zklora: Verifiable Multi-Party Inference for Evaluating Private LoRA Weights

Anonymous Authors

January 11, 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 multi-party 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 many tasks but can be expensive to store and fine-tune [1, 2]. Low-Rank Adaptation (LoRA) [4] addresses these costs by injecting small adapter matrices, making specialized fine-tuning more efficient [3].

However, if a *LoRA Owner* wants to keep these weights private while a *Base Model User* needs to validate them on a local dataset, a direct sharing of the LoRA parameters is impossible. `zklora` solves this dilemma by orchestrating multi-party inference: The Base Model User runs the base model layers locally, forwarding partial activations to the LoRA Owner for the private transform. A zero-knowledge proof is generated to demonstrate correctness, ensuring that final performance metrics are truly representative of the private LoRA.

2 Preliminary Results

We benchmarked `zklora` by measuring the creation and verification times across multiple base models and private LoRA configurations. Figures 1, 2, and 3 show the total LoRA parameter count (horizontal axis) versus:

- **Total Settings Time** (sec),
- **Total Proof Generation Time** (sec),
- **Total Verification Time** (sec),

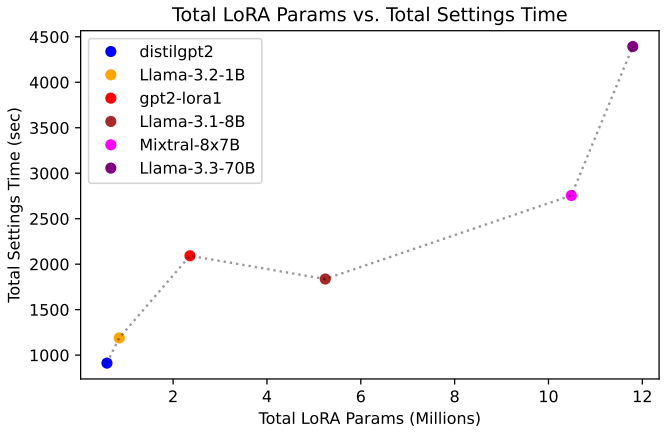

Figure 1: Total LoRA params (in millions) vs. total settings time. Each point is a different model.

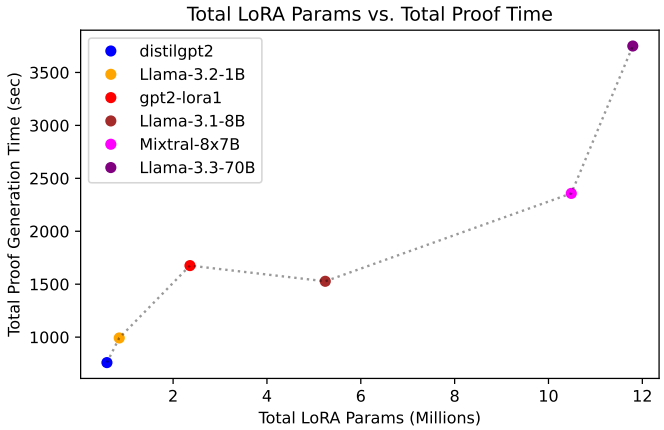

Figure 2: Total LoRA params (in millions) vs. total proof generation time. Each point is a different model.

respectively.

Notably, while proof-generation times can grow with model size, verification remains consistently fast (on the order of 1–2 seconds per proof for large models). This helps real-world "trial runs," letting the Base Model User confirm private LoRA quality without incurring excessive overhead.

3 Multi-Party Inference: zklora for Verifiable Evaluation

Our approach allows the *Base Model User* to incorporate a private LoRA module from the *LoRA Owner*:

1. **Base Model Forward Pass.** The user runs all layers without LoRA locally, producing partial activations.

2. **Forward to LoRA Adapter.** The partial activations are sent to the LoRA Owner for private transformations.

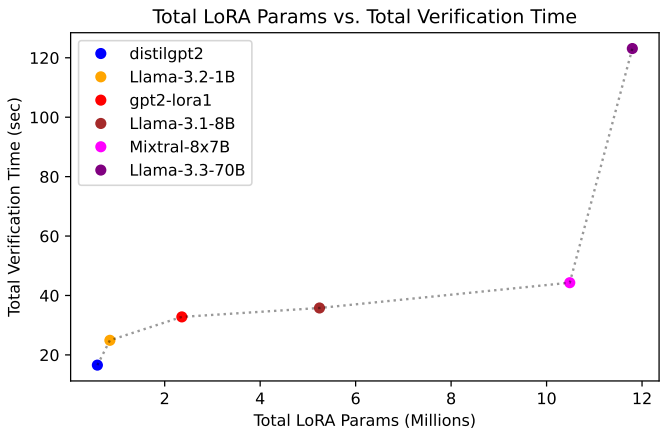

Figure 3: Total LoRA params (in millions) vs. total verification time. Each point is a different model.

3. **LoRA Computation & Proof Generation.** The LoRA Owner updates the activations and produces a zero-knowledge proof of correctness.

4. **Resuming the Forward Pass.** The updated activations are returned and merged into the local forward pass.

5. **Verification & Validation.** The Base Model User verifies the proof and measures final metrics (e.g., accuracy) to decide if the LoRA meets their needs.

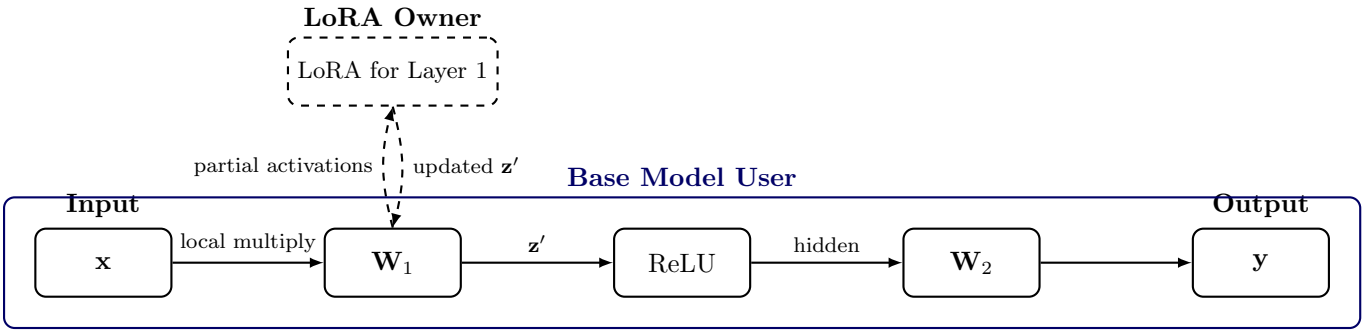

Figure 4: A simplified 2-layer MLP with LoRA on the first layer. The Base Model User (blue box) runs most computations locally, then requests the LoRA Owner (dashed box) to update partial activations.

4 Generating the Zero-Knowledge Proof

Central to `zklora` is the cryptographic proof that the LoRA computations were conducted faithfully. Below is a high-level outline:

Circuit Compilation (Model + JSON Input). The LoRA Owner provides an ONNX model that includes the LoRA parameters. We compile it into a "circuit" representing the layer compu-

tations. Next, the Base Model User's partial activations, saved as JSON, feed into this circuit as inputs.

Settings and Key Generation. We produce (1) a "settings" file that includes circuit metadata, (2) a proving key (PK) and verification key (VK), and optionally (3) a structured reference string (SRS) for polynomial commitments.

Witness Creation. We run the partial activations through the circuit, storing all intermediate wire values in a "witness" file that reflects the correct usage of the LoRA layer.

Proving. Using the PK, the witness, and the circuit, the LoRA Owner generates a proof file. This file certifies that the LoRA transformation was done correctly, matching the declared ONNX logic.

Verification. The Base Model User loads the proof and verifies it with the VK (plus SRS). If it checks out, the user trusts that the LoRA updates are accurate, without ever seeing the LoRA weights directly.

5 Related Work

5.1 Low-Rank Adaptation

LoRA [4] is widely adopted for parameter-efficient fine-tuning across GPT-2, GPT-Neo, and LLaMA. By focusing updates on low-rank adapters, it reduces memory overhead [3].

5.2 Incrementally Verifiable Computation

Earlier cryptographic frameworks [7, 6, 5] show how to prove correctness of multi-step computations. `zklora` combines these proof techniques with a multi-party inference flow, ensuring that LoRA transformations remain private yet verifiable.

6 Conclusion

We present `zklora` as a verifiable multi-party inference solution to validate private LoRA adapters. Our benchmarks show that proof generation scales with model size, while verification remains consistently quick. This enables the Base Model User to run iterative experiments on LoRA modules without risking the LoRA Owner's privacy. Future extensions might explore multi-owner LoRA composition or advanced multi-party computation frameworks that guard both user data and model parameters with cryptographic guarantees.

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.