ZKLoRA: Efficient Zero-Knowledge Proofs for LoRA Verification

Bidhan Roy [*1] **Peter Potash** [*1] **Marcos Villagra** [*1]

Abstract

Low-Rank Adaptation (LoRA) is a widely adopted method for customizing large-scale language models. In distributed, untrusted training environments, an open source base model user may want to use LoRA weights created by an external contributor, leading to two requirements: (1) the base model user must confirm that the LoRA weights are effective when paired with the intended base model, and (2) the LoRA contributor must keep their proprietary weights private until certain conditions have been met that allow the LoRA contributor to release the weights.

We present ZKLoRA, a zero-knowledge verification protocol that relies on succinct proofs and our novel Multi-Party Inference procedure to verify LoRA–base model compatibility without exposing LoRA weights. ZKLoRA produces *deterministic* correctness guarantees and validates each LoRA module in only 1–2 seconds on state-of-the-art large language models. This low-latency approach enables nearly real-time verification and promotes secure collaboration among geographically decentralized teams and contract-based training pipelines. The protocol ensures that the delivered LoRA module works as claimed, safeguarding the contributor's intellectual property while providing the base model user with verification of compatibility and lineage.

1. Introduction

Large Language Models (LLMs) have attained remarkable success (Brown et al., 2020; Devlin et al., 2018), but verifying fine-tuned modifications such as LoRA (Hu et al., 2021) in an untrusted, distributed training environment 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 base model. Crucially, the verification stage for each LoRA module in `ZKLoRA` remains about 1–2 seconds, even at scales of multi-billion parameter, state-of-the-art large language base models.

Code for the implementation can be found at `https://github.com/bagel-org/zkLoRA`.

[1]Bagel Labs. Correspondence to: Bidhan Roy <bidhan@bagel.net>, Peter Potash <peter@bagel.net>, Marcos Villagra <marcos@bagel.net>.

Proceedings of the ICML 2025 Workshop on Championing Open-source Development in Machine Learning (CODEML '25).

2. Related Work

2.1. Low-Rank Adaptation (LoRA)

Low-Rank Adaptation (LoRA) (Hu et al., 2021) is a technique for parameter-efficient fine-tuning of large language models (LLMs) that injects small, low-rank adapter matrices into specific modules 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 (Ding et al., 2022).

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.

2.2. 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), 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

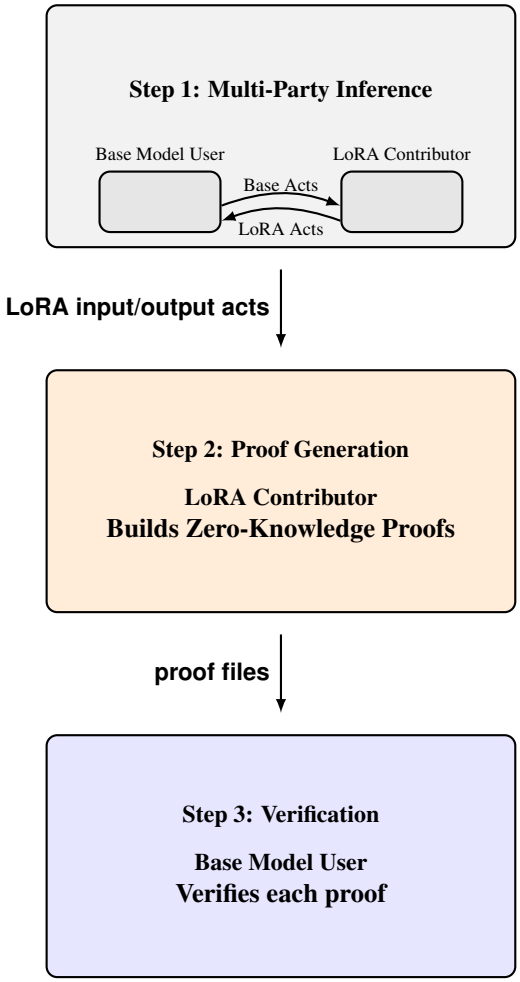

Figure 1. **Three-Step ZKLoRA Process (Vertical). (1)** Base Model User and LoRA Contributor exchange "Base Acts" and "LoRA Acts" in a multi-party inference. **(2)** The LoRA Contributor generates cryptographic proofs for correctness. **(3)** The Base Model User verifies these proofs, ensuring correct LoRA alignment without revealing private adapter weights.

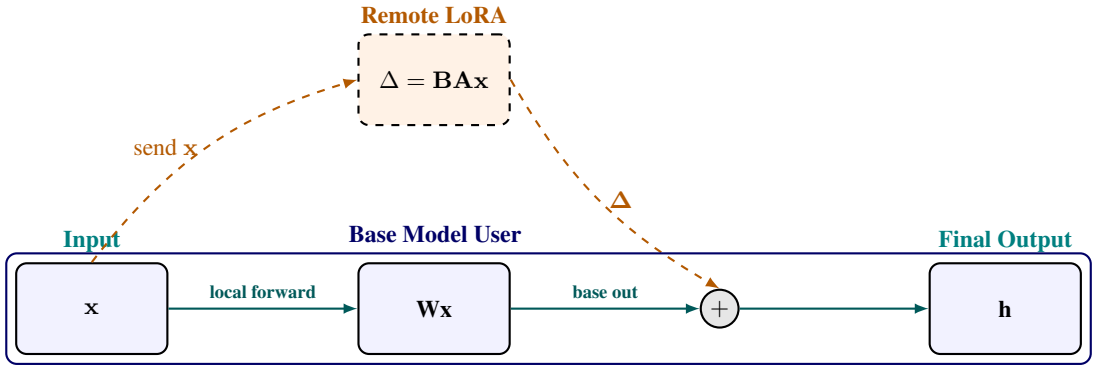

Figure 2. Flow in a Multi-Party Inference scenario between local base model and remote LoRA weights. The base model performs a local forward pass, computing $\mathbf{base\_out} = \mathbf{Wx}$. In parallel, the input $\mathbf{x}$ is sent to the remote LoRA module, which returns $\mathbf{\Delta} = \mathbf{BAx}$, where $\mathbf{B}$, $\mathbf{A}$ are the low-rank finetuned matrices. The final output is $\mathbf{base\_out} + \mathbf{\Delta}$. Note that $\mathbf{\Delta}$ is equivalent to the term "LoRA Acts" from the first box of Figure 1.

$c_0, c_1, \ldots, c_t$, then there exists 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.

Kothapalli et al. (2022) 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 (Kothapalli & Setty, 2024) 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 zk-SNARKs. 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 (Valiant, 2008).

3. ZKLoRA

ZKLoRA's design reflects the synergy between LoRA's parameter-efficiency and zero-knowledge cryptographic protocols: LoRA significantly shrinks the parameter footprint being proven, while the zero-knowledge aspect maintains confidentiality of the contributor's proprietary weights. By merging these ideas, ZKLoRA enables trust-driven collaboration across decentralized infrastructures, contract-based training, and other scenarios where proof-of-correctness is essential but the LoRA parameters remain private. Our approach also builds on incremental verification concepts (Valiant, 2008) and advanced proof systems such as Nova (Kothapalli et al., 2022) and HyperNova (Kothapalli & Setty, 2024), which allow us to scale proofs to large neural networks. Ultimately, this combination provides a practical pipeline for parameter-efficient fine-tuning while verifying correctness in a succinct and minimally intrusive manner.

We implement a protocol that not only supports multi-party inference with partial activations exchanged between a base model user and a LoRA contributor, but also produces cryptographic proofs that the LoRA transforms are valid. The overall workflow is shown in Figure 1, while Figure 2 gives a deeper look at how Multi-Party Inference with LoRAs functions within an individual module. In addition, pseudocode for ZKLoRA can be found in Appendix A. To begin the Multi-Party Inference, the base model user puts the dataset chosen for inference into the base model's first module. The forward pass continues through the base model until it hits a module that uses remote LoRA weights. When this occurs the base model user sends partial activations to the LoRA contributor for processing. These exchanged activations, shown conceptually in Figure 1, correspond to "Base Acts" from the base model user and "LoRA Acts" from the LoRA contributor.

After the multi-party inference finishes, the LoRA contributor shifts to a proof generation phase. At this stage, each LoRA module is compiled into a constraint system describing the LoRA transformations, and a key setup procedure yields the proving key, verification key, and possibly a structured reference string if the underlying zero-knowledge scheme requires one. The contributor then creates a "witness" by running partial activations through these constraints and finally produces the proof files.

Once the proof generation is done, the base model user receives each proof and runs a fast verification procedure, typically requiring about 1–2 seconds per module. As Figure 2 suggests, this does not require the LoRA contributor to reveal the actual low-rank matrices. Instead, the contributor only sends updates and proofs that these updates conform to the declared LoRA transformations. If any single proof fails, the base model user can reject the entire LoRA submission; otherwise, the system is accepted as consistent with the underlying base model.

4. Preliminary Results

We benchmarked ZKLoRA across several LLMs and smaller models with different numbers of LoRA modules. The input for inference is a batch of size 3 with sequence length 5. We keep the input dimensions fixed in order to focus on our primary question: 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 3 and Table 1 detail this trade-off.[2]

From Figure 3, 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

[2]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 yield one LoRA per layer. Conversely, targeting the Query, Key, and Value matrices yields three matrices per layer and makes the total number of LoRAs $3 \times num\_layers$.

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

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

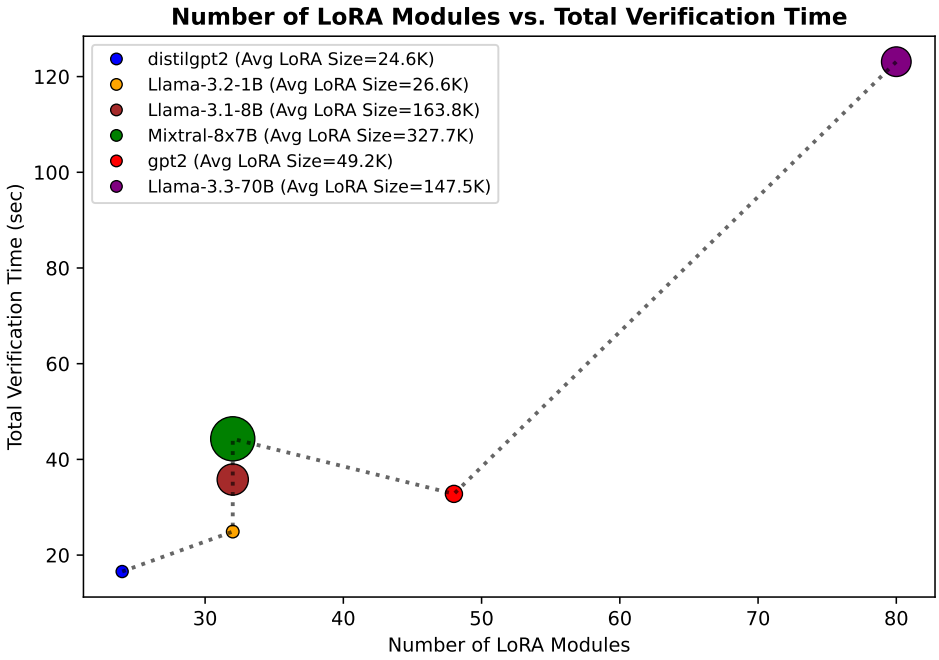

Figure 3. Total verification time (seconds) vs. number of LoRA modules, with dot size reflecting average LoRA size.

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 average *proof generation* and *settings time* scale with the size of a LoRA module, which, combined with the number of modules, gives the total times. 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 results confirm that ZKLoRA can handle large-scale implementation of LoRA modules with minimal overhead for verifying correctness, emphasizing the viability of repeated or multi-adapter scenarios in large-scale LLM pipelines.

5. Conclusions

ZKLoRA provides a fast, robust mechanism to ensure that private LoRA modules remain effective when combined with a large base model. Our evaluations indicate that each LoRA module's forward computation 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

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners, 2020.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

Ding, N., Zheng, Z., Tan, F., Chen, Y., Xie, X., Liu, Z., Dai, X., and et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models, 2022.

Hu, E., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. Lora: Low-rank adaptation of large language models, 2021.

Kothapalli, A. and Setty, S. Hypernova: Recursive arguments for customizable constraint systems. In *Annual International Cryptology Conference*, pp. 345–379. Springer, 2024.

Kothapalli, A., Setty, S., and Tzialla, I. Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*, pp. 359–388. Springer, 2022.

Valiant, L. G. Incrementally verifiable computation or IVC. https://dash.harvard.edu/handle/1/5026950, 2008. Harvard University, Technical Report.

A. ZKLoRA Pseudocode

In this section we present a pseudocode for ZKLoRA.

Algorithm 1 `ZKLoRA` Pseudocode

Require: *BaseModel* (public), *LoRAModel* (private), *Data*
Ensure: Verified outputs or `Reject`

Step 1: Multi-Party Inference
for each submodule s in *BaseModel* **do**
  if s **contains** LoRA layers **then**
    (a) run multi-party inference with LoRA Contributor for submodule s
  else
    (b) run local inference on submodule s (no remote calls)
  end if
end for

Step 2: Proof Generation
for each LoRA module m in *LoRAModel* **do**
  (1) Circuit Compilation: parse LoRA-augmented layers, produce cryptographic circuit
  (2) Key Setup: generate settings, proving key, verification key, and SRS if needed
  (3) Witness Creation: run partial activations through circuit, record wire values
  (4) Proof: construct zero-knowledge proof $\mathcal{P}_m$ for module m
end for

Step 3: Verification
for each proof $\mathcal{P}_m$ **do**
  if `Verify`($\mathcal{P}_m$) fails **then**
    return `Reject`
  end if
end for
return Verified outputs