

ESPEON: Verifying the Predestined Execution of Deterministic LLM Inference

You-Liang Huang
Boston University
yliangh@bu.edu

Cheng Tan
Northeastern University
c.tan@northeastern.edu

Yi Pan
University of California, Berkeley
yi.pan@berkeley.edu

Yigong Hu
Boston University
yigongh@bu.edu

Abstract

Today, platforms of Model-as-a-Service (MaaS) provides access to various foundation models to users. Users trust that the platform indeed provides the exact service as requested. However, this trust may be misplaced for benign or malicious intentions, where results from providers are not exactly the same as what the user requested. To detect such violations, this paper introduces *Espeon*, a large language model (LLM) inference verification framework. It enables reproducible deterministic inference through batch-invariant kernels and includes different verification variants for flexible computation and storage budgets of verification. We evaluated *Espeon* with single-point manipulation cases and *Espeon*'s performance of serving and verification. *Espeon* can detect such cases, and under the full verification mode it can achieve a speed-up of $4.56\times$ compared to serving time.

CCS Concepts

• **Security and privacy** → **Distributed systems security**; • **Software and its engineering** → *Software verification and validation*; • **Computing methodologies** → *Neural networks*.

Keywords

large language models, inference verification, deterministic inference

ACM Reference Format:

You-Liang Huang, Yi Pan, Cheng Tan, and Yigong Hu. 2026. **ESPEON: Verifying the Predestined Execution of Deterministic LLM Inference**. In *Rigorous LLM Systems (RigoLLM '26)*, November 15–18, 2026, Hong Kong, China. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3858610.3859272>

1 Introduction

In recent years, we have witnessed the rapid growth of the Model-as-a-Service (MaaS) paradigm [1, 13, 31]. Platforms that aggregate and route requests to various foundation models [4, 29, 32] are commonplace, offering scalable access to powerful AI capabilities. This paradigm, however, relies on a fundamental assumption of trust: users receive output from a provider's service and must trust that

the inference computation was executed faithfully. This trust may be misplaced, as incorrect results can stem from silent hardware errors [8, 17], software bugs [24], or even deliberate cost-saving manipulations by a provider, such as serving a quantized [12, 26] or pruned [11] substitute in place of the advertised model, or skipping layers to reduce compute [9]. Such substitutions are difficult for users to notice as outputs still remain fluent and plausible [19].

Current methods for verifying remote LLM inference force an undesirable compromise between verification strength and cost. One category of solutions, including statistical watermarking [22, 23], model fingerprinting [33, 41], and the use of smaller proxy models or learned verifiers for validation [25, 36], is inherently statistical: the uncertainty lies in the detector itself, so a manipulated response may simply go unflagged no matter how often it is tested. These approaches offer only probabilistic assurances and are often brittle, degrading under paraphrase, decoding-parameter changes, or adaptive providers [20]. On the other end of the spectrum, full re-execution of the inference task provides a definitive guarantee but is prohibitively expensive as it doubles the computational workload; because floating-point results are not reproducible across batch sizes and kernel configurations [16], prior re-execution schemes must compare against a tolerance [30] or model the nondeterminism explicitly [21], leaving a margin in which an adversary can hide. Advanced cryptographic techniques like ZK-SNARKs [3, 15, 35] offer strong proof but introduce significant overhead, making them impractical for large models and latency-sensitive applications. Hardware-based attestation [28, 38, 40] and auditing frameworks [18] avoid much of this overhead, but shift trust to the hardware vendor rather than removing it, and remain exposed to known side-channel attacks [39]. Optimistic schemes [5, 37] defer verification to a dispute phase, but still require a mechanism for resolving disputes cheaply.

In this work, we argue that the recent emergence of batch-invariant [16], deterministic inference technology presents a breakthrough that fundamentally reshapes this landscape. By enabling bit-for-bit reproducible computations [2], this technology makes it possible to verify an LLM's output with both high efficiency and complete certainty. The key insight is that while a provider must generate a sequence token-by-token through a slow auto-regressive process that is bound by memory bandwidth rather than arithmetic throughput [6, 34], a verifier holding the final sequence can validate it in a single, highly parallelized forward pass. This batched approach allows the verifier to compute the logits for many tokens



This work is licensed under a Creative Commons Attribution 4.0 International License. *RigoLLM '26, Hong Kong, China*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-3090-0/2026/11

<https://doi.org/10.1145/3858610.3859272>

simultaneously, checking the provider’s claimed sampling decisions at a fraction of the original generation cost.

While this batched verification is far more efficient than auto-regressive re-execution, its cost still scales up with the total sequence length. As context windows continue to grow [7, 13], the expense of verifying inference on very long sequences can remain a significant burden, potentially limiting the feasibility of verification for many users. This challenge necessitates a more resource-aware approach that can provide strong guarantees without demanding a linear cost relationship to the sequence length.

To address this, we present *Espeon*, a framework that operationalizes our efficient verification method while introducing novel mechanisms to manage its cost. *Espeon* moves beyond the all-or-nothing approach of verifying an entire sequence. Instead, it empowers the verifier to select specific, high-value portions of the total inference computation for a rigorous and deterministic check. This design provides a direct and tunable trade-off between the desired scope of the integrity guarantee and the computational resources the verifier is willing to expend. To prevent the provider from anticipating which region will be checked, *Espeon* has the provider commit to its intermediate states [27] before the verifier draws the layer and token range to examine, following the commit-then-challenge structure standard in interactive proofs [10, 14].

Espeon achieves this fine-grained control through a verification system that is partitioned along two orthogonal dimensions: model depth and sequence position. First, layer-wise verification allows a verifier to check the integrity of intermediate hidden states at any specific layer, confirming that the computation is correct up to that point without processing the entire model. Second, token-wise verification allows the verifier to focus its resources on a specific contiguous part of the sequence, such as a critical prefix or a randomly chosen suffix. These two dimensions can be combined to create hybrid strategies, providing deterministic guarantees for precisely targeted computational regions. Verification is exact wherever it looks, and random only in where it looks. Deviations inside the checked scope are always caught; coverage of the rest scales with the verifier’s budget. In summary, our contributions are:

- We identify that batch-invariant per-region deterministic inference enables a new, highly efficient paradigm for LLM verification based on a single, parallel forward pass, providing a strong deterministic guarantee without the high cost of full re-execution or complex cryptographic methods.
- We present *Espeon*, a flexible verification framework that manages verification costs through a multi-dimensional system, allowing for fine-grained, budget-aware checks across both model layers and token sequences.
- We evaluate *Espeon*, where it can detect single-point attack cases. *Espeon*’s serving overhead is only about 6%~9%, and the full verification can achieve a speed-up of 4.56×

2 Related Work

To verify the correctness of LLM inference results produced by untrusted providers, many efforts have been devoted to inference verification. LLM fingerprinting [33] is a representative approach for discriminating the identity of the model serving a request. It crafts a carefully designed set of probes to characterize the model’s

behavioral patterns and applies statistical or machine-learning techniques to infer the underlying model type and version with a confidence score. This statistics-based paradigm broadly characterizes a class of existing verification approaches. Other works, such as DiFR [21], TOPLOC [30], and SVIP [36], employ different forms of intermediate evidence, consistency checks, or statistical verification to assess whether the returned inference results are consistent with the expected execution. Although these methods achieve high detection accuracy in empirical evaluations, their guarantees are generally oriented toward identifying coarse-grained deviations or statistically significant inconsistencies. Consequently, it remains challenging to reliably detect sparse and fine-grained corruptions—for example, an adversarial provider modifying only a single token in one sequence among a large number of otherwise correct inference results. In addition to statistics-based approaches above, there are other cryptography-based verification and Trusted Execution Environment (TEE)-based verification. zkLLM [35] designs a zero-knowledge proof mechanism for LLM to enable the validation of LLM inference integrity. It can give the strongest guarantee for the verification, but it is infeasible to be deployed in real production due to exorbitant computing overhead. As for the TEE-based method, AFTUNE [18], its limitation lies in the TEE hardware requirement, and TEE itself cannot accommodate large-scale verification. In summary, to the best of our knowledge, no existing work can achieve both a strong guarantee and good performance at the same time.

3 Design

In this section, we describe the design of *Espeon*. There are two parties in *Espeon*’s defined scenario, a *provider* and a *verifier*, in which the former is responsible for providing serving services and passing information recorded during serving to the latter for verification. We first elaborate on the verification mechanism in *Espeon*, followed by its requirement and what actions *Espeon* takes to satisfy the requirement. Then, we introduce the tunable verification strategy in *Espeon*, where users can verify their desired scope at their own expense. In the end, we depict how *provider* interacts with *verifier* in the framework of *Espeon*.

3.1 Predesigned Generation Results

LLMs are auto-regressive models, and its inference works in an auto-regressive way, where the decoding stage, after the prefill stage, produces one token per step. The token newly produced by LLMs depends on older tokens (*i.e.*, prompts and previously generated tokens), as well as request’s sampling parameters and the current behavior of pseudorandom number generator (PRNG). When the computation is deterministic, given a request that has preset sampling parameters and PRNG seed, the generation results are deterministic. Here is our demonstration of the deterministic inference process.

Given a large language model M and an input sequence $S_{0:i}$, The model forward can be represented as follows.

$$S_{0:i} = (t_0, t_1, \dots, t_{i-1}) \quad (1)$$

$$Z_{0:i} = (z_0, z_1, \dots, z_{i-1}) = M(S_{0:i}) \quad (2)$$

where z_i is the resulting logit of the input token t_i . When sampling the next token, the behavior of PRNG and logit contribute to the decision. The process is abstracted as follows.

$$t_i = S(z_{i-1} | \text{PRNG}_{s,p}) \quad (3)$$

where S is the sampler, $\text{PRNG}_{s,p}$ denotes the behavior of PRNG at step p under given seed s . As we noticed, at any random step during decoding, the next token is deterministic upon previous tokens and current internal state of PRNG.

Therefore, we can assert that under deterministic inference, at the moment when a request arrives at *provider*, the generation result of any request is not yet known, but already predestined. The decode stage in *provider* is just manifesting the predestined trajectory token by token. The generated tokens can be seen as a deterministic state transition chain. Intuitively, for *verifier*, all it needs to do is check whether the generated tokens produced by *provider* match the deterministic state transition chain. Any deviation from the trajectory can be considered as a failure of verification.

This insight inspires us to verify the sequences by rigorously replicating the LLM's token generation process. However, replaying the LLM's decode can cause an exorbitant verification cost, making it almost impractical for *Espeon* to be put into practice. One of the goals of *Espeon* is to be able to use a limited resource to verify as many requests as possible. To achieve this goal, *Espeon* doesn't simply replay LLM's decode but leverages one inherent characteristic of LLM's computation to make the best use of limited resources.

Considering a request with input prompt length of m , output sequence length of n . In *provider*, the request goes through decoding, and tokens are generated iteratively, we have generated tokens:

$$\begin{aligned} S_{m:m+n} &= (t_m, \dots, t_{m+n-1}) \\ &= (S(z_{m-1} | \text{PRNG}_{s,0}), \dots, S(z_{m+n-2} | \text{PRNG}_{s,n-1})) \end{aligned} \quad (4)$$

When the request comes to *verifier* after the generation, *verifier* can naturally see all tokens in the request. Now, *verifier* only needs one prefill operation to be able to compute the corresponding logits of all tokens in parallel, i.e., $Z_{0:m+n} = (z_0, z_1, \dots, z_{m+n-1}) = \mathbf{M}(S_{0:m+n})$. Since PRNG's behavior is deterministic, *verifier* can precisely replicate sampling decisions by aligning PRNG's seed and internal states, along with derived logits $Z_{m-1:m+n-1}$. If *verifier* finds any mismatch between reproduced sampling decisions and tokens, the verification fails.

However, as serving and verification happen in different places, floating-point drift caused by different runtime environment can alter the decision of the sampler (i.e., disturbance of the decision boundary surpasses the margin). Therefore, we need to make the deterministic inference in *provider* reproducible in *verifier*. In the next subsection, we will describe the actions *Espeon* takes to satisfy this requirement.

3.2 Reproducible Deterministic Inference

If a *verifier* wants to check whether a request is processed as *provider* claims, reproducible deterministic computation is a must. To facilitate this, *Espeon* requires *verifier* to have at least one instance that has an identical accelerator hardware setup to *provider*. Furthermore, *Espeon* adopts containerization technique to eliminate non-determinism that comes from different system environment.

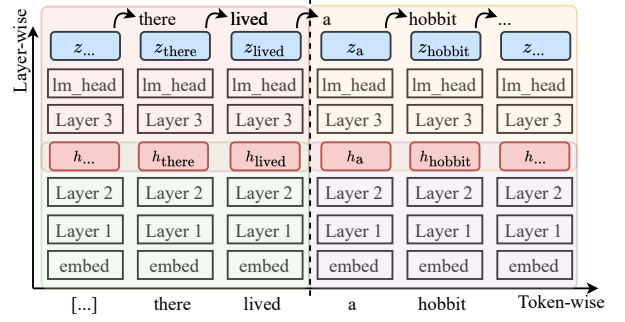


Figure 1: A high-level overview of verification methods in *Espeon*. Blocks of different colors indicate different parts of inference is verified.

Both *provider* and *verifier* run the same official image of a certain inference system. *Espeon* only applies hooks with minimal modification on non-critical path to adapt the same image to different roles, i.e., *provider* and *verifier*. Additionally, to eliminate numerical non-determinism in kernel's shape-specific optimization, *Espeon* replaces non-deterministic kernels with batch-invariant kernels [16]. With these actions, *Espeon* can enable reproducible floating-point computation across *provider* and *verifier*. In addition to computation, sampling is equally important for reproducible deterministic LLM inference. When *provider* processes requests, it records all sampling parameters (e.g., PRNG seed and temperature) and passes them to *verifier*. In *verifier*, with identical logits and sampling parameters given, *Espeon* can replay sampler's behaviors to reproduce the sampling results.

3.3 Verification

The verification described in §3.1 uses a complete sequence and a full model forward to get all logits, followed by sampling replication. It provides the strongest guarantee for the inference integrity, but it is computationally expensive. One important goal of *Espeon* is to let *verifier* be able to verify as many requests as possible. To achieve this goal, *Espeon* introduces multiple verification variants that verify requests on different parts of the model or token sequence. In this subsection, we will introduce all verification methods and how *Espeon* combines them under a unified verification framework.

The difference in verification variants in *Espeon* is the area of LLM inference that the verification tries to cover. We demonstrate such differences in Fig. 1. As shown in the figure, we can split LLM inference in two dimensions, layers and tokens. For the layer dimension, we can just reproduce a part of LLM inference in *verifier* and compare the reproduced output with outputs from *provider*. For example, the areas marked in different colors in Fig. 1 indicate the verification covers only a part of model forward. If the LLM inference of *verifier* ends after the last layer, i.e., *lm_head*, the verification criterion is to check if the sampling process can be reproduced, i.e., the red area and orange area. If the LLM inference in *verifier* ends in the middle of model forward, the criterion is to check if *verifier*'s hidden states bit-wise match with the hidden states transferred from *provider*, i.e., the green and purple area.

For the token dimension, a sequence can be split into two parts, prefix and suffix. To save compute, *Espeon* can just verify one part of

it, *i.e.*, areas split by dashed line in Fig. 1. If we choose to verify the prefix, the model forward starts with the token or hidden states, and no further actions need to be taken. However, if we decide to verify the suffix (area marked in orange and purple), the tokens or hidden states we have are insufficient to complete the model forward. This is because the attention calculation requires the presence of KV cache of previous tokens. Therefore, when verifying a suffix, *Espeon* requires the transfer of KV cache from *provider* to *verifier*. It trades costs of storage and transfer for the save of compute.

These two dimensions in verification are orthogonal and *Espeon* can mix them together into a hybrid and more fine-grained verification, such as performing a full model forward of the prefix, and a small part of the model forward of the suffix. Naturally, different variants offer guarantee of different strengths, and have different computation, storage, and transfer costs. Users can customize verification strategies according to their compute and storage budgets. In the next subsection, we will describe how *provider* interacts with *verifier* under the framework of *Espeon*.

3.4 Protocol Between Provider and Verifier

Espeon starts with the serving. To prevent *provider* from knowing verification strategies in advance, during the serving, *provider* records all information that *verifier* may need and makes a summary through hashing, then *provider* passes sequence’s metadata (*e.g.*, the token sequence and sampling parameters) and the data summary to *verifier*. All relevant data are transferred on *verifier*’s demand, and then first verified against the summary.

When *verifier* performs verification, it first chooses the verification strategy according to the compute and storage budget. Here cost calculation is straightforward, as now storage cost of a sequence is fixed and compute cost can be easily derived from the coverage of the prefill. After *verifier* finalizes the strategy, it can fetch relevant data from *provider* and complete the verification.

3.5 Implementation

We implement *Espeon* based on vLLM v0.27.0 and current prototype implementation only supports full verification. We enforce batch invariant kernels enabling to ensure deterministic inference. As for the reproducibility, we use two servers that have identical accelerators to act as *provider* and *verifier*. Details of setup can be found in the next section §4.

4 Evaluation

In this section, we evaluate *Espeon* in terms of two aspects. First, we evaluate *Espeon*’s effectiveness on single-point manipulation cases that previous deep learning-based work cannot detect. Second, we evaluate the serving and verification performance of *Espeon*.

The foundation model we use in evaluation is Meta-LLaMA-3-70B, with precision of fp16 and 80 decoder blocks in total. *Provider* has 4 NVIDIA H100 GPUs, and for *verifier* we use another server that also has 4 NVIDIA H100 GPUs to match parallelism.

4.1 Manipulation Detection

To evaluate the effectiveness of *Espeon*, we carefully designed two single-point attack cases, in these cases manipulation only happens

in one place. For the main evaluation, *Espeon* enforces full verification, executing a complete model forward from tokens and then checking if the sampling decisions match corresponding tokens. Additionally, we also analyze whether the verification variants can detect the attack. Here are two cases in our evaluation:

Output manipulation: In this case, at a random step amid decoding, *provider* ignores sampler’s decision and chooses another semantic-similar token as the next token, *e.g.*, replacing *cola* with *pepsi*, followed by completely normal serving of the rest part. Existing works like Token-DiFR [21] cannot detect such a case for following two reasons. First, the single-point error is statistically insignificant, especially in a long sequence. Second, the numeric error introduced by non-deterministic runtime environment can disturb the decision boundary, making the manipulated token seem absolutely normal. Such output manipulation can be detected by *Espeon*. Our *Espeon* has reproducible sampler behaviors by eliminating the non-determinism in LLM inference. When using full verification, *Espeon* can easily identify the inconsistency between the sampler’s decision and the manipulated token.

Activation Steering: In this case, at a random step amid decoding, *provider* steers the activation in order to alter the logits for manipulating the next token. For this case, existing works do not have strong guarantee on detection. There are many reasons for such failures, such as manipulation escapes from detection coverage [30] or steering is statistically insignificant [21, 36]. Our insight here is that these deep learning-based methods are unlikely to be able to detect single-point activation steering as there could be too much noise introduced to their calculation and how to decide an appropriate error threshold is heavily empirical. But for our *Espeon*, any steering that has impact on the next token to be generated or happened between the input and the check can be detected.

Settings	Throughput	TTFT	ITL	E2E Latency	Total
Default vLLM	1028 token/s	0.21 sec	18.1 ms	8.7 sec	72.7 sec
Deterministic vLLM	(↓61%) 398	(↑117×) 24.56	(↑4×) 72.9	(↑6.7×) 58.2	(↑2.6×) 185.5
w/ <i>Espeon</i>	(↓6%) 375	(↑9%) 26.85	(↑6%) 77.4	(↑6%) 61.8	(↑4%) 193.0

Table 1: *Provider*’s serving performance.

4.2 System Performance

Espeon includes a *provider* and a *verifier*. In this subsection, we evaluate both *provider*’s serving performance and also *verifier*’s verification performance. We randomly generate 128 requests with average prompt length of 112 tokens and maximal output length to be 1024. For serving, all these requests arrive at *provider* under a request rate of 2 requests per second. For each engine step, *provider*’s compute budget is 2048 tokens for continuous batching with 32 concurrent sequences in maximum. *Provider* is required to dump the hidden states at layer 40 and all KV cache. This is for aligning *provider* with *Espeon*’s design. Serving performance is reported in Table 1. When enabling *Espeon*, there is only about 6%~9% overhead regarding throughput and latency, *i.e.*, Time-to-first-token (TTFT), Inter-token Latency (ITL) and End-to-end (E2E) Latency. This is because in our current implementation the data dumping in *Espeon* is already well overlapped with the computation. With that being said, we can notice that vLLM with batch-invariant kernel enabling (*i.e.*, Deterministic vLLM) already suffers from severe performance degradation. Such significant performance degradation reveals a

huge gap between our proof-of-concept and production. How to mitigate this gap will be one goal of our future work.

For *verifier*, the vLLM engine arguments keep the same as *provider* and it uses the full-verification strategy, which starts with token ids and ends with sampling replication, to verify all sequences generated from *provider*. The performance is reported in Table 2. As we can notice, *verifier* achieves a speed-up of 4.56× regarding end-to-end verification time compared to the serving time. In which, sampling replication spends more time than model forward, making up 61.9% of the total. The exorbitant consumption comes from serial sampling replication that only runs on one accelerator. Even so, *verifier* can still achieve good performance, it has huge potential to improve performance further.

#Tokens	Forward (s)	Sampling (s)	Total (s)	Speed-up	Throughput	Coverage
72,901	16.13	26.16	42.29	4.56×	1,723	100%

Table 2: *Verifier's* verification performance.

5 Discussion & Conclusion

Today's Model-as-a-Service paradigm now faces emergent threats of user's trust being misplaced. Existing methods of low verification cost lack the capabilities of detecting single-point manipulation whereas methods with strongest guarantee are infeasible to be put to production for their exorbitant cost. Our paper introduced *Espeon*, an LLM inference verification framework that tries to reach a good balance between cost and strength. It leverages batch-invariant kernel to enable reproducible deterministic LLM inference, where *verifier* verifies *provider's* outputs through sampling replication and bit-to-bit checking. Our evaluation demonstrates that *Espeon* can detect single-point manipulation. However, *provider* exhibits exorbitant overhead when enforcing reproducible deterministic inference, which makes our *Espeon* infeasible for production. With that being said, we believe *provider's* performance degradation can be mitigated through a variety of optimizations and improvements regarding infrastructure. And *verifier's* performance can also be further improved. Upon promising performance improvement, our *Espeon* shows great potential for making LLM inference more reliable and making reliable LLM inference more feasible.

References

- [1] Anthropic. 2024. The Claude 3 Model Family: Opus, Sonnet, Haiku. https://www-cdn.anthropic.com/de8ba9b01e9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.
- [2] Naci Cankaya. 2026. Bit-Exact AI Inference Verification Without Performance Tradeoffs. In *ICML Workshop on Technical AI Governance Research*.
- [3] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. 2024. ZKML: An Optimizing System for ML Inference in Zero-Knowledge Proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys)*, 560–574.
- [4] Lingjiao Chen, Matei Zaharia, and James Zou. 2024. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *Transactions on Machine Learning Research* (2024).
- [5] KD Conway, Cathie So, Xiaohang Yu, and Kartin Wong. 2024. opML: Optimistic Machine Learning on Blockchain. *arXiv preprint arXiv:2401.17555* (2024).
- [6] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems*, Vol. 35, 16344–16359.
- [7] Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. 2024. LongRoPE: Extending LLM Context Window Beyond 2 Million Tokens. In *Proceedings of the 41st International Conference on Machine Learning*, Vol. 235, PMLR, 11091–11104.
- [8] Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, and Sriram Sankar. 2021. Silent Data Corruptions at Scale. *arXiv preprint arXiv:2102.11245* (2021).
- [9] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A. Aly, Beidi Chen, and Carole-Jean Wu. 2024. LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 12622–12642.
- [10] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology — CRYPTO '86*, Springer, 186–194.
- [11] Elias Frantar and Dan Alistarh. 2023. SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot. In *Proceedings of the 40th International Conference on Machine Learning*, Vol. 202, PMLR, 10323–10337.
- [12] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. *arXiv preprint arXiv:2210.17323* (2022).
- [13] Gemini Team, Google. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530* (2024).
- [14] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2015. Delegating Computation: Interactive Proofs for Muggles. *J. ACM* 62, 4, Article 27 (2015), 64 pages.
- [15] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology — EUROCRYPT 2016*, Springer, 305–326.
- [16] Horace He and Thinking Machines Lab. 2025. Defeating Nondeterminism in LLM Inference. *Thinking Machines Lab: Connectionism* (2025). doi:10.64434/tml.20250910 <https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/>.
- [17] Peter H. Hochschild, Paul Turner, Jeffrey C. Mogul, Rama K. Govindaraju, Parthasarathy Ranganathan, David E. Culler, and Amin Vahdat. 2021. Cores That Don't Count. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*, 9–16.
- [18] Heng Jin, Chaoyu Zhang, Hexuan Yu, Shanghao Shi, Ning Zhang, Y. Thomas Hou, and Wenjing Lou. 2026. Trusting What You Cannot See: Auditable Fine-Tuning and Inference for Proprietary AI. In *35th USENIX Security Symposium (USENIX Security 26)*, USENIX Association.
- [19] Renren Jin, Jiangcun Du, Wuwei Huang, Wei Liu, Jian Luan, Bin Wang, and Deyi Xiong. 2024. A Comprehensive Evaluation of Quantization Strategies for Large Language Models. In *Findings of the Association for Computational Linguistics: ACL 2024*, Association for Computational Linguistics, 12186–12215.
- [20] Nikola Jovanović, Robin Staab, and Martin Vechev. 2024. Watermark Stealing in Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning*, Vol. 235, PMLR, 22570–22593.
- [21] Adam Karvonen, Daniel Reuter, Roy Rinberg, Luke Marks, Adrià Garriga-Alonso, and Keri Warr. 2026. DiFR: Inference Verification Despite Nondeterminism. In *ICML Workshop on Technical AI Governance Research*.
- [22] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A Watermark for Large Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, Vol. 202, PMLR, 17061–17084.
- [23] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Manli Shu, Khalid Saifullah, Kezhi Kong, Kasun Fernando, Aniruddha Saha, Micah Goldblum, and Tom Goldstein. 2024. On the Reliability of Watermarks for Large Language Models. In *International Conference on Learning Representations*.
- [24] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 611–626.
- [25] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast Inference from Transformers via Speculative Decoding. In *Proceedings of the 40th International Conference on Machine Learning*, Vol. 202, PMLR, 19274–19286.
- [26] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*, Vol. 6, 87–100.
- [27] Ralph C. Merkle. 1988. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology — CRYPTO '87*, Springer, 369–378.
- [28] NVIDIA. 2023. Confidential Compute on NVIDIA Hopper H100. Whitepaper WP-11459-001 v1.0. <https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/HCC-Whitepaper-v1.0.pdf>.
- [29] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. 2025. RouteLLM: Learning to Route LLMs from Preference Data. In *International Conference on Learning Representations*, 34433–34448.
- [30] Jack Min Ong, Matthew Di Ferrante, Aaron Pazdera, Ryan Garner, Sami Jaghouar, Manveer Basra, Max Ryabinin, and Johannes Hagemann. 2025. TOPLOC: A Locality Sensitive Hashing Scheme for Trustless Verifiable Inference. In *Proceedings of the 42nd International Conference on Machine Learning*, PMLR, 47196–47211.
- [31] OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).

- [32] OpenRouter. 2025. OpenRouter. <https://openrouter.ai>. Accessed: 2026-09-03.
- [33] Dario Pasquini, Evgenios M. Kornaropoulos, and Giuseppe Ateniese. 2025. LLMmap: Fingerprinting for Large Language Models. In *34th USENIX Security Symposium (USENIX Security 25)*. USENIX Association, 299–318.
- [34] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently Scaling Transformer Inference. In *Proceedings of Machine Learning and Systems (MLSys)*, Vol. 5. 606–624.
- [35] Haochen Sun, Jason Li, and Hongyang Zhang. 2024. zkLLM: Zero Knowledge Proofs for Large Language Models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*. Association for Computing Machinery, 4405–4419.
- [36] Yifan Sun, Yuhang Li, Yue Zhang, Yuchen Jin, and Huan Zhang. 2025. SVIP: Towards Verifiable Inference of Open-source Large Language Models. In *Socially Responsible and Trustworthy Foundation Models at NeurIPS 2025*.
- [37] Jason Teutsch and Christian Reitwießner. 2017. A Scalable Verification Solution for Blockchains. <https://truebit.io/>. arXiv:1908.04756.
- [38] Florian Tramèr and Dan Boneh. 2019. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In *International Conference on Learning Representations*.
- [39] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 991–1008.
- [40] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted Execution Environments on GPUs. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, 681–696.
- [41] Jiashu Xu, Fei Wang, Mingyu Derek Ma, Pang Wei Koh, Chaowei Xiao, and Muhao Chen. 2024. Instructional Fingerprinting of Large Language Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 3277–3306.