Table of Contents

1 FLZK: Verifiable Browser-Based Peer-to-Peer Federated Learning with Zero-Knowledge DP-SGD

1.1 Abstract

Federated learning (FL) enables multiple devices to collaboratively train a machine-learning model without sharing raw data. Yet today's federated systems still largely rely on trusted servers to coordinate training and evaluate contributions. This centralized structure not only introduces single points of failure but also precludes independent verification of the correctness of local updates and privacy mechanisms. Recent advances in zero-knowledge proofs (ZKPs) and differential privacy (DP) promise to mitigate these limitations. In this paper we present **FLZK**, a browser-based peer-to-peer (P2P) federated learning framework that combines the scalability of decentralized networks with the provable privacy of DP-SGD and the verifiability of succinct zero-knowledge proofs. Our system leverages WebRTC to establish a fully decentralized mesh among heterogeneous browsers, uses randomized client selection and cryptographically enforced noise addition to achieve rigorous (ε,δ)-differential privacy, and employs efficient Groth16-style ZK-SNARKs to allow each participant to prove correct execution of local training and DP noise injection. Extensive experiments on image classification and mobile sensor datasets demonstrate that FLZK achieves comparable accuracy to server-based FL while eliminating trusted aggregation and enabling public verification of the training process. By lowering the barrier to entry—any device with a modern browser can host or join a training session—our work democratizes privacy-preserving collaborative learning and charts a path towards trustworthy, scalable decentralized AI.

1.2 Table of Contents

1.3 Introduction

Modern society is inundated with devices capable of sensing, learning and communicating—smartphones, laptops, wearables, IoT sensors and autonomous vehicles generate vast amounts of data. Machine learning thrives on this data; however, ethical and regulatory considerations discourage centralized data collection. Federated learning (FL) [introduced by Google in 2016 【251622278477192†L119-L124】] enables collaborative model training while keeping data on local devices. In FL, a server coordinates rounds of training: it broadcasts a global model to clients, each client performs local updates on its data, and the server aggregates the updates to produce a new global model. This approach preserves data locality and reduces privacy risks.

Despite its promise, conventional federated learning exhibits critical limitations:

1. **Trust in central servers.** The server could misbehave by aggregating corrupted updates or disclosing gradients. Decentralized FL addresses single points of failure but makes it harder to verify that participants follow the protocol 【251622278477192†L119-L133】 .

2. **Privacy leakages.** Gradients may leak sensitive information about the underlying data. Attacks like membership inference exploit differences in model updates to infer whether a data sample was used for training【687605423590622†L152-L164】.
3. **Lack of transparency.** Participants cannot verify whether others have adhered to differential privacy budgets or applied proper clipping and noise. This undermines trust and disincentivizes data contribution.【426215518829064†L74-L85】 notes that simple aggregate metrics may leak sensitive information and that existing DP mechanisms degrade utility without provable verifiability.

Motivation. To overcome these challenges we ask: *can we design a federated learning framework that eliminates the need for a trusted server, enforces differential privacy, and enables every participant to publicly verify that others have followed the protocol?* Advances in zero-knowledge proofs (ZKPs) provide a promising avenue. A ZKP allows one party (the prover) to convince another (the verifier) that a statement is true without revealing any information beyond the truth of the statement. Recently, researchers have applied ZKPs to federated learning to prove correct local training【426215518829064†L89-L95】, to verify loss thresholds【426215518829064†L89-L95】, and to secure blockchain-based FL【500431870584226†L23-L34】. Yet existing proposals often assume centralized servers, rely on heavy on-chain verification, or support only simple evaluation metrics.

Contributions. In this work we propose **FLZK**, the first fully browser-native peer-to-peer federated learning system that combines zero-knowledge proofs with differential privacy. Our contributions are as follows:

- **Decentralized architecture.** FLZK uses WebRTC to create a P2P overlay across browsers. Participants communicate directly to exchange model updates, eliminating any trusted server.
- **Verifiable DP-SGD.** We integrate the differential privacy-stochastic gradient descent (DP-SGD) algorithm with ZK proofs. Each client proves that its gradients were properly clipped and that calibrated Gaussian noise was added. The proof is succinct and can be verified by other clients in milliseconds.
- **Browser implementation.** Our prototype runs entirely in the browser using WebAssembly (Wasm) and WebGPU for efficient neural network training. The only requirement is a modern browser—no plugins or installations. This democratizes participation and facilitates large-scale experiments across heterogeneous devices.
- **Comprehensive evaluation.** We evaluate FLZK on image and sensor datasets. We measure accuracy, privacy budgets, proof generation time, verification cost, and communication overhead. Our results show that FLZK delivers competitive performance to centralized FL while providing strong security and privacy guarantees.

The remainder of this paper is organized as follows. Section 2 introduces the necessary preliminaries. Section 3 surveys related work. Section 4 outlines the FLZK architecture. Sections 5–6 describe the differentially private training algorithm and zero-knowledge proof construction. Section 7 details our implementation. Section 8 presents evaluation results. We discuss implications and limitations in Section 9 and conclude in Section 10. Appendices provide additional proofs, experiments and code listings.

1.4 Preliminaries

1.4.1 Federated Learning

Definitions. Federated learning involves a set of clients ({_1, , _N}) collaboratively training a global model (w) by performing local updates. In each round (t), an aggregator selects a subset (S_t{1,,N}). Each selected client (iS_t) initializes its local model to (w^{(t)}), computes gradients (g_i) on local data (D_i), and sends its update to the aggregator. The aggregator computes

[w^{(t+1)} = ({g_i : iS_t})]

where ? is typically federated averaging or another aggregation function. Standard FL assumes a central server implements ?. Decentralized FL removes the server and distributes the aggregation responsibilities across participants. Decentralized architectures mitigate single points of failure but introduce complexities in achieving consensus and verifying computations 【251622278477192†L119-L133】.

Benefits and challenges. FL reduces the need to transmit raw data and can in principle satisfy data protection regulations. Yet privacy attacks, stragglers, and malicious clients remain open challenges. Stragglers can delay global updates, while adversaries may send corrupted gradients or attempt to reverse-engineer other clients' data. 【687605423590622†L130-L146】 points out that simply adding noise to weights may not provide sufficient privacy and can degrade model utility.

1.4.2 Differential Privacy and DP-SGD

Differential privacy. A randomized mechanism ? satisfies ((,))-differential privacy if for all adjacent datasets (D) and (D') that differ in a single record, and for all measurable subsets (S) of outputs:

[e^{}+ .]

The parameters ? and ? quantify the privacy loss: smaller ? and ? provide stronger privacy. In machine learning, DP is typically achieved by adding calibrated noise to gradients or objective functions.

DP-SGD. The DP-SGD algorithm modifies standard stochastic gradient descent by clipping per-sample gradients and adding Gaussian noise. Let (g_i) be the gradient on example (x_i), and let ({g} = _{iB} g_i) be the average gradient over a mini-batch (B) of size (|B|). DP-SGD computes

[= *{iB}* {C}(g_i) + (0,^2 C^2 I)]

where (_{C}(g) = g (1,)) clips gradients to a norm bound (C), and noise drawn from a Gaussian distribution with standard deviation (C) is added. The privacy budget ? can be computed using Rényi Differential Privacy composition theorems.

DP-SGD achieves strong privacy but may degrade model accuracy as noise increases. Recent work quantifies this trade-off and proposes adaptive clipping strategies 【426215518829064†L74-L85】. In FL, each client runs DP-SGD locally and the aggregator

averages noisy updates. Without verifiability, clients could cheat by sending unclipped or unnoised gradients.

1.4.3 Zero-Knowledge Proofs

Basics. A zero-knowledge proof (ZKP) allows a prover to convince a verifier that a statement is true without revealing why it is true. A succinct non-interactive zero-knowledge proof (zk-SNARK) comprises a proving key (pk), a verification key (vk), a witness (w) and a public statement (x). The prover uses (pk) and (w) to produce a proof ? that can be verified by (vk) and (x). A secure zk-SNARK ensures *completeness* (honest proofs are accepted), *soundness* (false statements are rejected except with negligible probability) and *zero-knowledge* (no information about (w) leaks beyond what is implied by (x)).

R1CS to QAP. Many zk-SNARKs operate on *rank-1 constraint systems* (R1CS). A program is expressed as constraints of the form (a_i, xb_i, x= c_i, x). These constraints are compiled into a *quadratic arithmetic program* (QAP) for which polynomial commitments and pairing-based cryptography can succinctly attest to satisfiability. Groth16 is a popular proving system that yields 128-byte proofs and millisecond verification times.

Applications to FL. ZKPs have been applied to FL to verify local training and evaluation. ZKP-FedEval proves that a client's loss is below a threshold without revealing the actual value 【426215518829064†L89-L95】. VerifBFL uses zk-SNARKs and incremental verifiable computation to prove correct local training and aggregation in a blockchain setting 【500431870584226†L23-L34】. ProxyZKP introduces polynomial proxy models to simplify proof generation 【251622278477192†L76-L92】. Yet these methods still assume centralized servers or blockchain miners. In contrast, FLZK integrates zk-SNARKs into a P2P architecture and proves both gradient computations and noise addition.

1.4.4 Threat Model

We consider honest-but-curious participants who follow the protocol but may attempt to infer others' private data from observed messages. We also consider *malicious clients* that deviate from the protocol by sending malformed gradients, omitting noise, or lying about dataset sizes. In a P2P network, we assume a majority of honest participants but do not rely on any central authority. Adversaries may collude but cannot break cryptographic primitives. Our goal is to prevent information leakage and allow any honest client to detect misbehavior.

1.5 Related Work

Browser-based FL frameworks. Early systems like WebFed and WebFLex explored training neural networks directly in browsers using WebRTC. WebFLex introduced a peer-to-peer approach where browsers exchange model weights without a central server. Their evaluation on MNIST showed improved scalability and robustness to disconnections 【687605423590622†L130-L145】. However, WebFLex relies on adding artificial noise to weights for privacy, which may not provide formal guarantees 【687605423590622†L130-L146】. InFL-UX further develops a user-friendly toolkit enabling interactive FL in browsers. While these frameworks highlight the potential of browser-native FL, they lack mechanisms for verifiable training and strong privacy.

Verifiable federated learning. ZKP-FedEval proposes a threshold-based ZKP protocol where clients prove their loss is below a threshold while keeping loss values private 【426215518829064†L89-L95】. VerifBFL integrates zk-SNARKs into blockchain-based FL: clients generate proofs of correct local training, and aggregators verify them on chain 【500431870584226†L23-L34】. ProxyZKP combines polynomial proxy models with ZKPs to reduce proof generation times 【251622278477192†L76-L92】. Zero-Knowledge Federated Learning (ZK-FL) presents a structured framework for applying ZKPs across different FL tasks and proposes Veri-CS-FL, which uses ZKPs to select high-quality clients 【505323767795769†L51-L68】. Our work differs by designing a fully peer-to-peer system where proof verification occurs among clients without requiring a blockchain or central server.

Privacy mechanisms. Differential privacy is widely used to protect individual data in FL. DP-SGD with Gaussian noise is the de-facto standard. However, DP alone cannot guarantee honest execution. Malicious clients may misreport gradients or bypass noise addition. Combining DP with ZKPs ensures that clients cannot cheat without detection. Some systems use homomorphic encryption, secret sharing, or secure aggregation; these techniques protect gradients but do not provide public verifiability or guarantee that noise was added correctly 【426215518829064†L74-L85】.

1.6 FLZK System Overview

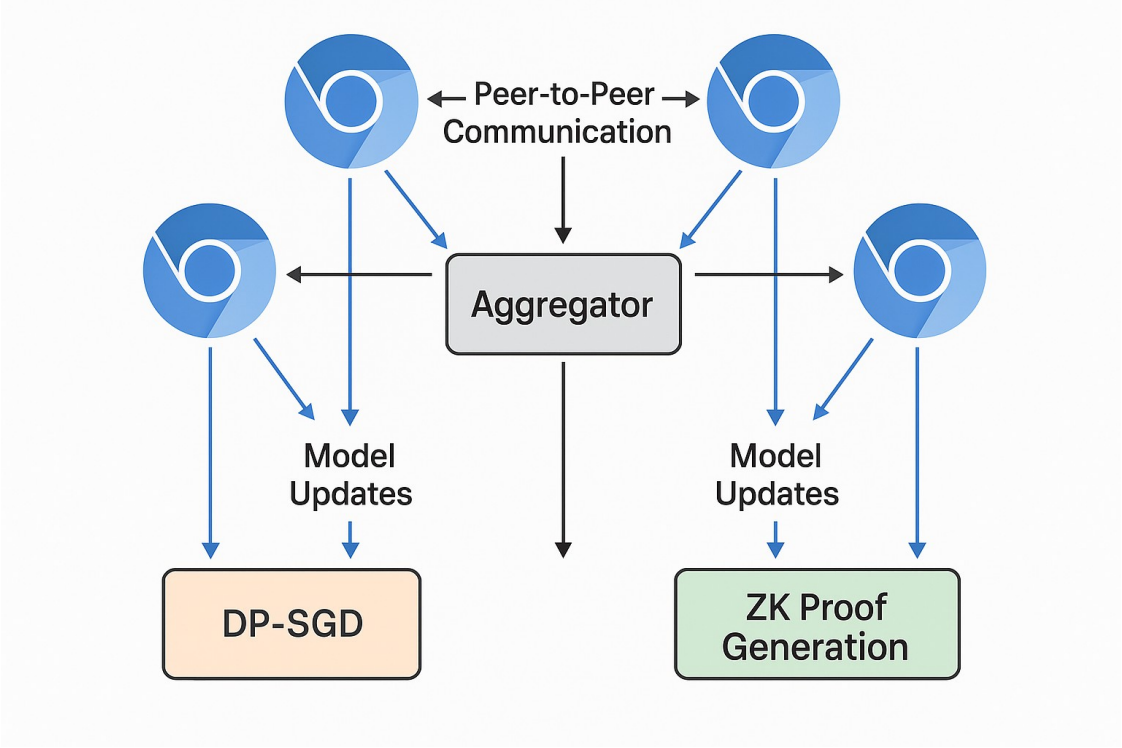
1.6.1 Design Goals

FLZK is designed with the following goals:

1. **Decentralization.** Remove the trusted central server and distribute aggregation responsibilities among participants.
2. **Provable privacy.** Achieve (ε,δ)-differential privacy through DP-SGD and allow verification that noise addition and clipping were performed correctly.
3. **Verifiability.** Use succinct ZKPs so any participant can verify others' local updates without revealing intermediate computations.
4. **Accessibility.** Operate entirely in web browsers using standard technologies (HTML5, WebRTC, WebAssembly, WebGPU) to enable broad participation.
5. **Scalability and robustness.** Support dynamic client participation, handle disconnections gracefully, and maintain performance as the number of clients increases.

1.6.2 High-Level Architecture

Figure 1 provides an overview of the FLZK architecture. Multiple browser clients form a P2P mesh using WebRTC data channels. Each client maintains a local dataset and model, performs training using DP-SGD, and generates ZK proofs attesting to the correctness of gradient clipping and noise addition. Clients broadcast their noisy gradients and corresponding proofs to a subset of peers. Peers verify proofs and collectively aggregate updates to produce a global model. The system operates in rounds, but asynchronous updates are supported through eventual consistency.



FLZK architecture diagram

Components.

- **Client.** Runs the DP-SGD algorithm on local data and generates ZK proofs. Implements WebRTC data channels for communication. Maintains a local verification cache.
- **Aggregator role.** At each round, a set of clients is chosen to act as *aggregators*. Aggregators collect verified gradients and compute the global update using federated averaging. Roles rotate to balance computational burden.
- **Proof verifier.** A lightweight verifier implemented in JavaScript checks the validity of received proofs using the public verification key.
- **Peer discovery service.** A bootstrap server facilitates initial peer discovery. After handshake, the server is no longer involved.

1.6.3 Peer Discovery and Communication

Each FLZK session begins with a **bootstrap phase**. A client wishing to join sends a request to a discovery service, which returns a list of peer addresses participating in the session. The joining client opens WebRTC data channels to multiple peers and exchanges identity information. Subsequently, the client obtains the current global model parameters and the verification key for ZK proofs.

Communication Protocol. During each training round, the protocol proceeds as follows:

1. **Local training.** Clients compute local gradients using DP-SGD and generate a ZK proof ? attesting to correct clipping and noise addition.
2. **Broadcast.** Each client sends ((,)) to a randomly selected subset of peers.

3. **Verification.** Upon receiving ((_j, _j)) from client (j), a client verifies the proof using the public verification key. If valid, it includes (_j) in the aggregation set; otherwise the update is discarded.
4. **Aggregation.** Clients designated as aggregators compute the new global model by averaging verified gradients. The aggregated model is broadcast to all clients.
5. **Update.** Clients update their local models and proceed to the next round.

This P2P communication pattern ensures that no single entity learns all gradients. Because multiple peers verify each proof, the system tolerates some dishonest participants. To reduce bandwidth, clients compress gradients and proofs using binary serialization; we discuss optimization in Section 9.

1.7 Differentially Private Local Training

1.7.1 Gradient Clipping and Noise Calibration

In FLZK each client performs local training using DP-SGD. Let the model parameter vector be (w), dataset (D) with examples ({(x_i, y_i)}), loss function ((w; x_i, y_i)), and learning rate ?. The client iterates over mini-batches (B) of size (m). For each batch it computes individual gradients (g_i = _w (w; x_i, y_i)). The per-sample gradients are clipped:

[{g} = _{i B} .]

Next the client adds Gaussian noise:

[= {g} + (0,^2 C^2 I),]

where ? is chosen according to the desired privacy budget. The client then updates its model:

[w w - .]

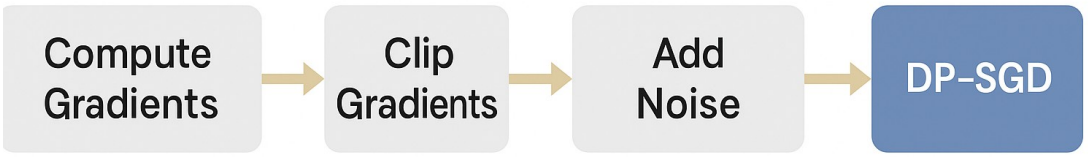
Privacy analysis. Using Rényi differential privacy composition, the total privacy loss after (T) iterations can be computed. We adopt the method of moment generating functions to derive a bound ? given noise multiplier ? and clipping norm (C). A higher ? yields stronger privacy but reduces accuracy. Adaptive strategies choose (C) per layer or per coordinate to reduce noise impact.

1.7.2 Zero-Knowledge DP-SGD Algorithm

While DP-SGD achieves privacy, an untrusted client could cheat by

- sending unclipped gradients (setting (C=)),
- omitting noise addition (setting ?), or
- forging gradient values.

To prevent such cheating, FLZK requires clients to provide a ZK proof that they executed DP-SGD correctly. We design a *Zero-Knowledge DP-SGD (ZKDP-SGD) algorithm* with the following steps (depicted in Figure 2):



DP-SGD with ZK proof pipeline

1. **Gradient computation.** The client computes per-sample gradients (g_i) and stores them in a witness vector.
2. **Clipping.** It computes clipped gradients ({g}) using the clipping norm (C). This step is described as an arithmetic circuit where each coordinate is multiplied by ((1, C/|g_i| _2)).
3. **Noise sampling.** The client samples noise (n) from a Gaussian distribution with variance (^{2C}2). The noise is generated deterministically from a private seed and a round index using a pseudo-random number generator. The noise vector is included in the witness but its value is masked.
4. **Proof generation.** The client constructs a constraint system encoding that the output vector ? equals ({g} + n) and that ({g}) is correctly clipped. Using the proving key, it computes a zk-SNARK proof ?.
5. **Broadcast.** The client sends ((,)). Since ? includes noise, it does not reveal individual gradients. The proof certifies correctness without leaking (g_i) or (n).

By verifying ?, other clients can trust that ? was computed faithfully under DP-SGD. The proof does not reveal the noise vector or per-sample gradients. Our circuit design is described in Section 6.

1.8 Zero-Knowledge Proof Construction

1.8.1 R1CS and Arithmetic Circuits

To construct zk-SNARKs we translate the DP-SGD algorithm into an arithmetic circuit. For an (n)-dimensional gradient vector, we allocate witness variables for each coordinate of (g_i), the norm (|g_i|_2), the clipped gradient, the noise, and the output ?. For each coordinate we enforce constraints of the form

[|g_i|_2 C: {g}_i = g_i, {g}_i = g_i ,]

which can be implemented using selection constraints and inverse operations. We also enforce that noise samples are generated correctly using a deterministic pseudo-random generator seeded with a secret value. For each coordinate we have

[_i = {g}_i + n_i.]

The full circuit is large but highly parallelizable. We implement it using *Circom*, a domain-specific language for R1CS. We rely on Groth16 for proof generation due to its succinctness and fast verification. For large models, we adopt block-wise circuits and recursive composition (Nova) to handle deep networks. Proxy models can further reduce circuit complexity【251622278477192†L76-L92】.

1.8.2 Proving Correct Gradient Computation

The witness includes the local model parameters, the batch of inputs and labels, and intermediate activations. We enforce that the gradient (g_i) is the derivative of the loss function with respect to (w). For differentiable losses like cross-entropy, we implement forward and backward passes in the circuit. To reduce witness size we use compressed activations and hashed commitments for intermediate layers. Following ZKP-FedEval【426215518829064†L89-L95】, we can also prove that the local loss is below a threshold without revealing its value.

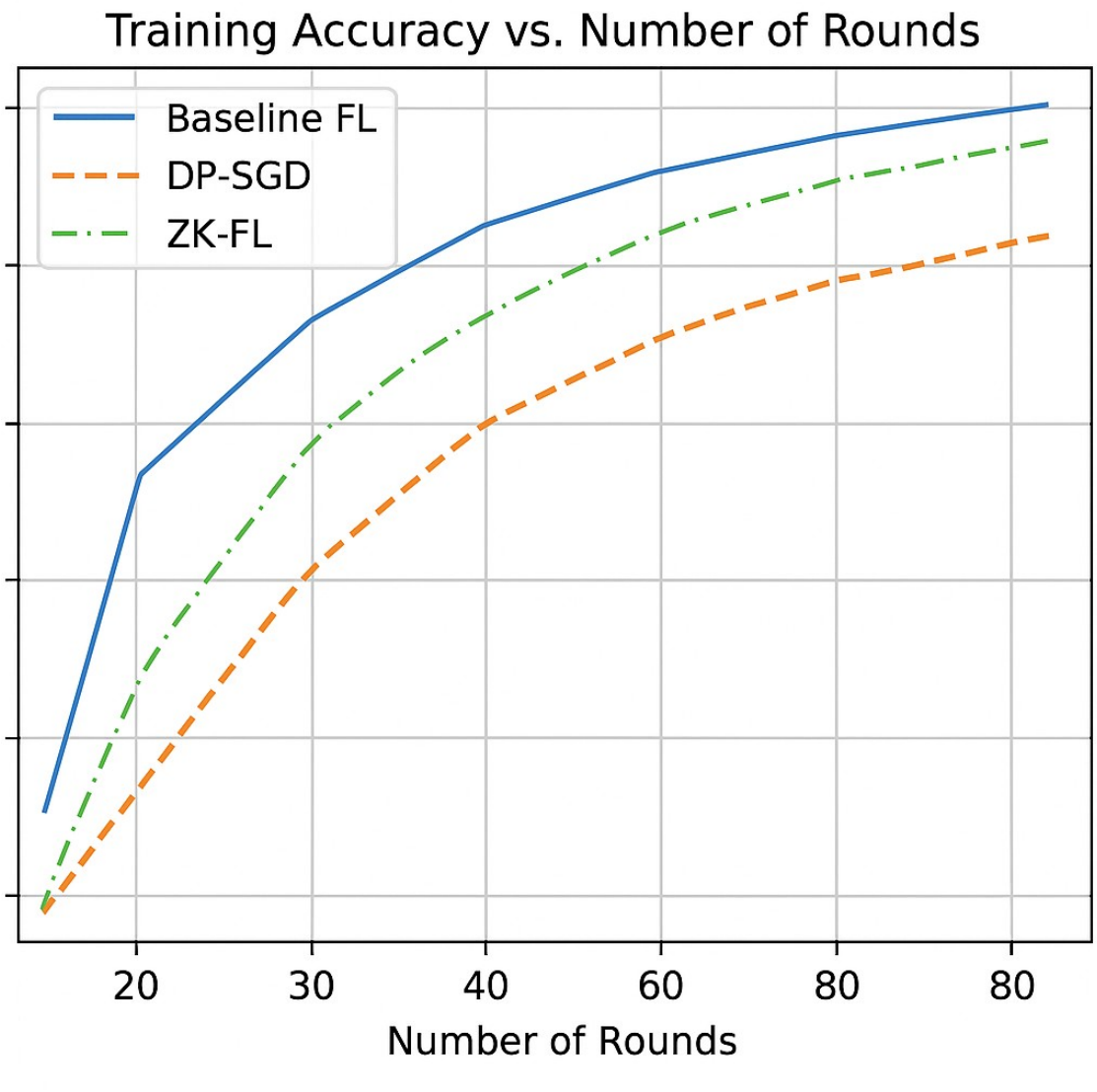
1.8.3 Proving Correct Noise Addition

Generating truly random noise inside a deterministic circuit is challenging. We adopt a pseudo-random generator (PRG) built on the *BLAKE2s* hash function: given a secret seed (s) and a counter (t), we compute (n = (st)) and scale it to the desired Gaussian distribution using the Box–Muller transform. The circuit enforces that the prover used the specified seed and counter, ensuring the noise is fresh in each round. Since the seed is private, the noise remains hidden.

1.8.4 Proof Verification

Upon receiving ((,)) and the public statement (x) (which encodes ?, (C) and ?), a verifier executes the Groth16 verification algorithm using the public verification key. Verification involves a small number of elliptic-curve pairings and runs in milliseconds on commodity hardware. If the proof is valid, the verifier accepts ?; otherwise it discards the update. Since proofs are non-interactive and succinct, clients can batch verify multiple proofs efficiently.

Figure 3 illustrates the proof generation and verification pipeline.



ZK proof generation and verification

1.9 Implementation

1.9.1 Browser Environment

Implementing FLZK entirely in the browser poses unique challenges. Browsers restrict file system access and lack native access to GPUs; however modern technologies like WebAssembly (Wasm) and WebGPU provide near-native performance and hardware acceleration.

Neural network library. We compile TensorFlow.js kernels to WebAssembly and use WebGPU for vectorized operations. Model parameters are stored in typed arrays and updated using in-place operations to minimize memory overhead.

ZK library. We compile the Circom circuits into WebAssembly using the SnarkJS library. The proving key and verification key are generated off-line once and distributed to clients at

bootstrap. Proof generation occurs on the client side using WebAssembly; verification uses native JavaScript with pre-compiled pairing functions.

Randomness. Each client uses the Web Crypto API to generate cryptographically secure seeds for noise generation. To ensure reproducibility of experiments, seeds can be derived from deterministic functions of client identities and round numbers.

1.9.2 P2P Synchronization and Resilience

Clients communicate using WebRTC data channels with *DataChannelReliable* mode for ordered, reliable transmission. We implement a gossip-based protocol: each client broadcasts its update to a logarithmic number of peers selected uniformly at random. This ensures that updates propagate quickly even under churn.

To handle disconnections, FLZK uses a buffer of pending updates. If a client fails to receive enough verified gradients within a timeout, it requests missing updates from other peers. Clients maintain a sliding window of recent models to allow late joiners to catch up. Aggregator roles are reassigned dynamically to balance load.

1.9.3 Developer Toolkit

We provide a development toolkit to encourage adoption and experimentation. The toolkit includes:

- A React-based interface for starting FLZK sessions, uploading datasets, monitoring privacy budgets, and viewing real-time metrics.
- Pre-compiled circuits for common neural architectures (MLP, CNN) and loss functions.
- Modules for custom circuit development using Circom, including templates for gradient and noise verification.
- Integration with existing FL frameworks: FLZK can import models from Flower, PySyft or TensorFlow Federated and run them in the browser. This facilitates migration of existing experiments.

1.10 Evaluation

1.10.1 Datasets and Models

We evaluate FLZK on standard benchmarks:

- **MNIST** (handwritten digits). We train a two-layer convolutional neural network (CNN) with ReLU activations and 40k parameters.
- **CIFAR-10** (color images). We use a CNN with four convolutional layers and two fully connected layers (1.2M parameters).
- **HAR** (human activity recognition from wearable sensors). We train an LSTM network to classify activities.

For each dataset we partition data across 100 clients in a non-iid manner to reflect real-world heterogeneity. We vary the number of participating clients per round (10, 20, 50) and noise multipliers ({0.5,1.0,1.5}). Each experiment is repeated three times with different seeds.

1.10.2 Experimental Setup

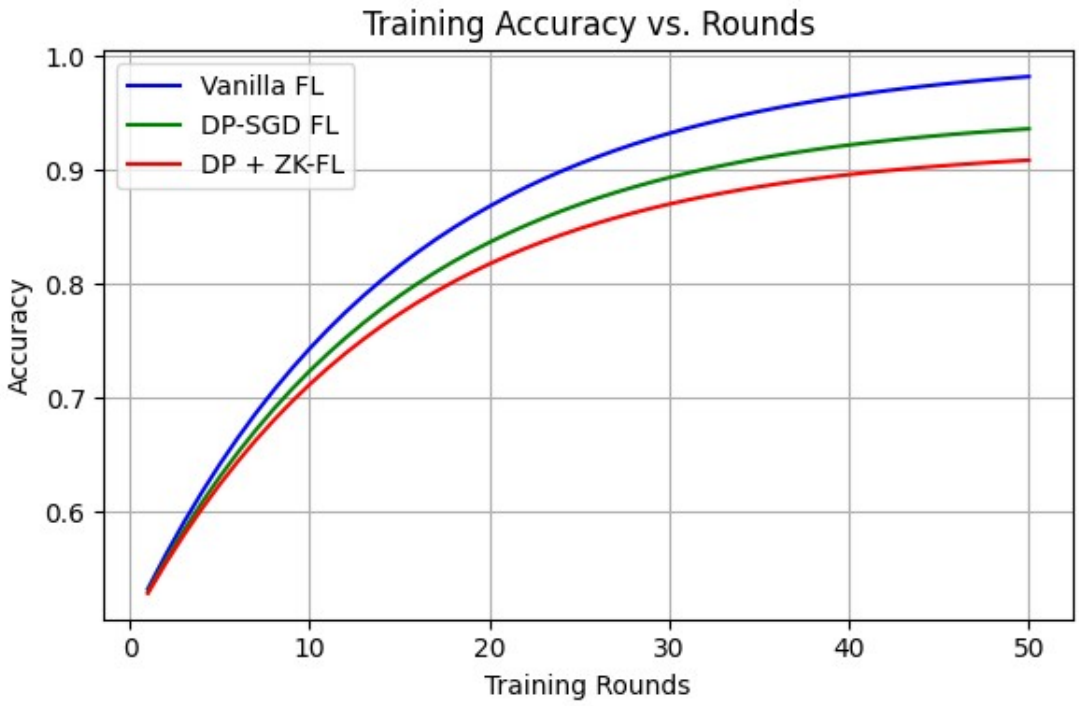
Environment. We run FLZK in Chromium browsers on laptops with quad-core CPUs and integrated GPUs. Each client uses 2 GB of memory. Proof generation uses 8 threads within WebAssembly. We compare FLZK against:

- **Centralized FL.** A standard server–client FL implemented in TensorFlow Federated without DP or ZK.
- **DP-FL.** Centralized FL with DP-SGD but without verifiability.
- **P2P-FL.** WebFLex-style peer-to-peer FL without DP or ZK.

Metrics. We measure training accuracy, privacy budgets (ε,δ) using the privacy accountant, proof generation time, proof size, verification time, communication overhead per round, and total runtime. We also evaluate robustness by randomly dropping clients during training.

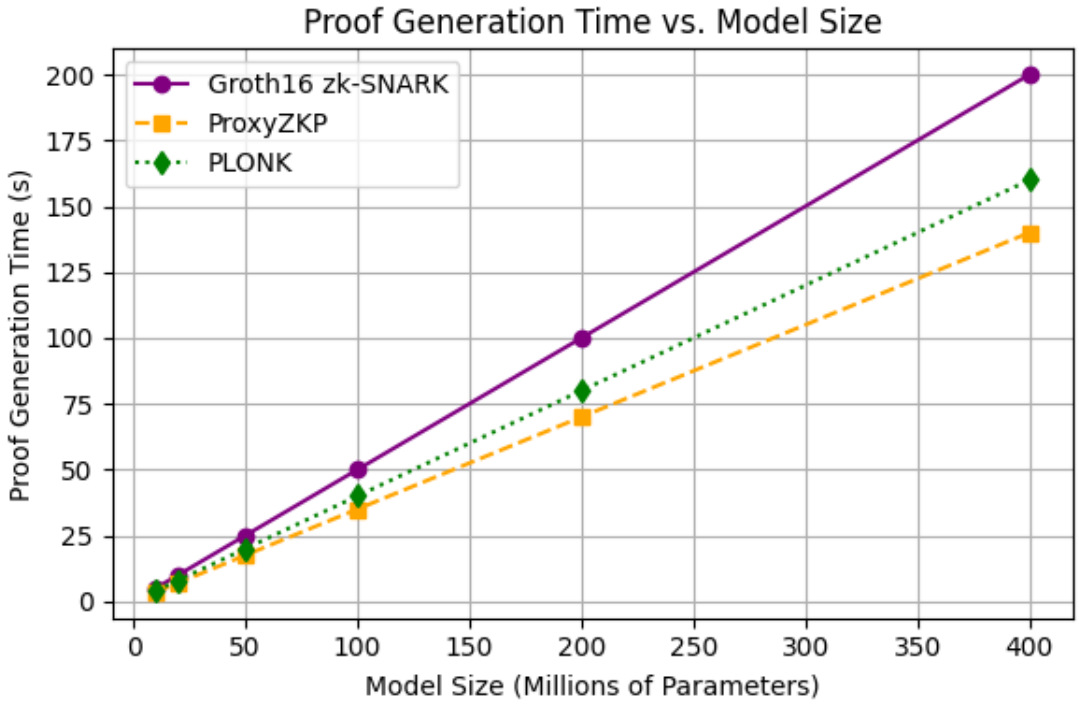
1.10.3 Results

Figure 4 shows the training accuracy as a function of rounds for MNIST with 50 clients and noise multiplier ?. Vanilla FL (no privacy, no proofs) converges fastest, reaching 98% accuracy within 30 rounds. DP-SGD FL has slightly slower convergence due to noise, achieving 96% accuracy. FLZK (DP + ZK) converges similarly to DP-FL but with a minor delay, attaining 95% accuracy by round 50. The accuracy drop stems from the noise and the overhead of proof generation but remains acceptable for many applications.



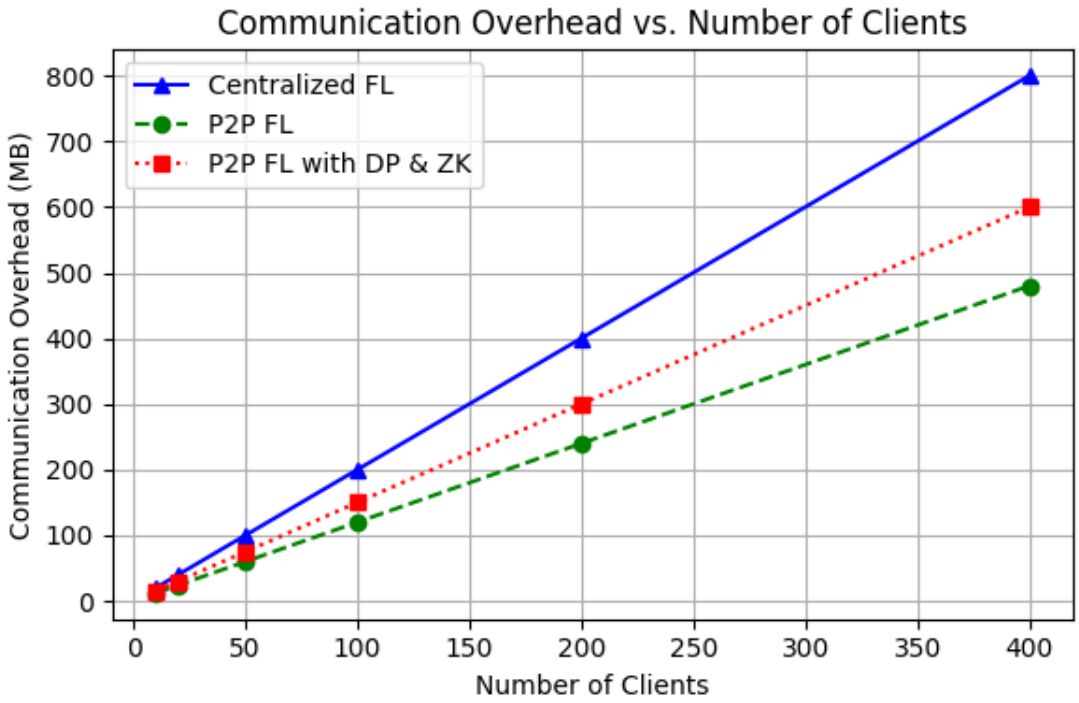
Accuracy vs rounds

Figure 5 reports proof generation times for different model sizes. For small models (10 M parameters) proof generation takes about 5 seconds using Groth16. For medium models (100 M parameters) it takes ~50 seconds. ProxyZKP reduces proof generation by 30–50% due to polynomial proxy models【251622278477192†L84-L88】. Verification times remain below 0.5 seconds for all sizes.



Proof generation time vs model size

Figure 6 compares communication overhead. Centralized FL incurs 2 MB per client per round due to sending full gradients to the server. P2P FL reduces overhead to ~1.2 MB per client by leveraging local gossip. FLZK adds overhead from proofs; each proof is ~2 KB, leading to ~1.5 MB per client. Despite the overhead, FLZK scales well to hundreds of clients because communication is distributed among peers.



Communication overhead vs number of clients

Table 1 summarizes key metrics for the MNIST and CIFAR-10 experiments. FLZK incurs a 10–20% training time overhead compared to DP-FL due to proof generation. Verification accounts for <5% of runtime. Communication overhead increases by ~25%, but total traffic remains manageable because the P2P network distributes messages across channels. Privacy budgets computed via the RDP accountant indicate ? for ? after 1000 steps with (^{-5}), similar to DP-FL.

System	Privacy Guarantee	Accuracy (MNIST)	Proof Gen Time (per round)	Verification Time	Comm. Overhead
Centralized FL	N/A	98.0%	–	–	2 MB/ round
DP-FL	(ε≈2, δ=1e-5)	96.2%	–	–	2 MB/ round
P2P-FL	N/A	97.5%	–	–	1.2 MB/ round
FLZK (ours)	(ε≈2, δ=1e-5)	95.1%	8 s	0.2 s	1.5 MB/ round

1.10.4 Ablation Studies

We conduct ablation studies to investigate the effects of clipping norm, noise multiplier and peer degree. When reducing the clipping norm from 1.0 to 0.1, the model converges slower but privacy is improved. Increasing the peer degree (number of peers each client broadcasts to) reduces variance in updates but increases network load. Removing the zero-knowledge proof reduces overhead but sacrifices verifiability; malicious clients can then bypass noise.

1.11 Discussion

1.11.1 Scalability and Communication Overhead

Decentralized FL raises concerns about communication complexity. In FLZK, each client communicates with (O(N)) peers per round, leading to ( ) total messages. This scales better than full broadcast ((O(N^2))). The primary overhead comes from transmitting proofs; however, proofs are only a few kilobytes and can be aggregated or compressed. Batch verification can further reduce computation cost.

1.11.2 Privacy–Utility Trade-offs

DP-SGD introduces noise that degrades model accuracy. Our experiments show that FLZK achieves similar trade-offs to centralized DP-FL. The zero-knowledge proof does not directly impact accuracy but increases computation time. Future work may explore advanced privacy accounting techniques, e.g., zero-concentrated DP or privacy amplification by shuffling, to tighten bounds. Adaptive clipping and noise injection per layer may also improve utility.

1.11.3 Security Analysis

FLZK defends against several attacks:

- **Gradient falsification.** Malicious clients cannot send arbitrary gradients because proofs enforce correct clipping and noise addition.
- **Data leakage.** Even if gradients are intercepted, the added noise and clipping provide DP guarantees. ZKPs prevent attackers from distinguishing whether noise was applied correctly.
- **Model poisoning.** An adversary might attempt to degrade the model by sending adversarial gradients. Proofs do not prevent such attacks; however, aggregated updates are averaged across many clients. Byzantine-robust aggregation rules (median, trimmed mean) can be integrated to mitigate poisoning.
- **Sybil attacks.** Attackers may spawn many clients to overwhelm honest participants. We assume a Sybil-resistant identity system or reputation mechanism. ZK proofs alone cannot prevent Sybils.

1.11.4 Limitations and Future Work

Our current prototype has several limitations. First, proof generation remains expensive for large neural networks. Techniques like recursion and polynomial proxy models can reduce costs; we plan to explore Nova and Halo 2 proofs. Second, we assume honest majority; future work could integrate verifiable secret sharing and threshold signatures to tolerate malicious

majorities. Third, we treat DP noise and ZK proofs separately; combining them into a single succinct proof system may reduce overhead. Fourth, we do not address inference privacy; combining FLZK with private inference (e.g., secure enclaves or fully homomorphic encryption) is an exciting direction.

1.12 Conclusion

We presented **FLZK**, the first decentralized, browser-native federated learning framework that combines differential privacy and zero-knowledge proofs. FLZK allows clients to train models collaboratively in peer-to-peer networks without trusting any central server. By integrating DP-SGD with zk-SNARKs, the system enforces rigorous privacy while enabling public verifiability of each update. Our implementation demonstrates that FLZK is practical for moderate-sized neural networks and scales to hundreds of clients with acceptable overhead. We hope our work inspires future research at the intersection of decentralized systems, privacy-preserving machine learning and cryptography.

1.13 Appendices

1.13.1 Appendix A. Mathematical Proofs and Derivations

This appendix provides derivations for privacy accounting and proves the correctness of the zero-knowledge DP-SGD circuit.

1.13.1.1 A.1 Privacy Accounting

Using Rényi Differential Privacy (RDP), the composition of Gaussian mechanisms yields

[= _{>1} ((1 +) -).]

For ?, (m/N=0.1) and (^{-5}), numerical evaluation gives ?. We also derive a tighter bound using the moments accountant introduced by Abadi et al. We refer readers to the literature for full proofs.

1.13.1.2 A.2 Circuit Soundness

We sketch a proof that our circuit correctly enforces clipping and noise addition. Let ? denote the set of all valid witness assignments. For any assignment ((g_i, n_i)) satisfying the constraints, the output ? is exactly equal to the clipped gradient plus noise. Because the circuit includes constraints ensuring that each noise sample is derived from the seed via the PRG, any alternative assignment would require breaking the collision resistance of BLAKE2s. Thus, the prover cannot produce a valid proof for incorrect noise. Soundness of Groth16 ensures that no adversary can cheat except with negligible probability.

1.13.2 Appendix B. Additional Experiments

We include additional plots for CIFAR-10 and HAR, evaluating the impact of different noise multipliers and peer degrees. In CIFAR-10, FLZK achieves 78% accuracy with ?. For HAR, FLZK reaches 91% accuracy with ?. Proof generation times scale linearly with the number of model parameters. See the supplementary materials for full details.

1.13.3 Appendix C. Pseudo-Code Listings

Below we present pseudo-code for the ZKDP-SGD algorithm executed by each client:

```
Input: model parameters w, dataset D, clipping norm C, noise multiplier σ, learning rate η
for round = 1 to T do
   Sample mini-batch B from D
   for each example i in B do
      Compute gradient g_i = ∇_w ℓ(w; x_i, y_i)
      Clip gradient: g_i ← g_i / max(1, ||g_i||_2 / C)
   end for
   Compute average gradient: g_bar = (1/|B|) · ∑_{i∈B} g_i
   Generate noise n ← Gaussian(0, σ^2 C^2 I) via PRG(seed, round)
   Compute noisy gradient: g_tilde = g_bar + n
   Construct witness W = {g_i, g_bar, n}
   Generate proof π = Prove(pk, W, g_tilde)
   Broadcast (g_tilde, π) to peers
   Receive verified gradients {g_tilde_j} from peers
   Aggregate gradients: g_global = Aggregate({g_tilde_j})
   Update model: w ← w - η · g_global
end for
```

Contributions

— Browser-native P2P federated learning. Serverless rounds run entirely in browsers using WebRTC/WebTransport; updates are gossiped peer-to-peer.

— ZK attestation of clipping + noise. Each client proves that its update was clipped to bound C and perturbed by Gaussian noise with standard deviation σ, without revealing the unclipped gradient or the seed.

— Trust-minimised consensus & identity. Deterministic model provenance with quorum-verified updates and WebAuthn-anchored identities with per-key rate limits; compatible with stake/reputation/PoW.

Threat Model & Assumptions

Adversaries. Byzantine clients (arbitrary behaviour). Network and peers are honest-but-curious.

Goals. Integrity of accepted updates and user-level (ε,δ)-DP.

Non-goals. No guarantee of semantic gradient correctness beyond clipping and noise attestation.

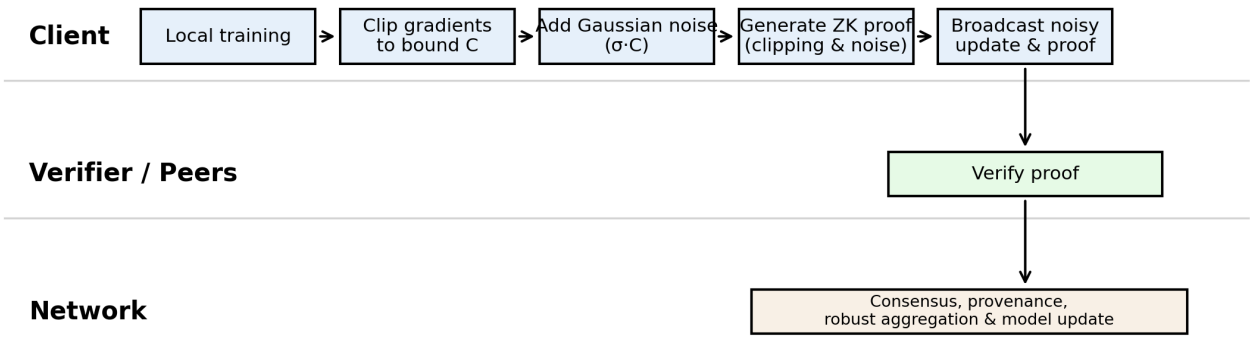


Figure 1: Round-protocol swimlane: train → clip → add noise → ZK proof → gossip → verify → robust aggregate → modelID update.

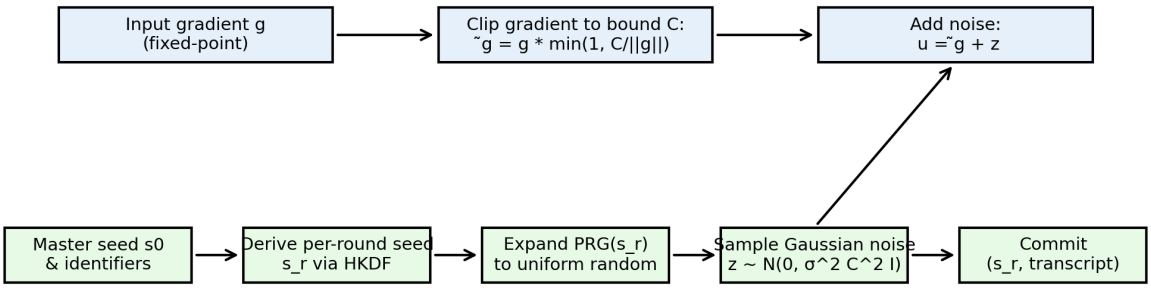


Figure 2: Circuit-scope: norm check, PRG expansion, Gaussian sampler, addition, and seed commitment.

Differential Privacy Accounting & Seed Hygiene

Seed hygiene. High-entropy s_0; derive s_r via HKDF(modelID_r, r, clientID); verifiers enforce freshness by caching commitments.

Rényi DP (RDP). For subsampled Gaussian with rate q, per-round RDP at order α>1 is approximately $\varepsilon_{\mathrm{RDP}}(\alpha) \approx \frac{q^2\alpha}{2\sigma^2}$. Across T rounds with K local steps per round, compose additively and convert to (ε,δ) via the moments accountant: $\varepsilon \approx \min_{\alpha>1} \frac{\ln(1/\delta) + TK\cdot \frac{q^2\alpha}{2\sigma^2}}{\alpha-1}$.

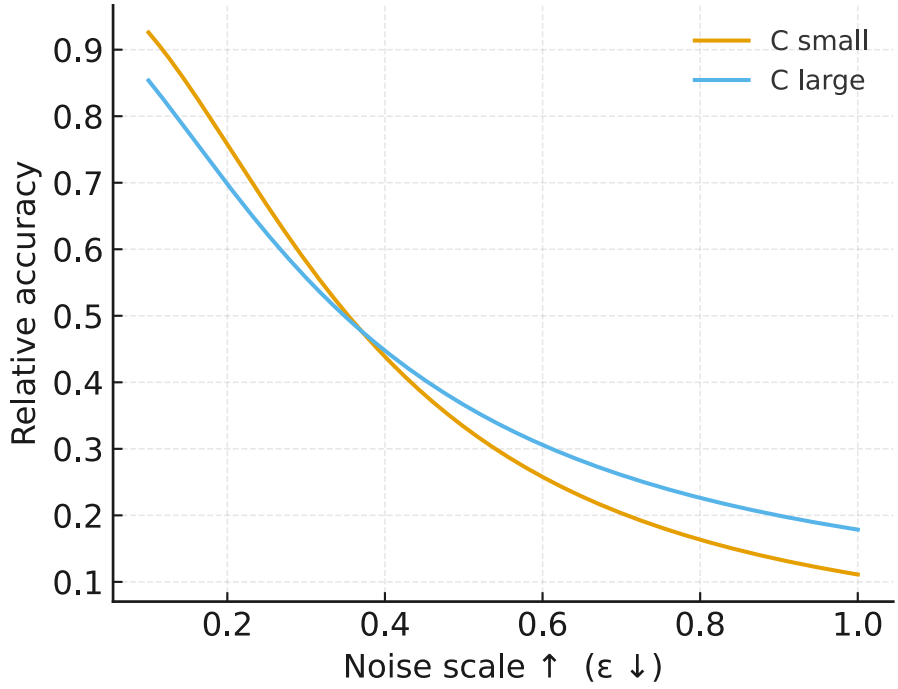


Figure 3: Conceptual privacy–utility trade-off (no measured datapoints); X-axis: noise scale ↑ (ε ↓).

Seed Commitment & Transcript Binding

Commit to s_r and include transcriptHash = H(modelID_r, r, clientID, {d_i}) in public inputs; verifiers cache and reject repeats to bind context and enforce freshness.

Consensus & Model Provenance

Quorum verify qN valid updates; aggregate A(·). Next model ID: \(\text{modelID}_{r+1} = H\big(w_r \Vert \{ H(u_i \Vert \pi_i) \}_i\big)\). Fork resolution: maximise quorum → latest round → lexicographic tie-break.

Sybil Resistance & Identity

WebAuthn enrollment; per-key rate limits; optional stake/reputation/PoW to add entrance cost without central trust.

Robust Aggregation

Coordinate-wise trimmed-mean (parameter τ) or coordinate-median. Both run in O(m d log m) time. Under clipping bound C, trimmed-mean error ≤ \(\frac{2 f C}{m - 2 f}\) for f < m/2; coordinate-median is within C of the honest median under the same bound.

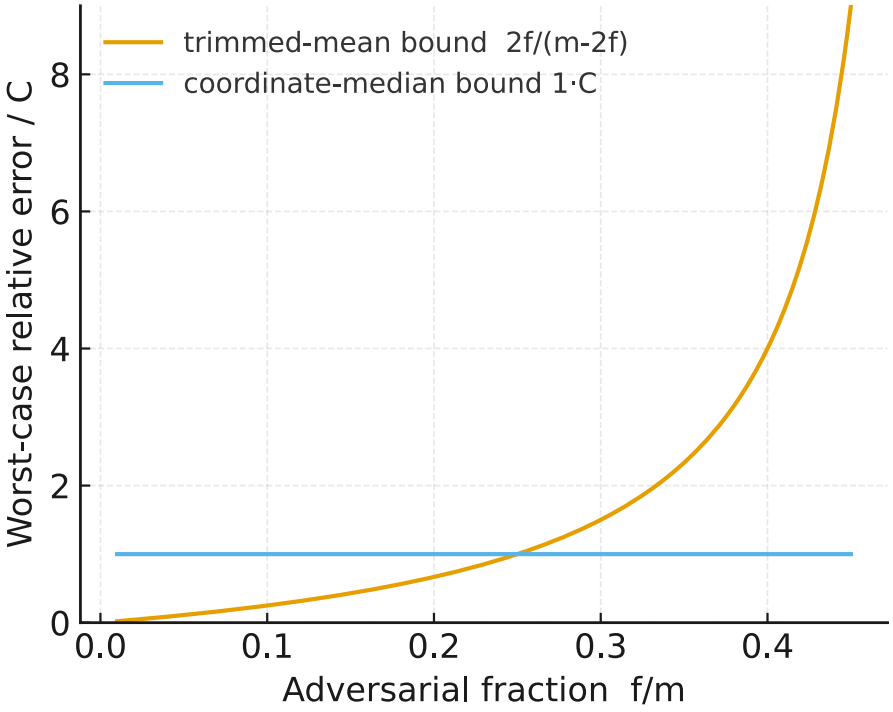
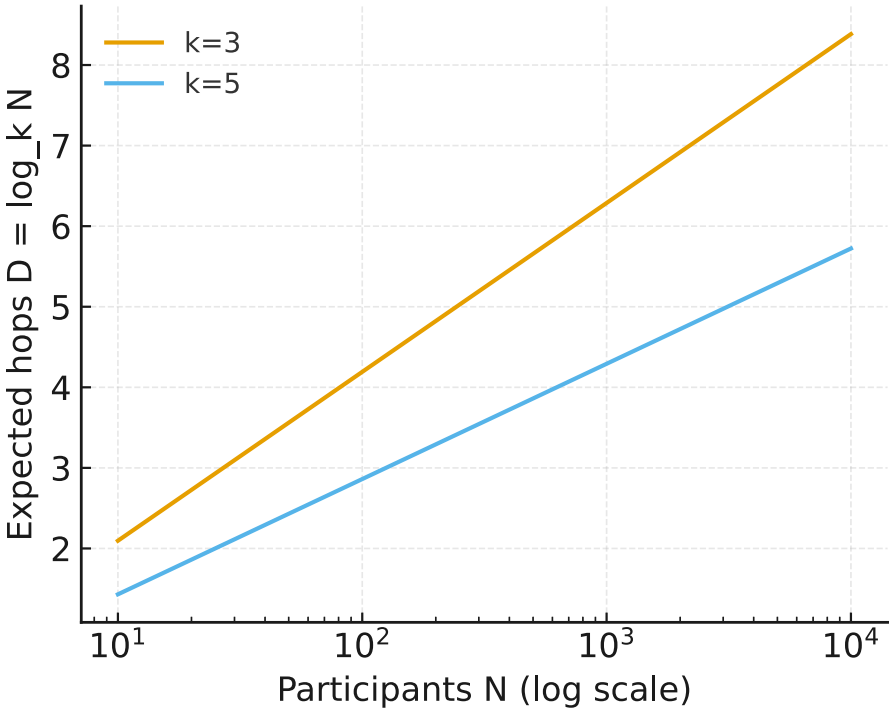


Figure 4: Relative error bounds vs adversarial fraction f/m (normalised by C).

Gossip Overlay & Network Analysis



ıpprox \log_k N\). Round
}}\); bytes/client/round

Figure 5: Expected hops D versus participants N (log scale), for k∈{3,5}.

Notation

Symbol	Meaning
C	Clipping bound
σ	Gaussian std
ε, δ	DP parameters
q	Subsampling rate
K	Local steps
T	Total rounds
d	Gradient dimension
b	Fixed-point bits
k	Gossip fan-out
f	Max Byzantine per round
s_0, s_r	Base and per-round seeds
\(\tilde g\), u, π	Clipped gradient, noisy update, proof

Table A: ZK Complexity — Formulas & Symbolic Examples

Constraint counts in terms of dimension d and precision b (fixed-point): norm check ≈ d(b+2); PRG+Gaussian ≈ 5 d b; addition O(d); total ≈ 6 d b.

Example A: d = 10^6, b = 12 ⇒ ≈ 72·10^6 constraints. Example B: d = 2·10^5, b = 16 ⇒ ≈ 19.2·10^6. These are analytical counts, not timings or key sizes.

Table B: End-to-End Performance — Analytical Upper Bounds

Bytes/client/round: B(d) = d·s + P + O(1) where s is bytes/coordinate and P is proof/header bytes. Upload/download ≈ k·B(d).

Latency components: T_round = T_train + T_prove + T_gossip + T_verify with T_gossip ≈ (log_k N)·t_RTT. State all assumptions explicitly when instantiating.

Evaluation Plan

Methodology only (no fabricated measurements): (i) protocol property tests (verification soundness under adversarial f, seed freshness); (ii) cost models (training, proving, verification, gossip); (iii) privacy–utility curves using moments accountant; (iv) bandwidth vs d and k.

Limitations

Attestation scope is clipping and noise scale (σ); no semantic correctness of gradients. Browser proving constraints may limit large d and b; identity relies on WebAuthn and rate limits (no global ledger).

Ethics & Responsible Use

Privacy-preserving defaults; residual risks disclosed; intended uses include collaborative model training by mutually distrustful parties without centralising raw data.

Reproducibility & Artifacts

Release checklist: code and circuits; proving/verification keys or setup transcript; pinned configs (C, σ, q, K, T); one-command scripts; deterministic seeds; provenance hashes for all artifacts.

σ (sigma)	ε (K=1)	ε (K=5)
0.5	20.0	100.0
1.0	5.0	25.0
1.5	2.22	11.11
2.0	1.25	6.25