Reliable and Cost-Efficient Dataset Storage for Distributed Machine Learning

1. Why the classical approach is outdated

Traditional dataset storage for ML often relies on full replication across several storage nodes. This leads to:

- **Massive disk overhead:** triple replication adds +200% storage cost;
- **Failure sensitivity:** a node or data-center outage can stall the pipeline for hours or days;
- **Trusted-operator model:** you must blindly trust storage providers;
- **Poor fit for federated ML:** regional copies multiply the cost.

2. Why do we need a new method?

Key challenges in distributed ML training

Modern pipelines must simultaneously:

- **Save space:** triple replication doubles your bill;
- **Stay available:** losing two or three nodes must not stop training;
- **Be verifiable:** storage integrity needs cryptographic guarantees, not blind faith.

Goal

Build a storage layer that simultaneously:

1. Keeps overhead within 20–40%;
2. Survives node failures with zero data loss;
3. Lets any client verify integrity without trusted parties.

(Plain: We want it **cheap**, **reliable** & **transparent** — all at once.)

3. High-level architecture

Our approach combines four foundational building blocks working as one cohesive system:

- **Erasure Coding** — slashes storage cost while preserving durability;
- **Leaderless Byzantine Consensus** — manages shards without central authority;
- **Zero-Knowledge Proofs (ZKP)** — prove data are stored correctly without revealing them;
- **On-Chain Anchor** — makes metadata tamper-proof and publicly auditable.

Together they form a **secure, scalable and verifiable pipeline**, from encoding to model training.

(Plain: Goal: marry distributed-systems reliability with cryptographic transparency.)

4. Component 1: Erasure Coding

Instead of three full copies we apply **erasure coding** — a more economical yet durable redundancy scheme.

How it works

- Split data into k data shards;
- Add $n-k$ parity shards via Reed–Solomon;
- Any k of n shards reconstruct the original.

Benefits

- Survives multiple node failures;
- Overhead only +37% (example k=8, n=11) vs +200% replication;
- Parallel fetch accelerates training.

(Plain: Less traffic, less storage — same durability.)

5. Component 2: Leaderless Byzantine Consensus

Problem: which nodes store which shards, and what if some nodes misbehave?

Solution: an asynchronous BFT protocol with no single leader:

- Every node votes — no single point of failure;
- Tolerates up to $\lfloor (n-1)/3 \rfloor$ faulty or malicious nodes;
- Cluster state and re-balancing are entirely decentralised.

(Plain: Leaderless consensus reliability without a centre.)

6. Component 3: Zero-Knowledge Proofs of Storage

How to be sure a remote node actually stores your shard without re-downloading it?

Idea: the node pre-publishes a cryptographic commitment; on request it returns a **compact ZK proof** of possession.

Key facts

- Proof size 128 bytes;
- Client verifies in 5 ms *without* the shard itself;

(Plain: Store honestly → prove instantly → no blind trust required.)

7. Component 4: On-Chain Anchor

For independent, tamper-proof verification the system stores key metadata on-chain.

Recorded on chain

- Merkle roots of all current commitments;
- Verifying keys for ZK proofs.

Why it matters

- Any client can verify a proof against the public root;
- Full history available for audit;

- Write cost on an L2 chain (e.g., zkEVM) — <0.50.

(Plain: The blockchain stores no data — only anchors the truth.)

8. End-to-End: full cycle

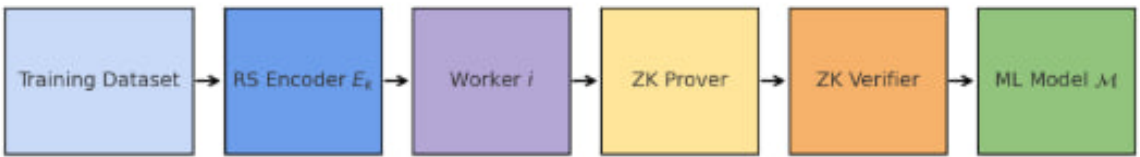

Figure 1: End-to-end data flow: the dataset is erasure-encoded, shards are distributed by a leaderless BFT cluster, their commitments are anchored on-chain, storage nodes return compact ZK proofs, and the client decodes the verified shards for model training.

9. Key implementation highlights

Implementation focuses on modularity and smooth integration with existing ML pipelines:

- **Shard format:** every sample is encoded into an independent shard set;
- **Coding layer:** Reed–Solomon with $k = 8, n = 11$ for durability;
- **Storage fetch:** shards treated as distributed objects, assembled on demand;
- **ZK check:** client verifies proof before download;
- **Integration:** verified tensors flow into `PyTorch DataLoader`;
- **Extensibility:** plug-and-play codecs, models, storage back-ends.

(Plain: The pipeline acts as an adaptable layer on top of mainstream ML tooling.)

10. Data recovery and model feed

Each training session starts with fetching and checking shards:

- Client requests an object ID and receives shards from multiple nodes;
- Each shard arrives with its ZK proof;
- After validation exactly k shards are downloaded;
- Original sample reconstructed via RS-decode;
- Resulting tensor is fed into the PyTorch model.

(Plain: The model learns only from verified data, even if part of storage is faulty.)

11. Test results

On a LeNet-5 model trained on protected data:

- Validation accuracy reached **99%** by epoch 7;
- Training loss dropped from **0.35** to **0.019** in 10 epochs;
- ZK proof verification takes **0.2 ms** and does not slow training;
- Overall storage reliability: >98.9% with overhead $1.37\times$.

Epoch	Train Loss	Train Acc	Val Loss	Val Acc
1	0.3459	90.04%	0.0977	97.03%
2	0.0919	97.14%	0.0587	98.12%
3	0.0611	98.12%	0.0476	98.47%
4	0.0473	98.53%	0.0364	98.86%
5	0.0380	98.81%	0.0360	98.77%
6	0.0332	98.92%	0.0362	98.81%
7	0.0278	99.11%	0.0318	99.00%
8	0.0255	99.23%	0.0344	98.83%
9	0.0216	99.30%	0.0338	98.97%
10	0.0194	99.37%	0.0297	98.98%

Table 1: Training and validation metrics per epoch

12. Visual results advantages

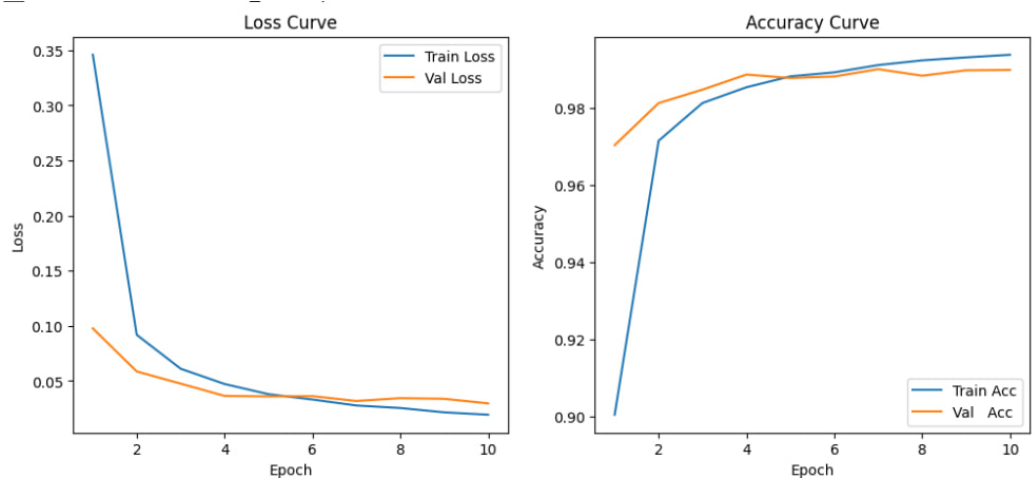

Figure 2: Decode time and reliability visualised.

13. Conclusion

We presented a storage architecture that merges modern cryptography with fault-tolerant distributed design, enabling reliable and verifiable ML pipelines at scale.
By replacing triple replication with efficient erasure coding, substituting trust in nodes with zero-knowledge proofs, and anchoring metadata on-chain, the system achieves security, durability and cost efficiency simultaneously.
This paves the way for machine learning where **security, reliability and economy coexist**, with no need to sacrifice one for another.