ZKLoRA: Efficient Zero-Knowledge Proofs for LoRA Verification

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

January 17, 2025

Abstract

Parameter-efficient fine-tuning (PEFT) techniques, including Low-Rank Adaptation (LoRA), are widely adopted to customize large-scale language models in distributed training environments. A base model user may want to use LoRA weights developed by an external contributor, creating two requirements: the base model user must verify that the LoRA weights are effective when combined with the designated open-source model, and the contributor must protect proprietary parameters until compensation is assured.

We propose **ZKLoRA**, a zero-knowledge protocol that leverages polynomial commitments, succinct proofs, and advanced cryptographic designs to confirm LoRA–base model compatibility *without* disclosing private LoRA weights. Our evaluations show that verification requires only **1–2 seconds** per LoRA module for typical large models. This short verification time enables trust-driven collaboration across decentralized infrastructures and contract-based training pipelines, ensuring that the delivered LoRA is valid while preserving the contributor's intellectual property.

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.

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

2 Preliminary Results

We benchmarked `ZKLoRA` across several LLMs and smaller models with different numbers of LoRA modules. Our central question is how verification times, as well as settings and proof generation times, grow with **the number of LoRA modules**, while also considering each LoRA's average parameter size. Figure 1 and Table 1 detail this trade-off. [1]

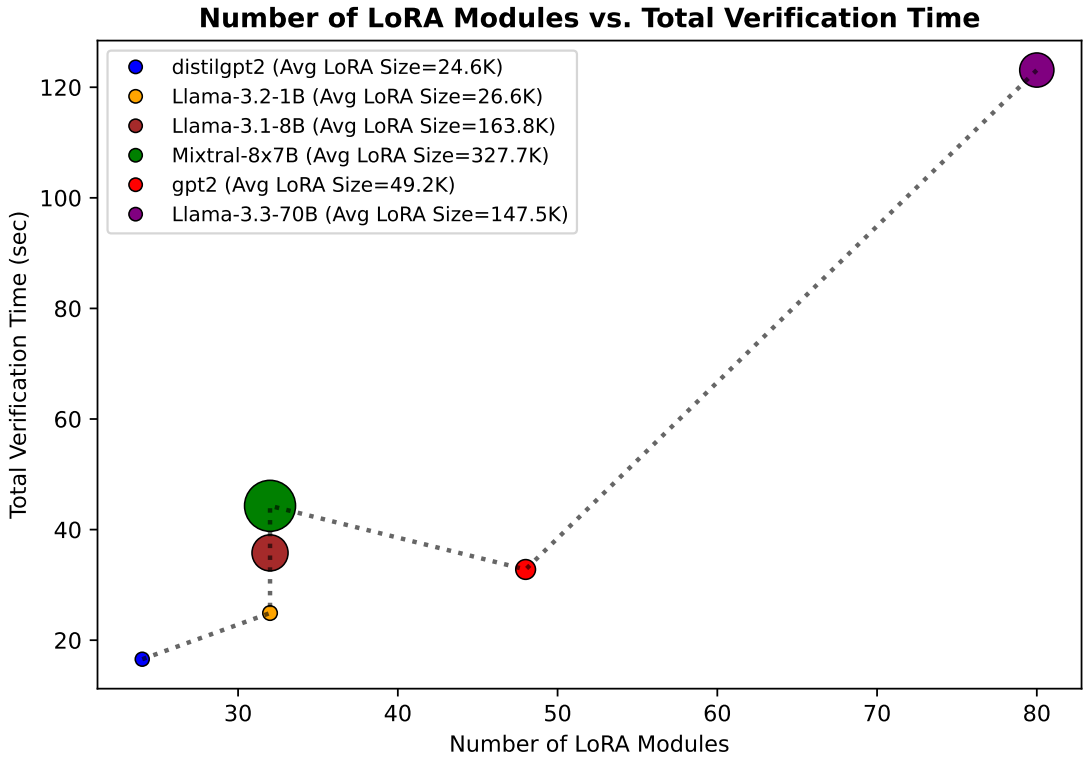

Figure 1: Total verification time (seconds) vs. number of LoRA modules, with dot size reflecting average LoRA size.

Base Model	# of LoRAs	Avg LoRA Size	Avg Settings	Avg Proof
distilgpt2	24	24576	38.0	31.6
openai-community/gpt2	48	49152	43.6	34.9
meta-llama/Llama-3.2-1B	32	26624	37.2	31.0
meta-llama/Llama-3.3-70B-Instruct	80	147456	54.9	46.9
meta-llama/Llama-3.1-8B-Instruct	32	163840	57.4	47.7
mistralai/Mixtral-8x7B-Instruct-v0.1	32	327680	86.1	73.7

Table 1: Model benchmark results for Settings and Proof generation time averaged by number of LoRA modules.

[1] Note that the number of LoRA modules in a given model is not purely a function of the number of layers – it is also a choice of which weight matrices within each layer is targeted. For example, just targeting one matrix per layer (ie the Query matrix in the Attention Block) will yeild one LoRA per layer. Conversely, targeting the Query, Key, and Value matrices yields three matrices per layer and makes the total number of LoRAs $3 \times num\_layers$.

From Figure 1, we see that models with a higher LoRA count (e.g., 80 modules for a large 70B model) can indeed lead to larger total verification time overall. However, the slope remains modest due to ZKLoRA's succinct design. For instance, even if each module is verified individually at around 1–2 seconds, verifying 80 modules can be done in just a few minutes, which is still practical for real-world usage.

Table 1 similarly shows that *proof generation* and *settings time* scale with the number of LoRA modules, but also depend heavily on each module's average size (indicated by larger or smaller dot size). These two steps (proof generation on the LoRA contributor's side and cryptographic circuit setup) can become more expensive, yet remain feasible in decentralized settings or paid contract relationships. The *Base Model User*, meanwhile, benefits from the relatively short verification overhead.

Overall, these plots confirm that ZKLoRA can handle multiple LoRA modules with minimal overhead for verifying correctness, emphasizing the viability of repeated or multi-adapter scenarios in large-scale LLM pipelines.

3 Ensuring LoRA–Base Model Compatibility

A typical workflow involves:

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

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

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 Contributor applies private LoRA forward passes based on partial activations, 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).

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.

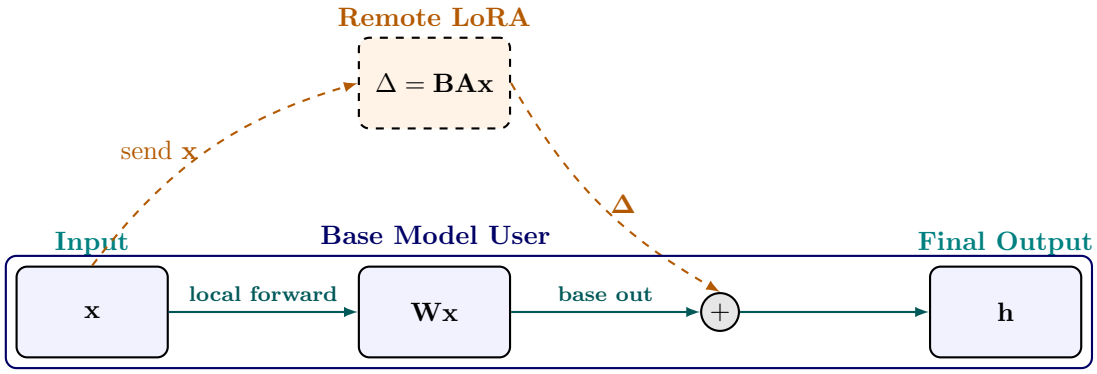

Figure 2: Flow in a Multi-Party Inference scenario between local base model and remote LoRA weights. The base model performs a local forward pass, computing $\mathbf{base\_out} = \mathbf{Wx}$. In parallel, the input $\mathbf{x}$ is sent to the remote LoRA module, which returns $\boldsymbol{\Delta} = \mathbf{BAx}$, where $\mathbf{B}, \mathbf{A}$ are the low-rank finetuned matrices. The final output is $\mathbf{base\_out} + \boldsymbol{\Delta}$.

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

(4) Proof. Using the witness plus proving key, the LoRA Contributor 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].

Beyond the clear advantage of reduced storage, LoRA also facilitates swapping multiple domain-specific adapters into a single base model, making it straightforward to maintain separate skill sets without instantiating an entire new copy of the model. These adapters can target specialized tasks (e.g., medical or legal text) with minimal overhead, driving LoRA's widespread adoption. Yet verifying that a proprietary LoRA truly aligns with a base model (without revealing the adapter) remains problematic—precisely the gap `ZKLoRA` fills.

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 computations are both done and done 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 of IVC is to produce compact, updatable proofs of correctness for a sequence of computations, so that each new step can be verified on its own while building on the guarantees of 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 less than 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. In terms of future work, the most relevant and immediate work would be adding polynomial commitments of the base model's activations (those that are sent as input to the LoRA contributor). This would take us one step closer to providing end-to-end verifiability of inference computation for LoRA-finetuned models. Other avenues of expansion could be integrating multi-contributor 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.