Poster: Verification of Machine Learning Programs via Bitwise-Reproducible Operators

Arasu Arun[1,2], Adam St. Arnaud[1], Jacob Lin[1], Alexey Titov[1], Brian Wilcox[1],
Viktor Kolobarić[1], Marc Brinkmann[1], Oguzhan Ersoy[1], Ben Fielding[1], Joseph Bonneau[2]
[1]Gensyn [2]NYU

***Abstract*—Modern machine learning programs are extremely compute and memory intensive, requiring clients to outsource them to model providers. This outsourcing is typically done with no correctness guarantees, leaving clients vulnerable to attacks. Dishonest model providers could perform training-time attacks, such as backdoor insertion, and inference-time attacks, such as model downgrading. To solve this, we propose adapting the cryptographic notion of *refereed delegation* to the ML setting. This approach enables a client to delegate a program to multiple untrusted model providers, with a guarantee of obtaining the correct result if at least one of them is honest. This framework captures practical use cases like auditing and securing decentralized networks. We make the following two contributions to enable refereed delegation for ML programs: (1) Verde is an arbitration protocol that enables a computationally limited client to resolve disputes when model providers disagree on a program's output; (2) RepOps is a library that eliminates hardware "non-determinism" by controlling the order of floating-point operations performed on all hardware. This enables bitwise-reproducible inference and training of LLMs, including Llama-1B and Llama-8B, across various GPUs with practical overheads.** [1]

How can clients who outsource programs be assured that the results they receive is honest? Cryptographic proof systems like SNARKs offer the strongest guarantees to this problem, but they are extremely costly, incurring at least four orders of magnitude of overhead relative to the program itself. Heuristic, ML-specific techniques, such as Proof-of-Learning [4] or Proof-of-Training-Data [2], trade off the formal guarantees of cryptographic proofs for efficiency, making assumptions about the pattern of weight updates during training. These techniques thus lack rigorous security guarantees, and in fact, many practical attacks have been demonstrated [3].

Thus, we are still motivated to look for a method to provide concrete guarantees while imposing practical overheads on model providers. A simple approach is for a client to delegate the same task to multiple model providers. Naively, the client could take a majority vote and ensure correctness, assuming most of them are honest. In fact, the client can do better and ensure correctness if *any single provider* is honest. The key idea is efficient *dispute resolution*: if two providers produce different outputs, the client needs to determine which is incorrect.

A naive approach is to ask a trusted party, the "referee" (which could be the client themselves), to re-run the computation to determine the right output. Canetti et al. [1] shows an approach to significantly reduce the work done by the referee based on a simple observation: if two parties disagree on the output of a program, there must be some *point of divergence* in their computation path. They propose using interaction between the referee and the model providers and succinct *commitments* to program states to efficiently find the point of divergence via binary search, without sending the full computation trace to the referee. Once it is found, the referee need only re-execute a single computation step at the point of divergence, which by definition at least one of the model providers must have executed incorrectly. Hence, the referee can prove that at least one party is incorrect.

Refereed delegation is also referred to as "optimistic verification" and "bisection games" and has popularly been used to secure blockchain rollup protocols. In this work, we apply this model to ML programs, as shown in Figure 1, making the following contributions.

Contribution 1: Verde: Dispute resolution for ML.

The challenge: Existing refereed delegation protocols are designed for CPU programs and do not efficiently translate to the scale of modern neural networks. Program states, consisting of neural network parameters, are many gigabytes in size, and program steps involve highly parallelized code (often run on GPUs or similar hardware). Moreover, programs are represented as graph-based neural networks and are often compiled to run on hardware using proprietary compilers that do not provide machine-level code to developers.

Our contribution: We propose a dispute resolution protocol tailored to the graph-based computation of neural networks. The referee interacts with the disputing parties to narrow down the dispute to the diverging epoch, then divering training step, and finally down to the diverging ML operator in the graph. The referee only needs to compute this final operator to resolve the dispute. The model providers are only required to store and hash intermediate training checkpoints on top of re-executing small intervals of training, thus

1. This poster is built upon the work in https://arxiv.org/abs/2502.19405.

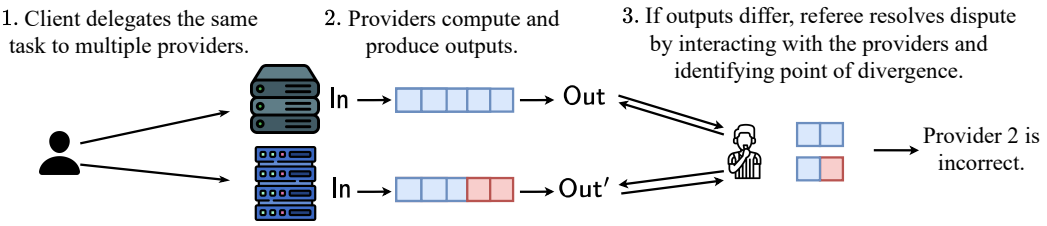


Figure 1: Refereed delegation with a client outsourcing the same task (that is, same model, same prompts, same training data) to two providers. Note that the model parameters are known to all parties involved. The blue and red bars indicate the computational transcript of the providers and where they diverged. If the outputs produced are different, the referee (which could be the client) interacts with the providers to determine the point of divergence and decide which party is incorrect.

incurring low overheads.

We implement Verde using the PyTorch framework (version 2.4) with neural network models defined as ONNX computational graphs [5]. We observe that there is a natural time-space trade-off for model trainers: for training a 1B or 8B model, the trainer is expected to store around a few hundred GBs or a few TBs of checkpoint data (with full FP32 precision), while incurring under a 6% in re-execution cost during dispute resolution. The referee is only required to re-execute one operator: even with a large sequence length, like $L = 8192$, this requires them to have just 8 GB of space, which is much lower than loading the models themselves.

Contribution 2: RepOps reproducible operators library.

The challenge: Ensuring bitwise reproducibility across hardware setups. The refereed delegation model assumes that honest servers will always compute the same result for the same program, but this may not be true if they are using different hardware. As floating point operations are not guaranteed to be associative (i.e., $(a+b)+c$ and $a+(b+c)$ could be different), the same program may result in different numerical results due to the architecture differences leading to different orders of floating point operations.

Our contribution. To overcome this, we design RepOps (Reproducible Operators), a library that implements bitwise reproducible versions of popular ML operators. It works by ensuring that floating point operations are performed in the same order, eliminating hardware non-determinism. We implement RepOps using the CUDA framework (version 12.1). All benchmarks are performed with FP32 precision.

Figures 2 and 3 outline our evaluation of RepOps. In brief, RepOps achieves a steady state overhead of about 60%-70% for large matrix multiplication, which is the bottleneck in ML programs. RepOps can also perform end-to-end inference and training of Llama 1B and 8B models with overheads of well under 100% in many settings. When working with smaller models or smaller-memory GPUs, the overhead can increase to over 300%.

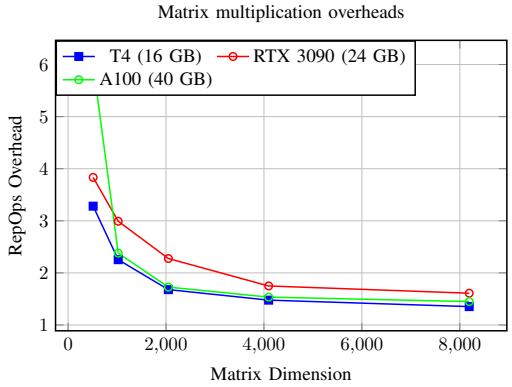


Figure 2: RepOps overhead for matrix-matrix multiplication. At larger matrix dimensions, RepOps matches or beats the performance of PyTorch with cuDNN.

Hardware	DistilBERT		Llama-1B	
	Infer.	Train.	Infer.	Train.
T4 (16 GB)	74%	258%	218%	374%
A100 (40 GB)	84%	312%	58%	67%
A100 (80 GB)	394%	354%	104%	54%

	Llama-8B	
	Infer.	LoRA.
A100 (80 GB)	98%	126%

Figure 3: RepOps training and inference overheads for DistilBERT, Llama-1B and Llama-8B.

References

[1] Ran Canetti, Ben Riva, and Guy N. Rothblum. Refereed delegation of computation. *Information and Computation*, 226:16–36, 2013. Special Issue: Information Security as a Resource.

[2] Dami Choi, Yonadav Shavit, and David Duvenaud. Tools for verifying neural models' training data. In *37th International Conference on Neural Information Processing Systems*, NeurIPS '23, 2024.

[3] Congyu Fang, Hengrui Jia, Anvith Thudi, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-Learning is Currently More Broken Than You Think. In *8th IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2023.

[4] Hengrui Jia, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-Learning: Definitions and Practice. *IEEE Symposium on Security and Privacy (S&P)*, pages 1039–1056, 2021.

[5] ONNX. Onnx: Open neural network exchange, 2017. Accessed: 2025-01-30.

Verde and RepOps: Verification of ML Programs via Bitwise-Reproducible Operators

Arasu Arun[1,2], Adam St. Arnaud[1], Jacob Lin[1], Alexey Titov[1], Brian Wilcox[1],
Viktor Kolobarić[1], Marc Brinkmann[1], Oguzhan Ersoy[1], Ben Fielding[1], Joseph Bonneau[2] [1]Gensyn [2]NYU

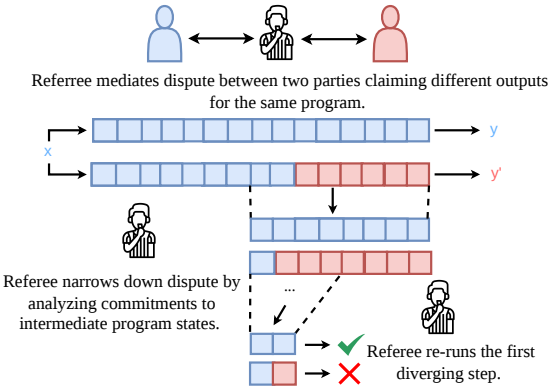
Two parties trained a model but got different outputs. How can we efficiently determine which party is incorrect?

Approach: Refereed Delegation (Canetti et al., 2013)

- Find first point of divergence via binary search on intermediate states
- Re-execute only that step to identify correct computation path
- Guarantee: If ≥ 1 party is honest, all incorrect outputs are identified

Challenges for refereed delegation of machine learning:

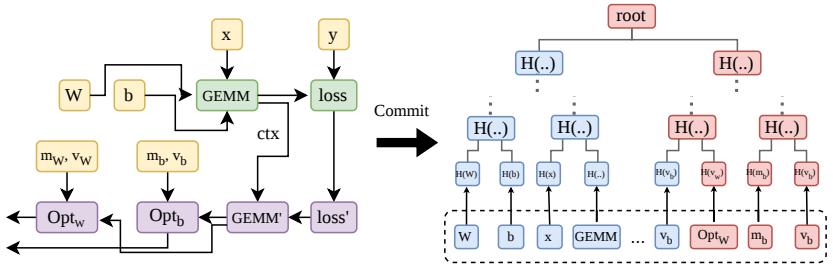
- **Scale:** LLMs are computational graphs, so we need commitments and dispute resolution over graph states
- **Non-determinism:** Honest parties may compute different numerical outputs when using GPUs



Verde: Dispute resolution for graph-based LLMs

Problem: even a single step is costly (e.g., Llama-8B ≈40 GB RAM)
Verde lets referees narrow disputes down further:

1. Augment graph nodes with I/O for binding
2. Commit to graph states via a Merkle tree
3. Use binary search to find first diverging node to re-execute



RepOps: Reproducible operators

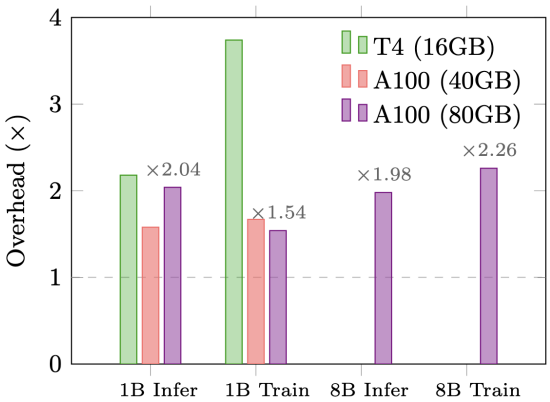
Floating point ops are not associative:
$(a + b) + c \neq a + (b + c)$

- Different hardware → different order → different outputs → different checkpoint hashes
- **RepOps** library: implements ML operations with a fixed operation order
- Matrix multiplication example:

```
# repops matrix multiplication
for i = 0 to M-1: # any order
    for j = 0 to N-1: # any order
        sum = 0
        for k = 0 to K-1: # fixed order
            a = A[i][k]
            b = B[k][j]
            sum = sum + a * b
        C[i][j] = sum
```

Verde and RepOps incur reasonable overheads

RepOps incurs 1–4× overhead for Llama 1B/8B.



Verde costs: hashing and storing checkpoints + re-execution during disputes. Llama 8B: ≈ 1 TB storage + 1–6% re-execution

Applications

Auditing Audits for attacks (e.g., data poisoning) can be performed across hardware setups, with dispute resolution

Volunteer compute networks SETI@home-style inference & training; volunteer nodes audit one other via Verde

Decentralized blockchains Enable verifiable model inference on chains like Ethereum

Reproducibility of research Replicate experimental results across heterogeneous hardware setups

Future Work

- **Model parallelism:** Extend Verde, RepOps to use pipeline parallelism
- **More hardware support:** Currently only CPUs and Nvidia GPUs

Verde paper | RepOps repo