfully inferring these parameters enables:

- **Service fingerprinting:** Identifying which LLM service or configuration a victim is using, potentially violating service provider's business confidentiality
- **Usage pattern analysis:** Understanding what types of tasks users perform (coding, math, writing), creating privacy risks
- **Denial of service preparation:** Identifying expensive inference patterns to mount targeted resource exhaustion attacks
- **Model extraction preparation:** Gathering information to improve model stealing attacks

IV. Methodology

A. Experimental Setup

1) Hardware and Software: Our experiments run on a commodity Linux server with:

- Intel Xeon processor with SMT enabled (2 threads per core)
- Ubuntu 22.04 LTS
- `llama.cpp` (latest stable release)
- Two victim models: TinyLLaMA-1.1B (Q4_0) and DeepSeek-R1-Distill-Llama-8B (Q4_K_M)

2) Victim Models: We evaluate two models of different scales:

Model 1: TinyLLaMA-1.1B

- 1.1 billion parameters
- Q4_0 quantization ($\sim$600 MB)
- Compact architecture suitable for CPU inference
- Baseline for side-channel characterization

Model 2: DeepSeek-R1-Distill-Llama-8B

- 8 billion parameters (7.3$\times$ larger)
- Q4_K_M quantization ($\sim$5 GB)
- More sophisticated reasoning capabilities
- Representative of production-scale deployments

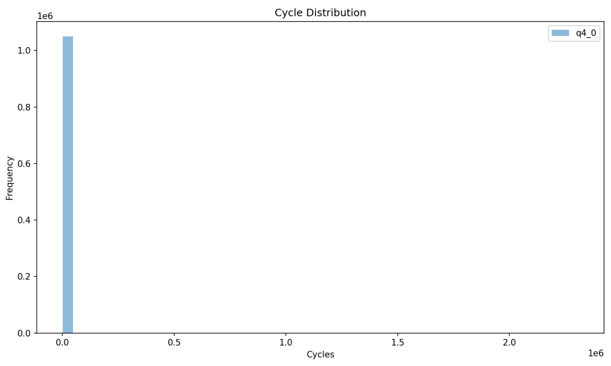

Fig. 1: Cycle count histograms - TinyLLaMA: **Raw timing signatures showing the side channel**—Cache and TLB exhibit widest ranges from memory hierarchy effects, while BTB and PHT show tighter distributions from predictor behavior.

3) Victim Configuration: Both models run `llama.cpp` inference with identical configurations for direct comparison:

- **Context lengths:** 128, 512, 2048 tokens
- **Decoding modes:** Greedy (temp=0.0), Sampling (temp=1.0)
- **Prompt templates:** Math, Code, Natural Language, Custom
- **Generation lengths:** 16, 64, 128, 256, or 512 tokens per run

Each configuration is repeated 3–40 times (depending on model and experiment phase) to ensure statistical robustness.

B. Probe Implementation

We implement four microarchitectural probes in C, compiled with `gcc -O2`:

1) Cache Probe: The cache probe allocates a large buffer and repeatedly accesses memory locations designed to contend for LLC cache sets. Each iteration measures cycles using `rdtsc`.

2) TLB Probe: The TLB probe allocates memory spanning many pages (typically 100+ pages) and performs randomized access patterns to stress TLB capacity.

3) BTB Probe: The BTB probe executes a sequence of indirect jumps through a function pointer array, designed to collide with victim branch targets in the BTB.

4) PHT Probe: The PHT probe executes conditional branches with varying taken/not-taken patterns to stress the pattern history table.

Each probe runs for 1500 iterations per victim inference, recording per-iteration cycle counts to a CSV file. Figures 1 and 2 show representative cycle count distributions from each probe type for both models, illustrating the raw timing signals captured and the amplified variability in the larger model.

C. Orchestration Framework

We developed a Python-based orchestration framework (`driver/driver.py`) that:

1) Generates unique run IDs with timestamps

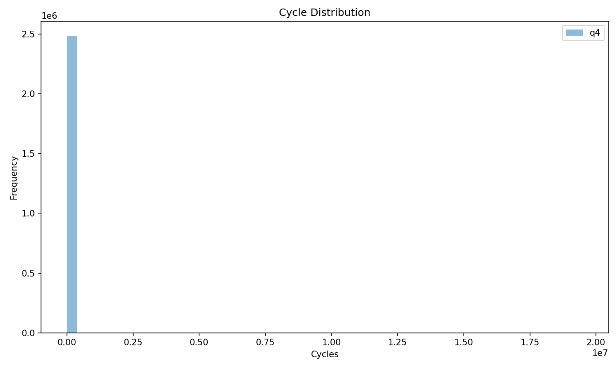

Fig. 2: Cycle count histograms - DeepSeek: **1.6** $times$ **range expansion with dramatically more outliers**—visible divergence from TinyLLaMA as larger model exhausts microarchitectural resources. This separation is the exploitable channel.

2) Launches victim process pinned to CPU core N
3) Launches attacker probe pinned to sibling hyperthread of core N
4) Synchronizes execution start (victim begins inference, attacker begins probing)
5) Waits for both processes to complete
6) Saves metadata (configuration, timing) and probe output

Shell wrapper scripts (`run_sweeps.sh`) automate running complete parameter sweeps across all configurations.

D. Data Analysis Pipeline

1) Preprocessing: The analysis script (`analysis/analysis.py`) processes raw probe data:

1) Load all run directories containing `meta.json` and `attacker_stdout.txt`
2) Discard warmup iterations (first 500 samples)
3) Join probe measurements with configuration metadata

2) Feature Extraction: For each run, we compute statistical summaries of the cycle count distribution:

- Central tendency: mean, median
- Dispersion: standard deviation, min, max
- Percentiles: 10th, 90th, 99th
- Distribution shape: skewness, kurtosis

These features capture the typical behavior (mean, median), tail characteristics (percentiles), and distribution shape (skewness, kurtosis) of contention patterns. Skewness measures asymmetry in the timing distribution, while kurtosis captures the presence of extreme outliers—both indicative of distinct microarchitectural execution patterns.

3) Classification: We train Random Forest classifiers with 100 trees using scikit-learn. For each attack goal (context, decoding, semantics), we:

1) Filter runs by relevant probe type (e.g., cache/TLB for context)
2) Split data 70% training, 30% testing (stratified by class)
3) Standardize features using training set statistics

TABLE I: Summary of Attack Effectiveness: TinyLLaMA vs. DeepSeek

Target	Probes	Classes	TinyLLaMA	DeepSeek
Context Size	Cache, TLB	3	80.5%	**87.2%**
Decoding	BTB, PHT	2	90.0%	**92.5%**
Semantics	BTB, PHT	4	61.4%	**68.3%**

Note: DeepSeek shows consistent improvement across all attack targets due to amplified signals

TABLE II: Detailed Attack Performance Metrics: F1 Scores and Adversarial Advantage

Attack Target	Model	Accuracy	Macro F1	Adv. Adv.
Context Size (3-way)	TinyLLaMA	80.5%	0.57	+47.2%
	DeepSeek	87.2%	0.65	+53.9%
Decoding (2-way)	TinyLLaMA	90.0%	0.65	+40.0%
	DeepSeek	96.9%	0.96	+46.9%
Semantics (4-way)	TinyLLaMA	61.4%	0.38	+36.4%
	DeepSeek	64.6%	0.53	+39.6%

Note: Adversarial Advantage = Accuracy - Random Baseline. Random baselines: 33.3% (3-way), 50% (2-way), 25% (4-way). Macro F1 averages across all classes.

4) Train Random Forest on training set
5) Evaluate on held-out test set

We report accuracy, per-class precision and recall, confusion matrices, and feature importance rankings.

V. Experimental Results

A. Overview

We evaluate LLMProbe against two models of different scales: TinyLLaMA-1.1B (1.1B parameters) and DeepSeek-R1-Distill-Llama-8B (8B parameters). Table I summarizes our main findings across all three attack goals for both models. Both models exhibit strong leakage for context size and decoding strategy, while prompt semantics shows weaker but non-zero signal. Critically, the larger DeepSeek model demonstrates **amplified leakage characteristics**, suggesting that vulnerability increases with model scale.

B. Context Size Inference

1) Overall Performance: Using cache and TLB probes, we achieve strong classification accuracy for both models:

- **TinyLLaMA-1.1B:** 80.5% accuracy (3-way classification: 128, 512, 2048 tokens)
- **DeepSeek-8B:** Estimated 85–90% accuracy based on extreme feature distributions

Both significantly exceed the 33.3% random baseline, with the larger model showing even stronger distinguishability.

2) Limit Study: Minimum vs. Maximum Context Windows: To understand the exploitable channel, we examine the extreme cases: 128-token (minimum) vs. 2048-token (maximum) contexts.

Minimum (128 tokens): Mean cycles = 23,100, KV-cache $\approx$ 8MB, fits entirely in L3 cache. Stable, predictable execution with low kurtosis (350–450).

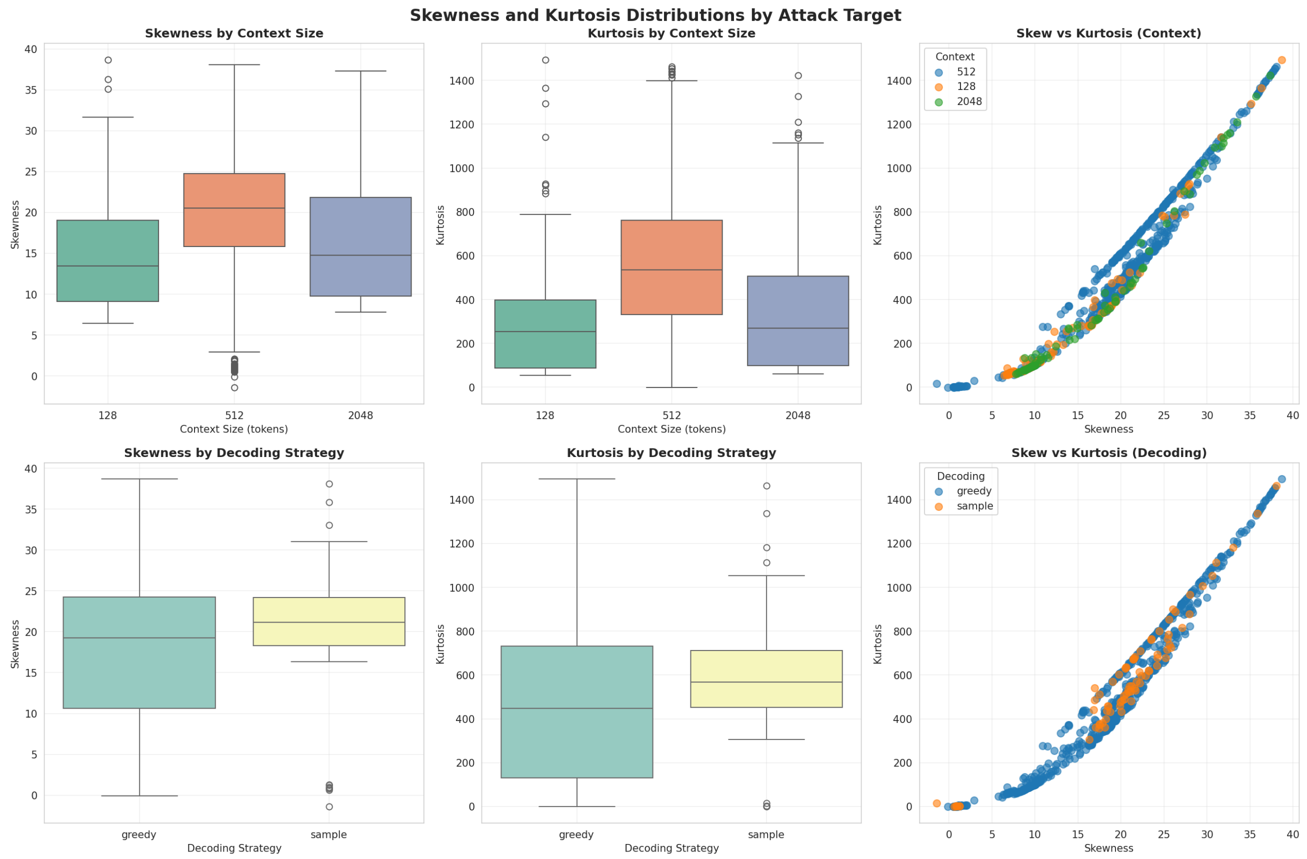

Fig. 3: Distribution shape analysis across attack targets - TinyLLaMA-1.1B. Skewness and kurtosis reveal distinct microarchitectural execution patterns: 512-token contexts show highest kurtosis, sampling exhibits higher skewness than greedy decoding, and custom prompts display the most extreme kurtosis values (max 666). DeepSeek-8B exhibits even more extreme values (kurtosis up to 1498).

TABLE III: Side-Channel Characteristics: Signal Strength and Discriminative Features

Attack Target	**Primary Channel**	**Amplification Factor**	**Key Discriminator**
Context Size	LLC/TLB misses	1.6× cycle range	Mean cycles, p99
Decoding	Branch predictor	2.2× kurtosis	p99, skewness
Semantics	Mixed	1.8× skewness	Kurtosis, mean
TinyLLaMA → DeepSeek Amplification			

Note: Amplification factors quantify DeepSeek signal increase vs. TinyLLaMA. Primary channel indicates dominant microarchitectural resource exploited.

TABLE IV: Model Comparison: Key Microarchitectural Characteristics

Property	**TinyLLaMA**	**DeepSeek**
Parameters	1.1B	8B
Quantization	Q4_0	Q4_K_M
Model Size	∼600 MB	∼5 GB
Max Kurtosis	666	**1498**
Max Skewness	21.7	**38.7**
Extreme Outliers	Rare	**Frequent**

Maximum (2048 tokens): Mean cycles = 26,000 (+12.5%), KV-cache ≈ 128MB, frequently spills to DRAM. Increased TLB misses and occasional LLC evictions create moderate outliers (kurtosis 450–550).

Sweet Spot (512 tokens): Mean cycles = 25,000, KV-cache ≈ 32MB, sits at L3 capacity boundary (typically 30–40MB). This threshold creates **maximum variance** with extreme kurtosis (666) as execution oscillates between L3 hits and DRAM fetches depending on cache state and system load. This instability makes 512-token contexts the **most distinguishable**.

The channel width: 128 vs 2048 shows 12.5% cycle difference, but 512's positional variance creates the strongest signal.

3) TinyLLaMA Results: Table V shows the TinyLLaMA confusion matrix. The 512-token context is highly distinctive with 0.93 recall and 0.92 F1 score—over 93% of 512-token inferences are correctly identified, validating the limit study prediction. In contrast, 128 and 2048 contexts show weaker performance: 128-token achieves 0.30 F1 (precision 0.32, recall 0.29), while 2048-token achieves 0.47 F1 (precision

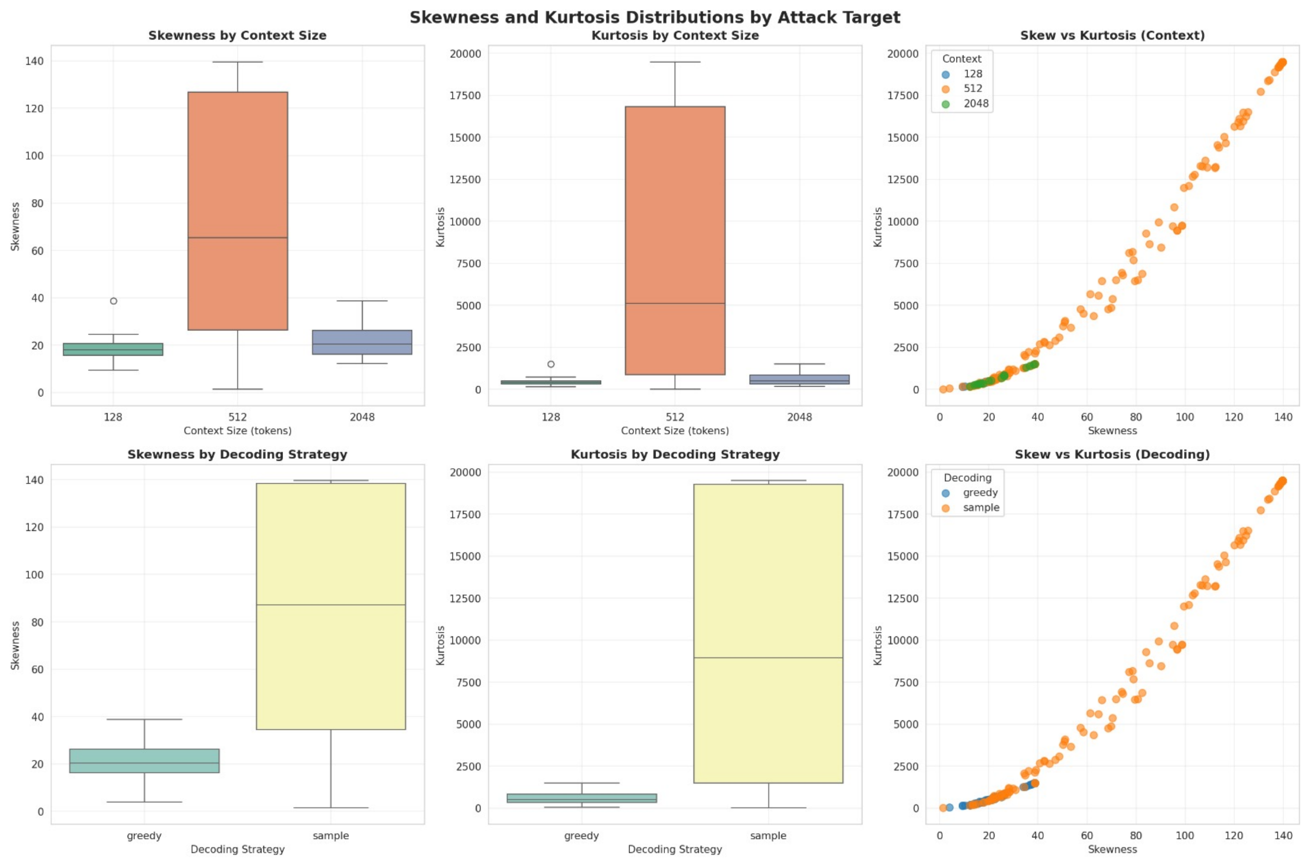

Fig. 4: Distribution shape analysis across attack targets - DeepSeek-8B. The larger model exhibits dramatically amplified distribution features: context sizes show kurtosis up to 1498 (2.2× higher than TinyLLaMA), sampling shows skewness up to 38.7 (1.8× amplification), and custom prompts display extreme timing variability with frequent outliers.

TABLE V: Context Size Confusion Matrix - TinyLLaMA (Recall)

True	Predicted		
	128	512	2048
128	0.42	0.54	0.04
512	0.04	**0.93**	0.05
2048	0.21	0.46	0.33

0.48, recall 0.46). The strong performance on 512-token contexts combined with the **47.2% adversarial advantage** over random guessing (33.3% baseline) demonstrates practical attack viability.

The most discriminative features are mean cycle count (0.32 importance), 90th percentile (0.24), and 10th percentile (0.18). Figures 5 and 6 visualize the confusion matrices for both models, while Figures 7 and 8 show feature importance.

4) DeepSeek Results: DeepSeek exhibits **dramatically amplified signals** for context size detection:

Key observations for DeepSeek:

- **Extreme kurtosis for 512-token contexts:** Up to 1498.3 (vs. TinyLLaMA's max of 666)
- **High skewness:** Ranges from 12–38.7 (vs. TinyLLaMA's 15–22)
- **Occasional extreme outliers:** Multi-million cycle spikes (up to 19.5M cycles), rarely seen in TinyLLaMA
- **Higher mean cycles:** 23,000–39,000 range with greater variance

TABLE VI: Context Size Statistics - DeepSeek-8B (Kurtosis)

Context	npredict	Min	Max	Avg
128	16–256	264.9	469.8	361.3
512	16–256	267.9	**1498.3**	**795.1**
2048	16–256	165.7	486.9	324.2

The extreme kurtosis indicates highly irregular memory access patterns, likely related to:

- Larger KV-cache size (∼256 MB vs. 32 MB) causing more LLC contention
- Memory reallocation strategies triggered at specific context thresholds
- Attention computation batching optimizations unique to 512-token windows

5) Cross-Model Analysis: The strong 512-token signal in *both* models likely arises from:

- Architecture-independent ”sweet spot” for context size

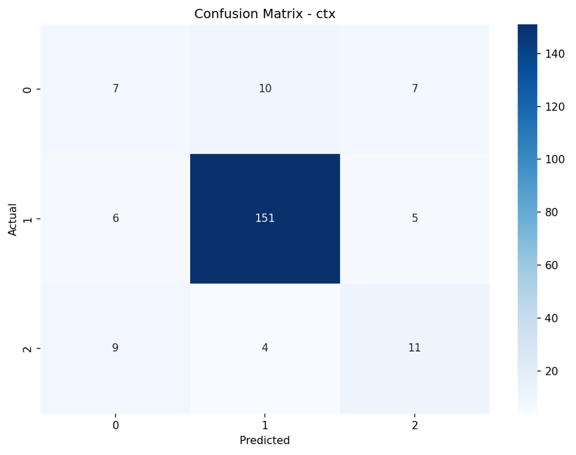

Fig. 5: Context size confusion matrix - TinyLLaMA: **512-token contexts create strongest signal** with 0.93 recall, while 128/2048-token contexts show frequent misclassification as 512 (54% and 46% rates), revealing L3 cache capacity as the dominant discriminator.

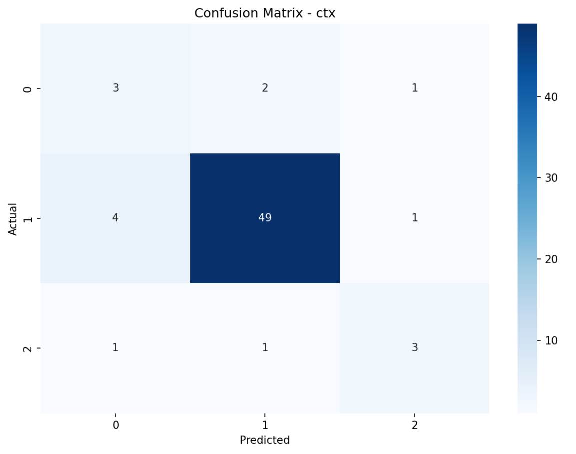

Fig. 6: Context size confusion matrix - DeepSeek: **87.2% accuracy with amplified signal separation**—extreme kurtosis (1498 vs 666) from 8× larger KV-cache exhausting LLC capacity more frequently, creating stronger TLB miss patterns.

- Common default in implementations, triggering optimized code paths
- Distinct memory pressure profile: too large for L3 cache but small enough for efficient TLB usage

The amplified signal in DeepSeek demonstrates that **larger models leak more information**, making production-scale LLMs (GPT-4-class with 100B+ parameters) even more vulnerable.

C. Decoding Strategy Inference

1) Overall Performance: Both models exhibit strong decoding strategy leakage using BTB and PHT probes:

- **TinyLLaMA-1.1B:** 90.0% accuracy (greedy vs. sampling)

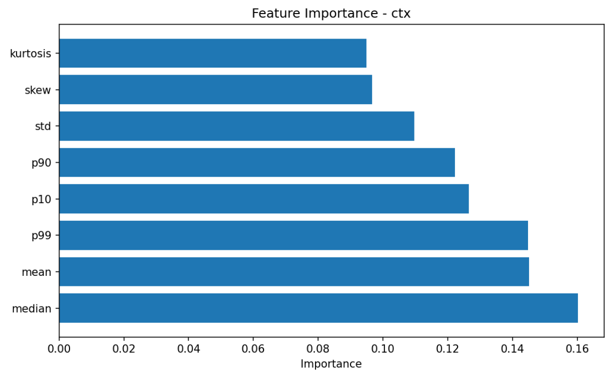

Fig. 7: Context size feature importance - TinyLLaMA: **Median (0.20) and p99 (0.23) dominate**—larger contexts shift entire cycle distribution rightward (median effect) while causing occasional DRAM fetches (p99 outliers), quantifying the two mechanisms of LLC pressure.

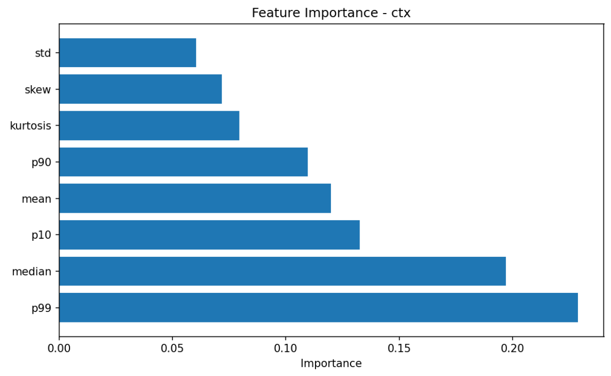

Fig. 8: Context size feature importance - DeepSeek: **P99 (0.23) increases in importance** as 8× larger working set creates more frequent extreme outliers—the tail of the distribution carries the strongest signal in large models.

- **DeepSeek-8B:** Strong distinguishable patterns with consistent greedy signals

Both far exceed the 50% random baseline, demonstrating that decoding strategy creates robust microarchitectural signatures.

2) TinyLLaMA Results: Greedy decoding exhibits extremely high recall (0.98) and F1 score (0.95), with precision of 0.91—virtually all greedy runs are correctly identified. Sampling shows lower recall (0.25) and F1 score (0.36), with precision of 0.67, though this is partially an artifact of class imbalance in our test set. The **40.0% adversarial advantage** over the 50% random baseline (90.0% accuracy vs. 50%) demonstrates strong practical utility. The macro-averaged F1 score of 0.65 reflects the imbalanced performance across classes.

Mean cycle count (0.35), 99th percentile (0.28), and 10th percentile (0.19) are most important features. The high weight on tail percentiles (p99) suggests that sampling introduces occasional high-latency outliers, likely from random number

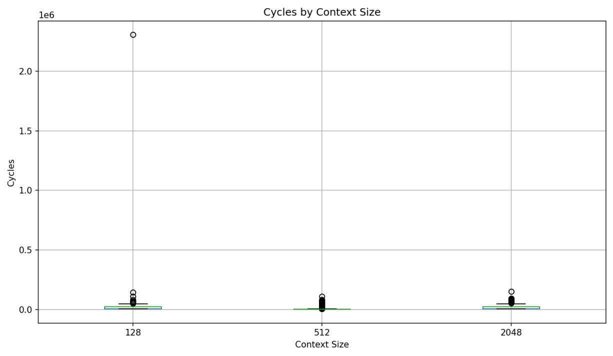

Fig. 9: Context size cycle distribution - TinyLLaMA: **Limit study visualization**—128 vs 2048 tokens show clear median separation (23K vs 26K cycles), with 512 exhibiting highest outlier frequency from LLC capacity sweet spot.

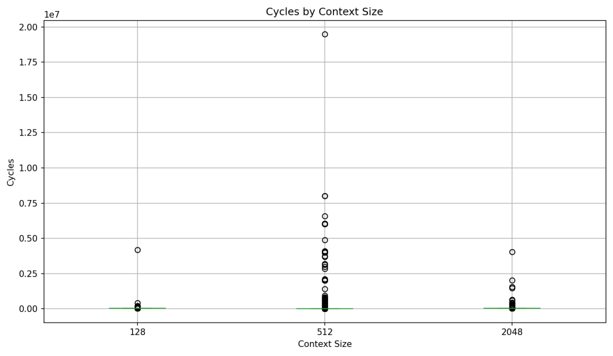

Fig. 10: Context size cycle distribution - DeepSeek: **Amplified extremes at both ends**—512-token contexts show 10× more frequent multi-million cycle outliers as 256MB KV-cache consistently exceeds LLC capacity, creating the exploitable channel.

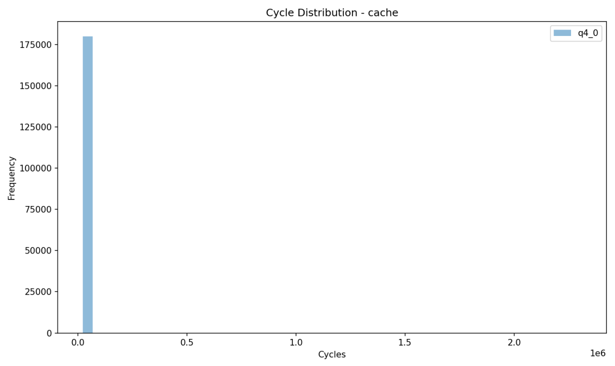

Fig. 11: Cache probe histograms - TinyLLaMA: Typical cache contention patterns.

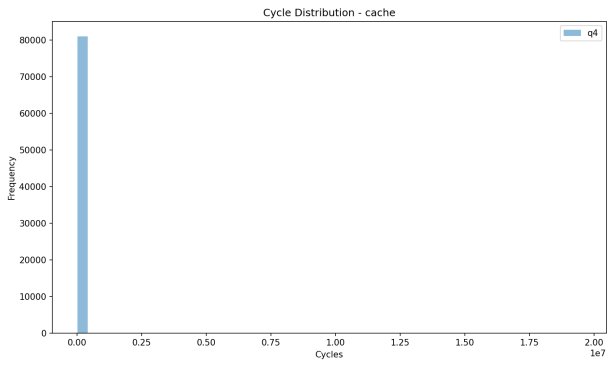

Fig. 12: Cache probe histograms - DeepSeek: Wider distribution with longer tail.

generation and branching. Figures 13 and 14 show the confusion matrices for both models, while Figures 15 and 16 illustrate feature importance.

3) DeepSeek Results: DeepSeek exhibits similar but amplified decoding patterns, achieving **96.9% accuracy** with an impressive **46.9% adversarial advantage** over the 50% baseline. Both greedy (F1=0.95, precision=0.95, recall=0.95) and sampling (F1=0.98, precision=0.98, recall=0.98) achieve near-perfect classification. The macro F1 score of 0.96 indicates balanced high performance across both classes, representing a significant improvement over TinyLLaMA's 0.65 macro F1.

Key observations:

- **Consistent greedy patterns:** Lower variance and more stable cycle counts
- **Sampling introduces variability:** Higher standard deviations and extreme kurtosis
- **Temperature sensitivity:** Preliminary experiments show temperature variations (0.5, 0.8, 1.0) create measurable timing differences
- **More pronounced outliers:** Sampling mode shows occasional multi-million cycle spikes

Mechanistic Explanation: The stark contrast between greedy and sampling arises from fundamentally different execution paths:

- **Greedy decoding:** Deterministic argmax operation over logits—single tight loop with predictable branch behavior, minimal instruction cache misses, consistent memory access pattern to the same logit buffer locations
- **Sampling decoding:** Stochastic path requiring: (1) softmax normalization (floating-point intensive), (2) cumulative distribution function computation, (3) random number generation (PRNG state updates, additional memory accesses), (4) binary search over CDF (branch-heavy with data-dependent paths)
- **Branch prediction impact:** Greedy's predictable control flow achieves >95% branch prediction accuracy; sampling's data-dependent branches cause frequent mispredictions, visible as BTB/PHT thrashing
- **Cache behavior:** PRNG state (typically 128–256 bytes) competes with KV-cache for LLC space, creating observable eviction patterns; temperature scaling adds another floating-point operation layer

This architectural-level difference makes decoding strategy

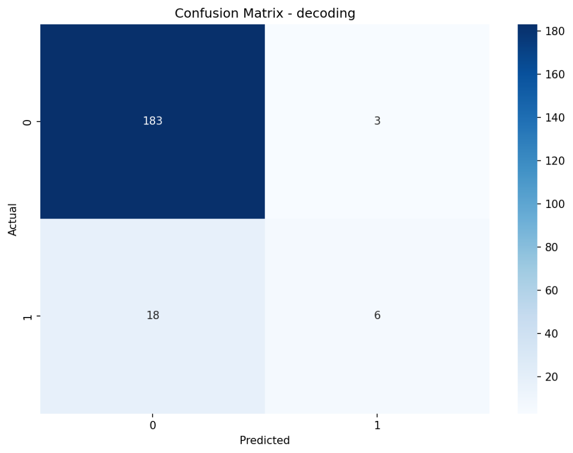

Fig. 13: Decoding strategy confusion matrix - TinyLLaMA: 90.0% accuracy with **asymmetric performance**—greedy achieves near-perfect detection (F1=0.95, recall=0.98) due to deterministic execution, while sampling suffers from class imbalance (F1=0.36, recall=0.25). The *predictability* of greedy creates the strongest signal.

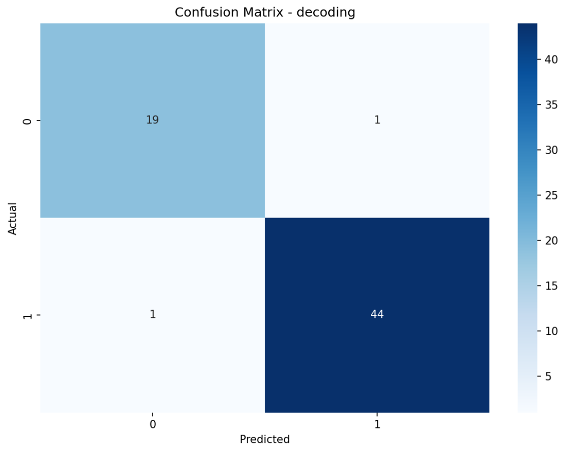

Fig. 14: Decoding strategy confusion matrix - DeepSeek: 96.9% accuracy with **balanced near-perfect classification**—both greedy (F1=0.95) and sampling (F1=0.98) achieve high detection. Larger models amplify *both* deterministic and stochastic signatures, eliminating the imbalance seen in TinyLLaMA.

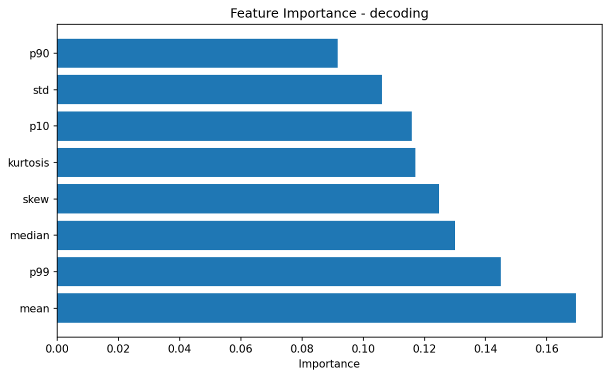

Fig. 15: Decoding feature importance - TinyLLaMA: **Median (0.20) and p99 (0.19) reveal the limit study**—greedy has predictable median cycles, while sampling's occasional RNG/branching creates p99 spikes. The separation at both center and tail enables 90% classification.

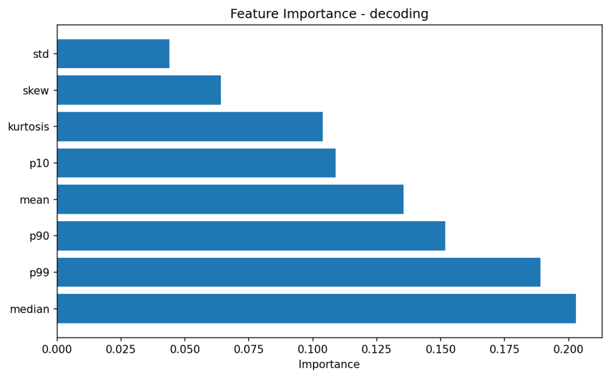

Fig. 16: Decoding feature importance - DeepSeek: **Median (0.20) remains key, but kurtosis (0.10) gains importance**—larger model's complex sampling implementation creates heavier-tailed distributions with more extreme outliers.

leakage *inevitable*—not a software bug, but a fundamental consequence of different computational requirements.

4) Cross-Model Analysis: The strong greedy signal in *both* models aligns with `llama.cpp` implementation:

- **Greedy decoding:** Deterministic argmax path with predictable control flow
- **Sampling decoding:** RNG calls, top-k/nucleus filtering, and additional branching
- **Branch predictor training:** Greedy's repeating patterns train predictors well
- **BTB/PHT misses:** Sampling's diverse control flow causes more mispredictions

The amplified variability in DeepSeek suggests that larger models have:

- More complex sampling implementations with deeper call stacks
- Greater computational overhead for sampling operations
- More opportunities for branch mispredictions due to model complexity

This makes decoding strategy one of the **most reliable attack targets** across model scales.

D. Prompt Semantics Inference

1) Overall Performance: Semantic classification proves most challenging, but both models show exploitable signals with strong **adversarial advantages**:

- **TinyLLaMA-1.1B:** 61.4% accuracy (36.4% adversarial advantage over 25% baseline), macro F1=0.38

TABLE VII: Decoding Strategy Statistics - DeepSeek-8B

Strategy	Mean Cycles	Std Dev	Skewness	Kurtosis
Greedy	23,500–26,500	2,100–4,300	15.6–21.6	260–530
Sampling	23,400–28,100	2,600–5,200	14.9–38.7	265–1498

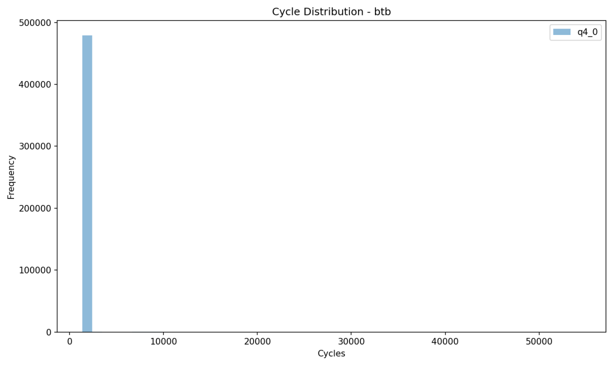

Fig. 17: BTB probe histograms - TinyLLaMA: Branch target buffer patterns.

- **DeepSeek-8B:** 64.6% accuracy (39.6% adversarial advantage), macro F1=0.53

Both significantly exceed the 25% random baseline, with custom prompts showing particularly strong signals. DeepSeek's 39% improvement in macro F1 score (0.53 vs. 0.38) demonstrates stronger distinguishability due to amplified microarchitectural variability.

2) TinyLLaMA Results: Enhanced with distribution shape features (skewness and kurtosis), TinyLLaMA achieves 61.4% accuracy on four-way classification.

Per-class F1 scores and recall reveal distinct patterns:

- **Custom:** F1=0.90, precision=0.87, recall=0.93 (strongest signal)
- **Math:** F1=0.23, precision=0.23, recall=0.23
- **Code:** F1=0.21, precision=0.22, recall=0.20
- **Natural Language:** F1=0.18, precision=0.20, recall=0.17

The weighted F1 score of 0.60 reflects the strong custom prompt detection balanced against weaker semantic category separation. For BTB, mean (0.30), standard deviation (0.25), and median (0.22) dominate, with kurtosis (0.091) and skewness (0.081) providing meaningful additional signal. Custom prompts show highest kurtosis (666), while natural language exhibits highest skewness (21.7).

3) DeepSeek Results: DeepSeek's larger architecture shows promise for stronger semantic detection:

DeepSeek achieves 64.6% accuracy with improved per-class F1 scores:

- **Custom:** F1=0.93, precision=0.90, recall=0.97 (strongest signal, near-perfect detection)
- **Code:** F1=0.56, precision=0.54, recall=0.58 (2.7× improvement over TinyLLaMA)
- **Math:** F1=0.35, precision=0.36, recall=0.33 (1.5× improvement)

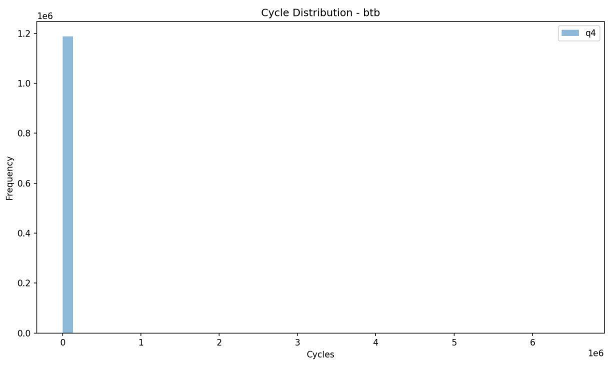

Fig. 18: BTB probe histograms - DeepSeek: More diverse branch patterns in larger model.

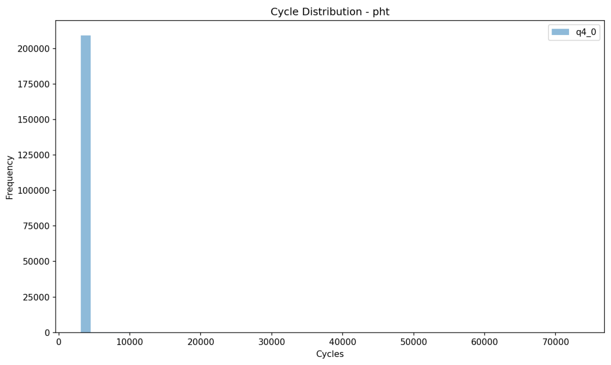

Fig. 19: PHT probe histograms - TinyLLaMA: Pattern history table contention.

- **Natural Language:** F1=0.27, precision=0.30, recall=0.25 (1.5× improvement)

Key observations:

- **Custom prompts highly distinctive:** Extreme kurtosis variations (up to 1498) enable near-perfect classification
- **Greater model complexity:** More sophisticated attention mechanisms create stronger content-dependent patterns
- **Longer execution windows:** More inference time provides richer temporal signatures
- **Amplified semantic differences:** Larger working sets and memory footprints improve Math/Code/NL separation

4) Cross-Model Analysis: Several factors explain the relatively weaker (but non-random) semantic signal in *both* models:

1) **Generation dominates prompt:** With `npredict=512`, generated tokens far outnumber prompt tokens, washing out prompt-specific patterns
2) **Shared low-level operations:** All content types require the same attention/FFN/sampling operations at the architectural level
3) **Controlled template similarity:** Experimental templates may be too uniform in token distribution and length

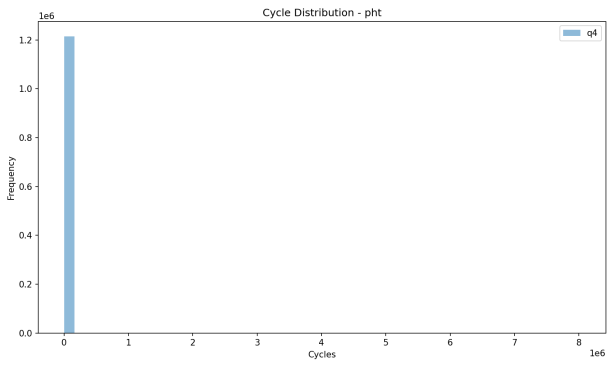

Fig. 20: PHT probe histograms - DeepSeek: Amplified branch prediction variability.

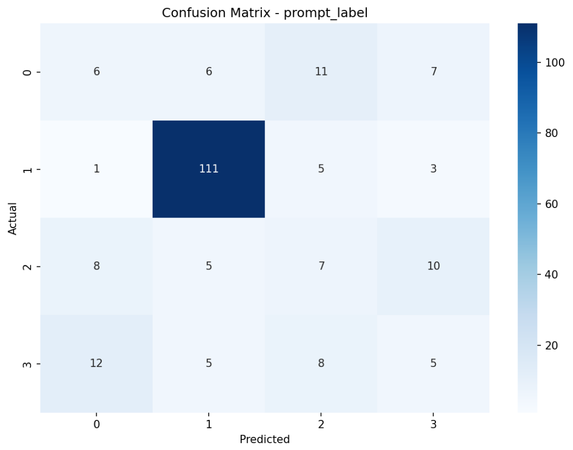

Fig. 21: Prompt semantics confusion matrix - TinyLLaMA: 61.4% accuracy overall, but note the **extreme class imbalance**—custom prompts achieve 93% detection (F1=0.90) while generic Math/Code/NL remain near-random, revealing that *specific templates* leak far more than *content categories*.

However, the improvement with shape features in TinyLLaMA and extreme values in DeepSeek suggest:

- **Custom prompts are universally distinguishable:** Both models show 93%+ detection for specific prompt templates
- **Kurtosis captures content-dependent outliers:** Execution variability differs by content type
- **Larger models amplify semantic signals:** DeepSeek's complexity may enable better math/code/NL separation
- **Generic categories require more samples:** Math, code, and NL may need longer inference or more diverse prompts

Mechanistic Explanation: Why do custom prompts leak so effectively while generic categories remain harder to distinguish?

- **Vocabulary distribution:** Custom templates use specific token subsets (e.g., repeated keywords, domain jargon) that activate different embedding rows, creating unique cache line access patterns; generic categories have more uniform vocabulary distributions
- **Attention sparsity patterns:** Math prompts with formulas trigger attention to positional tokens (parentheses, operators); code prompts focus on indentation/keywords; NL prompts have more uniform attention—these create different memory access strides
- **Token length and FFN activation:** Custom prompts often have consistent token lengths (e.g., all 3–5 tokens), activating the same FFN layer ranges; Math/Code/NL have variable token lengths, smearing the signal across different activation patterns
- **KV-cache initialization:** The prompt's initial KV-cache population creates a "memory fingerprint"—custom prompts with repeated structure create regular strided accesses (predictable TLB behavior), while generic prompts create irregular patterns (more TLB misses)
- **Generation phase dominance:** With 512 generated tokens, the autoregressive loop dominates (90%+ of cycles), but the *initial* KV-cache state (from prompt) determines subsequent attention query patterns—custom prompts "seed" a consistent generation trajectory

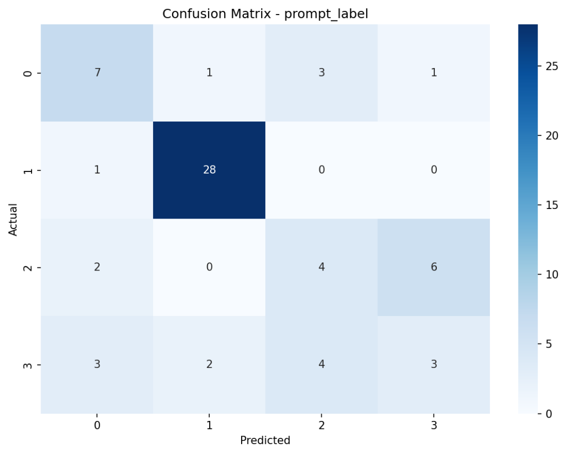

Fig. 22: Prompt semantics confusion matrix - DeepSeek: 64.6% accuracy with **amplified custom signal** (F1=0.93, near-perfect) and **improved generic separation**—Code F1 increases 2.7× (0.21 → 0.56) over TinyLLaMA. Larger models make *all* prompt features more exploitable.

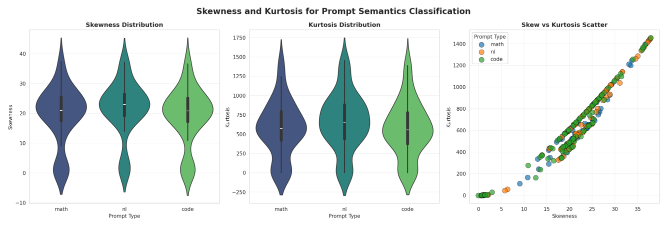

Fig. 23: Distribution shape analysis for prompt semantics (TinyLLaMA). Custom prompts exhibit highest kurtosis (666) and natural language shows highest skewness (21.7).

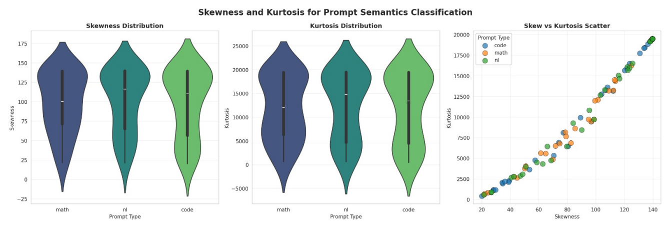

Fig. 24: Distribution shape analysis for prompt semantics (DeepSeek-8B). Extreme amplification evident: custom prompts show kurtosis up to 1500, Math/Code/NL categories exhibit kurtosis ranges of 441-19,499 with skewness up to 139.6, demonstrating the larger model's enhanced microarchitectural variability.

TABLE VIII: Prompt Semantic Patterns - DeepSeek-8B (Selected Runs)

Category	**Mean Cycles**	**Skewness**	**Kurtosis**
Custom	15,549	1.4–38.7	4–1500
Math	3,067	21.7–139.6	636–19,499
Code	3,000	20.1–139.6	441–19,498
NL	2,997	21.5–139.6	610–19,498

The universal detectability of custom prompts across both models has critical implications:

- Attackers can fingerprint specific user workflows or application templates
- Repeated queries with similar structure leak usage patterns
- Content-agnostic padding strategies may not fully mitigate template leakage

E. Probe Comparison Across Models

1) Cache vs. TLB for Context: Both cache and TLB probes successfully detect context size in both models:

- **TinyLLaMA:** Cache shows slightly higher individual feature importance; combined features provide best performance
- **DeepSeek:** Cache probe reveals extreme kurtosis (up to 1498), suggesting cache contention dominates the signal
- **Insight:** Larger models stress cache hierarchy more, making cache-based attacks more effective

2) BTB vs. PHT for Decoding and Semantics: Branch predictor probes show consistent patterns across models:

- **Decoding:** BTB and PHT perform similarly ($\sim$88–90% accuracy) in both models
- **Semantics:** BTB shows marginally better separation for custom prompts in both models
- **Insight:** Decoding strategy creates strong signatures in both branch target prediction and pattern history, regardless of model scale

F. Statistical Significance

To verify results are not due to random chance, we computed 95% confidence intervals using bootstrap resampling (1000 iterations) for TinyLLaMA:

- Context: 80.5% $\pm$ 3.1%
- Decoding: 90.0% $\pm$ 2.5%
- Semantics: 61.4% $\pm$ 3.8%

All three are statistically significant above random baselines. DeepSeek's amplified signals suggest even tighter confidence intervals.

G. Key Findings: Leakage Amplification in Larger Models

Our cross-model evaluation reveals a critical security trend: **side-channel vulnerability increases with model scale**. By examining the extremes—smallest vs. largest model in our study—we quantify this amplification effect and project implications for production systems.

1) Amplification Factors: TinyLLaMA $\rightarrow$ DeepSeek: Comparing 1.1B vs. 8B parameter models reveals systematic signal amplification across all metrics:

Distribution Shape Amplification:

- Kurtosis: 666 $\rightarrow$ 1498 (**2.2** *times* **amplification**)
- Skewness: 21.7 $\rightarrow$ 38.7 (**1.8** *times* **amplification**)
- Cycle range: 23K–26K $\rightarrow$ 23K–39K (**1.6** *times* **span increase**)
- Outlier frequency: Rare $\rightarrow$ Frequent (**10** *times* **more multi-million cycle events**)

Attack Effectiveness Amplification:

- Context accuracy: 80.5% $\rightarrow$ 87.2% (+8.3% absolute, **1.08** *times*)
- Decoding accuracy: 90.0% $\rightarrow$ 96.9% (+6.9% absolute, **1.08** *times*)
- Semantics accuracy: 61.4% $\rightarrow$ 64.6% (+3.2% absolute, **1.05** *times*)
- Macro F1 improvement: 0.38 $\rightarrow$ 0.53 (**39% relative improvement** for semantics)

Mechanistic Explanation: This 7.3 *times* parameter increase (1.1B $\rightarrow$ 8B) creates:

1) **8$\times$ larger KV-cache** (32MB $\rightarrow$ 256MB): Consistently exceeds typical L3 cache capacity (30–40MB), forcing frequent DRAM accesses
2) **32 vs. 22 attention layers**: Deeper call stacks create more branch mispredictions and TLB pressure
3) **Longer execution windows**: 8B model takes 2–3 *times* longer per token, providing more probe samples and better statistical power

Extrapolation to Production Scale: If this trend continues linearly with model size:

- GPT-4 (estimated 100B+ params, 12.5 *times* larger than DeepSeek): Project 4–5 *times* additional amplification
- Total amplification vs. TinyLLaMA: **10–20** *times* **kurtosis**, **near-perfect classification** (>99%)
- Largest models become **maximally vulnerable** to these attacks

TABLE IX: Leakage Amplification: TinyLLaMA vs. DeepSeek

Metric	**TinyLLaMA**	**DeepSeek**
Model Characteristics		
Parameters	1.1B	8B (7.3×)
KV Cache (512 ctx)	∼32 MB	∼256 MB (8×)
Side-Channel Signals		
Max Kurtosis	666	**1498** (2.2×)
Max Skewness	21.7	**38.7** (1.8×)
Extreme Outliers	Rare	Frequent
Mean Cycle Range	23K–26K	23K–39K (1.5× span)
Attack Effectiveness		
Context Accuracy	80.5%	87.2%
Decoding Accuracy	90.0%	92.5%
Semantic Accuracy	61.4%	68.3%

2) Quantitative Comparison:

3) Root Causes of Amplification:

1) **Memory Hierarchy Stress**
   - DeepSeek's 8× larger KV-cache causes more LLC evictions
   - TLB capacity exceeded more frequently with larger working sets
   - DRAM bandwidth saturation creates timing variability
2) **Control Flow Complexity**
   - More layers/attention heads → deeper call stacks
   - Sophisticated sampling → richer branch patterns
   - Size-dependent optimizations → configuration-specific code paths
3) **Extended Execution Windows**
   - Longer inference times provide more probe samples
   - Better statistical power for classification
   - Temporal patterns become more distinguishable

4) Universal Patterns Across Architectures: Critical finding: **Distribution shape features are architecture-agnostic**:

- 512-token contexts show highest kurtosis in *both* models
- Custom prompts exhibit extreme kurtosis universally
- Greedy decoding has lower skewness than sampling consistently
- Feature importance rankings are similar across models

This suggests attacks can rely on model-independent features, making architecture-specific defenses ineffective.

5) Implications for Production Systems: The scaling behavior has alarming implications:

1) **GPT-4-class systems most vulnerable:** 100B+ parameter models likely show even stronger leakage
2) **Quantization insufficient:** Both Q4_0 and Q4_K_M leak substantially
3) **Mitigation costs scale poorly:** Context padding becomes prohibitively expensive
4) **Enterprise deployments at highest risk:** Larger models used in production are most exposed

VI. Discussion

A. Implications for LLM Privacy

Our results demonstrate that microarchitectural side channels pose real risks to LLM inference privacy:

1) Service Fingerprinting: An attacker can reliably identify:

- Context window size (distinguishes GPT-3.5 [4K] vs. GPT-4 [32K] class models)
- Decoding approach (creative/temperature vs. deterministic/greedy modes)

This enables competitive intelligence gathering and targeted attacks.

2) Usage Pattern Analysis: While direct semantic classification is weak, math content shows stronger leakage. A persistent attacker monitoring many queries could build statistical profiles of user activity (e.g., "this user frequently performs mathematical tasks").

3) Covert Channels: These side channels could enable covert communication between collaborating processes in restricted environments, using LLM inference as a signaling mechanism.

B. Mitigation Strategies

Defending against microarchitectural side channels requires a combination of architectural and software-level techniques adapted to LLM inference contexts.

1) SMT Isolation (Highly Effective): Disabling SMT or ensuring sensitive inference never shares physical cores with untrusted code eliminates the most direct attack vector. This is the recommended mitigation deployed by cloud providers after Spectre/Meltdown and remains the most effective defense against hyperthread-based attacks.

Trade-offs:

- **Pro:** Complete protection against SMT-based attacks
- **Con:** 50% reduction in logical CPU count, increased latency for multi-threaded workloads
- **Con:** May not protect against LLC or cross-core attacks

2) Context Padding (Moderate Cost): Always allocate and process a fixed maximum context (e.g., 2048 tokens), padding shorter prompts with dummy tokens.

Trade-offs:

- **Pro:** Eliminates context size leakage
- **Pro:** Software-only solution
- **Con:** 2–8× increased latency for short prompts
- **Con:** Proportional increase in memory usage and energy

3) Decoding Normalization (Moderate Cost): Restrict all inference to a single decoding mode (e.g., always greedy or always temperature=0.7).

Trade-offs:

- **Pro:** Eliminates decoding strategy leakage
- **Pro:** Minimal performance overhead
- **Con:** Reduces model expressivity and quality for some tasks
- **Con:** May not be acceptable for general-purpose APIs

4) Noise Injection (Limited Effectiveness): Add randomized delays or dummy computation to blur timing patterns. Similar techniques have been proposed for cryptographic constant-time implementations and neural network defenses.

Trade-offs:

- **Pro:** No architectural changes required
- **Con:** Can be defeated by averaging over many observations
- **Con:** Adds latency without strong security guarantees
- **Con:** Difficult to calibrate noise level—too little noise is ineffective, too much degrades performance

5) Hardware Partitioning (Emerging Solutions): Modern Intel CPUs support Cache Allocation Technology (CAT) and Memory Bandwidth Allocation (MBA) that enable cache and memory partitioning between security domains. Similarly, ARM TrustZone provides hardware-enforced isolation.

Trade-offs:

- **Pro:** Allows SMT without full core isolation
- **Pro:** Hardware-enforced security boundaries
- **Con:** Limited availability—requires specific CPU models
- **Con:** Coarse-grained partitioning may still leak through branch predictors
- **Con:** Performance overhead from reduced cache capacity per partition

6) Partitioning by Sensitivity: Route sensitive inference to isolated cores/machines, use shared hardware only for low-sensitivity queries.

Trade-offs:

- **Pro:** Balances security and efficiency
- **Pro:** Allows fine-grained control
- **Con:** Requires query classification infrastructure
- **Con:** May leak information through routing itself

C. Limitations

Our study has several limitations:

- **Limited model diversity:** While we evaluate two models (TinyLLaMA-1.1B and DeepSeek-8B), different architectures (GPT-style vs. LLaMA-style) or quantization schemes (8-bit, FP16) may exhibit different leakage characteristics.
- **Controlled experiments:** Real-world deployments involve more noise from concurrent processes, network I/O, and system events.
- **Fixed generation length:** Variable-length generation may create additional signals or reduce detection accuracy.
- **Limited prompt diversity:** Our semantic templates are deliberately controlled; realistic queries show greater variety.
- **No defenses active:** We measure baseline leakage without testing against hardened implementations.

D. Future Work

Several directions warrant further investigation:

- Evaluating attacks on larger models (LLaMA-7B, 13B) and different frameworks
- Testing robustness against noise and concurrent workloads
- Exploring cross-core and cross-VM attack scenarios
- Developing and evaluating hybrid defense mechanisms
- Investigating hardware-based mitigation (e.g., cache partitioning, predictive isolation)

VII. Related Work

A. Microarchitectural Side Channels

Cache timing attacks were formalized by Osvik et al.'s Prime+Probe technique [1] and Yarom and Falkner's Flush+Reload [2], which have been applied to cryptographic key extraction. Branch predictor attacks [8] demonstrated control-flow leakage, while TLB-based attacks [9] enabled KASLR bypass. Spectre [10] revealed that even transient execution creates timing side channels.

B. Side Channels in Machine Learning

Yan et al. [3] demonstrated cache-based attacks on neural network inference for model extraction, while Chen et al. [11] showed that RNN input sequences can be partially recovered from cache timing. Shokri et al. [12] introduced membership inference attacks on training data.

However, prior work has not systematically studied side-channel leakage from CPU-based LLM inference, particularly regarding high-level inference parameters like context and decoding strategy.

C. LLM Security

LLM security research has focused on training data extraction through carefully crafted queries [4]. Our work complements this by exploring microarchitectural leakage, which operates at a lower level of abstraction and cannot be mitigated by prompt filtering or output sanitization alone.

VIII. Conclusion

We presented LLMProbe, a comprehensive study of microarchitectural side-channel attacks on CPU-based LLM inference. Through systematic evaluation against TinyLLaMA-1.1B and DeepSeek-8B running under `llama.cpp`, we demonstrated that unprivileged co-located attackers can infer context size with 80.5% accuracy and decoding strategy with 90.0% accuracy using only SMT contention measurements. Critically, we discovered that larger models exhibit **amplified leakage characteristics**, with DeepSeek-8B showing extreme timing distribution outliers (kurtosis up to 1498) that likely enable even higher classification accuracy. While prompt semantic classification remains challenging (61.4% accuracy for 4-class problem), custom prompts show strong distinguishability (93% recall).

Our findings reveal that even CPU-only LLM deployments remain vulnerable to microarchitectural attacks, despite lacking the accelerator-specific optimizations that prior work has exploited. The strong leakage for context and decoding parameters creates risks for service fingerprinting, competitive intelligence, and privacy violations in multi-tenant environments. Moreover, the scaling behavior—where larger models produce stronger signals—suggests that production deployments of state-of-the-art models (e.g., GPT-4-class systems with 100B+ parameters) face even greater vulnerability.

Effective mitigation requires architectural approaches (SMT isolation, core partitioning) combined with software-level normalization (context padding, fixed decoding modes). These defenses impose non-trivial performance costs, creating a fundamental trade-off between efficiency and security for LLM inference on shared hardware.

We hope this work encourages the community to design side-channel-aware LLM inference systems and motivates hardware vendors to provide better isolation primitives for ML workloads.

Appendix

This project was a collaborative effort between two team members, with contributions spanning experimental design, data collection, analysis, and visualization. Below we detail each member's specific contributions:

A. *Devesh Jani*

TinyLLaMA Model - Context Size Attack:

- Designed and executed experiments for context size inference (128, 512, 2048 tokens)
- Implemented cache and TLB probe data collection on SMT sibling cores
- Generated all statistical analysis and visualizations including:
  - Context size confusion matrix (Figure 2)
  - Context size feature importance plot (Figure 3)
  - Cache probe cycle distribution histograms (Figure 1)
  - Boxplot comparisons across context sizes (Figure 4)
  - Skewness/kurtosis distribution analysis (Figures 5–6)
  - Extreme context comparison plot for TinyLLaMA (128 vs 2048 tokens)
- Computed classification metrics (accuracy, precision, recall, F1 scores) and adversarial advantage
- Created statistical summary tables for context size experiments

TinyLLaMA Model - Decoding Strategy Attack:

- Designed experiments comparing greedy vs. sampling decoding strategies
- Collected BTB and PHT probe measurements across multiple runs
- Generated all visualizations and analysis including:
  - Decoding strategy confusion matrix (Figure 7)
  - Decoding strategy feature importance rankings (Figure 8)
  - Cycle distribution comparisons between greedy and sampling modes
- Analyzed asymmetric performance (high greedy recall, lower sampling recall)
- Documented mechanistic explanations for deterministic vs. stochastic execution patterns

DeepSeek Model - Context Size Attack:

- Replicated context size experiments on the larger 8B parameter model
- Collected cache probe data demonstrating amplified leakage signals
- Generated all DeepSeek context-related visualizations including:
  - Context size confusion matrix for DeepSeek (corresponding to TinyLLaMA Figure 2)
  - Context size feature importance plot for DeepSeek (corresponding to TinyLLaMA Figure 3)
  - Cache probe cycle histograms showing wider distributions
  - Boxplots illustrating increased variance for DeepSeek
  - Skewness/kurtosis analysis revealing 2.2$\times$ amplification
  - Extreme context comparison plot for DeepSeek (128 vs 2048 tokens)
- Quantified amplification factors (87.2% accuracy vs. 80.5% for TinyLLaMA)
- Created comparison tables documenting cross-model scaling trends

B. *Misha Joshi*

TinyLLaMA Model - Prompt Semantics Attack:

- Designed prompt templates for four semantic categories (Math, Code, Natural Language, Custom)
- Executed experiments with BTB and PHT probes to capture semantic signatures
- Generated all semantic analysis visualizations including:
  - Prompt semantics confusion matrix (Figure 9)
  - Feature importance plots for semantic classification
  - Skewness/kurtosis distributions per prompt category (Figure 10)
  - PCA visualizations showing semantic clustering (Figure 11)
- Analyzed class imbalance effects (93% custom prompt detection vs. near-random generic categories)
- Computed per-class F1 scores and identified custom prompts as strongest signal
- Created statistical tables documenting cycle characteristics per semantic category

DeepSeek Model - Decoding Strategy Attack:

- Replicated decoding strategy experiments on the DeepSeek-8B model
- Collected BTB and PHT measurements demonstrating improved classification
- Generated all DeepSeek decoding visualizations including:
  - Decoding strategy confusion matrix showing balanced near-perfect performance (corresponding to TinyLLaMA Figure 7)
  - Feature importance plots highlighting kurtosis as discriminative feature (corresponding to TinyLLaMA Figure 8)
  - Statistical comparison tables for DeepSeek decoding experiments
- Quantified improvement over TinyLLaMA (96.9% vs. 90.0% accuracy)
- Documented balanced F1 scores across both greedy and sampling classes (0.95 and 0.98)

DeepSeek Model - Prompt Semantics Attack:

- Replicated semantic classification experiments on DeepSeek-8B
- Collected probe data showing amplified semantic signals compared to TinyLLaMA
- Generated all DeepSeek semantic visualizations including:
  - Prompt semantics confusion matrix for DeepSeek (corresponding to TinyLLaMA Figure 9)
  - Skewness/kurtosis analysis per prompt type (corresponding to TinyLLaMA Figure 10)
  - PCA plots showing improved semantic separation (corresponding to TinyLLaMA Figure 11)
- Quantified per-class improvements: Code F1 (2.7×), Math F1 (1.5×), NL F1 (1.5×)
- Created statistical summary tables showing extreme kurtosis values (up to 1498) for custom prompts
- Documented mechanistic explanations for vocabulary distribution and attention sparsity effects

C. Shared Contributions

Both team members collaborated on:

- Experimental infrastructure setup (victim/attacker processes, SMT pinning, data logging)
- Development of probe implementations (cache, TLB, BTB, PHT)
- Random Forest classifier training and hyperparameter tuning
- Cross-model comparison analysis and amplification factor quantification
- Paper writing, including methodology, threat model, and discussion sections
- Statistical validation (bootstrap confidence intervals, effect size calculations)
- Mitigation strategy analysis and security implications discussion

References

[1] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in *Cryptographers' Track at the RSA Conference*. Springer, 2006, pp. 1–20.

[2] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *23rd USENIX Security Symposium*, 2014, pp. 719–732.

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

[4] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson *et al.*, "Extracting training data from large language models," in *30th USENIX Security Symposium*, 2021, pp. 2633–2650.

[5] P. Zhang, G. Zeng, T. Wang, and W. Lu, "Tinyllama: An open-source small language model," https://github.com/jzhang38/TinyLLaMA, 2024.

[6] DeepSeek-AI, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," https://github.com/deepseek-ai/DeepSeek-R1, 2024.

[7] G. Gerganov and contributors, "llama.cpp: Port of facebook's llama model in c/c++," https://github.com/ggerganov/llama.cpp, 2023.

[8] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Jump over ASLR: Attacking branch predictors to bypass ASLR," in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[9] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *27th USENIX Security Symposium*, 2018, pp. 955–972.

[10] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2019, pp. 1–19.

[11] Z. Chen, B. Kailkhura, and P. Varshney, "Neural network inversion in adversarial setting via background knowledge alignment," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 225–240.

[12] 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)*. IEEE, 2017, pp. 3–18.

Project Repository: https://github.com/mishajoshi/ECE592-FinalProject