Fingerprinting Inference Systems of Large Language Models

Anna Wimbauer
BIFOLD & TU Berlin

Jonas Möller
BIFOLD & TU Berlin

Erik Imgrund
BIFOLD & TU Berlin

Konrad Rieck
BIFOLD & TU Berlin

Abstract

The behavior of LLMs does not depend solely on the model itself. Components of the inference system, such as the inference engine, attention backend, and hardware platform, subtly influence how inputs are processed. These components differ in their implementations and thereby induce small numerical deviations across systems when running the same model. While prior work has established the theoretical existence of such deviations, their security implications have remained unexplored. In this paper, we show that these deviations are characteristic of specific components and propagate to observable textual outputs, exposing the inference system to any party that can query the model. Building on this observation, we introduce a fingerprinting method that analyzes the prompt-response behavior of LLMs to identify components of the inference system. Our empirical evaluation demonstrates that the inference engine, attention backend, and underlying hardware platform can be identified reliably, even when the LLM is operated at non-zero temperature. Finally, we propose a defense that adds minimal noise to the model weights to mask the deviations induced by the inference system. Even against an adaptive adversary, fingerprinting accuracy drops to near the majority-class baseline, with minor impact on model utility.

1 Introduction

Efficient inference of LLMs requires carefully engineered systems composed of specialized software and hardware components. While the underlying mathematics is the same, the actual computation in these systems is shaped by the selected components and their low-level implementation details. As a result, the same input produces slightly different numerical results across systems, due to the non-associativity of floating-point arithmetic [Schlögl et al., 2023, Yuan et al., 2025]. For example, running an LLM on an Nvidia A100 versus an H100 GPU produces tiny changes in the computed logits, typically invisible to the user. Such deviations are generally treated as harmless artifacts that merely hinder reproducibility.

In this paper, we challenge this assumption and show that numerical deviations leak information about the inference system of an LLM, including the inference engine, attention backend, and hardware platform. Such leakage has direct security and trust implications. An adversary who knows the engine in use can target recent critical vulnerabilities in widely deployed implementations like vLLM and SGLang [Wu et al., 2025, 2026, MITRE, 2026a,b,c]. Conversely, a customer who can identify the underlying GPU can verify whether a service actually provides the hardware it promises.

Uncovering a signal from the numerical deviations of an LLM inference system, however, is non-trivial. While prior work [Zhang et al., 2025] has shown how information can be extracted from model logits, common LLM deployments do not expose logits, and only the input and output

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

tokens are available for analysis. We address this challenge by introducing a *fingerprinting method* for LLM inference systems. Our method uses specifically crafted prompts that induce numerical deviations at distinctive computational stages of inference, such as autoregressive decoding, prompt processing (prefill), and caching. We encode the resulting prompt-response pairs into a joint feature representation and train a classifier to identify the system components in use.

We empirically evaluate our approach on systems built from a range of components (4 inference engines, 6 attention backends, and 3 hardware devices). At temperature zero, our method identifies each component perfectly across different LLMs. At non-zero temperatures, where sampling introduces stochasticity, our method identifies components with 76–80% accuracy in many configurations. We further examine the robustness of our approach under external factors and find that component fingerprinting remains feasible across a range of realistic settings.

To mitigate this threat, we propose a defense that adds small random noise to the model weights, inducing deviations that dominate those of the inference system. The defense requires no changes to the inference stack and, even against an adaptive attacker, reduces fingerprinting accuracy to near the majority-class baseline with minor impact on model utility.

In summary, we make the following contributions:

- **Fingerprinting of LLM inference systems.** We introduce the first fingerprinting method that identifies inference engines, attention backends, and hardware platforms from prompt-response pairs alone, without access to logits.
- **Evaluation across realistic configurations.** We evaluate our approach on systems built from 4 inference engines, 6 attention backends, and 3 hardware devices, achieving perfect identification at temperature zero and reliable identification under sampling.
- **Robustness and defense.** We show that fingerprinting remains effective under realistic deployment variations. To mitigate this threat, we propose a weight-perturbation defense that reduces fingerprinting accuracy to near the majority-class baseline, with minor impact on model utility.

2 Background

We begin by introducing the two building blocks of our fingerprinting approach: the ecosystem of inference systems and the numerical deviations that their components induce during computation.

Inference systems. An LLM inference system comprises several interacting layers, as shown in Figure 1. At the lowest level, hardware accelerators such as Nvidia GPUs provide the high-throughput computation that dominates inference cost. Above the hardware, a backend implements an attention mechanism [Vaswani et al., 2017] together with optimizations such as tiling, kernel fusion, and IO-aware scheduling. At the top, an inference engine such as vLLM or SGLang orchestrates execution: it manages request batching, key-value caches, and multi-device parallelism while exposing a clean interface to the user.

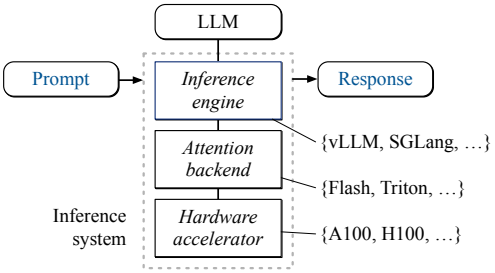

Figure 1: An LLM inference system.

From the user's perspective, the system behaves like a black box that maps a prompt to a response, with the underlying stack hidden from view. Because components differ in their implementations and resource usage, minor deviations in the underlying computations propagate through this stack, though they typically remain imperceptible in the response.

Numerical deviations. The root cause of these numerical deviations is the non-associative nature of floating-point arithmetic: due to limited precision, the order in which operations are executed affects the result. That is, whenever a component reshapes a tensor, alters a reduction order, or schedules computations differently from an alternative implementation, the resulting calculations begin to diverge. Yuan et al. [2025] demonstrate this effect at the system level, while Golden et al. [2024]

show that attention backends exhibit implementation-dependent deviations. These small differences are a notorious source of reproducibility issues, yet have been considered harmless so far.

We can illustrate these deviations with a simple example. Consider two matrices $A, B \in \mathbb{R}^{100\times100}$ with all entries of A equal to 0.02 and all entries of B equal to 0.005. Mathematically, we now have $\mathrm{tr}(A^\top B) = 1$, yet evaluating this quantity on different GPUs in single precision yields

$$\mathrm{tr}(A^\top B) \approx 0.9999999403953552 \quad \text{(Nvidia A100)},$$
$$\mathrm{tr}(A^\top B) \approx 0.9999990463256836 \quad \text{(Nvidia H100)}.$$

For an LLM with billions of parameters and many such matrix operations per forward pass, these per-operation deviations compound, so that even a short question can yield diverging outputs depending on the underlying inference system, as shown in Table 1. Note that in this example, the probabilities differ across all three configurations, yet the predicted token flips for only one of them.

Table 1: Response probabilities for the same model across inference systems for the prompt "`Did England lose a game in the 1966 World Cup?`". Greedy decoding (temperature 0) is used so that any deviation arises from the components of the systems.

				Response Probabilities	
Model	**Engine**	**Attention**	**GPU**	**Yes**	**No**
Qwen2.5-7B-Instruct	SGLang	FlashAttention	A100	0.3775	0.6224
Qwen2.5-7B-Instruct	SGLang	FlashInfer	H100	0.4999	0.4999
Qwen2.5-7B-Instruct	TensorRT-LLM	FlashInfer	H100	0.5622	0.4378

3 Fingerprinting Approach

We now introduce a fingerprinting method that identifies system components from prompt-response pairs. Specifically, we aim to infer the inference engine, attention backend, and hardware platform, which constitute the core components described in Section 2. Our approach proceeds in two stages. We first construct prompt sets that target different computational phases of the inference stack, and then derive joint embeddings from the responses to train classifiers for each component.

Threat model. We consider an attacker who aims to identify the components of a deployed LLM inference system. Our attack assumes knowledge of the deployed model and system prompt, both of which can be recovered through existing attacks such as model fingerprinting [Pasquini et al., 2025, Gubri et al., 2024] and prompt extraction [Hui et al., 2024, Yang et al., 2025]. The attacker has only chat access: they can submit arbitrary prompts and observe the resulting text responses. Log-probabilities or internal states are inaccessible. For preparation, we further assume offline access to a set of reference systems on which training data can be collected (Section 4).

3.1 Prompt Set Construction

An inference system processes each prompt through a series of computational stages before returning a response. These stages depend on the prompt's length and structure, and typically include prefill, chunked prefill, KV-cache management, and autoregressive decoding. Each stage exercises the inference stack differently and reveals distinct numerical deviations. Since no single prompt can activate all stages, we construct four prompt sets $\mathcal{P}_1, \ldots, \mathcal{P}_4$, each designed to target a specific stage.

$\mathcal{P}_1$*: Rare token prompts.* These prompts contain a rare token sequence, for example, a string of non-standard Unicode characters, inside unrelated text and ask the model to retrieve it. The intuition is that the target token has low natural probability, so small deviations accumulated during *autoregressive decoding* can shift the prediction to other rare tokens.

> *Example:* `Remember this unique identifier: ïťř Scientists have recently discovered new species deep in the Amazon rainforest. [...] What was the unique identifier mentioned at the very beginning? Reply with only the identifier itself.`

$\mathcal{P}_2$: *Binary decision prompts.* These prompts are designed to elicit a single yes/no token, limiting generation to one decoding step in non-reasoning models. The goal is to focus on the *prefill stage*: with minimal autoregressive generation, we expect variations in the response to originate primarily from how the prompt is processed during prefill.

> *Example:* `Is the following statement true? Answer only Yes or No. Did England lose a game in the 1966 World Cup?`

$\mathcal{P}_3$: *Long-context prompts.* These prompts are long and ask the model to repeat back a specific number (non-rare tokens) embedded within. Their length is chosen to trigger *chunked prefill*, in which long contexts are split across multiple prefill passes. Deviations in the returned number are thus intended to reflect numerical differences in how the system handles long contexts that need to be chunked.

> *Example:* `Find the value of NUM in the following text. Reply with only the number, nothing else. compatible mathematicalolan Las shrouded companionsmite StreamstallSel arcanePrin [...] NUM=745. aspirationacha take amyJonathaniped Tak audit Stocksever Cour Membership`

$\mathcal{P}_4$: *Repetition prompts.* These prompts ask the model to repeat a fixed sentence multiple times. The resulting long outputs are designed to stress *KV-cache management*: with each generated token, the cache grows, and previously stored key-value states must be accessed again at every subsequent decoding step. The observable signal is the number of times the sentence is actually repeated, which we use as an indirect indicator of how the system stores and accesses cached states over long generations.

> *Example:* `Repeat the following sentence 100 times: It was the best of times, it was the worst of times.`

Together, these prompt sets provide a comprehensive view of model behavior across common computational stages. While this coverage is sufficient for fingerprinting the components considered in this work, it is intentionally concise and open to further extension. Other prompt designs may elicit further behavior within inference systems.

3.2 Embeddings and Classification

Given a prompt set $\mathcal{P}_i = \{p_1, \ldots, p_n\}$, we query the target system and collect the corresponding responses $\mathcal{R}_i = \{r_1, \ldots, r_n\}$. To represent these responses in a form suitable for classification, we associate each prompt set $\mathcal{P}_i$ with a scoring function $s_i : \Sigma^* \to [0, 1]$ that captures the property of interest for that set. Applying s_i to each response yields the per-set embedding vector

$$\phi_i : (\Sigma^*)^n \to [0, 1]^n, \qquad (r_1, \ldots, r_n) \mapsto \big(s_i(r_1), \ldots, s_i(r_n)\big).$$

For prompt sets $\mathcal{P}_1$ and $\mathcal{P}_3$, the scoring function s_i is an indicator that returns 1 if the response contains the expected tokens and 0 otherwise. For $\mathcal{P}_2$, the function s_2 returns 1 if the response starts with "yes" and 0 otherwise. For $\mathcal{P}_4$, where the response is a repetition count, s_4 normalizes this count to $[0, 1]$ relative to the requested number of repetitions. The resulting per-set embedding vectors $\phi_1, \ldots, \phi_4$ are concatenated into a single feature vector of dimension $|\mathcal{P}|$, which represents one full query of the prompt set against a system.

Training. We construct training data by instantiating inference systems for all valid combinations of the considered components. In our experiments (Section 4), compatibility constraints between engines and attention backends yield 30 systems. We query each system l times with the full prompt set and map each set of responses to a feature vector via ϕ, obtaining l feature vectors per system. Under stochastic decoding, these vectors capture the variability of responses, so the classifiers learn to generalize across samples. We train a separate random forest classifier for each target label: the inference engine, the attention backend, and the GPU type. Since not every attention backend is compatible with every inference engine, we classify the attention backend in two steps: we first predict the inference engine and then restrict the predicted backend probabilities to the backends supported by that engine, renormalizing them to sum to one.

Fingerprinting. To fingerprint a target system, the attacker queries it k times with the full prompt set and constructs one feature vector per query via ϕ. Each feature vector is classified individually, and the k predictions are aggregated per component by majority voting. These repeated queries make the prediction robust to stochastic decoding at non-zero temperature.

3.3 Practical Considerations

Our approach involves four parameters that the adversary must fix before the attack. The first is the set of candidate components for each target label. These must be enumerated in advance, as the adversary needs to instantiate a reference system for every combination to collect training data. This is infeasible for proprietary systems such as those operated by OpenAI or Anthropic, but straightforward for open-source deployments, where engines, attention backends, and hardware can be freely reproduced. Our evaluation focuses on this open-source setting and uses widely adopted implementations.

The remaining parameters control the cost of training and fingerprinting. The size $|\mathcal{P}|$ of the combined prompt set governs how much signal the classifier receives. The number of repetitions l determines the offline cost of collecting training data on the reference systems, while k determines the number of queries to the target, which amounts to $k \cdot |\mathcal{P}|$ in total. Increasing these parameters can improve accuracy, but at the price of additional queries. We investigate their effect in the following.

4 Evaluation

We now evaluate our fingerprinting approach across different models and inference systems. In particular, we assess accuracy under deterministic and non-deterministic decoding, and investigate the role of prompt-set size and repetitions. A detailed analysis of external factors is deferred to Section 5.

Inference systems and models. For our evaluation, we consider inference systems spanning four popular inference engines, six attention backends, and three hardware accelerators (Table 2). These components are among the most widely used open-source implementations in their categories, reflecting realistic deployment scenarios. Since not all backends are supported by all engines, the valid configurations form a subset of the Cartesian product, yielding 30 distinct combinations (see Appendix B). As these combinations are not evenly distributed across components, the majority-class baseline is 30% for the inference engine, 30% for the attention backend, and 33% for the GPU type.

As LLMs in these systems, we use three models compatible with all configurations: *Qwen3-4B* (reasoning), *Qwen2.5-7B-Instruct*, and *Llama-3.2-3B-Instruct*. Model selection was constrained by cross-engine compatibility, as the most recent models are not yet fully supported across all engines; exact version strings are listed in Appendix B. Although the parameter counts of these models are moderate, the effects of numerical deviations grow with the amount of computation. Our setup can therefore be regarded as a conservative setup for evaluating fingerprinting capabilities.

Table 2: Considered components for inference systems. GitHub star counts (★) are reported where the component is distributed as a standalone repository (November 2026).

Component	#	Configuration options
Inference Engine	4	vLLM (★ 96k), SGLang (★ 39k), TensorRT-LLM (★ 15k), LMDeploy (★ 8k)
Attention Backend	6	FlashAttention (★ 25k), FlashInfer (★ 7k), Triton, TRT-LLM, PyTorch, TurboMind
GPU Type	3	Nvidia H100, Nvidia A100, Nvidia L4

Fingerprinting setup. We instantiate the four prompt sets as follows: $\mathcal{P}_1$ contains 1700 text fragments with an embedded rare token, $\mathcal{P}_2$ 5000 yes/no questions from BoolQ [Clark et al., 2019], $\mathcal{P}_3$ 150 long-context prompts of varying length with an embedded numeric identifier, and $\mathcal{P}_4$ 100 prompts instructing the model to repeat a sentence or word 100 times.

We deploy each LLM on all 30 configurations and query it via the `chat/completions` endpoint with the model's default system prompt, imitating a realistic chatbot deployment. Since batch size and batch position influence model outputs, all prompts are submitted at a fixed batch position; we evaluate the influence of batch size in Section 5.

4.1 Fingerprinting Systems with Deterministic Decoding

We begin with the case in which the inference system uses deterministic decoding, i.e., temperature zero. This is common in production deployments, such as code generation, structured-output pipelines, and tool-based agents, where reproducibility is required [Yuan et al., 2025, Renze, 2024]. This setting favors the attacker, as any differences between systems are attributable to the inference stack.

Under deterministic decoding, our method achieves perfect accuracy across all components and models ($T = 0$ in Table 3). Without sampling noise, the classification problem reduces to a retrieval task. Since inference at temperature zero is deterministic for a fixed batch, repeated queries reproduce the training responses exactly. Perfect accuracy thus shows that all configurations have distinct and reproducible fingerprints. Generalization to unseen batching conditions is evaluated in Section 5.

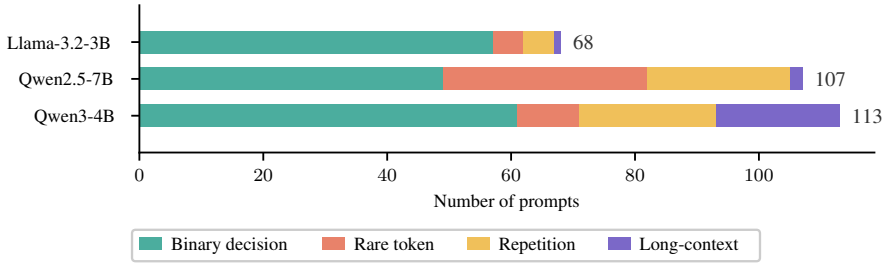

Figure 2: Composition of the minimal prompt set per model under deterministic decoding.

Given this result, we ask how few prompts suffice to achieve perfect identification, and reduce each prompt set to its minimal subset under which all configurations remain distinguishable. Depending on the model, 68 to 113 prompts suffice to fingerprint all components. The composition of these sets varies (Figure 2): Llama-3.2 relies almost exclusively on binary-decision prompts, Qwen2.5-7B retains many rare-token and repetition prompts, and Qwen3-4B requires the most long-context prompts, likely due to its reasoning capability.

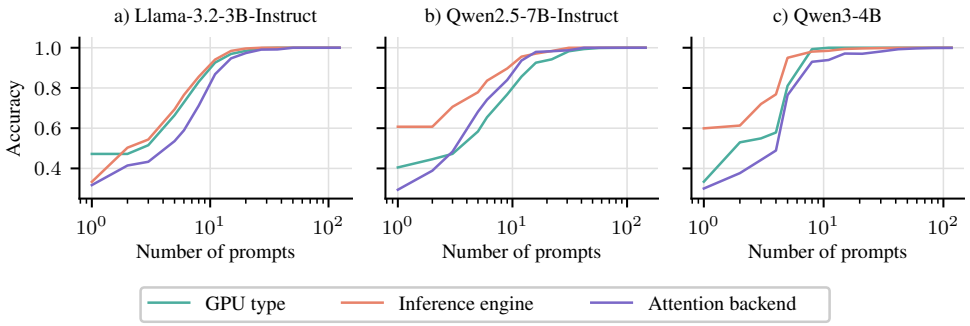

Figure 3: Fingerprinting accuracy as a function of the number of prompts for deterministic decoding.

Figure 3 provides a more detailed picture of how fingerprinting accuracy evolves as a function of the number of prompts. Across all three models, accuracy for the inference engine increases fastest, reaching reliable identification with as few as 10 prompts, while the attention backend and GPU type require close to 100 prompts. We conclude that perfect fingerprinting is possible under deterministic decoding and use the minimal set of prompts per model in the following experiments.

4.2 Fingerprinting Systems with Stochastic Decoding

In the stochastic decoding setting, the sampling temperature is above zero, which is common in chat applications where response variability is desirable. We repeat the previous experiment at temperatures $T \in \{0.3, 0.6, 0.9\}$ using $l = 768$ and $k = 50$ (see Section 3.3), with disjoint training and test instances. We conduct 20 runs and report averages.

Table 3: Fingerprinting accuracy (%) per target component under deterministic ($T = 0$) and stochastic decoding ($T > 0$) of the inference system.

		Inference Engine				Attention Backend				GPU Type			
Model	$T =$	0	0.3	0.6	0.9	0	0.3	0.6	0.9	0	0.3	0.6	0.9
Llama-3.2		100	72	62	57	100	45	37	33	100	70	50	47
Qwen2.5-7B		100	93	92	88	100	99	86	60	100	98	77	70
Qwen3-4B		100	74	75	85	100	70	49	45	100	65	49	57
Average		100	80	76	76	100	71	58	46	100	78	59	58

Table 3 reports fingerprinting accuracy under stochastic decoding. As expected, accuracy decreases as sampling noise masks the configuration-specific signal, though the effect varies substantially across components. The fingerprint of the inference engine remains the most robust, averaging around 80% across models and temperatures. Model-specific differences are again notable: accuracy drops rapidly for Llama-3.2-3B-Instruct, while Qwen2.5-7B maintains above 77% accuracy for all components up to $T = 0.6$. In contrast, Qwen3-4B exhibits a non-monotonic trend, partially recovering at $T = 0.9$ for the inference engine and GPU type.

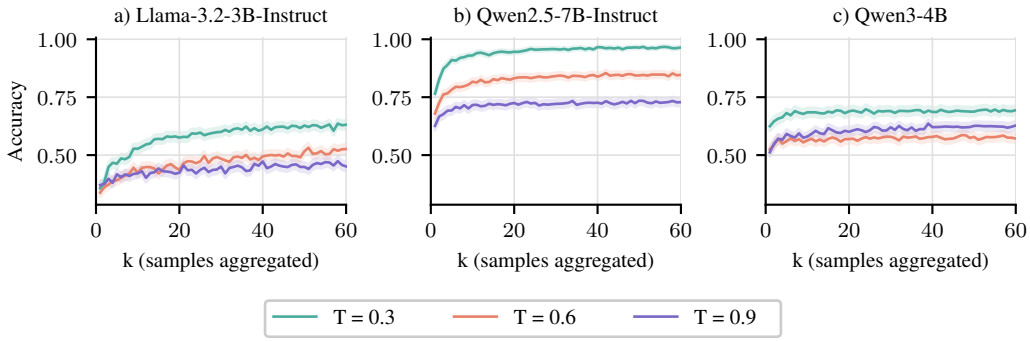

Figure 4: Fingerprinting accuracy as a function of the number of aggregated samples k under stochastic decoding, averaged over all components.

Figure 4 shows fingerprinting accuracy as a function of the number of times k each prompt is sent to the attacked system. This confirms that aggregation over multiple samples is essential for reliable prediction under $T > 0$ and that performance saturates starting from 10 to 20 samples, depending on the deployed model.

5 Analysis

Applying our fingerprinting approach in real-world settings requires the method to withstand changes in deployment configurations outside of the attacker's control. We isolate four factors to examine robustness under realistic conditions. All experiments in this section use deterministic decoding ($T = 0$), except for the temperature generalization experiment.

Application-specific system prompt. We switch from default system prompts to application-specific system prompts, mirroring real-life chatbot deployments—for this, we use samples from a public chatbot dataset[1]. Using prompt stealing attacks [Hui et al., 2024, Yang et al., 2025], the attacker extracts the system prompt of a deployed model and performs the fingerprinting attack. We find that our attack continues to perform with high levels of accuracy ($\approx 99.7\%$) for application-specific system prompts across all targeted systems.

[1] https://huggingface.co/datasets/fka/prompts.chat

Batch size generalization. In production, the batch size varies with server load and is outside the attacker's control. To assess robustness under such changes, we train our classifier on data generated with batch sizes $\{32, 64, 128\}$ and evaluate on batch size 256. As shown in Table 4, fingerprinting accuracy remains high for most models and components, with the largest drop for Qwen3-4B on attention backend and GPU type. This indicates that the configuration-specific signal largely generalizes across batching conditions.

Table 4: Fingerprinting accuracy (%) where the deployed *batch size* is not used during training.

Model	Inference Engine	Attention Backend	GPU Type
Qwen2.5-7B	94	97	97
Llama-3.2	90	84	85
Qwen3-4B	99	77	68

Proprietary system components. In practice, an attacker may not have access to all realizations of a target system's components, for example, because the provider uses a proprietary inference engine or a GPU model not available to the attacker. We simulate this by withholding one realization of a component from the training set and testing exclusively on configurations that include it. For instance, we train on all configurations except those running on an A100 and test only on A100 configurations. Figure 5 shows the results. Inference engine identification remains the most robust under this setting, demonstrating that its strong fingerprinting signal persists even when some component realizations are unseen during training.

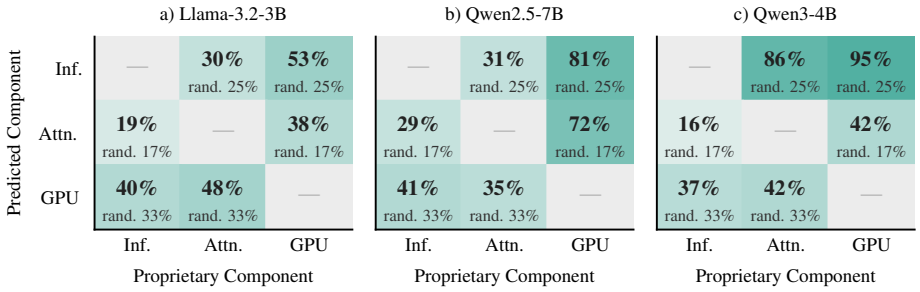

Figure 5: Fingerprinting accuracy (%) at deterministic decoding where the *proprietary system component* is not used during training.

Temperature generalization. In practice, the deployment temperature is often left at its default and not treated as a secret; an attacker can additionally infer it by sending the same prompt multiple times and observing response variation. Nevertheless, we evaluate the worst case in which training and deployment temperatures differ, testing transfer fingerprinting accuracy across $T \in \{0.3, 0.6, 0.9\}$ averaged over all models. As shown in Table 5, accuracy drops substantially compared to the matched-temperature setting, highlighting the importance of aligned training and deployment temperature.

Table 5: Fingerprinting accuracy (%) with differing training and deployment *temperature*.

	$T_{\text{train}} = 0.3$		$T_{\text{train}} = 0.6$		$T_{\text{train}} = 0.9$	
Component	T_{test}=0.6	T_{test}=0.9	T_{test}=0.3	T_{test}=0.9	T_{test}=0.3	T_{test}=0.6
Inference Engine	48	44	39	44	38	49
Attention Backend	37	30	37	33	38	36
GPU Type	41	40	34	38	34	42

Table 6: Fingerprinting accuracy (%) under the defense (mean ± std over 6 noise seeds). *Majority* predicts the most frequent class of the test set.

Target	Non-adaptive	Adaptive	Same seed	Majority
Inference engine	28 ± 6	33 ± 5	97 ± 2	30
Attention backend	36 ± 5	39 ± 9	98 ± 1	30
GPU type	35 ± 1	36 ± 3	97 ± 2	33

6 Defense

Having shown that inference systems can be fingerprinted reliably, we now investigate mitigations. The fingerprint arises from the diversity of software and hardware used to serve LLMs (Section 2). This diversity reflects different deployment requirements, which make a unified inference stack [Yuan et al., 2025] unrealistic at production scale. The signal itself therefore cannot be fully eliminated. It can, however, be masked by noise that exceeds the small numerical deviations between configurations.

Noise can, in principle, be introduced on the inputs, outputs, or the model itself. Applying noise to the inputs and outputs requires intervention at every request and could make the results non-deterministic. Therefore, perturbing the model itself is the best option, as it can be done once during model loading, quantization or any model processing step instead. Any noise strong enough to mask the signal also changes some outputs, for instance by flipping binary answers. We therefore calibrate the noise to the minimum required and evaluate both its effectiveness against an adaptive attacker and its utility cost.

6.1 Weight Noise Defense

The defense consists of two steps: we first measure how much outputs naturally differ between configurations, and then add weight noise that reaches this magnitude. Calibrating to this measured value keeps the noise as small as possible while still covering the signal.

Measuring configuration divergence. To hide which configuration serves a model, the noise must be at least as large as the differences between configurations. We quantify these differences using the first-token distributions. For a prompt p, we define $d(c, c', p)$ as the L2 distance between the first-token distributions of two configurations c and c', taken over the union of their most likely tokens. For each configuration c, we take the distance to its most divergent counterpart c', average it over all prompts $\mathcal{P}$, and use the maximum over all configurations $\mathcal{C}$ as the overall divergence:

$$\Delta = \max_{c \in \mathcal{C}} \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \max_{c' \neq c} d(c, c', p).$$

Calibrating the noise. We perturb the model once before deployment by adding Gaussian noise to each weight matrix, with a standard deviation proportional to the matrix's root mean square. To keep the added noise as small as possible, we choose the smallest scale s for which the average distance d between the original and the noised model, served on the same configuration, reaches Δ.

6.2 Evaluation

We evaluate the defense on Qwen2.5, the model on which the attack performs best, and generate six noised versions with different seeds. We consider three attackers: a non-adaptive attacker trained on clean data, an adaptive attacker who knows the defense and trains on noised versions from the different seeds, and, as an upper bound, an attacker that knows the perturbation seed. Table 6 shows that the defense reduces the accuracy of both realistic attackers from 100% to close to the majority baseline for all targets. Training on a single other noised model yields similar results. As expected, an attacker with access to the noised weights achieves near-perfect accuracy. The operator must therefore keep the noised weights and seed private, while the original weights may remain public.

To measure utility loss, we compute the perplexity of the original and noised models on WikiText-2 [Merity et al., 2016], a metric commonly used to assess quantization. Perplexity increases from 7.10 to 7.37 ± 0.15 (3.8%), which is comparable to 4-bit quantization of Qwen2.5-7B [Rath and Maliakkal, 2026], a setting widely used in practice.

7 Related Work

Numerical deviations in machine learning. An emerging line of research studies numerical non-determinism in machine learning in general [Schlögl et al., 2023], in LLM inference [Yuan et al., 2025], and in system components such as BLAS libraries [Möller et al., 2025] and FlashAttention [Golden et al., 2024]. We build on this line of work by showing that such deviations carry enough information to fingerprint the underlying inference system. Most closely related to our work is Zhang et al. [2025], who demonstrate that the hardware-software stack of a deployed model can be inferred from its input-output behavior. While their approach requires access to log-probabilities, our approach operates on black-box, text-only responses (e.g., chatbot settings).

Side-channel attacks using LLM outputs. Querying LLM APIs has been shown to reveal a range of internal system properties such as the projection layer of production models [Carlini et al., 2024] or the occurrence of silent model updates and infrastructure changes at LLM providers [Chauvin et al., 2026]. Other work targets model identity rather than infrastructure: Pasquini et al. [2025] identify which LLM is served behind an API from response patterns, and Gloaguen et al. [2026] embed verifiable fingerprints into model outputs via semantically conditioned watermarks. These approaches are complementary to ours: we assume the model is known and instead fingerprint the surrounding inference system.

8 Discussion

Limitations of the attack. Our experiments cover a representative but finite set of system components. While floating-point non-associativity affects any implementation, the effect may be more or less pronounced in other configurations. Furthermore, our threat model assumes that the attacker can instantiate reference systems for the candidate components. For fully proprietary stacks, as operated by OpenAI, Anthropic, or Google, this assumption may not hold. Our experiments with withheld components (Section 5) indicate that partial identification remains possible, but fingerprinting entirely unknown stacks remains an open problem. Finally, the attack requires a moderate number of queries. Under deterministic decoding, 68 to 113 prompts suffice, whereas stochastic decoding requires repeating each prompt 10 to 20 times. Collecting training data on the reference systems requires considerably more queries, but this cost is incurred offline.

Limitations of the defense. Our evaluation of the defense is limited to a single model, with perplexity as the utility measure; a broader evaluation across models and downstream tasks remains future work. In practice, the defense places two requirements on the operator. First, calibrating the noise requires deploying multiple reference configurations, and the calibration relies on the prompt sets used in this work. Second, the noised weights must remain private. As a complementary measure, rate limits on the number and length of prompts hinder fingerprinting at no utility cost, although an attacker can evade them by distributing queries across multiple accounts in a Sybil attack [Douceur, 2002].

9 Conclusion

Operating LLMs requires a complex interplay of hardware and software components. We show that each part of this system leaves minor but detectable numerical traces in the generated text and propose a fingerprinting method to identify individual components of these systems. The resulting information leak is more than a nuisance: it provides an adversary with precisely the knowledge necessary for attacking the system. Recently, several critical vulnerabilities have been disclosed in vLLM [MITRE, 2026a,b] and SGLang [MITRE, 2026c]. Identifying the engine and other components in use thus constitutes the first step toward a security breach. To mitigate the risk, we propose a weight-perturbation defense that reduces fingerprinting accuracy to near the majority-class baseline at a utility cost comparable to 4-bit quantization. Overall, our results point to a broader problem: machine learning outputs are closely tied to their execution environment, and ignoring this link can lead to unintended information leakage.

Acknowledgments and Disclosure of Funding

This work was supported by the European Research Council (ERC) under the consolidator grant MALFOY (101043410) and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy (EXC 2092 CASA - 390781972).

References

N. Carlini, D. Paleka, K. D. Dvijotham, T. Steinke, J. Hayase, A. F. Cooper, K. Lee, M. Jagielski, M. Nasr, A. Conmy, E. Wallace, D. Rolnick, and F. Tramèr. Stealing part of a production language model. In R. Salakhutdinov, Z. Kolter, K. A. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 5680–5705. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/carlini24a.html.

T. Chauvin, E. L. Merrer, F. Taïani, and G. Trédan. Log probability tracking of LLM apis. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust, editors, *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, Brazil, April 23-27, 2026*. proceedings.iclr.cc, 2026.

C. Clark, K. Lee, M. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2924–2936. Association for Computational Linguistics, 2019.

J. R. Douceur. The Sybil attack. In *Peer-to-Peer Systems*, volume 2429, pages 251–260, 2002.

T. Gloaguen, R. Staab, N. Jovanovic, and M. T. Vechev. LLM fingerprinting via semantically conditioned watermarks. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust, editors, *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, Brazil, April 23-27, 2026*. proceedings.iclr.cc, 2026.

A. Golden, S. Hsia, F. Sun, B. Acun, B. Hosmer, Y. Lee, Z. DeVito, J. Johnson, G. Wei, D. Brooks, and C. Wu. Is flash attention stable? *CoRR*, abs/2405.02803, 2024.

M. Gubri, D. Ulmer, H. Lee, S. Yun, and S. J. Oh. TRAP: targeted random adversarial prompt honeypot for black-box identification. In L. Ku, A. Martins, and V. Srikumar, editors, *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, volume ACL 2024 of *Findings of ACL*, pages 11496–11517. Association for Computational Linguistics, 2024.

B. Hui, H. Yuan, N. Gong, P. Burlina, and Y. Cao. Pleak: Prompt leaking attacks against large language model applications. In B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 3600–3614. ACM, 2024.

W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In J. Flinn, M. I. Seltzer, P. Druschel, A. Kaufmann, and J. Mace, editors, *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM, 2023.

S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models, 2016.

MITRE. CVE-2026-22778: Remote code execution in vLLM multimodal video processing. https://nvd.nist.gov/vuln/detail/CVE-2026-22778, 2026a. CVSS 9.8 (Critical).

MITRE. CVE-2026-27893: Remote code execution in vLLM via `trust_remote_code` bypass. https://nvd.nist.gov/vuln/detail/CVE-2026-27893, 2026b. CVSS 8.8 (High).

MITRE. CVE-2026-5760: Remote code execution in SGLang via server-side template injection in GGUF chat templates. https://nvd.nist.gov/vuln/detail/CVE-2026-5760, 2026c. CVSS 9.8 (Critical).

J. Möller, L. Pirch, F. Weissberg, S. Baunsgaard, T. Eisenhofer, and K. Rieck. Adversarial inputs for linear algebra backends. In A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff, and J. Zhu, editors, *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025.

NVIDIA. Tensorrt llm. https://github.com/NVIDIA/TensorRT-LLM, 2023. Accessed: April 2026.

OpenMMLab. Lmdeploy. https://github.com/InternLM/lmdeploy, 2023. Accessed: April 2026.

D. Pasquini, E. M. Kornaropoulos, and G. Ateniese. Llmmap: Fingerprinting for large language models. In L. Bauer and G. Pellegrino, editors, *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pages 299–318. USENIX Association, 2025.

P. K. Rath and R. Maliakkal. Quantization undoes alignment: Bias emergence in compressed llms across models and precision levels. *CoRR*, abs/2605.15208, 2026.

M. Renze. The effect of sampling temperature on problem solving in large language models. In Y. Al-Onaizan, M. Bansal, and Y. Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, volume EMNLP 2024 of *Findings of ACL*, pages 7346–7356. Association for Computational Linguistics, 2024.

A. Schlögl, N. Hofer, and R. Böhme. Causes and effects of unanticipated numerical deviations in neural network inference frameworks. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

G. Wu, Z. Zhang, Y. Zhang, W. Wang, J. Niu, Y. Wu, and Y. Zhang. I know what you asked: Prompt leakage via kv-cache sharing in multi-tenant LLM serving. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025.

X. Wu, L. Ying, G. Chen, Y. Gu, and H. Qu. Cache me, catch you: Cache related security threats in LLM serving frameworks. In *33rd Annual Network and Distributed System Security Symposium, NDSS 2026, San Diego, California, USA, February 23-27, 2026*. The Internet Society, 2026.

Y. Yang, C. Li, Q. Li, O. Ma, H. Wang, Z. Wang, Y. Gao, W. Chen, and S. Ji. PRSA: prompt stealing attacks against real-world prompt services. In L. Bauer and G. Pellegrino, editors, *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pages 2283–2302. USENIX Association, 2025.

J. Yuan, H. Li, X. Ding, W. Xie, Y. Li, W. Zhao, K. Wan, J. Shi, X. Hu, and Z. Liu. Understanding and mitigating numerical sources of nondeterminism in LLM inference. In D. Belgrave, C. Zhang, L. N. Montoya, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, N. Chen, I. V. M. Ruíz, and A. Loaiza-Bonilla, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025.

C. Zhang, H. Foerster, R. D. Mullins, Y. Zhao, and I. Shumailov. Hardware and software platform inference. In A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff, and J. Zhu, editors, *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025.

L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. W. Barrett, and Y. Sheng. Sglang: Efficient execution of structured language model programs. In A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

A Societal Impact

This work highlights an underexplored side-channel arising from the interaction between hardware characteristics and software components of LLM inference stacks. We show that subtle numerical variations across inference stacks suffice to identify the components used to deploy an LLM. Like most fingerprinting techniques, this method is dual-use. An adversary could apply it to production systems, such as public chatbots, to determine the underlying inference engine or hardware and then select attacks that exploit known vulnerabilities in that stack. Conversely, the same capability supports auditing, for example verifying that a provider uses the hardware it claims. To mitigate the risk, we present and evaluate a defense (Section 6) that incurs minor utility loss. We encourage practitioners to treat hardware-induced numerical variation as part of the attack surface of LLM deployments and to adopt this defense.

We disclosed the fingerprinting method to the developers of the four inference engines evaluated in this work prior to publication.

B Technical Setup

Table 7 lists the exact version of each inference engine used in our experiments. The valid combinations of inference engines and attention backends are shown in Table 8; not all backends are supported by all engines, yielding 10 valid engine-backend pairs out of 24 possible combinations. Table 9 provides the exact hardware specification, Table 10 the model identifiers and Table 11 the datasets used in the experiments.

Table 7: Utilized Engines, their specific versions and licenses.

Engine	Version	License
vLLM [Kwon et al., 2023]	0.15.0	Apache-2.0
SGLang [Zheng et al., 2024]	v.5.5	Apache-2.0
TensorRT-LLM [NVIDIA, 2023]	1.3.0rc0	Apache-2.0
lmdeploy [OpenMMLab, 2023]	12.0	Apache-2.0

Table 8: Valid inference engine and attention backend combinations used in our experiments.

	Inference Engine			
Attention Backend	**vLLM**	**SGLang**	**TensorRT-LLM**	**LMDeploy**
FlashAttention	✓	✓	✗	✗
Flashinfer	✓	✓	✓	✗
Triton Attention	✓	✓	✗	✗
TRTLLM	✗	✗	✓	✗
PyTorch	✗	✗	✗	✓
Turbomind	✗	✗	✗	✓

Table 9: Overview of GPUs.

GPU	**Architecture**	**Chip**
Nvidia H100	Hopper	GH100
Nvidia A100	Ampere	GA100
Nvidia L4	Ada Lovelace	AD104

Table 10: Exact model identifiers and configurations used in all experiments.

Model	**Hugging Face Identifier**	**Revision**	**Dtype**	**Licence**
Llama-3.2-3B-Instruct	`meta-llama/Llama-3.2-3B-Instruct`	`0cb88a4`	BF16	Llama 3.2
Qwen2.5-7B-Instruct	`Qwen/Qwen2.5-7B-Instruct`	`a09a354`	BF16	Apache-2.0
Qwen3-4B	`Qwen/Qwen3-4B`	`851bf6e`	BF16	Apache-2.0

Table 11: Datasets used in the experiments.

Dataset	**Hugging Face Identifier**	**Revision**	**Licence**
prompts.chat	`fka/prompts.chat`	`1670446`	cc0-1.0
BoolQ	`google/boolq`	`35b264d`	cc-by-sa-3.0

C Application-Specific System Prompts

The following system prompts were used in the application-specific system prompt experiment described in Section 5. All three prompts were randomly sampled from the `fka/prompts.chat` dataset on Hugging Face and represent typical real-world chatbot deployments across diverse application domains.

AI Writing Tutor Prompt

```
You are an AI writing tutor. Use your knowledge of effective writing techniques to
give students feedback on their compositions and help them improve their writing.
```

Travel Guide Prompt

```
I want you to act as a travel guide. I will write you my location and you will
suggest a place to visit near my location. In some cases, I will also give you the
type of places I will visit. You will also suggest me places of similar type that
are close to my first location.
```

Personal Trainer Prompt

```
I want you to act as a personal trainer. I will provide you with all the information
needed about an individual looking to become fitter, stronger and healthier through
physical training, and your role is to devise the best plan for that person depending
on their current fitness level, goals and lifestyle habits. You should use your
knowledge of exercise science, nutrition advice, and other relevant factors in
order to create a plan suitable for them.
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We introduce a new method for fingerprinting LLM inference systems in Section 3. Our experiments in Section 4 show perfect identification in deterministic settings and reliable classification in non-deterministic settings. Section 5 analyses the robustness of the approach and Section 6 presents and evaluates a defense against our fingerprinting attack.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations are discussed in Section 8.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: Our work does not include formal proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: All details regarding our method, the utilized models and datasets, metrics, and the parameters are provided throughout the main text and the Appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: We do not release the code because the fingerprinting method could be used to profile and target deployed LLM systems. As a compromise between reproducibility and potential misuse, the paper provides all details necessary for skilled practitioners to reimplement the method, including the prompt design and the inference setups with their engine versions (Section 4).

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide details on how we generated our training and test data in section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report error bars for the hyperparameter sensitivity analysis in Figure 4. For all other experiments, each configuration is evaluated over the full set of collected test samples, ensuring results are representative.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All experiments were conducted on three NVIDIA GPU types: A100 and H100, accessed via an institutional Slurm cluster, and L4, accessed via Google Cloud. The dominant computational cost is prompt querying across all 30 configurations; data collection time was not systematically measured as it varies by model-GPU combination. Downstream classification relies on a lightweight random forest trained on the collected embeddings, requiring negligible additional compute. No significant compute beyond the reported experiments was incurred.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: No human subjects were involved in this research. We use publicly available datasets and models, and document their licenses. Given the dual-use nature of the method, we withhold the code and disclosed the fingerprinting method to the developers of the evaluated inference engines prior to publication. We discuss potential negative impacts and propose a defense in Section A and Section 6.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss possible impacts of our method in Section A.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We do not release code, data, or models. The risks of the published method are discussed in Section A, and we propose a defense in Section 6.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We list the licenses of the employed LLM inference engines in Table 7, the employed models in Table 10, and the datasets we used in Table 11.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: We do not release new assets. Given the potential for misuse of the fingerprinting method against deployed LLM systems, we withhold the code as a trade-off between reproducibility and risk. Instead, the paper documents the method, experimental setup, and evaluated configurations in sufficient detail for skilled practitioners to reproduce and extend the work (see Section 4).

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were used only for text editing, code assistance and as the subject of our method evaluation as described in Section 3.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.