Quantamination: Dynamic Quantization Can Leak Your Data Across the Batch

Anonymous Authors[1]

Abstract

Dynamic quantization emerged as a practical approach to increase the utilization and efficiency of the machine learning serving flow. Unlike static quantization, which applies quantization offline, dynamic quantization operates on tensors at run-time, adapting its parameters to the actual input data. Today's mainstream machine learning frameworks – including ML compilers and inference engines – frequently recommend dynamic quantization as an initial step for optimizing model serving. This is because dynamic quantization can significantly reduce memory usage and computational load, leading to faster token generation and improved model serving efficiency without substantial loss in model accuracy. In this paper, we reveal a critical vulnerability in dynamic quantization: an adversary can exploit such quantization strategy to steal sensitive user data placed in the same batch as the adversary's input. Our analysis demonstrates that dynamic quantization, when improperly implemented or configured, can create side channels that expose information about other inputs within the same batch. We call this phenomenon ***Quantamination***, describing contamination from quantization. Specifically, we show that at least 4 of the most popular ML frameworks in use today either default to or can use configurations that leak data across the batch boundary. This data leakage, in theory, allows attackers to partially or even fully recover other users' batched input data, representing a serious privacy risk for existing ML serving frameworks.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

1. Introduction

Modern LLM serving frameworks such as vLLM (Kwon et al., 2023), SGLang (Zheng et al., 2024), and DeepSpeed (Rasley et al., 2020) routinely batch concurrent requests from multiple users into shared forward passes to maximize GPU utilization and throughput. To further reduce inference cost and memory footprint, these systems increasingly deploy post-training quantization, converting high-precision activations to lower-precision formats like INT8 or FP8 at runtime. These activation quantization parameters can either be determined statically ahead of time through a calibration set, or computed dynamically during the forward pass for greater accuracy. While many modern serving frameworks compute such dynamic activation quantization parameters for each token separately, several also support computing these parameters across the entire batch.

We find that batching and quantization can create unexpected privacy leakages when configured to compute quantization parameters dynamically across the batch. In this setting, all samples in the batch jointly determine the shared quantization parameters through a global `min`/`max` computation over the combined activation tensor, creating a cross-user side channel: the victim's input influences the quantization applied to the adversary's activations, producing measurable differences in the adversary's output. By crafting an input, co-locating it in the same batch as the victim, and observing the resulting perturbation, an adversary can systematically infer properties of the victim's data.

We demonstrate practical attacks exploiting this side channel in both LLM and classification settings, achieving 99.6–100% token recovery accuracy in LLMs and exact image identification in classification tasks, requiring only batch co-location and observation of top-1 log probabilities. We survey the vulnerability surface across major inference frameworks, identifying at-risk configurations in vLLM's default FP8 online quantization, SGLang's per-tensor FP8 mode, and the dynamic quantization APIs of ONNX Runtime and PyTorch. Notably, per-token dynamic quantization, already the standard in many LLM serving frameworks for both accuracy and computational efficiency reasons, eliminates this side channel entirely, highlighting that per-tensor dynamic activation quantization cannot only be suboptimal for performance but also introduces a concrete privacy vulnerability

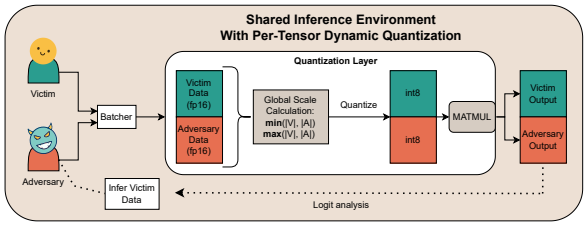

Figure 1. Overview of the per-tensor dynamic activation quantization side channel. A victim and an adversary's inputs are batched together in a shared forward pass. During per-tensor dynamic quantization, a global scale factor is computed from the combined activations of both inputs, causing the victim's data to influence the quantization applied to the adversary's activations. By observing perturbations in its own output logits, the adversary can infer properties of the victim's input.

in multi-tenant serving environments. Finally, we discuss real-world deployment challenges that complicate exploitation in production, including non-determinism from hardware and software heterogeneity and possible provider-side adaptations that conflict with the supposedly deterministic temperature 0 setting.

2. Related Works

Batch-based privacy attacks. Several recent works demonstrate that shared computation in batched inference creates cross-user information leakage. Yona et al. (2024) show that in Mixture-of-Experts (MoE) models, an attacker co-located in the same batch can steal victim prompts by exploiting the tie-handling behavior of Expert-Choice-Routing, and Hayes et al. (2024) demonstrate that an adversary can manipulate expert routing by overflowing expert buffers, causing both output degradation and targeted manipulation of co-batched victims' outputs. Küchler et al. (2025) take a complementary approach, demonstrating that architectural backdoors can be embedded into models to enable within-batch data stealing and inference manipulation. Wu et al. (2025) exploit prefix-level KV cache sharing across requests in multi-tenant LLM serving (SGLang, vLLM) to reconstruct prior users' prompts via cache-timing side channels. In federated learning, Yin et al. (2021) demonstrate that batch-averaged gradients can be inverted to recover individual training images, with batch normalization statistics providing additional reconstruction signal. Our attack differs from all of the above in that it requires no architectural modification, no MoE routing, no cache sharing, and is not in a federated learning training setting. Instead it exploits a side channel inherent to inference frameworks that perform per-tensor dynamic quantization, a default or optional configuration in several widely deployed systems.

Numerical error as information channel. A growing body of work recognizes that numerical artifacts in neural network computation carry exploitable information. Schlögl et al. (2021); Schlögl et al. (2021); Schlögl et al. (2023) demonstrate that framework- and hardware-specific floating-point deviations serve as forensic fingerprints, enabling identification of the hardware used to produce a given model output. Zhang et al. (2025) systematize this observation, showing that numerical differences across platforms are sufficient to reliably infer the hardware and software environment in which a model was executed. Clifford et al. (2024) exploit similar device-specific numerical behavior to bind models to specific hardware, while Jia & Rinard (2021) and Zombori et al. (2021) show that floating-point error can be weaponized to fool neural network verifiers. In the quantization domain specifically, Mishra et al. (2026) identify temporal information leakage through block quantization scales during training, and Egashira et al. (2024) show that an adversary can craft weights that activate malicious behavior only after quantization (an integrity attack on the model supply chain). Our work reveals a distinct inference-time privacy risk that requires no adversarial crafting: the data-dependent quantization parameters computed during standard batched inference create exploitable cross-sample information flow, exposing numerical artifacts as a privacy side channel.

3. Background

3.1. Post-Training Quantization

Post-training quantization reduces the numerical precision of neural network weights and activations from high-precision floating-point (FP32/FP16) to short number formats (INT8, FP8, INT4) to accelerate inference and reduce memory footprint. The quantization process relies on calibrated parameters to clamp, shift, and scale weights and activations before rounding to low-precision grids to recover model performance.

Weight quantization is usually performed offline before deployment. Given a trained network, weights are considered as constants; thus, complex calibration and quantization algorithms can be applied to yield low-precision weights before deployment. Modern weight-only approaches like GPTQ (Frantar et al., 2022) and AWQ (Lin et al., 2024) achieve 3–4× model compression by quantizing weights to INT4/INT8, with per-token/channel calibrated scales to minimize accuracy loss. Unlike quantization of weights, activation quantization is more challenging as the activations change with each input at inference time. **Activation quantization** parameters can be determined in two ways: (1) computed offline from calibration data and fixed for all inputs (*static* activation quantization), or (2) computed at runtime from the observed values of each forward pass (*dynamic* activation quantization) (Krishnamoorthi, 2018; Wu et al., 2020). Dynamic quantization introduces more

runtime cost compared to static quantization, but is usually more effective for extremely low precisions like INT4. Several inference frameworks provide static activation quantization — for example, TensorRT-LLM's default FP8 and INT8 SmoothQuant modes calibrate per-tensor or per-token activation scales offline and bake them into the model at conversion time (NVIDIA, 2024). Because these scales do not depend on runtime inputs, static quantization does not create cross-sample information flow and is outside the scope of our attack. Dynamic activation quantization adapts to input variation and usually achieves higher accuracy, but introduces data-dependent computation of scaling parameters during inference. It is this runtime dependence that our attack exploits.

Existing dynamic activation quantization methods operate at different granularities. *Per-tensor* quantization computes a single scale factor from the global `min`/`max` across an entire activation tensor. This approach degrades accuracy when activation magnitudes vary significantly, a common occurrence in large language models processing diverse sequences (Dettmers et al., 2022; Xiao et al., 2023). *Per-token* quantization addresses this by computing separate scale factors for each token, preserving accuracy for inputs with heterogeneous activation distributions. Methods like LLM.int8() (Dettmers et al., 2022) and ZeroQuant (Yao et al., 2022) demonstrate that per-token dynamic quantization maintains model performance at scale. On modern GPUs, per-token quantization can also be *faster* than per-tensor: computing per-tensor `min`/`max` requires a global reduction across all $B \times D$ elements with expensive cross-block synchronization, whereas per-token computes B independent reductions over D elements each, mapping naturally to GPU parallelism without cross-block communication. Consequently, per-token dynamic quantization has become the standard in LLM-serving frameworks for INT8 precision. However, per-tensor dynamic quantization persists in two important settings: (1) FP8 online quantization modes in LLM-serving frameworks such as vLLM (vLLM Team, 2024), where it is the *default*; and (2) general-purpose frameworks such as ONNX Runtime (Microsoft, 2024) and PyTorch (PyTorch Team, 2024), whose INT8 dynamic quantization predates the per-token designs adopted by LLM-specific systems. **When per-tensor dynamic quantization is applied during batched inference, all samples in the batch contribute to the shared `min`/`max` computation, creating cross-batch information flow that enables the privacy attacks demonstrated in this work.**

3.2. Framework Deployment Landscape

We survey dynamic activation quantization configurations across major inference frameworks in Table 4 (Appendix A). The vulnerability surface concentrates in two places: FP8 online quantization in LLM serving frameworks, where per-tensor remains the default (vLLM's –quantization fp8, SGLang's `fp8dq-per_tensor`), and INT8 dynamic quantization in general-purpose frameworks (ONNX Runtime's `quantize_dynamic()`, PyTorch's observer API), which predate the per-token designs adopted by LLM-specific systems. By contrast, LLM-serving INT8 paths (SGLang, vLLM, DeepSpeed) have standardized on per-token granularity, eliminating the side channel. Static-scale configurations (TensorRT-LLM defaults, LiteRT full-integer) are excluded since fixed scales do not create runtime cross-batch flow. Per-tensor dynamic quantization is therefore a worst-case scenario: it compromises user privacy while simultaneously degrading model performance.

4. Methodology

4.1. Threat Model

We consider an adversary who seeks to infer sensitive information about a victim's input by exploiting side-channels introduced by dynamic quantization in batched inference settings.

Notation. Let s denote the victim's secret input, a denote the adversarial input controlled by the attacker, and $C = \{c_1, \ldots, c_k\}$ represent a candidate set of possible victim inputs. We use $\text{logits}(a, x)$ to denote the model's logit output for input a when batched with input x. For LLM token recovery, we denote the victim's token sequence as $s = [s_1, s_2, \ldots, s_n]$ and the adversary's token sequence as $a = [a_1, a_2, \ldots, a_m]$, where $s_i, a_j \in V$ and V is the model's vocabulary.

Attack surface and goal. In the per-tensor setting, dynamic quantization computes a single scale and zero-point from the combined activation statistics of all inputs in the batch, so a and s share quantization parameters within each forward pass. The adversary's goal is to recover the victim's token sequence s (LLM setting) or identify the victim's class or nearest candidates in C (classification setting).

Adversary capabilities. We assume the adversary can (i) co-locate a with s in the same batch, (ii) observe $\text{logits}(a, \cdot)$ as top-1 log probabilities (LLM) or the full logit vector (classification), (iii) access the vocabulary V (LLM) or a same-distribution public dataset as C (classification), and (iv) know the target model and that dynamic quantization is used. We evaluate both white-box (local model access, enabling arbitrary KV-cache states and optimized probe selection) and black-box (API-only) variants; the black-box LLM setting additionally assumes greedy decoding so that the adversary's sequence is reproducible across queries.

Justification. Batch co-location arises naturally in multi-tenant serving systems that group concurrent requests for throughput. We use a batch size of 2 in experiments to isolate the quantization effect cleanly; an attacker facing a larger production batch N can submit $N-1$ identical items to reduce to the same setup, so larger batches add noise but do not remove the shared quantization parameter. Appendix B gives the full enumeration of capabilities and their justifications.

4.2. Dynamic Quantization Mechanics in Batched Inference

Quantization parameter computation Dynamic INT8 quantization converts floating-point activations to integers at inference time. Given an input batch $\mathbf{X} \in \mathbb{R}^{b,d}$ where b denotes batch size and d denotes hidden dimension, the per-batch scale α and the zero point β are calculated as

$$\alpha = \frac{\max(\mathbf{X}) - \min(\mathbf{X})}{255}, \beta = -\text{round}(\frac{\min(\mathbf{X})}{\alpha}). \quad (1)$$

where $\max(\cdot)$ and $\min(\cdot)$ calculate the maximum and minimum values across the entire matrix, and $\text{round}(\cdot)$ denotes elementwise rounding to the nearest (RTN).

The input batch is then quantized on the fly:

$$\mathbf{X}_q = \text{clip}\left(\text{round}(\frac{\mathbf{X}}{\alpha} + \beta),\ 0,\ 255\right) \quad (2)$$

where $\text{clip}(\cdot)$ clamps every element to the INT8 range (0-255).

Key observation: An input sample with a larger activation magnitude dominates quantization parameters (α and β) of a batch. Different co-batched inputs create measurably different quantization effects, observable in final logits through $\Delta(a, s, c) = \|\text{logits}(a, s) - \text{logits}(a, c)\|$. In multi-layer networks, per-layer quantization effects compound in complex ways (discussed further in Section 4.4).

4.3. Attack Methodology: LLM Token Recovery

The LLM setting is particularly vulnerable because the vocabulary V is known and fixed. We recover the secret token sequence $s = [s_1, \ldots, s_n]$ iteratively, exploiting batching behavior during the decode phase of LLM inference.

Batching in LLM inference. LLM inference consists of two phases with distinct batching characteristics. The *prefill phase* processes the entire prompt in parallel, where multiple tokens from the same sequence (or multiple sequences) can be batched together, creating complex quantization interactions. The *decode phase* generates tokens sequentially, one per forward pass, where each forward pass in batched serving typically processes one new token per active sequence. We target the decode phase where each step processes exactly two tokens, the adversarial token a_i and the victim's token s_i, with shared quantization parameters, creating a direct observable side-channel.

Batching persistence assumption. Modern LLM serving systems (e.g., vLLM, TensorRT-LLM) use continuous batching where sequences are grouped together and remain co-batched throughout their generation. Once the adversary's sequence and victim's sequence are batched together in the decode phase, they continue processing in the same batch until one completes. This persistence is realistic because re-batching mid-generation would require expensive KV cache reorganization, which serving systems avoid for efficiency.

Token-by-token recovery algorithm.

Initial token recovery ($i = 1$):

For the first token, we have no context to constrain the search space, requiring broader exploration:

1. *Query candidate tokens:* For each $t^{(j)} \in V$ measure:
$$\Delta_1(t) = \|\text{logits}(a_1, s_1) - \text{logits}(a_1, [t^{(j)}])\|_2. \quad (3)$$
2. *Match with early termination:* Return the first token satisfying:
$$\hat{s}_1 = \arg\min_{t \in V} \Delta_1(t) \quad \text{subject to} \quad \Delta_1(\hat{s}_1) < \epsilon, \quad (4)$$
where ϵ is a match threshold.
3. *Optional: Prior-based ordering:* To reduce queries, order candidates by sentence-initial probability $p_{\text{init}}(t)$ before testing. Tokens like `<BOS>`, `"The"`, `"I"` are tested first, exploiting statistical biases in natural language.

Subsequent token recovery ($i > 1$):

For later tokens, context $s_{<i} = [s_1, \ldots, s_{i-1}]$ provides strong constraints, enabling efficient search:

1. *Rank by language model prior:* Compute $p(t|s_{<i})$ for all $t \in V$ and sort in descending order of probability.
2. *Sequential testing:* Iterate through ranked candidates $t^{(1)}, t^{(2)}, \ldots$ where $p(t^{(j)}|s_{<i}) \geq p(t^{(j+1)}|s_{<i})$:
$$\Delta_i(t^{(j)}) = \|\text{logits}(a_i, s_i) - \text{logits}(a_i, t^{(j)})\|_2. \quad (5)$$
3. *Early termination:* Stop at the first candidate satisfying:
$$\hat{s}_i = t^{(j)} \quad \text{if} \quad \Delta_i(t^{(j)}) < \epsilon. \quad (6)$$

Depending on threat model, this recovery algorithm needs to be interpreted with slightly different assumptions.

White-box setting (open-source model): The adversary runs the model locally for candidate testing. For testing candidate tokens $t^{(j)} \in V$ at position i, the adversary prefills the KV cache with previously found secret sequence context $\hat{s}_{<i}$ and uses the actual adversarial token sequence $[a_1, \ldots, a_i]$ observed during victim interaction to measure $\Delta_i(t^{(j)})$. No assumptions about the adversary's decoding strategy are needed.

Black-box setting (closed-source model): Candidate search must be performed through API queries. To replicate the adversary's token sequence $[a_1, \ldots, a_i]$ for candidate testing, we assume the adversary used greedy decoding (selecting $\arg\max$ token at each step). This allows deterministically reconstructing the adversarial sequence state for testing each candidate. Without this assumption, the adversary would need to re-sample their sequence multiple times, hoping to randomly recreate the exact same token sequence $[a_1, \ldots, a_i]$ observed during victim interaction. The probability of matching this sequence decreases exponentially with sequence length, particularly when sampling strategies (temperature, top-k, nucleus sampling) frequently select lower-probability tokens.

4.4. Attack Methodology: Classification Setting

The classification setting is more challenging because exact matches for s may not exist in the adversary's candidate set C. Unlike the discrete, finite vocabulary in LLMs, the space of possible images is effectively infinite in practice. In our evaluation, we test two scenarios: (1) a setting where the entire test set (including the secret) is used as the candidate set, representing cases where the adversary aims to identify which specific image from a known dataset the victim queried, and (2) a setting where all test samples except the secret are used as candidates, representing a more realistic scenario where the adversary cannot enumerate all possible images and the victim's input may not exist in any available dataset. As in the LLM token recovery, we assume that the secret input produces a distinct dynamic quantization effect on the adversary's output logit calculation. In the first scenario, we test whether this effect can pinpoint the exact secret if it exists in the candidate set. In the second scenario, we test whether inputs from the same class produce more similar quantization effects than inputs from different classes, allowing class-level inference even without an exact match.

Attack strategy. For each candidate $c_j \in C$ and adversarial probe a, we compute:

$$\text{score}(c_i) = \|\text{logits}(a, s) - \text{logits}(a, c_i)\|_2. \quad (7)$$

The best candidate is:

$$\hat{c} = \arg\min_{c_i \in C} \text{score}(c_i). \quad (8)$$

In multi-layer networks, each layer applies dynamic quantization independently based on that layer's activation statistics. At each layer, the quantization parameters (scale, zero-point) are determined by the minimum and maximum activations in the batch. Either one input sets both extremes (dominating that layer), or each input sets one extreme (split control). Crucially, which input dominates varies layer by layer: input a might set the max at layer 1 but the min at layer 2. These per-layer effects can partially cancel when propagated through the network, making aggregate final logits ambiguous. However, as network depth increases, both the probability that one input dominates across all layers and the probability that per-layer effects cancel out perfectly decrease exponentially, making coincidental aggregate matches increasingly unlikely. Hence, in a significantly deep network, for a candidate c to produce matching logits for an arbitrary adversarial probe a, c must exhibit nearly identical per-layer quantization behavior to s.

White-box setting: With local model access, the adversary can optimize probe selection by computing per-layer activation ranges for potential probes and selecting one with particularly small activation ranges across multiple layers, maximizing its influence on quantization parameters.

Black-box setting: Limited to API queries, the adversary uses random selection for both adversarial probes and candidates from the available dataset, relying solely on aggregate logit scoring.

5. Evaluation

5.1. LLM Token Recovery

We evaluate the token recovery attack on three small language models with varying architectures and tokenizers: TinyStories-1M (Eldan & Li, 2023), Pythia-70M (Biderman et al., 2023), and SmolLM2-135M (Allal et al., 2025). Table 1 summarizes the model specifications. All experiments use batch size two during the decode phase, with one token from the adversary's sequence and one token from the victim's sequence per forward pass.

The diversity in tokenizer vocabulary sizes and training corpora allows us to assess whether the attack generalizes across different tokenization schemes. TinyStories uses the GPT-2 tokenizer (Radford et al., 2019) (50,257 tokens) similar to GPT-Neo (Black et al., 2021), Pythia uses the GPT-NeoX-20B tokenizer (Black et al., 2022) trained specifically on The Pile (Gao et al., 2020), and SmolLM2 uses a custom tokenizer (Allal et al., 2024) trained on the Smollm Corpus.

Table 1. Language models and tokenizers used for token recovery evaluation.

Model	Parameters	Vocab Size	Tokenizer
TinyStories-1M (Eldan & Li, 2023)	1M	50,257*	GPT-2 Tokenizer (Radford et al., 2019)
Pythia-70M (Biderman et al., 2023)	70M	50,257	GPT-NeoX-20B Tokenizer (Black et al., 2022)
SmolLM2-135M (Allal et al., 2025)	135M	49,152	SmolLM tokenizer (Allal et al., 2024)

*TinyStories models use only the top 10,000 most frequent tokens during training, though the full GPT 2/GPT Neo vocabulary contains 50,257 tokens.

We sample secret sequences of up to 20 tokens from various datasets and measure the number of candidate tokens evaluated before finding the correct match. All evaluations were run on Intel Xeon Ice Lake CPUs and to ensure practical efficiency, we impose a 4-hour timeout per sequence. Runs exceeding this limit were excluded from the analysis (affecting a small percentage of sequences, especially for out-of-distribution cases). To create a more realistic and challenging attack scenario, we convert the full logit vectors to log probabilities and extract only the top-1 prediction (argmax) rather than using the complete logit distribution. This single scalar value per forward pass represents what an attacker could at most observe from typical model serving APIs. We set the matching threshold to $\epsilon = 10^{-6}$ for the L2 distance between these top-1 log probability values, which we found empirically provides a robust threshold.

Table 2. Token recovery attack results across models and datasets. All models achieve near-perfect recovery with varying efficiency based on model-data distribution alignment. All settings were run 100 times with a 4-hour timeout per sequence. Missing runs in the table are runs that did not finish in the imposed time interval. This highlights the inefficiency of recovering out-of-distribution sequences compared to in-distribution sequences.

Model	Dataset	Runs	Accuracy	Mean Queries
Specialized model				
TinyStories-1M	TinyStories	100	100.0%	424
TinyStories-1M	WikiText	12	99.6%	8,095
TinyStories-1M	AG News	8	100.0%	7,489
General-purpose models				
Pythia-70M	WikiText	96	99.6%	1,343
SmolLM2-135M	WikiText	88	100.0%	985
Theoretical baseline (random search)				
All models	Any	—	—	$\lvert V\rvert/2 \sim 25{,}000$

Attack success. Table 2 summarizes our results. The attack achieves near-perfect token recovery across all settings, with 99.6-100% accuracy. With our chosen threshold $\epsilon = 10^{-6}$, we observe occasional false positives (0.4% of tokens for Pythia-70M, 0% for others). These occur when two different tokens produce nearly identical quantization effects across layers. Stricter thresholds might be able to eliminate these errors.

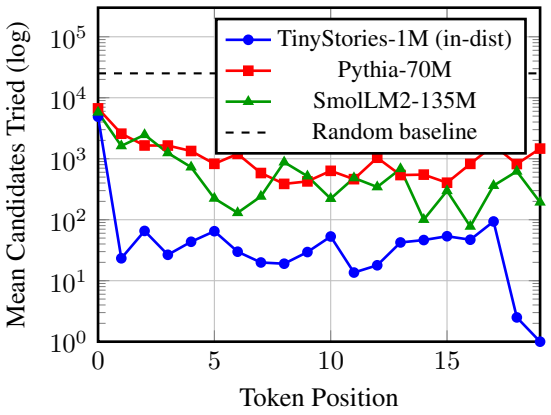

Figure 2. Query efficiency improves dramatically with position as autoregressive context accumulates. The first token (no context) requires thousands of queries, while subsequent tokens leverage language priors to narrow the search space. TinyStories-1M shows the strongest efficiency gain on in-distribution data, while general-purpose models (Pythia, SmolLM2) maintain consistent mid-range efficiency. All models vastly outperform random search baseline (∼25,000 queries).

Attack efficiency and distribution alignment. The attack significantly outperforms random search, which would require $|V|/2 \approx 25{,}000$ queries per token on average. In realistic attack scenarios, victim secrets would be generated by the model itself during decoding, ensuring in-distribution alignment. However, to understand how distribution alignment affects attack efficiency, and to evaluate the attack on more meaningful sequences than randomly sampled tokens, we deliberately tested recovery across different dataset sources. This reveals that efficiency varies dramatically based on the alignment between model training distribution and secret text distribution.

TinyStories-1M, a model specialized for simple children's stories, illustrates this effect. For in-distribution secrets from its native domain, the attack achieves exceptional efficiency (424 queries/token on average, 59× faster than random). However, for out-of-distribution datasets like WikiText and AG News, efficiency degrades severely to 7,500-8,000 queries per token, with many sequences timing out after 4 hours. While such out-of-distribution scenarios are unrealistic in practice, this comparison demonstrates how critical language priors are to the computational feasi-

bility of this attack. For general-purpose models trained on diverse web text (Pythia-70M and SmolLM2-135M), the attack maintains practical efficiency (985-1,343 queries/token) on WikiText, achieving 18-25× speedup over random search and demonstrating the general computational feasibility of this attack. We expect that sequences generated directly by these models (the realistic threat scenario) would be even more efficient to recover, as they would exhibit stronger alignment with the model's learned distribution.

Context-based acceleration. Figure 2 demonstrates how autoregressive context dramatically accelerates recovery. The first token (no context) requires 5,000-7,000 queries across all models, but efficiency improves exponentially at subsequent positions as each recovered token constrains the next. TinyStories-1M exhibits the most dramatic improvement on in-distribution data: after the costly first token (∼4,900 queries), positions 1-19 average only 20-65 queries, reflecting its strong priors over familiar story structures. In contrast, general-purpose models (Pythia-70M, SmolLM2-135M) maintain more consistent mid-range efficiency (200-2,500 queries per position), reflecting broader but less specialized language knowledge. Despite these differences in pattern, all models vastly outperform random search (∼25,000 queries) at every position, with the advantage growing as context accumulates.

5.2. Classification Model Class Recovery

We evaluate the classification attack on MNIST (Lecun, 1998), a 10-class handwritten digit classification benchmark with 10,000 test images (1,000 per class). We test three CNN architectures with increasing depth: a simple 3-layer CNN (3 convolutional layers + 2 fully connected layers), ResNet18 (He et al., 2016) (18 layers), and ResNet50 (He et al., 2016) (50 layers). All models are trained to convergence on MNIST and achieve >98% test accuracy. We apply 8-bit dynamic quantization during inference using the same batching setup as the LLM experiments (batch size 2: one adversarial probe a, one victim input s).

5.2.1. Results: Secret in Candidate Set

Table 3. Exact recovery success rate when the secret exists in the candidate set. Each model is evaluated 10 times per probe strategy (20 trials total per model).

Model	Random Probe	Layer-Diverse Probe
Simple CNN (3 layers)	8/10 (80%)	10/10 (100%)
ResNet18 (18 layers)	10/10 (100%)	10/10 (100%)
ResNet50 (50 layers)	10/10 (100%)	10/10 (100%)

Table 3 shows near-perfect exact recovery when the secret exists in the candidate set. For deeper networks (ResNet18, ResNet50), even randomly selected probes achieve 100% success, confirming that per-layer quantization effects are unlikely to coincidentally cancel across many layers. The simple CNN occasionally fails with random probes (80% success) because strong random probes can dominate quantization parameters, obscuring the victim's signal. Optimized layer-diverse probes with minimal activation ranges restore perfect recovery. This demonstrates that probe selection matters for shallow networks but becomes less critical as network depth increases.

5.2.2. Results Excluding Secret From Candidate Set

When the secret is excluded from the candidate set (9,999 candidates remaining), exact recovery is impossible. We instead evaluate whether class-level inference is feasible.

Evidence for class-recoverable quantization signatures. Figure 3 suggests that class recovery may be possible. The upper row shows 2D t-SNE projections of each test sample's per-layer quantization scales, colored by true class. Clustering becomes more pronounced as network depth increases. While the simple CNN shows weak separation, ResNet18 exhibits clearer class structure particularly for class 1, and ResNet50 shows the most distinct clustering separating most distinctly classes 0, 1, and 2. Although overlap remains even for ResNet50, the lower row demonstrates meaningful class correlation: for an average sample, over 50% of its 10 nearest neighbors (by quantization signature similarity) belong to the same class, well above the 10% random baseline. This indicates that per-layer quantization scales do carry class information in deep networks.

Attack performance and limitations. Despite this promising signal, our attack achieves limited success at class inference. We use optimized layer-diverse probes (to exclude poorly-performing random probes) and evaluate 50 runs per model. We test multiple ranking strategies: top-1 candidate selection, top-3 candidate selection, and clustering-based approaches where we cluster the top 100 candidates into 10 groups by scale profile similarity and select either the closest cluster on average or the cluster containing the top-3 candidates. None of these strategies significantly improve performance. For top-1 candidate accuracy, we achieve 16% for the simple CNN, 14% for ResNet18, and 10% for ResNet50, only marginally above the 10% random baseline, despite the strong class clustering visible in the scale profiles.

Why class inference fails. We hypothesize this failure stems from a fundamental information asymmetry. The clustering analysis in Figure 3 directly examines each image's per-layer quantization scale profile. However, during the attack, we cannot observe the secret's scale profile di-

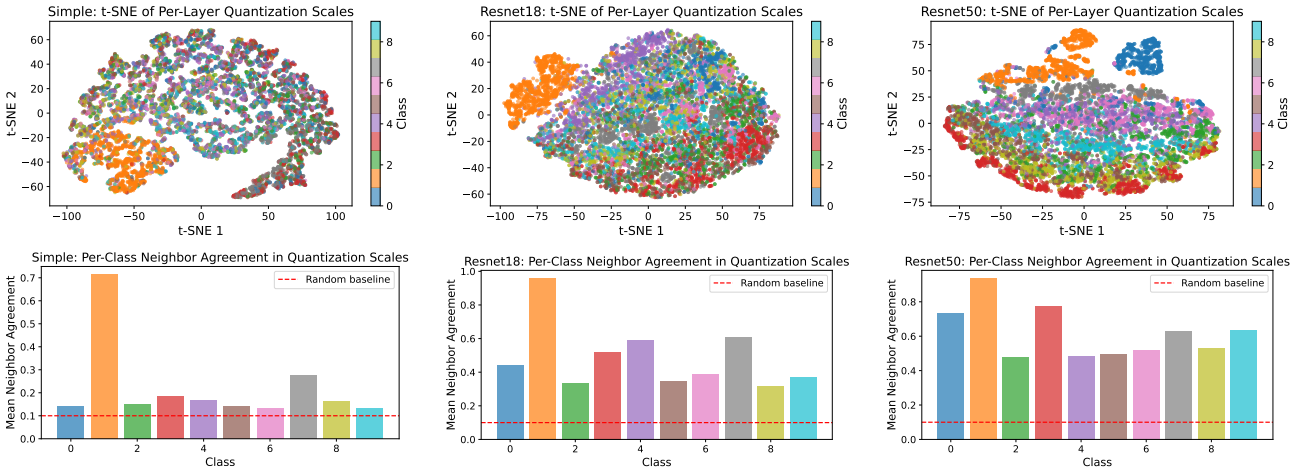

Figure 3. **From left to right:** Analysis of a simple 3-layer CNN, Resnet18, and Resnet50's per-layer dynamic quantization scales aggregated together. **Upper row:** 2D t-SNE projection of each test sample's per-layer quantization scales, colored by true class. Distinct clusters indicate that different classes induce separable quantization behavior across layers. **Lower row:** For each class, the average proportion of a sample's 10 nearest neighbors that belong to the same class, measured by similarity of per-layer quantization scales. Bars above the random baseline (red) indicate that samples with similar quantization profiles tend to share a class.

rectly. We only observe the difference in logit output it produces when batched with our adversarial probe. This indirect signal limits our ability to locate the correct cluster of candidates with similar scale profiles. Even when classes form distinct clusters in scale space, the logit-based side-channel may not reliably map to those clusters. Future work might explore alternative attack strategies that better exploit the class structure present in quantization signatures.

6. Discussion

Real-world deployment challenges: Non-determinism. Our attack relies on detecting small logit differences caused by cross-batch quantization effects, which raises the question of whether such signals survive the noise present in production deployments. Querying the same prompt 20 times with temperature 0 through OpenRouter reveals non-trivial variation in log probabilities across runs, even for identical inputs (see Figure 4 in Appendix C). Several factors may contribute to this non-determinism. First, hardware and software heterogeneity can introduce systematic variation through differences in how calculations are implemented (Zhang et al., 2025; Schlögl et al., 2023). Second, even when a user requests temperature zero, it is unclear whether the serving endpoint performs pure greedy decoding. While frameworks like vLLM explicitly disable dynamic sampling parameters (min-p, top-p, top-k) when temperature falls below a small epsilon (vLLM Contributors, 2024), providers routing requests through platforms like OpenRouter may apply their own sampling pipelines, custom logits processors, or model-specific generation configurations that deviate from these defaults. More broadly, the growing ecosystem of dynamic, context-aware sampling methods, including min-p truncation (Minh et al., 2025), entropy-based adaptive sampling (xjdr & doomslide, 2024), and adaptive attention temperature (Veličković et al., 2025), reflects a trend toward inference-time modifications that are increasingly difficult for external users to audit, making it uncertain whether any given API endpoint truly implements deterministic decoding even when explicitly requested.

7. Conclusion

We have shown that per-tensor dynamic activation quantization, a default in several widely deployed inference frameworks, introduces a cross-batch side channel that enables practical privacy attacks against co-located users: an adversary can recover LLM token sequences at 99.6–100% accuracy using only top-1 log probabilities, and can exactly identify a victim's input when it belongs to a known candidate set. The mitigation is already adopted by most LLM-serving frameworks for INT8: move to per-token dynamic quantization, which eliminates the cross-batch dependency while typically improving accuracy and throughput on modern GPUs. More broadly, our findings reinforce a recurring theme in recent work on batched inference: optimization choices that appear purely numerical, such as routing mechanisms, cache sharing policies, or quantization granularity, can become security boundaries once batches cross trust boundaries, and as serving infrastructure continues to optimize for throughput through shared state across users, the set of such boundaries should be audited explicitly rather than assumed to be benign.

References

Allal, L. B., Lozhkov, A., Bakouch, E., von Werra, L., and Wolf, T. Smollm - blazingly fast and remarkably powerful, 2024. URL https://huggingface.co/blog/smollm. Blog post.

Allal, L. B., Lozhkov, A., Bakouch, E., Blázquez, G. M., Penedo, G., Tunstall, L., Marafioti, A., Kydlíček, H., Lajarín, A. P., Srivastav, V., Lochner, J., Fahlgren, C., Nguyen, X.-S., Fourrier, C., Burtenshaw, B., Larcher, H., Zhao, H., Zakka, C., Morlon, M., Raffel, C., von Werra, L., and Wolf, T. Smollm2: When smol goes big – data-centric training of a small language model, 2025. URL https://arxiv.org/abs/2502.02737.

Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Black, S., Leo, G., Wang, P., Leahy, C., and Biderman, S. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021. URL https://doi.org/10.5281/zenodo.5297715. If you use this software, please cite it using these metadata.

Black, S., Biderman, S., Hallahan, E., Anthony, Q., Gao, L., Golding, L., He, H., Leahy, C., McDonell, K., Phang, J., et al. Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of BigScience Episode# 5–Workshop on Challenges & Perspectives in Creating Large Language Models*, pp. 95–136, 2022.

Clifford, E., Saravanan, A., Langford, H., Zhang, C., Zhao, Y., Mullins, R., Shumailov, I., and Hayes, J. Locking machine learning models into hardware. *arXiv preprint arXiv:2405.20990*, 2024.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35:30318–30332, 2022.

Egashira, K., Vero, M., Staab, R., He, J., and Vechev, M. Exploiting llm quantization. *Advances in Neural Information Processing Systems*, 37:41709–41732, 2024.

Eldan, R. and Li, Y. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Google. Post-training quantization. https://ai.google.dev/edge/litert/conversion/tensorflow/quantization/post_training_quantization, 2024. Accessed: 2025-02-04.

Hayes, J., Shumailov, I., and Yona, I. Buffer overflow in mixture of experts. *arXiv preprint arXiv:2402.05526*, 2024.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Jia, K. and Rinard, M. Exploiting verified neural networks via floating point numerical error. In *Static Analysis: 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17–19, 2021, Proceedings 28*, pp. 191–205. Springer, 2021.

Kelly, A. Faster dynamically quantized inference with xnnpack. https://blog.tensorflow.org/2024/04/faster-dynamically-quantized-inference-with-xnnpack.html, April 2024. Accessed: 2025-02-09.

Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

Küchler, N., Petrov, I., Grobler, C., and Shumailov, I. Architectural backdoors for within-batch data stealing and model inference manipulation. *arXiv preprint arXiv:2505.18323*, 2025.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Lecun, Y. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998. URL https://cir.nii.ac.jp/crid/1571417126193283840.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

Microsoft. Onnx runtime quantization documentation. https://onnxruntime.ai/docs/performance/model-optimizations/quantization.html, 2024. Accessed: 2025-02-04.

Minh, N. N., Baker, A., Neo, C., Roush, A. G., Kirsch, A., and Shwartz-Ziv, R. Turning up the heat: Min-p sampling for creative and coherent LLM outputs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=FBkpCyujtS.

Mishra, A., Pope, R., Neelam, S., Heinlein, D., Cvicek, V., Vasania, Z., and Hill-Khurana, J. Future leakage in block-quantized attention. https://matx.com/research/leaky_quantization, January 2026.

NVIDIA. Tensorrt-llm precision reference. https://nvidia.github.io/TensorRT-LLM/reference/precision.html, 2024. Accessed: 2025-02-04.

PyTorch Contributors. PerTensor — PyTorch documentation, 2025. URL https://docs.pytorch.org/docs/stable/generated/torch.ao.quantization.observer.PerTensor.html. Accessed: 2025-02-22.

PyTorch Team. Pytorch quantization documentation. https://pytorch.org/docs/stable/quantization.html, 2024. Accessed: 2025-02-04.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 3505–3506, 2020.

Red Hat AI and vLLM Project. LLM Compressor, 8 2024. URL https://github.com/vllm-project/llm-compressor.

Schlögl, A., Kupek, T., and Böhme, R. Forensicability of deep neural network inference pipelines. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2515–2519, 2021. doi: 10.1109/ICASSP39728.2021.9414301.

Schlögl, A., Kupek, T., and Böhme, R. iNNformant: Boundary samples as tell-tale watermarks. In *Workshop on Information Hiding and Multimedia Security (IH&MMSec)*, pp. 81–86. ACM, 2021.

Schlögl, A., Hofer, N., and Böhme, R. Causes and effects of unanticipated numerical deviations in neural network inference frameworks. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 56095–56107. Curran Associates, Inc., 2023.

SGLang Team. Sglang quantization documentation. https://docs.sglang.io/advanced_features/quantization.html, 2025. Accessed: 2025-02-09.

Veličković, P., Perivolaropoulos, C., Barbero, F., and Pascanu, R. Softmax is not enough (for sharp size generalisation). In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=S4JmmpnSPy.

vLLM Contributors. Sampling parameters, 2024. URL https://docs.vllm.ai/en/v0.6.4/dev/sampling_params.html.

vLLM Team. Fp8 quantization. https://docs.vllm.ai/en/latest/features/quantization/fp8/, 2024. Accessed: 2025-02-04.

Wu, G., Zhang, Z., Zhang, Y., Wang, W., Niu, J., Wu, Y., and Zhang, Y. I know what you asked: Prompt leakage via kv-cache sharing in multi-tenant llm serving. In *Proceedings of the 2025 Network and Distributed System Security (NDSS) Symposium. San Diego, CA, USA*, 2025.

Wu, H., Judd, P., Zhang, X., Isaev, M., and Micikevicius, P. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

xjdr and doomslide. Entropix: Entropy based sampling and parallel cot decoding, 2024. URL https://github.com/xjdr-alt/entropix.

Yao, Z., Yazdani Aminabadi, R., Zhang, M., Wu, X., Li, C., and He, Y. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in neural information processing systems*, 35: 27168–27183, 2022.

Yin, H., Mallya, A., Vahdat, A., Alvarez, J. M., Kautz, J., and Molchanov, P. See through gradients: Image batch recovery via gradinversion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16337–16346, 2021.

Yona, I., Shumailov, I., Hayes, J., and Carlini, N. Stealing user prompts from mixture of experts. *arXiv preprint arXiv:2410.22884*, 2024.

Zhang, C., Foerster, H., Mullins, R. D., Zhao, Y., and Shumailov, I. Hardware and software platform inference. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=kdmjVF1iDO.

Zheng, L., Yin, L., Xie, Z., Sun, C. L., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Zombori, D., Bánhelyi, B., Csendes, T., and Jelasity, M. Fooling a complete neural network verifier. In *International Conference on Learning Representations (ICLR)*, 2021.

A. Extended Framework Deployment Landscape

Table 4. Dynamic activation quantization configurations across major inference frameworks. Only methods that quantize activations *at runtime* are shown; weight-only methods (e.g., INT4 AWQ/GPTQ) leave activations in FP16/BF16, and static calibrated activation scales (e.g., TensorRT-LLM defaults, LiteRT full-integer quantization) use fixed scales from conversion time— neither creates runtime cross-batch information flow. **Default**: granularity obtained when enabling the configuration without additional flags; **Opt.**: requires selecting a specific flag or variant. Most LLM-serving frameworks implement per-token dynamic activation quantization for INT8; per-tensor dynamic quantization appears in FP8 online modes and in general-purpose frameworks (ONNX Runtime, PyTorch).

Framework	Configuration	Precision	Setting	Act. Gran.
*Per-tensor — **vulnerable to cross-batch leakage***				
vLLM (vLLM Team, 2024)	`--quantization fp8` (online)	W8A8 FP8	Default	Per-tensor
SGLang (SGLang Team, 2025)	`--torchao-config fp8dq-per_tensor`	W8A8 FP8	Opt.	Per-tensor
ONNX RT (Microsoft, 2024)	`quantize_dynamic()`	W8A8 INT8	Default	Per-tensor
PyTorch (PyTorch Contributors, 2025)	`torch.ao.quantization...PerTensor`	W8A8 FP8/INT8	Default	Per-tensor
*Per-token / per-row — **not vulnerable***				
SGLang (SGLang Team, 2025)	`--torchao-config fp8dq-per_row`	W8A8 FP8	Opt.	Per-token
SGLang (SGLang Team, 2025)	`--torchao-config int8dq`	W8A8 INT8	Default	Per-token
SGLang (SGLang Team, 2025)	`--quantization w8a8_fp8/int8`	W8A8 FP8/INT8	Default	Per-token
vLLM (Kwon et al., 2023)	W8A8 FP8/INT8 (llm-compressor)	W8A8 FP8/INT8	Default	Per-token
DeepSpeed (Yao et al., 2022)	ZeroQuant W8A8	W8A8 INT8	Default	Per-token

Table 4 surveys dynamic activation quantization configurations across major inference frameworks. Frameworks that use static calibrated activation scales (e.g., TensorRT-LLM's default `fp8` and `int8_sq` modes (NVIDIA, 2024), LiteRT's full-integer quantization (Google, 2024)) embed fixed scales into the model at conversion time and are excluded, as they do not create runtime cross-batch information flow. Among LLM-serving frameworks, per-token dynamic activation quantization is the standard for INT8: SGLang (Zheng et al., 2024), DeepSpeed-Inference (Yao et al., 2022), vLLM (Kwon et al., 2023), and optional dynamic mode of LiteRT (Kelly, 2024) all default to per-token granularity for their INT8 W8A8 modes.

The vulnerability surface lies primarily in FP8 online quantization, where per-tensor remains the default. In vLLM (vLLM Team, 2024), launching with `--quantization fp8` dynamically quantizes activations per-tensor at runtime; users must instead use the llm-compressor pipeline (Red Hat AI & vLLM Project, 2024) with `FP8_DYNAMIC` to obtain dynamic per-token FP8 scales. SGLang (SGLang Team, 2025) exposes the choice explicitly through its torchao integration: `fp8dq-per_tensor` and `fp8dq-per_row` select between the vulnerable and safe granularities respectively, while its `w8a8_fp8/int8` and `int8dq` modes default to per-token. Beyond LLM-serving, general-purpose frameworks also exhibit the vulnerability. ONNX Runtime's `quantize_dynamic()` (Microsoft, 2024) computes a single per-tensor activation scale at runtime. Similarly, PyTorch's quantization observer API defaults to per-tensor activation granularity (via `torch.ao.quantization.observer.PerTensor`); in recent versions, this path shares the same torchao backend used by SGLang's `fp8dq-per_tensor` mode. With our side-channel attack, we highlight that per-tensor dynamic activation quantization is not only harmful for accuracy but also introduces a concrete security vulnerability exploitable across co-located batch samples. In other words, it represents a worst-case scenario: it compromises user privacy while simultaneously degrading model performance.

B. Extended Threat Model

This appendix provides the full enumeration of adversary capabilities and their justifications, a condensed version appears in Section 4.1.

Adversary capabilities. The adversary possesses the following capabilities:

1. **Batch co-location:** The adversary can inject a data point a into a batch that is processed jointly with the victim's input s. For simplicity, we assume a batch size of 2, where one data point is from the adversary and the other from the victim.
2. **Output observation:** The adversary can observe the model's output (logits or predictions) corresponding to their own injected data points.

3. **Candidate set access:**
   - *LLM setting:* The adversary knows the vocabulary V.
   - *Classification setting:* The adversary has access to a public dataset from the same distribution as the victim's data point. We assume two settings, where either the dataset contains the victim's data point or does not contain the victim's data point.
4. **Model knowledge:** The adversary knows which model is being used and knows that dynamic quantization is employed during inference.
5. **Model access:**
   - *White-box (open-source):* The adversary can run the model locally.
     - LLM setting: Enables testing candidates with arbitrary KV cache states.
     - Classification setting: Enables an optimized adversarial probe.
   - *Black-box (closed-source):* The adversary queries via API only. This adds the assumption that the adversary is able to fill many batches with only their own data points when testing with adversary and candidate data points.
     - LLM setting: Requires greedy decoding assumption for reproducible adversarial sequences.
     - Classification setting: Limits to random adversarial probe selection.

Threat model justification. *Batch co-location* arises naturally in multi-tenant serving systems that batch concurrent requests for throughput optimization. For experiments in the paper we use a batch size of 2 to simplify analysis and clearly isolate quantization effects. In practice, an attacker for a batch size N can submit $N-1$ items that are the same, thus effectively reducing the setting to our 2-item batch setup. Larger batch sizes in production systems may add more noise to these quantization effects but would not eliminate the core vulnerability, since any co-located inputs can influence shared quantization parameters. *Output observation* is a realistic assumption, as APIs like OpenRouter expose parameters like top-k logprobs. We assume access to top-1 logprobs for the LLM case and the full logit vector for the classification case, as we only test on a 10-class problem. *Candidate set access* is inherent for LLMs as tokenizers are publicly available for all major models. For classification, we evaluate two scenarios: (1) the victim's input exists in a public dataset from the same distribution, representing a threat where the adversary identifies which specific datapoint the victim queried, and (2) the victim's input is excluded from the candidate set, where the adversary seeks similar inputs or infers the victim's class label. *Model knowledge* is reasonable as adversaries typically initiate queries and can infer model details from API behavior or documentation. For *Model access*, white-box attacks on open-source models (LLaMA, Mistral, Qwen) are highly practical given the prevalence of locally-runnable models and represent the strongest adversarial capability. Black-box attacks on closed APIs require additional assumptions (e.g., greedy decoding for LLMs, random probe selection for classification) but still constitute a meaningful threat model.

C. Non-Determinism in Production APIs

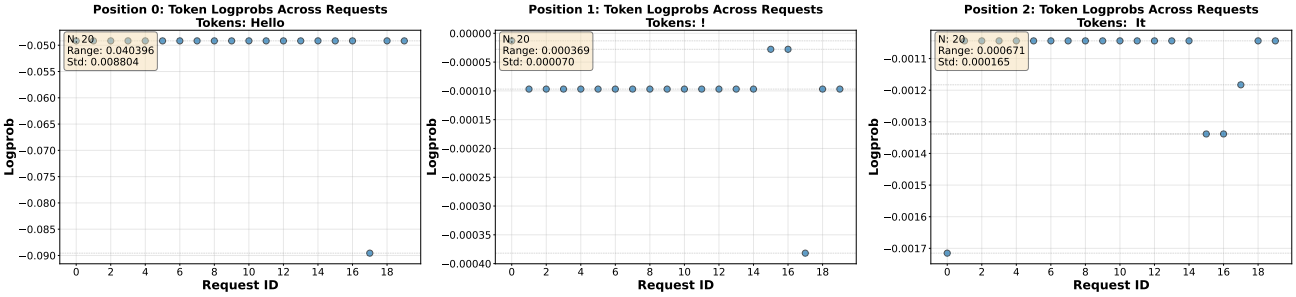

Figure 4. Llama log probability analysis of the first 3 output tokens after querying "Hello my name is" 20 times with temperature 0 on OpenRouter with the "Crusoe" provider. Despite deterministic settings, log probabilities vary across runs, with ranges of $\sim$0.04 at position 0 and $\sim 10^{-3}$ at positions 1 and 2.