ZKLoRA: Efficient Verification of LoRA-Base Model Compatibility for Large Language Models

Bidhan Roy, Peter Potash, Marcos Villagra
Bagel Research Team*
`bidhan@bagel.net, peter@bagel.net, marcos@bagel.net`

January 12, 2025

Abstract

Verifying that a private Low-Rank Adaptation (LoRA) module is truly compatible with a given large language model (LLM) can be daunting, especially when those LoRA parameters cannot be disclosed. We introduce `ZKLoRA`, a library for generating zero-knowledge proofs that demonstrate correct LoRA-base model compatibility *without sharing LoRA weights*. A key insight is that the *verification step* for each LoRA module remains on the order of **1–2 seconds**, enabling repeated or batch checking at scale. Our empirical evaluations confirm that `ZKLoRA` maintains this rapid per-module verification time even for multi-billion-parameter LLMs.

1 Introduction

Large Language Models (LLMs) have attained remarkable success [1, 2], but verifying fine-tuned modifications such as LoRA [4] can be difficult when the updated weights must remain private. Traditionally, one might re-run an entire forward pass or inspect thousands of parameters to ensure correctness, which is infeasible for massive models. `ZKLoRA` addresses this by generating a zero-knowledge proof of correctness for each LoRA module, guaranteeing that the private LoRA genuinely fits the public base model. Crucially, **the verification stage for each LoRA module** in `ZKLoRA` remains about 1–2 seconds, even at scales of multi-billion parameter base models. This rapid, per-module check allows the Base Model User to efficiently, independently verify the compatibility of the private LoRA parameters.

2 Preliminary Results

We benchmarked `ZKLoRA` across several LLMs and smaller models with varying LoRA parameter counts. Our focus is verifying that, regardless of total parameter scale, *the per-module verification step* remains swift. Figures 1, 2, and 3 illustrate how total LoRA parameters correlate with:

- **Verification Time per Module** (Base Model User's overhead),
- **Total Settings Time** (preparatory steps and key generation),
- **Total Proof Generation Time** (LoRA Owner's overhead).

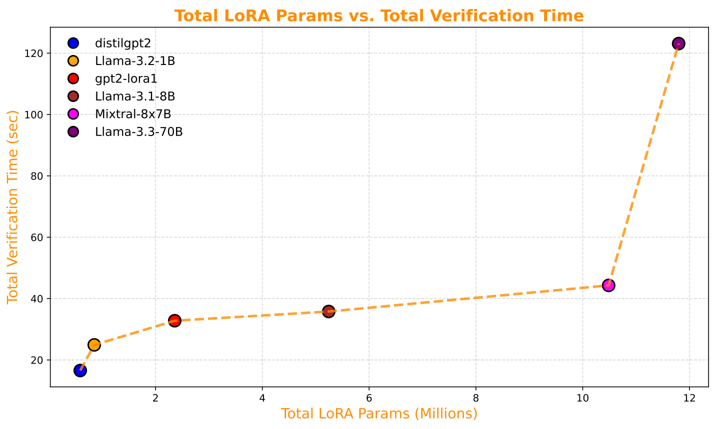

Figure 1: LoRA params (millions) vs. Total verification. Even at higher scales, verifying each LoRA module remains about 1–2 seconds. For example, even for a 70-billion parameter model, the total verification time of the additional LoRA parameters is only 2 minutes.

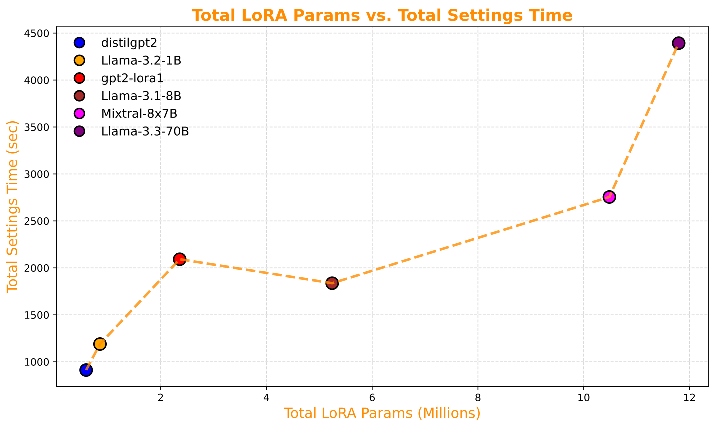

Figure 2: LoRA params (millions) vs. total settings time. Each point corresponds to a different LLM or smaller model.

Although proof generation may scale with parameter size, verification for each LoRA module is consistently on the order of seconds. We consider this *fast, per-module verification* to be `ZKLoRA`'s core advantage in production LLM environments.

3 Ensuring LoRA–Base Model Compatibility

A typical workflow involves:

1. **Base Model User** possessing a large LLM.
2. **LoRA Owner** having proprietary low-rank adapter weights.

However, these two parties must confirm the LoRA truly "fits" the base model. Using `ZKLoRA`, the LoRA Owner proves the adapter's compatibility *without* exposing private weights:

[1]Bagel is a research lab, making open source AI monetizable.

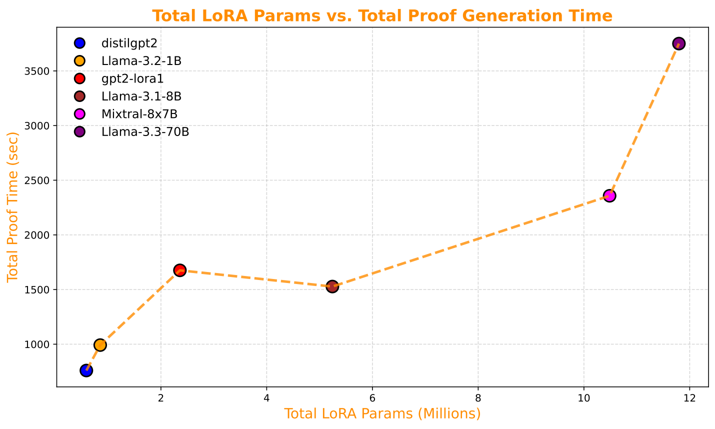

Figure 3: LoRA params (millions) vs. total proof generation time (LoRA Owner's overhead).

1. **Partial Forward Pass.** The Base Model User runs the unaltered layers of the LLM, producing partial activations.
2. **LoRA Transform + Proof Gen.** The LoRA Owner applies private LoRA updates, constructing a zero-knowledge proof to attest correct usage.
3. **Verification per LoRA Module (1–2 sec).** The user verifies each proof quickly, ensuring the final outputs reflect a valid LoRA for the base model.
4. **Evaluate.** With trust in the LoRA's correctness, the user measures performance (e.g., loss).

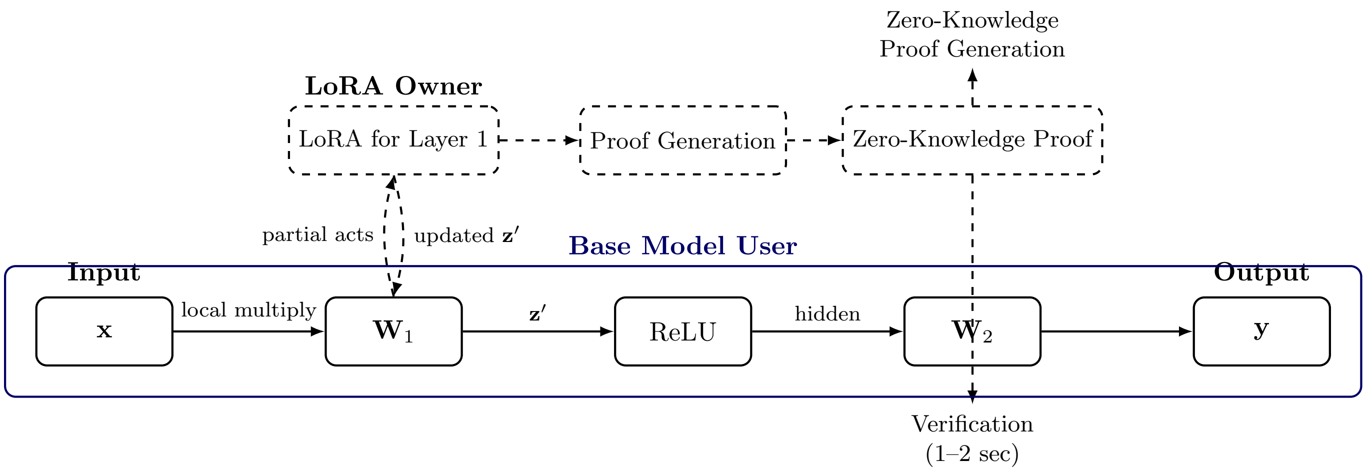

Figure 4: ZKLoRA inference flow on a 2-layer neural network where the first layer has private LoRA weights added. The LoRA Owner generates a zero-knowledge proof to ensure compatibility, and the Base Model User verifies this proof quickly.

4 Zero-Knowledge Proof Generation

ZKLoRA compiles an ONNX model (with LoRA included) into a constraint circuit. The user's partial activations feed the circuit as inputs:

(1) Circuit Compilation. We parse the LoRA-augmented layers, producing a cryptographic circuit capturing each parameter operation.

(2) Key Setup. A settings file, proving key, and verification key are generated. A structured reference string (SRS) may also be needed.

(3) Witness Creation. The LoRA Owner runs the partial activations through the circuit, tracking all wire values in a "witness."

(4) Proof. Using the witness plus proving key, the LoRA Owner constructs a proof.

(5) Verification per Module. Each LoRA module's proof is tested with the verification key, taking **1–2 seconds**. This minimal overhead is ZKLoRA's main advantage for frequent or large-scale checks.

5 Related Work

5.1 Low-Rank Adaptation (LoRA)

Low-Rank Adaptation (LoRA) [4] is a technique for parameter-efficient fine-tuning of large language models (LLMs) that injects small, low-rank adapter matrices into specific layers of a pre-trained model. By isolating the fine-tuning process to these low-rank components, LoRA drastically reduces memory overhead compared to full-model fine-tuning. This design choice is especially appealing for massive LLMs where training or even storing all parameters can be prohibitive [3].

5.2 Incrementally Verifiable Computation

In a decentralized world, trust is a resource that is hard to achieve. In decentralized computation, we need to make sure the computation are being done, and are being correctly. In a seminal paper by Valiant (2008) [7], it was shown that proofs of knowledge can be used to assert the correct execution of general computations. That is, if M is a machine that runs for t steps producing a sequence of configurations $c_0, c_1, \ldots, c_t$, then there exist an efficient and effective way to produce a computationally sound proof for the computation $c_0 \xrightarrow{t} c_t$. This idea is referred to as Incrementally Verifiable Computation or IVC.

The main goal is IVC is to produce compact, updatable proofs of correctness for a sequence of computations, so that each new step in the computation can be verified on its own while building on the guarantees of the previous steps. This technique significantly reduces the verification overhead for long or evolving computations, which is invaluable in scenarios like decentralized networks, outsourced computation, and any application requiring frequent correctness checks.

Kumar et al. (2021) [6] introduced the proof system NOVA and the idea of recursive proofs, which are proofs that can "prove the correctness of other proofs." Recursive proof composition is key to IVC where each proof attests to the correctness of both a step's output and the validity of the previous step's proof.

HyperNova [5] is a novel recursive argument system optimized for customizable constraint systems (CCS) that generalizes and improves upon prior approaches like Nova. It achieves efficiency through a folding scheme where the prover's cryptographic costs are minimized and achieves zero-knowledge without relying on zkSNARKs.

An IVC system allows the construction of proofs in zero-knowledge where the proofs reveal no information about the underlying computation or its inputs beyond the validity of the claim [7].

6 Conclusion

`ZKLoRA` provides a fast, robust mechanism to ensure that private LoRA modules remain consistent with a large base model. Our evaluations indicate that each LoRA module's correctness can be verified in about 1–2 seconds, even for multi-billion-parameter LLMs. This efficiency bridges the gap between privacy-preserving LoRA development and practical, real-time validation in large-scale deployments. Future expansions could integrate multi-owner LoRAs, advanced zero-knowledge proofs for further performance gains, or partial data-privacy frameworks to shield user inputs as well as LoRA parameters.

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.