zklora: Verifiable Split Inference for Evaluating Private LoRA Weights

Anonymous Authors

January 10, 2025

Abstract

In decentralized computing environments, one party may wish to fine-tune a publicly available base model with Low-Rank Adaptation (LoRA) but keep those adapter weights private, while another party wants to evaluate performance on new data. We introduce `zklora`, a library designed to help the *Base Model User verifiably assess* the quality of a private LoRA module for a target task—for instance, to run a validation experiment without directly accessing the private weights. By adopting a split inference approach and generating *verifiable computations*, `zklora` enables the *Base Model User* to confirm that remote LoRA computations are correct, all without seeing the private weights themselves. We demonstrate the feasibility of `zklora` via benchmark experiments, showing that our approach remains computationally viable across various model sizes, with notably fast verification times.

1 Introduction

Large Language Models (LLMs) excel at a variety of tasks but come with significant computational and storage costs [1, 2]. Low-Rank Adaptation (LoRA) [4] addresses these costs by introducing a small number of trainable parameters into the model, drastically reducing memory overhead for fine-tuning [3].

When two entities, a *LoRA Owner* and a *Base Model User*, collaborate on a shared LLM, they may have conflicting constraints. The *LoRA Owner* wants to keep custom LoRA weights private (e.g., for competitive or regulatory reasons), while the *Base Model User* wants to *test* or *validate* these LoRA weights on a target task before deciding whether to deploy them at scale. Under normal circumstances, either the LoRA must be fully shared (sacrificing privacy), or the Base Model User must trust the LoRA Owner's claims about performance with little evidence.

In this paper, we present `zklora`, a library that provides a *verifiable computation* solution for the *Base Model User* to *evaluate* a private LoRA module. By orchestrating a split inference, `zklora` allows the Base Model User to run the base model locally while sending partial activations to the LoRA Owner. Crucially, the LoRA Owner returns not only updated activations but also a cryptographic *proof* that the correct LoRA transformation was applied. In this way, the Base Model User can confirm the final outputs truly reflect the LoRA module's capabilities. After verifying and measuring results on a validation dataset, the Base Model User can later choose how to adopt the LoRA at larger scale.

2 Related Work

2.1 Low-Rank Adaptation

LoRA [4] has emerged as a popular way to efficiently fine-tune LLMs by injecting small, low-rank matrices into specific layers. This approach scales well, requiring less GPU memory than full fine-tuning. Various frameworks build on LoRA for GPT-2, GPT-Neo, LLaMA, and others [3]. By confining most updated parameters to low-rank adapter matrices, LoRA drastically reduces memory overhead during fine-tuning. As a result, domain-specific LoRA modules can be swapped in or out of the same base model, avoiding the need to retrain or store complete copies of large networks.

2.2 Incrementally Verifiable Computation

Valiant [7] introduced the notion of Incrementally Verifiable Computation (IVC), which offers updatable, compact proofs of correctness over multi-step computations. Nova [6] and HyperNova [5] refine these ideas, leveraging folding schemes and zero-knowledge properties to enable scaling of verifiable computations in multi-step neural network inference. IVC-based approaches align well with repeated computations needing multiple checkpoints of trust, such as multiple forward passes or partial training loops.

3 Methodology: zklora for Verifiable Evaluation

Our aim is to enable the *Base Model User* to run a *validation experiment* on their local data, incorporating the private LoRA module from the *LoRA Owner*. The Base Model User can measure task performance (e.g., cross-entropy loss, accuracy) to decide whether the LoRA meets performance requirements. Figure 1 illustrates a simple scenario.

1. **Base Model Forward Pass (Base Model User).** The *Base Model User* runs layers without LoRA parameters locally, generating partial activations.

2. **Forward to LoRA Adapter (Base Model User).** Whenever the next layer includes a LoRA transform, the *Base Model User* sends these partial activations to the *LoRA Owner*, who holds the private LoRA module.

3. **LoRA Computation & Proof Generation (LoRA Owner).** The *LoRA Owner* applies the LoRA transformation to the activations and returns the updated activations alongside a *cryptographic proof* of correctness. In our setup, `zklora` interfaces with `ezkl` to produce these proofs.

4. **Resuming the Forward Pass (Base Model User).** The *Base Model User* injects the updated activations back into the local model state and continues the forward pass.

5. **Verification & Validation (Base Model User).** The *Base Model User* verifies the LoRA proof. If valid, the outputs from the combined base + LoRA model are guaranteed to match the LoRA Owner's claimed computation. The Base Model User can now measure final metrics (loss, accuracy, etc.) to assess *quality* on the target task.

After confirming performance with `zklora`, the Base Model User can move forward using the LoRA weights with confidence, knowing their underlying quality combined with the base model.

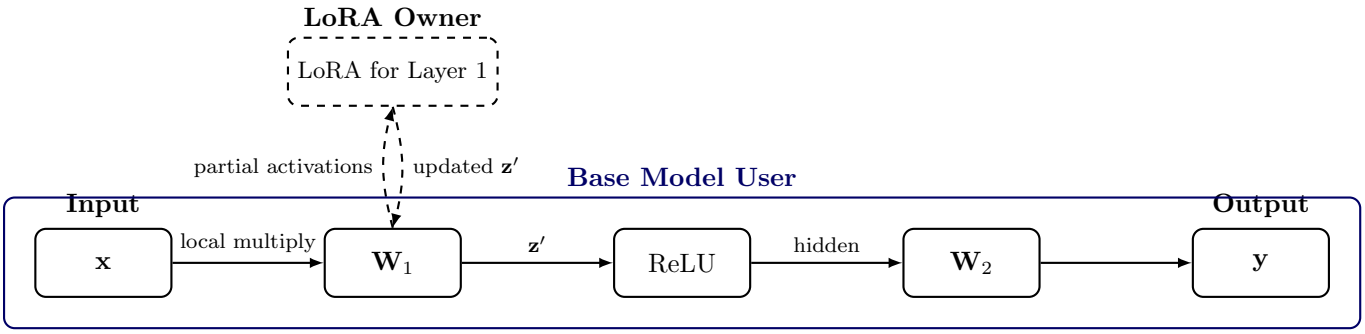

Figure 1: A simplified 2-layer MLP with LoRA on the first layer. The Base Model User (blue box) runs most computations locally but calls the LoRA Owner (dashed box) to update partial activations. After verifying correctness, the Base Model User can measure final metrics to decide whether to adopt the private LoRA more extensively.

The core objective with `zklora` is to *verify* that the private LoRA weights indeed perform as claimed, without revealing them.

4 Experiments

We benchmarked `zklora` by measuring the creation and verification times for our verifiable computations across multiple base models and private LoRA configurations. For each model, we computed a forward pass for a batch of 3 items with sequence length 5. Each experiment simulates the workflow where the *Base Model User* uses local data to evaluate the base model + LoRA combination, requesting remote LoRA computations from the *LoRA Owner*. We report total and average proof-generation times (`total_prove`, `avg_prove`) and verification times (`total_verify`, `avg_verify`), as well as relevant model parameters.

base_model	#lora	total_params	avg_params	total_prove	avg_prove	total_verify	avg_verify
distilgpt2	24	589824	24576.0	759.33	31.64	16.56	0.69
gpt2	48	2359296	49152.0	1675.58	34.91	32.79	0.68
Llama-3.2-1B	32	851968	26624.0	991.93	31.00	24.91	0.78
Llama-3.3-70B-Instr.	80	11796480	147456.0	3749.76	46.87	123.11	1.54
Llama-3.1-8B-Instr.	32	5242880	163840.0	1527.40	47.73	35.79	1.12
Mixtral-8x7B-Instr.	32	10485760	327680.0	2357.61	73.68	44.30	1.38

Table 1: Benchmark results for `zklora` showing split inference overhead on various model + LoRA combinations. Times are in seconds.

As shown in Table 1, proof-generation times tend to grow with model size, particularly for 70B-scale systems. However, the verification step remains fast, often averaging under two seconds per proof, even at large scales. This efficiency is crucial for real-world "trial runs", allowing the Base Model User to confirm performance on a validation set without incurring prohibitive overhead. In larger-scale usage scenarios, once trust in the LoRA module is established, `zklora` may no longer be needed.

5 Conclusion

`zklora` enables a *Base Model User* to verify the quality of a private LoRA module without revealing the LoRA weights. By pairing split inference with cryptographic proofs, `zklora` ensures that the user can trust the computed outputs during validation experiments. After verifying performance, the Base Model User may adopt any approach they choose to integrate or deploy the private LoRA more widely, since the module has been proven effective. Our experiments show this approach is feasible across a range of model sizes, with *fast verification times* minimizing the overhead of a verifiable evaluation phase. Future directions include further circuit optimization and exploring multi-owner scenarios, in which multiple private LoRA modules could be composed and verified within the same pipeline.

References

[1] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners, 2020.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[3] Ning Ding, Zhuosheng Zheng, Fei Tan, Yuxian Chen, Xipeng Xie, Zhiyang Liu, Xinze Dai, and et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models, 2022.

[4] Edward Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[5] Rahul Kothapalli et al. Hypernova: High-degree polynomial folding via customizable constraint systems, 2024.

[6] Abhiram Kumar, Mary Maller, Pratyush Mishra, Shashank Siri, Giovanni Tessaro, Pratik Vasudevan, and Yupeng Zhao. Nova: Recursive zero-knowledge arguments from folding schemes, 2022.

[7] Leslie G Valiant. Incrementally verifiable computation or ivc. `https://dash.harvard.edu/handle/1/5026950`, 2008. Harvard University, Technical Report.