Do Privacy Mechanisms Enable Cheap Verifiable Inference of LLMs? An Initial Exploration

Anonymous Author(s)
Affiliation
Address
email

Abstract

As large language models (LLMs) continue to grow in size, users often rely on third-party hosting and inference service providers. However, in this setting, there is a lack of guarantees on the computation performed by the inference provider. For example, a dishonest provider may replace an expensive large model with a cheaper-to-run weaker model and return the results from the weaker model to the user. Existing tools to verify inference typically rely on methods from cryptography such as zero-knowledge proofs (ZKPs), but these typically add significant computational overhead to vanilla inference. In this work, we develop a new insight – that given a method for performing *private* LLM inference, one can obtain forms of *verified* inference at marginal extra cost. Specifically, we propose two new protocols, the *logit fingerprint* protocol and the *append key* protocol, each of which leverage privacy-preserving LLM inference in order to provide different guarantees over the inference that was carried out. Both approaches are cheap, requiring the addition of a few extra tokens of computation respectively, and have little to no downstream impact. Our work provides novel insights in the connections between privacy and verifiability in the domain of LLM inference.

1 Introduction

Large language models (LLMs) have increased significantly in size over the last few years. Recent *open-weights* models achieve cutting-edge performance [DeepSeek-AI et al., 2025, Qwen et al., 2025, Team et al., 2025], for example, now often contain hundreds of billions of parameters. The hardware requirements to run these are thus now often too high for individuals or organizations to run on their own, leading to a significant growth in demand demand for third-party LLM inference providers. However, this trend raises critical concerns about the integrity and trustworthiness of the services provided, particularly in the burgeoning decentralized inference space. In this setting, any entity with surplus computational resources can offer to complete computational tasks, such as LLM inference, for another user. As the providers in this setting are often individuals or small companies, and do not typically undergo strict vetting, it is imperative to ensure that the service paid for is actually one that is performed by the provider.

Traditionally, the verification of outsourced computation has been addressed through cryptographic methods, such as zero-knowledge proofs (ZKPs). Although offering strong theoretical guarantees, these methods often introduce substantial computational overhead for either the prover (the inference provider) or the verifier (the user), or both. Despite significant progress in recent years, the state-of-the-art for ZK verification of LLM inference remains thousands of times slower than vanilla inference [Sun et al., 2024], rendering it infeasible for large models, which are particularly likely to be in demand for third-party inference provision.

A related concern for third-party compute provision is that of *privacy-preservation*. Performing LLM inference for another party requires the user to share their prompts, resulting in a loss of privacy. Therefore, methods such as secure multi-party computation (SMPC), fully homomorphic encryption (FHE), and trusted

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

execution environments (TEE) have been utilized in recent work to prevent the third-party viewing the user prompt.

Our work examines the question: **if a privacy gadget is already in use, can this be leveraged to provide verification of the LLM inference computation as well?** We answer this in the affirmative; specifically, we propose two simple but novel protocols, 'logit fingerprinting' and 'key appending', that use privacy to obtain differing levels of verification guarantees. We demonstrate that our protocols are extremely cheap compared to methods such as ZK when a privacy mechanism is already used. Although our protocols have limitations and do not offer the same level of guarantees of ZK, we hope that introducing the idea of connecting privacy and verification for LLM inference spurs the creation of improved protocols and further research in this area.

2 Background & Related Work

Due to space constraints, we provide a background on SMPC, FHE, TEEs and ZKP, and a discussion of related work, in Appendix A.

3 Protocol 1: Logit Fingerprinting

Our first proposal for obtaining verification cheaply given access to a privacy-preserving method of LLM inference is **logit fingerprinting**. We hypothesize that the logit vector returned by performing a forward pass on any set of tokens on modern LLMs is a highly unique 'fingerprint' of the model. Our proposed protocol leverages this property to provide inference verification as follows:

1. First, the user inserts K *sentinel tokens* into the tokenized prompt, at random positions within the prompt. Call these positions $p_1, p_2, ..., p_K$. These K tokens are taken randomly from a public cache that is available, consisting of many such length K sequences.
2. Next, the user creates the 2D attention mask to be used by the LLM by taking their desired attention mask (e.g., lower triangular for decoder-only LLMs) and inserting rows and columns as follows.
   - Add a row at p_i that is 0 everywhere except positions $p_j \; \forall \, j \leq i$, where it is set to 1.
   - Add a column at p_i that is 0 everywhere except positions $p_j \; \forall \, j \geq i$, where it is set to 1.
3. The attention mask and augmented tokenized prompt are given to the inference provider under a privacy-preserving scheme, and the inference provider carries out a forward pass, and returns the output logit vector at all token positions to the user.
4. The user verifies that the sentinel token logits match against a precomputed, publicly available cache for that specific model.

The construction of the attention mask is such that the sentinel tokens do not attend to, and are not attended by, any of the original prompt tokens, but they do attend to each other in standard autoregressive fashion. This also ensures that sentinel tokens have no downstream impact on the original prompt when inference is performed.

3.1 Cost Analysis

Inference provider (Prover) Excluding the overhead of the private inference scheme, the total number of extra operations is a factor of $\frac{K}{N}$, where N is the length of the original prompt. As we discuss in Section 3.2, K can be set to be as small as 3 and retain strong security properties, so this is very small for reasonably sized N. Furthermore, if the privacy scheme supports parallel inference, as is the case for GPU-enabled TEEs, for example, this can add almost no additional runtime.

User (Verifier) The verifier is required to pick a sequence from a public cache and perform a matching on the returned logits against the same cache. The cost of this is minimal and does not require specialized hardware.

Construction of the Cache Constructing the cache both entails an initial computational cost and also must be performed by a trusted party, since it is a one-time operation underpinning the correctness of the protocol. Ideally, this responsibility is delegated to an entity with sufficient computational resources to produce a verifiable proof of correctness, for example, in the form of a zero-knowledge proof. Despite the potential computational expenses of the construction, the cost is incurred only once and can be amortized across all subsequent uses of the cache.

3.2 Security Analysis

In this section, we assume that logits are indeed unique fingerprints of models. We perform analysis across a range of models in Section 3.3 to verify this is the case.

In order for the inference provider to not be able to guess the logits to return for the sentinel tokens, the set of sentinel tokens must be randomly chosen from a large set of possibilities. The crux of this protocol is that the inference provider cannot determine which of the possibilities is specifically being asked for in any particular instance due to the privacy gadget.

We propose that it is sufficient to precompute a cache of 1000 different sentinel sequences. For example, if $K = 3$, then 1000 unique sequences of length 3 are sampled from the model's token vocabulary, and a forward pass is performed on these (with a standard unidirectional attention mask). The logits of the final token in the sequence are stored in the cache, totaling ~ 400MB in this setting.

Probabilistic Attacks This protocol utilizes two elements of randomization: the choice of the sentinel tokens, and their positions. For the former, if the user selects the sequence uniformly at random from a cache of size $|C|$, then a dishonest inference provider can guess it with probability at best $1/|C|$. For the latter, under a privacy-preserving gadget that also preserves tensor structure (such as SMPC), correctly guessing of the sentinel tokens' exact positions is sufficient for a successful attack: the inference provider can perform a forward pass on only those components. However, this occurs with probability $\binom{N+K}{K}^{-1}$, where N is the length of the original prompt. When $K = 3$, for example, with $N = 14$, this is less than $1\mathrm{e}{-3}$, and it drops further with $N = 100$ to circa $1\mathrm{e}{-6}$.

A related attack is to perform computation only on a random subset of the token indices (adjusting the attention mask correspondingly). In the most extreme case, a dishonest provider takes $N + K - 1$ tokens, i.e. excludes exactly one token. The probability that all sentinel tokens are still selected (hence successfully passing verification) is $\frac{N}{N+K}$, requiring an infeasibly large K to make secure.

Approximation Attacks We consider attacks focusing on attempts to use a different model – especially, cheaper-to-run models – that still succeed in passing verification. Such alternatives could include smaller models from the same model family or approximations to the models by using e.g. low-rank projections of the weights. We perform experiments to test the robustness of the protocol to each of the above in Section 3.3 and find that our protocol fails immediately when any of the above are attempted.

3.3 Experiments

Setup We test the claim from Section 3.2 that pre–softmax logits can serve as model fingerprints. For each model m, we sample $N = 50{,}000$ token sequences of fixed length $K = 3$ from the model's tokenizer vocabulary (excluding special tokens). Given a sequence $\mathbf{t} = (t_1, t_2, t_3)$, we run a forward pass and record the next-token *logit vectors* at each position, $\ell_m^{(k)}(\mathbf{t}) \in \mathbb{R}^{V_m}$ for $k \in \{1, 2, 3\}$, where V_m is the vocabulary size of model m. We define the *logit fingerprint*

$$\phi_m(\mathbf{t}) \;=\; \mathrm{concat}\big(\ell_m^{(1)}(\mathbf{t}),\, \ell_m^{(2)}(\mathbf{t}),\, \ell_m^{(3)}(\mathbf{t})\big) \in \mathbb{R}^{3V_m},$$

and compare fingerprints using L1 distance. We test on Llama 3.2 Instruct 1B, 3B, and 8B [Grattafiori et al., 2024], and on Qwen 2.5 Instruct 0.5B, 1.5B, 3B and 7B [Qwen et al., 2025]. Comparisons are performed on FP32 logits; dropout is disabled.

Floating Point Non-determinism We first provide context on the expected L1 distance due to non-determinism of floating-point operations [Shanmugavelu et al., 2024], We run the *same* sequence multiple times with different batch sizes on GPU to measure this. We observe a maximum L1 deviation in doing so across all models tested of 3.168.

Intra-model Within each model, we compute the nearest-neighbor similarity among fingerprints from *distinct* sequences (i.e. $\mathbf{t} \neq \mathbf{s}$). Across $N = 50$k samples per model, there are no exact matches; the closest pair has an L1 distance of 2909.

Within-family For the Llama family the smallest L1 distance of logits we obtain is 335399. For the Qwen family the minimum cross-model distance is 791218. These results indicate that even with a family of models, the logits are significantly different and suitable as fingerprints.

Cross-family To enable comparisons across families with different vocabularies, we align dimensions by truncating the larger logit vectors to the smaller vocabulary size (i.e. comparing the first $\min(V_m, V_{m'})$ coordinates). Under this conservative alignment, Llama–Qwen comparisons exhibit substantially higher distances than the within-family maxima reported above (qualitatively, well above 800000).

Low-rank factorization We approximate the linear layers of Llama 3.2 1B Instruct by replacing each weight matrix $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ with a rank-r factorization $W \approx UV^\top$, where $U \in \mathbb{R}^{d_{\text{in}} \times r}$ and $V \in \mathbb{R}^{d_{\text{out}} \times r}$. The default hidden dimension of this model is 2048, so we test with $r \in \{2047, 2040, 2000\}$. Comparing fingerprints of 50k sequences between the full-rank and the low-rank variants, the minimum L1 distances observed observed are:

$$r = 2047 : 833.97, \quad r = 2040 : 8029.45, \quad r = 2000 : 31194.02.$$

Quantization We next load Llama 3.2 1B Instruct in 8-bit precision using `bitsandbytes` and compare fingerprints to the full-precision (bfloat16) baseline. The minimum L1 distance is 32137, again easily separated from the original model.

Fine-tuning Finally, we evaluate robustness against model fine-tuning by comparing Llama 3.2 1B Instruct with a finetuned variant on a single sample from FineWeb dataset for a single step. The minimum observed distance is 471.10, consistent with the previous cases and again easily separable from the original model.

In our experiments, the minimum L1 distance observed between two different sequences was 833.97, as seen in the low-rank setting, while the maximum deviation caused by floating point non-determinism was only 3.168. Based on these results, we recommend using a matching threshold in the range of 5–10. Sequences whose logits differ by less than this threshold can be confidently regarded as originating from the same model; and even a single step of fine-tuning is easily detectable with this threshold.

3.4 Limitations

The main limitation of this protocol is that it can only be used to verify a single forward pass at a time, i.e. only generate a single new token, before requiring the user to repeat the protocol above with a fresh set of sentinel tokens and positions; otherwise, a dishonest provider could honestly perform the first forward pass (to pass verification) and provide spurious outputs for all subsequent forward passes. Thus, this protocol inherently requires user interaction for every step of token decoding. Another limitation is the vulnerability to the subsetting attack mentioned in Section 3.2. As such, we recommend that this protocol not be used in isolation with privacy gadgets that retain tensor structure, such as SMPC methods.

4 Protocol 2: Key Appending

A second and conceptually distinct proposal for obtaining verification cheaply under privacy assumptions is **key appending**. The high-level idea of this protocol is to ask the LLM to emit a randomly generated key at the end of its normal response – for example, 'strawberry reticent gestalt' – wrapped in a specific tag structure, and to verify at the end of inference that the key was correctly replicated.

System prompt In addition to the user's input, we prepend a system-level instruction that enforces the verification protocol and prevents premature conversation termination:

```
You are a helpful assistant who should never speak in two
consecutive turns.  At the end of your response, repeat the
key mentioned at the end of the prompt.  You must print the key
between tags like the following structure:  <key> *insert key here*
</key>.
```

This system prompt explicitly conditions the model to always conclude with the verification key enclosed in `<key>...</key>` tags, and discourages it from generating the turn-termination token prior to key emission.

User prompt augmentation Given a user's original prompt p and a randomly sampled key $w_1, w_2, \ldots, w_K$ of K words, we augment p by appending the following instruction to the end:

> `At the end of your response, repeat this key: <key>` w_1 w_2 $\cdots$ w_K `</key>\nDO NOT print anything else after it.`

Stopping criterion Unlike standard decoding, where inference continues until the model produces an end-of-sequence token, we adopt a custom stopping rule: decoding halts once the closing tag `</key>` is generated. This ensures that inference completes exactly after the verification key is produced, with no trailing tokens.

Verification The encrypted response is returned to the user, who decrypts it and parses the contents of the `<key>...</key>` span. Verification succeeds if and only if the extracted string matches the originally sampled key. Using HTML-/XML-like tags facilitates robust parsing and prevents ambiguity in locating the verification key within the model's output.

We also tested the same protocol but instead of appending the key repeating prompt to the end of the user's prompt, we insert it in any whitespace randomly. This approach performs less well – we report results for this in Appendix B.

4.1 Cost Analysis

Inference provider (Prover) Adding an extra t tokens to the prompt adds an overhead of a factor of $\frac{t}{N}$ operations (in addition to the system prompt and remaining augmentations, which are of constant length). Given that, for most tokenizers and English words, words are approximately 1–2 tokens in length, and that the protocol offers good security with just $K = 3$ words (see Section 4.3), this therefore introduces little extra overhead.

User (Verifier) Similar to the logit fingerprinting protocol, the verifier is required to perform minimal work. They must select a sequence of K words $w_1 w_2 \cdots w_K$ and append them together with the augmentation template to the prompt, as well as prepend the system prompt. When the inference is complete, the verifier checks the words between the `key` tags of the decoded output against the original words $w_1 w_2 \cdots w_K$. Again, no specialized hardware is necessary.

4.2 Security Analysis

Probabilistic Attacks Suppose that tokenizing the K words results in a total of t tokens. An adversarial party must correctly guess each token exactly out of the total vocabulary, resulting in a success probability of $\frac{1}{|V|^t}$, where $|V|$ is the vocabulary size. As modern LLMs typically have $|V| \geq 1\text{e}5$, with a key length of only 3 tokens, this is already on the order of $1\text{e}{-15}$ or lower.

Approximation Attacks This protocol is potentially vulnerable to the model approximation attacks as described in Section 3.2, especially if they largely retain the instruction following capabilities of the original model. We again perform experiments to test them in Appendix C.

4.3 Experiments

Key Transcription Capability We first examine the capability of LLMs to perform our protocol successfully; this is analogous to the cryptographic property of *completeness*. We evaluate multiple open-source models on a random sample of 1000 prompts from the Databricks dolly-15k dataset [Argilla, 2023]. The key is sampled uniformly at random from the standard Ubuntu `words` file provided by the `wordlist` package (any similar list of English words suffices) and appended via the protocol; success is recorded if the words enclosed in the `key` tags match the key exactly.

Table 1 reports the transcription success rate. We observe near-perfect adherence for models at or above the 3–4B scale. Therefore, a simple capacity criterion suffices in practice: models with $\geq$3B parameters reliably satisfy the protocol's instruction, making them suitable drop-in choices for verified inference with key appending.

Table 1: Transcription success rate on 1000 prompts of our 'key appending' verification protocol, with keys of length $K = 3$ words. We see that models with parameter sizes of 3B and above obtain very high transcription rates of $> 98\%$.

Model	**Transcription rate**
Llama 3.2 1B	56.6%
Gemma 3 1B	73.1%
Llama 3.2 3B	98.1%
Gemma 3 4B	99.7%
Mistral 7B	99.6%
Llama 3.1 8B	98.7%
Gemma 3 12B	98.0%
Mistral 24B	100.0%
Qwen 2.5 32B	99.9%
Llama 3.1 70B	99.6%

Table 2: LiveBench scores for models under the 'key appending' protocol. Higher is better. There is a relatively large degradation in performance for smaller models, but models of size 8B and larger exhibit much smaller relative degradation.

Model	**Vanilla**	**Append**	Δ
Llama 3.2 1B	10.7	6.7	-4.0
Gemma 3 1B	14.7	9.9	-4.8
Llama 3.2 3B	20.5	16.9	-3.6
Qwen 2.5 3B	24.2	18.3	-5.9
Gemma 3 4B	30.2	26.4	-3.8
Mistral 7B	20.4	14.5	-5.9
Llama 3.1 8B	25.4	25.6	$+0.2$
Gemma 3 12B	41.0	36.8	-4.2
Mistral 24B	30.5	32.1	$+1.6$
Qwen 2.5 32B	42.7	41.7	-1.0
Llama 3.1 70B	42.3	39.8	-2.5

Downstream Performance Impact We now test the impact of the additional key-transcription instruction on model downstream performance. To quantify this, we evaluate all models on LiveBench [White et al., 2025] (30–05–2025 release), a benchmark testing model performance on a range of tasks including data analysis, instruction following, language, math, and reasoning. Table 2 compares overall performance of the models in vanilla inference against our protocol. Extended tables showing complete results for all LiveBench categories are presented in Appendix D and Appendix B.

We find that there there is a relatively large performance impact for smaller models. However, for larger models, the performance impact is reduced. Indeed, for Mistral 24B, the performance is actually slightly higher – which we attribute to the natural variability inherent in the benchmark. In practice, using a model of size 8B or larger seems sufficient to ensure minimal relative downstream performance impact from applying this protocol.

Approximation Attack Experiments As described in Section 4.2, this protocol is potentially vulnerable to model approximation attacks. We perform an in-depth examination of the viability of such in the specific case of use of SMPC for privacy preservation, with one honest party, in Appendix C. We find that in this setting, any such approximation does fail.

4.4 Limitations

The main limitation of this protocol is that it cannot specify exactly the model that is being used; it only guarantees that the model is capable of performing the verification task. In the SMPC setting, if there is at least one honest participant, then we show in Appendix C that it is any approximations result in the verification failing. However, in the FHE or TEE setting, this remains a limitation. Furthermore, this protocol does not offer 100% completeness, although we see figures close to this (see Table 1).

5 Conclusion

We have introduced two protocols for verifying LLM inference, given the use of privacy-preserving tools. We have shown that these protocols are cheap for both the prover and the verifier, and have little to no downstream impact. Each protocol retains their own set of limitations. We hope that by introducing the idea of connecting privacy and verifiability, particularly in the LLM domain, that we can inspire future work to devise new and improved protocols that address these shortcomings.

References

Yoshimasa Akimoto, Kazuto Fukuchi, Youhei Akimoto, and Jun Sakuma. Privformer: Privacy-preserving transformer with mpc. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 392–410, 2023. doi: 10.1109/EuroSP57164.2023.00031.

Argilla. argilla/databricks-dolly-15k-curated-en [dataset]. https://huggingface.co/datasets/argilla/databricks-dolly-15k-curated-en, 2023. Downloaded from Hugging Face Hub on 2025-08-29.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*, pages 409–437. Springer, 2017.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Ye Dong, Wen jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Chen. Puma: Secure inference of llama-7b in five minutes, 2023. URL https://arxiv.org/abs/2307.12533.

Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585062. doi: 10.1145/1536414.1536440. URL https://doi.org/10.1145/1536414.1536440.

O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery. ISBN 0897912217. doi: 10.1145/28395.28420. URL https://doi.org/10.1145/28395.28420.

S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery. ISBN 0897911512. doi: 10.1145/22145.22178. URL https://doi.org/10.1145/22145.22178.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong,

Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir

Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In *Advances in Neural Information Processing Systems*, volume 35, pages 15718–15731, 2022.

Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 809–826, 2022.

Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 21–30, 2007.

Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. Trusted execution environments: Properties, applications, and challenges. *IEEE Security Privacy*, 18(2):56–60, 2020. doi: 10.1109/MSEC.2019.2947124.

B. Knott, S. Venkataraman, A.Y. Hannun, S. Sengupta, M. Ibrahim, and L.J.P. van der Maaten. Crypten: Secure multi-party computation meets machine learning. In *arXiv 2109.00984*, 2021.

Zhengyi Li, Kang Yang, Jin Tan, Wen jie Lu, Haoqi Wu, Xiao Wang, Yu Yu, Derun Zhao, Yancheng Zheng, Minyi Guo, and Jingwen Leng. Nimbus: Secure and efficient two-party inference for transformers, 2024. URL `https://arxiv.org/abs/2411.15707`.

Jinglong Luo, Guanzhong Chen, Yehong Zhang, Shiyu Liu, Hui Wang, Yue Yu, Xun Zhou, Yuan Qi, and Zenglin Xu. Centaur: Bridging the impossible trinity of privacy, efficiency, and performance in privacy-preserving transformer inference, 2024. URL `https://arxiv.org/abs/2412.10652`.

Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. THOR: Secure transformer inference with homomorphic encryption. Cryptology ePrint Archive, Paper 2024/1881, 2024. URL `https://eprint.iacr.org/2024/1881`.

Krishna Giri Narra, Zhifeng Lin, Yongqin Wang, Keshav Balasubramaniam, and Murali Annavaram. Privacy-preserving inference in machine learning services using trusted execution environments. *arXiv preprint arXiv:1912.03485*, 2019.

Jack Min Ong, Matthew Di Ferrante, Aaron Pazdera, Ryan Garner, Sami Jaghouar, Manveer Basra, Max Ryabinin, and Johannes Hagemann. Toploc: A locality sensitive hashing scheme for trustless verifiable inference, 2025. URL `https://arxiv.org/abs/2501.16007`.

Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. BOLT: Privacy-preserving, accurate and efficient inference for transformers. Cryptology ePrint Archive, Paper 2023/1893, 2023. URL https://eprint.iacr.org/2023/1893.

Wenjie Qu, Yijun Sun, Xuanming Liu, Tao Lu, Yanpei Guo, Kai Chen, and Jiaheng Zhang. zkgpt: An efficient non-interactive zero-knowledge proof framework for llm inference. In *34st USENIX Security Symposium (USENIX Security 25)*, 2025.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/Ispa*, volume 1, pages 57–64. IEEE, 2015.

Sanjif Shanmugavelu, Mathieu Taillefumier, Christopher Culver, Oscar Hernandez, Mark Coletti, and Ada Sedova. Impacts of floating-point non-associativity on reproducibility for hpc and deep learning applications, 2024. URL https://arxiv.org/abs/2408.05148.

Haochen Sun, Jason Li, and Hongyang Zhang. zkllm: Zero knowledge proofs for large language models, 2024. URL https://arxiv.org/abs/2404.16109.

Yifan Sun, Yuhang Li, Yue Zhang, Yuchen Jin, and Huan Zhang. Svip: Towards verifiable inference of open-source large language models, 2025. URL https://arxiv.org/abs/2410.22307.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2025. URL https://arxiv.org/abs/2507.20534.

Rahul Thomas, Louai Zahran, Erica Choi, Akilesh Potti, Micah Goldblum, and Arka Pal. Cascade: Token-sharded private llm inference, 2025. URL https://arxiv.org/abs/2507.05228.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. Livebench: A challenging, contamination-limited llm benchmark, 2025. URL https://arxiv.org/abs/2406.19314.

Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982. doi: 10.1109/SFCS.1982.38.

Mu Yuan, Lan Zhang, and Xiang-Yang Li. Secure transformer inference protocol, 2024. URL `https://arxiv.org/abs/2312.00025`.

Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure transformer inference made non-interactive. Cryptology ePrint Archive, Paper 2024/136, 2024. URL `https://eprint.iacr.org/2024/136`.

Fei Zheng, Chaochao Chen, Zhongxuan Han, and Xiaolin Zheng. Permllm: Private inference of large language models within 3 seconds under wan, 2024. URL `https://arxiv.org/abs/2405.18744`.

A Background and Related Work

In this section, we provide a brief background on general methods of privacy-preserving function computation, general methods of verification, and their application to LLM inference in particular.

A.1 Privacy-Preservation

There are four main families of privacy-preserving inference of LLMs that have been proposed in the literature: **SMPC** (Secure Multi-Party Computation), **FHE** (Fully Homomorphic Encryption), **TEEs** (Trusted Execution Environments), and **statistical methods**. Here we provide brief background on each of these.

SMPC SMPC protocols split the required computation among multiple parties. The key ideas were originally developed in the 1980s [Yao, 1982, Goldreich et al., 1987] and provide mathematical guarantees that no single party can reconstruct the data on their own. Recently, the methodologies of SMPC have been applied to LLMs [Huang et al., 2022, Hao et al., 2022, Pang et al., 2023, Akimoto et al., 2023, Dong et al., 2023, Li et al., 2024]. A difficulty uniformly faced by these protocols is efficient computation of the many non-linearities present in transformer-based LLMs; most of the works attempt to ameliorate this by using piecewise polynomial approximations which are more well-suited for MPC algorithms. However, this approximation leads to degraded inference results, and remains more expensive than direct computation of the non-linearities. The requirement of multiple parties also engenders significant communication overheads, and the further non-collusion requirement among the parties may be difficult to guarantee.

FHE FHE protocols require only a single party and make use of cryptographic methods to ensure that the result of the computation on the ciphertext is the same as that performed on the plaintext. The adjective 'fully' indicates the capability of performing arbitrary computations, not limited to a particular type or complexity. The first plausible construction of an FHE scheme was described in Gentry [2009]; a more modern and widely used incarnation is CKKS [Cheon et al., 2017]. Recently, CKKS has been further optimized and applied to LLM inference [Moon et al., 2024, Zhang et al., 2024], but similar issues arise with the non-linearities as SMPC methods. The overheads both for linear and non-linear operations are typically even larger than those in the SMPC setting.

TEE Trusted Execution Environments (TEEs) [Sabt et al., 2015, Narra et al., 2019] create secure and isolated enclaves at the hardware level. This ensures confidentiality via memory encryption – allowing only the process running in the enclave to read the data. Furthermore, TEEs support integrity via attestation mechanisms. However, a significant concern is the vulnerability to side-channel attacks [Jauernig et al., 2020]. Furthermore, attestation is only provided at boot-time and is not equivalent to an ongoing verification process. This process typically involves the TEE measuring the code and its environment, signing these measurements cryptographically, and sending a report for external verification. However, this is often a one-time check at the start and does not guarantee the integrity of the TEE throughout its execution. Finally, in cloud environments, attestation can rely on the cloud provider's services, which means users must trust the provider's proprietary attestation process without full transparency. This introduces a level of trust in the cloud provider's integrity, as these attestation services can be opaque "black boxes" that are not open to external audit. Moreover, there may be no independent way to verify the boot measurements provided by the cloud provider's infrastructure.

Statistical Methods A more broad and diverse grouping than the above is what we term 'statistical methods'. These are protocols without the mathematical guarantees of FHE or SMPC approaches, or the hardware-based guarantees of TEEs, but that instead employ statistical or empirical arguments to support the difficult of reversing ciphertext. Some ideas in this domain include the use of permutation-based security [Zheng et al., 2024, Yuan et al., 2024, Luo et al., 2024] or token-sharding based security [Thomas et al., 2025]. These methods typically trade off the stronger guarantees of the above methods for greatly reduced overheads, sometimes approaching similar speeds to vanilla inference.

A.2 Verification

Zero-Knowledge Proofs (ZKP) ZKPs are a class of methods that allows one party (the prover) to prove to another party (the verifier) that a statement is true, without revealing any additional information beyond the proof itself. The main properties that ZKPs satisfy are completeness (an honest prover can convince a verifier that they performed the work as stated), soundness (a dishonest prover cannot convince a verifier that they performed the work), and the zero-knowledge property of not revealing any further information than the fact the work was done as stated. The first ZK protocol was introduced in 1985 in Goldwasser et al. [1985]. Recently, ZK methods have been applied as proofs of inference for machine learning models, and specifically LLMs, in works such as Sun et al. [2024], Qu et al. [2025]. However, these approaches remain thousands of times slower than vanilla inference – for example, zkLLM takes 15 minutes for generating a proof of a single forward pass for Llama-2-13B, compared to milliseconds for vanilla inference.

Statistical Methods Analogously to statistical methods of privacy-preservation, very recent work has investigated methods of relaxing the standard of proof of work provided in order to reduce computational overhead. Ong et al. [2025] encodes and validates the most salient features of the last hidden state tensor of an LLM using a compact, verifiable proof, which is then recomputed in parallel by the verifier. Although the authors demonstrate how to set up a commitment scheme that has relatively little overhead to the prover, and verification is faster than full recomputation thanks to parallelization, there is still a requirement for the verifier to perform a full LLM forward pass, potentially necessitating specialized hardware. Sun et al. [2025] proposes the use of a 'proxy task' based on the last hidden layer features of an LLM that can then be utilized by the user to compare to a label that they would expect based on their original input. The method proposed requires trust assumptions from the platform for generation of the proxy-task feature extractor and labeller networks, as well as secret generation/embedding, and adds the overhead of computation to perform all of the above.

A.3 Connections Between Privacy and Verifiability

The connection between privacy and verification has not been extensively studied previously. Perhaps the closest work to approach this is MPC-in-the-Head [Ishai et al., 2007], which introduced a zero-knowledge verification protocol by utilizing any SMPC protocol. The protocol comes with steep costs for both the prover and verifier. For example, the prover must not only locally simulate every party in the underlying MPC execution but also repeat the computation multiple times. On the verifier's end, the party must perform several confirmation tasks, including recomputing opened views, consistency checks, and typically engage in multiple rounds of checking to achieve acceptable soundness. The crucial distinction of our suggested protocols to MPC-in-the-Head is that we use the privacy scheme *directly* to encode inexpensive secrets that are easily verifiable. To the best of our knowledge, there has not previously been any work that specifically examines the relationship between privacy-preserving LLM inference and verifiable inference of LLMs in this way.

B Random Whitespace Protocol

As an alternative to the key appending protocol of Section 4, we also tested requiring the model to repeat the key in a random whitespace.

Protocol description. This protocol adopts the same stopping criterion and verification method as the key–appending protocol, but modifies both the system prompt and the user prompt transformation.

System prompt. We prepend the following system instruction:

```
You are a helpful assistant who should never speak in two
consecutive turns.  Somewhere in the prompt, you will find a
secret key of {K} words in curly braces right after the marker
<{marker}>.  Print that key at the very end of your response.
Ignore what is inside the <verify> </verify> tags when answering
the main prompt.\nYou must print the key between tags like the
following structure:  <key> *insert key here* </key>.
```

This system prompt enforces the placement of the key at the end of the model's response and ensures that the verification tags are ignored during the main task, preventing interference with downstream output.

User prompt transformation. Given an original user prompt p, the user selects a random whitespace location and inserts the following structure:

```
<verify> <{marker}> {key} </{marker}> </verify>.
```

Table 4: LiveBench scores for models under the 'random whitespace' protocol. Higher is better. There is generally a larger degradation in performance than for the key appending protocol.

Model	**Vanilla**	**Random Whitespace**	**Δ**
Llama 3.2 1B	10.7	5.4	-5.3
Gemma 3 1B	14.7	9.2	-5.5
Llama 3.2 3B	20.5	14.2	-6.3
Qwen 2.5 3B	24.2	17.2	-7.0
Gemma 3 4B	30.2	21.3	-8.9
Mistral 7B	20.4	12.2	-8.2
Llama 3.1 8B	25.4	20.7	-4.7
Gemma 3 12B	41.0	29.1	-11.9
Mistral 24B	30.5	25.3	-5.2
Qwen 2.5 32B	42.7	39.3	-3.4
Llama 3.1 70B	42.3	32.1	-10.2

Here, the marker is a randomly generated four-character ASCII string, and the key consists of three English words sampled uniformly at random, as in the key–appending protocol.

Design rationale. The system prompt explicitly instructs the model to ignore the inserted tags when answering the main query, which tries to minimize the impact of the injected verification key on downstream task performance. Moreover, we deliberately employ HTML-like tags for three reasons:

1. Large language models are extensively exposed during pretraining to HTML/XML patterns, which aids reliable parsing and generation.
2. Wrapping the marker–key pair inside `<verify>` tags avoids accidental collisions with ordinary prompts (e.g., programming queries that might already include custom markers).
3. Randomly generating the marker string reduces the probability of unintentional matches with existing content, while including the outer `<verify>` tags improves transcription accuracy compared to using only `<marker>` … `</marker>`.

The transcription rates and downstream performance impact of the protocol are shown in Table 3 and Table 4 respectively. Although the transcription rates match that of key appending for models of size 8B and above, the downstream performance impact is significantly larger than that of key appending.

Table 3: Transcription success rate on 1000 prompts of our 'Random Whitespace' verification protocol, with keys of length $K = 3$ words. We see that models with parameter sizes of 8B and above obtain very high transcription rates of $> 98\%$.

Model	**Transcription rate**
Llama 3.2 1B	6.7%
Gemma 3 1B	2.2%
Llama 3.2 3B	88.8%
Gemma 3 4B	79.0%
Mistral 7B	86.6%
Llama 3.1 8B	98.5%
Gemma 3 12B	99.0%
Mistral 24B	99.6%
Qwen 2.5 32B	99.2%
Llama 3.1 70B	99.3%

C Approximation Attack Experiments – Key Appending

We perform approximation attack tests in the SMPC setting. We assume the existence of at least one honest party; in the case where all parties are dishonest (i.e. performing the same, matching approximation), the approximated model still can potentially accurately produce the key and evade detection. We use the CrypTen Python library [Knott et al., 2021].

For the dishonest party, we uniformly reduce the rank of all weight matrices in the models to various proportions of the original rank, and test the protocol to see whether the approximated model is still capable of correctly outputting the key. We select the following for our parameters:

1. Models: Llama 3.2 3B Instruct, Qwen 2.5 3B. We select two models with very high key transcription rates in the non-attack setting from different model classes.
2. We test the reduction of original ranks of very weight matrix M to the following percentage reductions: 1%, 5%, 10%, 25%, 50%, 75%, 90%, 99%, using SVD. We desire to test a wide variety of different ranks, ranging from an extremely significant reduction in rank to a slight decrease in rank, and we hence select the previously listed percentages for significant coverage of all of these possibilities.

For each combination of model and rank, we run the framework as described at the beginning of this section, selecting $n = 20$ prompts and $K = 3$ words.

The results of such experiments revealed that regardless of the model used or rank approximated to, the model was *always unable to output the key* (i.e. 0 of the 20 tests succeeded). Notably, even in the 99% test, both models were unable to produce anything legible, and tokens outputted were entirely random: an example decoded result from one prompt was "deesestiftigiongh" with random unicode characters inserted inside.

Quantizer Attacks A malicious actor can also potentially quantize the model's weights to a different precision, which is straightforward to test: given a model, we quantize all its weights to a different precision and perform the common tests to determine performance.

We again perform tests in an SMPC setting encrypted with CrypTen with two parties, one honest and one dishonest. We again note the potential weakness of this strategy when both parties are dishonest or a different encryption scheme is used.

1. Models: Llama 3.2 3B Instruct, Qwen 2.5 3B. We select the same models as for the low-rank approximations, due to their ordinarily high transcription rates.
2. Precisions: 8-bit and 4-bit floats. The weights in the Llama model tested are 16-bit floats at full precision, and in the Qwen model are 32-bit floats. Therefore, to reduce precision, we test quantization to 8- and 4-bit precision.

Once again, we run the common testing framework with $n = 20$ prompts and $K = 3$ words. Similar to the low-rank tests, in all cases, the models were *never able to output* the key, or in fact anything legible, revealing the effectiveness of the key appending protocol in defending against quantization attacks.

D Key Appending – Extended LiveBench Results

Table 5: LiveBench category scores for vanilla inference.

Model	Average	Data Analysis	Instr. Follow.	Language	Math	Reasoning
Llama 3.2 1B	10.7	12.9	25.1	0.0	9.5	5.8
Gemma 3 1B	14.7	12.0	42.1	3.7	13.1	2.9
Llama 3.2 3B	20.5	23.3	48.4	3.5	15.5	11.9
Qwen 2.5 3B	24.2	29.0	43.2	10.7	23.5	14.6
Gemma 3 4B	30.2	38.3	61.5	6.3	33.0	11.8
Mistral 7B	20.4	26.4	46.2	1.5	13.4	14.4
Llama 3.1 8B	25.4	36.0	48.0	13.8	15.9	13.1
Gemma 3 12B	41.0	46.4	71.2	19.3	39.5	28.8
Mistral 24B	30.5	42.1	50.4	17.3	19.0	23.4
Qwen 2.5 32B	42.7	50.7	61.2	27.3	43.9	30.4
Llama 3.1 70B	42.3	52.6	65.9	30.3	31.4	31.4

Table 6: LiveBench category scores under the 'key appending' protocol. Higher is better.

Model	**Average**	**Data Analysis**	**Instr. Follow.**	**Language**	**Math**	**Reasoning**
Llama 3.2 1B	6.7	0.9	24.9	0.0	2.3	5.5
Gemma 3 1B	9.9	3.3	32.1	2.3	4.8	6.6
Llama 3.2 3B	16.9	8.0	43.4	7.8	11.7	13.5
Qwen 2.5 3B	18.3	18.2	30.4	5.8	21.4	15.9
Gemma 3 4B	26.4	38.6	41.3	8.0	24.5	19.8
Mistral 7B	14.5	21.7	31.8	6.7	6.6	5.5
Llama 3.1 8B	25.6	35.3	51.7	9.3	15.2	16.6
Gemma 3 12B	36.8	46.1	60.5	16.5	37.0	24.0
Mistral 24B	32.1	41.1	47.2	24.0	21.0	26.9
Qwen 2.5 32B	41.7	47.2	58.2	24.0	44.2	35.0
Llama 3.1 70B	39.8	50.4	69.2	22.0	29.1	28.5