Jolt-FL: A General-Purpose Verifiable Federated Learning Framework Powered by zkVM

From Jolt zkVM to the Future of zkML

Hoa Viet Nguyen (Member, IEEE)
Hoang D. Le (Member, IEEE)
Anh T. Pham (Senior Member, IEEE)

Computer Communications Laboratory
The University of Aizu, Japan

IEEE Globecom 2025 – CISS-11
Taipei, Taiwan

- **Track:** CISS-11 – Communication and Information System Security
- **Talk:** Verifiable Federated Learning over zkVM

Outline

- Motivation: **why verifiable federated learning?**
- From zk-SNARKs to zkVMs and zkML
- Jolt zkVM
- Jolt-FL design: proving each local training step
- Evaluation on MNIST and robustness to malicious clients
- Jolt vs. Jolt-FL and the road ahead for zkML

Outline

- Motivation: **why verifiable federated learning?**
- From zk-SNARKs to zkVMs and zkML
- Jolt zkVM
- Jolt-FL design: proving each local training step
- Evaluation on MNIST and robustness to malicious clients
- Jolt vs. Jolt-FL and the road ahead for zkML

Outline

- Motivation: **why verifiable federated learning?**
- From zk-SNARKs to zkVMs and zkML
- Jolt zkVM
- Jolt-FL design: proving each local training step
- Evaluation on MNIST and robustness to malicious clients
- Jolt vs. Jolt-FL and the road ahead for zkML

Outline

- Motivation: **why verifiable federated learning?**
- From zk-SNARKs to zkVMs and zkML
- Jolt zkVM
- Jolt-FL design: proving each local training step
- Evaluation on MNIST and robustness to malicious clients
- Jolt vs. Jolt-FL and the road ahead for zkML

Outline

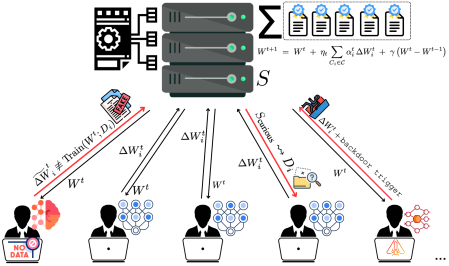
- Motivation: **why verifiable federated learning?**
- From zk-SNARKs to zkVMs and zkML
- Jolt zkVM
- Jolt-FL design: proving each local training step
- Evaluation on MNIST and robustness to malicious clients
- Jolt vs. Jolt-FL and the road ahead for zkML

Outline

- Motivation: **why verifiable federated learning?**
- From zk-SNARKs to zkVMs and zkML
- Jolt zkVM
- Jolt-FL design: proving each local training step
- Evaluation on MNIST and robustness to malicious clients
- Jolt vs. Jolt-FL and the road ahead for zkML

Why Verifiable Federated Learning?

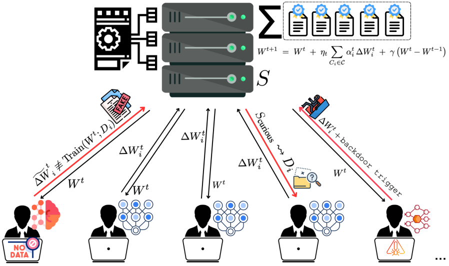
- **Federated Learning (FL):** clients collaboratively train a model without sharing raw data.
- In practice, many clients are **untrusted**: phones, IoT devices, vehicles, or edge servers.



- Malicious clients can send:
  - Random or adversarial gradients (Byzantine / poisoning attacks).
  - Model replacement or free-rider updates without any real training.

Why Verifiable Federated Learning?

- **Federated Learning (FL):** clients collaboratively train a model without sharing raw data.
- In practice, many clients are **untrusted**: phones, IoT devices, vehicles, or edge servers.



- Malicious clients can send:
  - Random or adversarial gradients (Byzantine / poisoning attacks).
  - Model replacement or free-rider updates without any real training.

Why Verifiable Federated Learning? (cont.)

- Existing defenses: robust aggregation, anomaly detection, heuristics.
- **Goal:** cryptographically guarantee that each accepted update really came from running the prescribed training algorithm on some data.

FL update rule

$$W_{t+1} = W_t + \frac{1}{|C_t|} \sum_{i \in C_t} \Delta W_i$$

$$\Delta W_i = \mathrm{Train}(W_t, D_i)$$

From zk-SNARKs to zkVMs and zkML

- Early zkML: hand-written circuits for specific models or layers
  - E.g., zkCNN, specialized SNARKs for inference on small networks.
- Limitations:
  - Hard to maintain as models and training code change.
  - Each new architecture requires new circuit engineering.
- **zkVM approach** (a16z, others):
  - Treat the prover as emulating a CPU (e.g., RISC-V) inside a SNARK.
  - Same VM can run arbitrary Rust / C / ML code, no new circuits.
- Vision for **zkML**:
  - Generic, developer-friendly ZK infrastructure for training and inference.
  - Think of zkVMs as a "ZK coprocessor" similar to GPUs for linear algebra.

From zk-SNARKs to zkVMs and zkML

- Early zkML: hand-written circuits for specific models or layers
  - E.g., zkCNN, specialized SNARKs for inference on small networks.
- Limitations:
  - Hard to maintain as models and training code change.
  - Each new architecture requires new circuit engineering.
- **zkVM approach** (a16z, others):
  - Treat the prover as emulating a CPU (e.g., RISC-V) inside a SNARK.
  - Same VM can run arbitrary Rust / C / ML code, no new circuits.
- Vision for **zkML**:
  - Generic, developer-friendly ZK infrastructure for training and inference.
  - Think of zkVMs as a "ZK coprocessor" similar to GPUs for linear algebra.

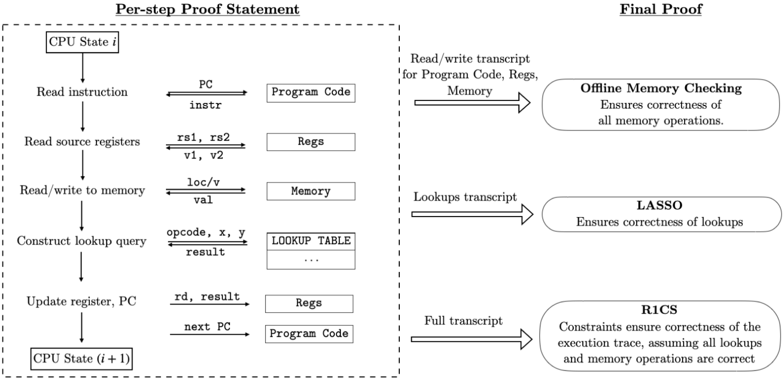
From zk-SNARKs to zkVMs and zkML

- Early zkML: hand-written circuits for specific models or layers
  - E.g., zkCNN, specialized SNARKs for inference on small networks.
- Limitations:
  - Hard to maintain as models and training code change.
  - Each new architecture requires new circuit engineering.
- **zkVM approach** (a16z, others):
  - Treat the prover as emulating a CPU (e.g., RISC-V) inside a SNARK.
  - Same VM can run arbitrary Rust / C / ML code, no new circuits.
- Vision for **zkML**:
  - Generic, developer-friendly ZK infrastructure for training and inference.
  - Think of zkVMs as a "ZK coprocessor" similar to GPUs for linear algebra.

From zk-SNARKs to zkVMs and zkML

- Early zkML: hand-written circuits for specific models or layers
  - E.g., zkCNN, specialized SNARKs for inference on small networks.
- Limitations:
  - Hard to maintain as models and training code change.
  - Each new architecture requires new circuit engineering.
- **zkVM approach** (a16z, others):
  - Treat the prover as emulating a CPU (e.g., RISC-V) inside a SNARK.
  - Same VM can run arbitrary Rust / C / ML code, no new circuits.
- Vision for **zkML**:
  - Generic, developer-friendly ZK infrastructure for training and inference.
  - Think of zkVMs as a "ZK coprocessor" similar to GPUs for linear algebra.

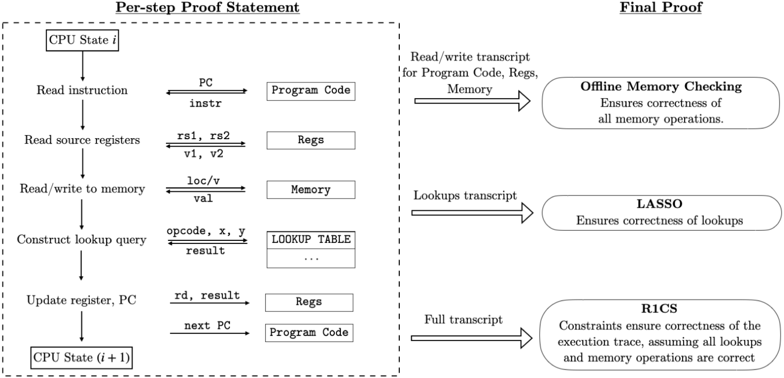
Jolt zkVM (1/3)



Source: Arun et al., “Jolt: SNARKs for Virtual Machines via Lookups”, 2024.

- Jolt is a **high-performance zkVM** from a16z crypto.
- Targets a **RISC-V** ISA; each instruction is enforced via a lookup into a large instruction table.

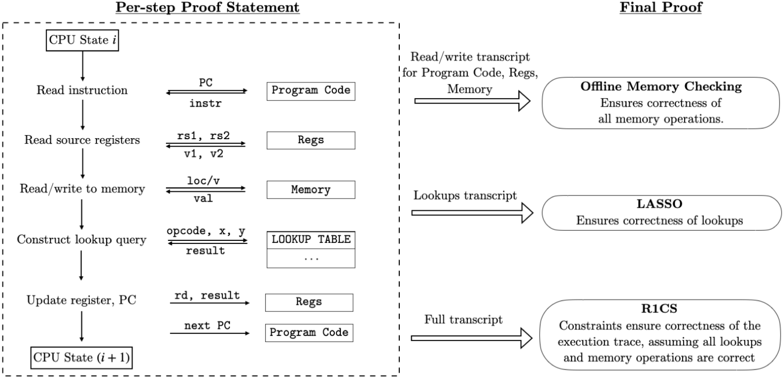
Jolt zkVM (2/3)



Source: Arun et al., "Jolt: SNARKs for Virtual Machines via Lookups", 2024.

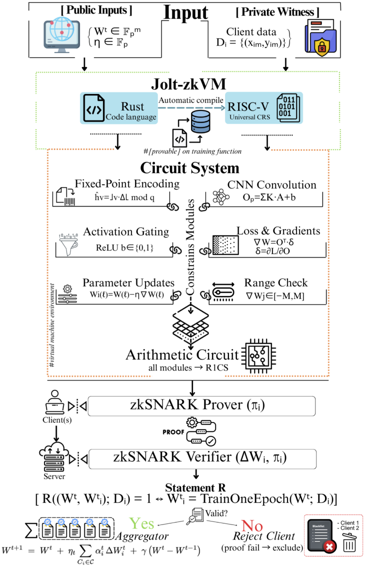
- **Key ingredients:**
  - Lasso-style **lookup argument** for instruction semantics.
  - Spice-style **RAM consistency** checking.
  - **Polynomial commitments** (e.g., KZG) enabling recursion.

Jolt zkVM (3/3)



- **Benefits for ML:**
  - Fixed-point–friendly field for 32/64-bit arithmetic.
  - Uniform zkVM backend for training, preprocessing, and aggregation.
  - No model-specific circuits: we prove execution of normal ML code.

Jolt-FL: High-Level Idea



- **Each client:** run local training program inside Jolt zkVM.
- Output: model update ΔW_i and proof π_i of correct training.
- **Server:** verifies π_i and aggregates only valid ΔW_i.

What Does One Jolt-FL Proof Attest To?

- **Public statement:**
  - Initial model W_t and client output W_i^{t+1} (or update ΔW_i).
  - Round number, learning rate η, and other hyperparameters.
- **Witness (private):**
  - Client dataset D_i (MNIST images and labels in our prototype).
  - Full execution trace of the training program inside Jolt.
- **The proof π_i guarantees:**
  - Running the fixed training program P on (W_t, D_i) inside Jolt produces W_i^{t+1}.
  - All forward, backward, and update steps follow the specified equations.
  - Zero-knowledge: the verifier learns nothing about D_i beyond ΔW_i.

Informal NP statement:

$$R\big((W_t, W_i^{t+1}), (D_i, \tau_i)\big) = 1 \iff \tau_i : P(W_t, D_i) \to W_i^{t+1}.$$

What Does One Jolt-FL Proof Attest To?

- **Public statement:**
  - Initial model W_t and client output W_i^{t+1} (or update ΔW_i).
  - Round number, learning rate η, and other hyperparameters.
- **Witness (private):**
  - Client dataset D_i (MNIST images and labels in our prototype).
  - Full execution trace of the training program inside Jolt.
- **The proof π_i guarantees:**
  - Running the fixed training program P on (W_t, D_i) inside Jolt produces W_i^{t+1}.
  - All forward, backward, and update steps follow the specified equations.
  - Zero-knowledge: the verifier learns nothing about D_i beyond ΔW_i.

Informal NP statement:

$$R\big((W_t, W_i^{t+1}), (D_i, \tau_i)\big) = 1 \iff \tau_i : P(W_t, D_i) \to W_i^{t+1}.$$

What Does One Jolt-FL Proof Attest To?

- **Public statement:**
  - Initial model W_t and client output W_i^{t+1} (or update ΔW_i).
  - Round number, learning rate η, and other hyperparameters.
- **Witness (private):**
  - Client dataset D_i (MNIST images and labels in our prototype).
  - Full execution trace of the training program inside Jolt.
- **The proof π_i guarantees:**
  - Running the fixed training program P on (W_t, D_i) inside Jolt produces W_i^{t+1}.
  - All forward, backward, and update steps follow the specified equations.
  - Zero-knowledge: the verifier learns nothing about D_i beyond ΔW_i.

Informal NP statement:

$$R\big((W_t, W_i^{t+1}), (D_i, \tau_i)\big) = 1 \iff \tau_i : P(W_t, D_i) \to W_i^{t+1}.$$

What Does One Jolt-FL Proof Attest To?

- **Public statement:**
  - Initial model W_t and client output W_i^{t+1} (or update ΔW_i).
  - Round number, learning rate η, and other hyperparameters.
- **Witness (private):**
  - Client dataset D_i (MNIST images and labels in our prototype).
  - Full execution trace of the training program inside Jolt.
- **The proof π_i guarantees:**
  - Running the fixed training program P on (W_t, D_i) inside Jolt produces W_i^{t+1}.
  - All forward, backward, and update steps follow the specified equations.
  - Zero-knowledge: the verifier learns nothing about D_i beyond ΔW_i.

Informal NP statement:

$$R\big((W_t, W_i^{t+1}), (D_i, \tau_i)\big) = 1 \iff \tau_i : P(W_t, D_i) \to W_i^{t+1}.$$

Encoding CNN Training in Jolt zkVM (1/3)

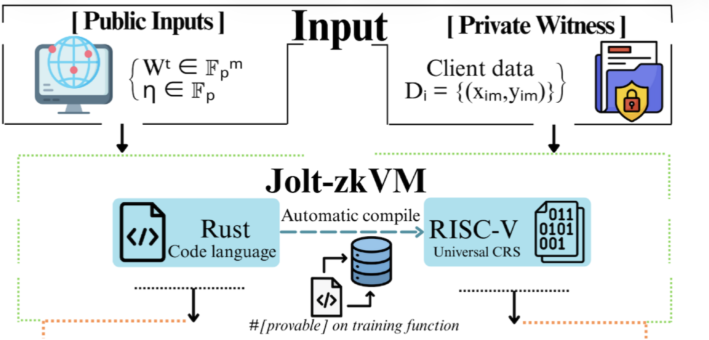
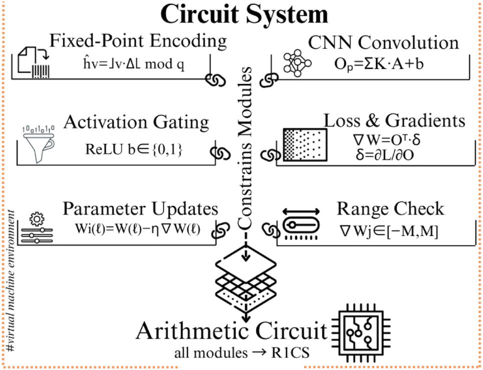


Figure: CNN code in Rust compiled to RISC-V and executed inside Jolt.

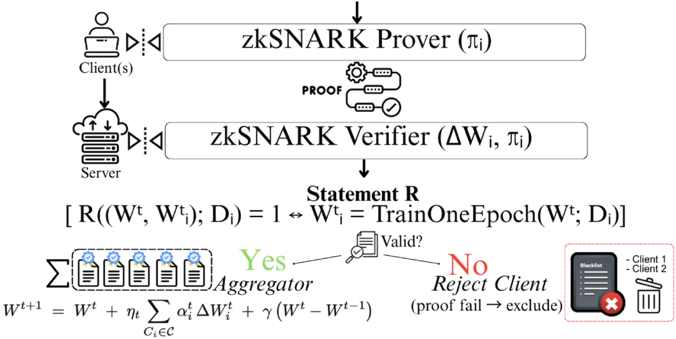
- We implement an end-to-end CNN in **Rust** and compile it to **RISC-V**.
- Jolt then arithmetizes the RISC-V execution into constraints inside the zkSNARK.

Encoding CNN Training - constrained inside the zkVM. (2/3)



- **Forward pass:** convolutions, ReLUs, pooling, fully connected layer.
- **Loss and gradient:** cross-entropy loss, backpropagation all layers.
- **Parameter update:** enforce $W_i^{t+1} = W_t - \eta \nabla_W \mathcal{L}(W_t; D_i)$.
- Constraints tightly link the gradient to the actual data batch D_i, so any deviation immediately breaks the proof.

Encoding CNN Training in Jolt zkVM (3/3)



- **Client side:** each client is a zkSNARK *prover* and sends its update (ΔW_i) together with a proof π_i.
- **Server side:** the server is the *verifier*; it checks each proof. Valid updates are marked **Yes**, invalid ones are **rejected**.
- **Aggregation:** the new global model is computed from *only* the verified updates; bad clients will be blacklisted.

Zero-Knowledge FL Round (Algorithm Sketch)

Jolt-FL round t

1. Server broadcasts (W_t, η_t) to participating clients.
2. Each client C_i:
   - 2.1 Runs one local epoch of training on D_i **inside Jolt.**
   - 2.2 Computes $\Delta W_i = W_i^{t+1} - W_t$.
   - 2.3 Generates proof π_i for (W_t, W_i^{t+1}).
   - 2.4 Sends $(\Delta W_i, \pi_i)$ to the server.
3. Server verifies each π_i and discards invalid updates.
4. Aggregates verified updates:

$$W_{t+1} = W_t + \frac{1}{|C_t|} \sum_{i \in C_t} \Delta W_i.$$

Zero-Knowledge FL Round (Algorithm Sketch)

Jolt-FL round t

1. Server broadcasts (W_t, η_t) to participating clients.
2. Each client C_i:
   - 2.1 Runs one local epoch of training on D_i **inside Jolt.**
   - 2.2 Computes $\Delta W_i = W_i^{t+1} - W_t$.
   - 2.3 Generates proof π_i for (W_t, W_i^{t+1}).
   - 2.4 Sends $(\Delta W_i, \pi_i)$ to the server.
3. Server verifies each π_i and discards invalid updates.
4. Aggregates verified updates:

$$W_{t+1} = W_t + \frac{1}{|C_t|} \sum_{i \in C_t} \Delta W_i.$$

Zero-Knowledge FL Round (Algorithm Sketch)

Jolt-FL round t

1. Server broadcasts (W_t, η_t) to participating clients.
2. Each client C_i:
   2.1 Runs one local epoch of training on D_i **inside Jolt.**
   2.2 Computes $\Delta W_i = W_i^{t+1} - W_t$.
   2.3 Generates proof π_i for (W_t, W_i^{t+1}).
   2.4 Sends $(\Delta W_i, \pi_i)$ to the server.
3. Server verifies each π_i and discards invalid updates.
4. Aggregates verified updates:

$$W_{t+1} = W_t + \frac{1}{|C_t|} \sum_{i \in C_t} \Delta W_i.$$

Zero-Knowledge FL Round (Algorithm Sketch)

Jolt-FL round t

1. Server broadcasts (W_t, η_t) to participating clients.
2. Each client C_i:
   - 2.1 Runs one local epoch of training on D_i **inside Jolt.**
   - 2.2 Computes $\Delta W_i = W_i^{t+1} - W_t$.
   - 2.3 Generates proof π_i for (W_t, W_i^{t+1}).
   - 2.4 Sends $(\Delta W_i, \pi_i)$ to the server.
3. Server verifies each π_i and discards invalid updates.
4. Aggregates verified updates:

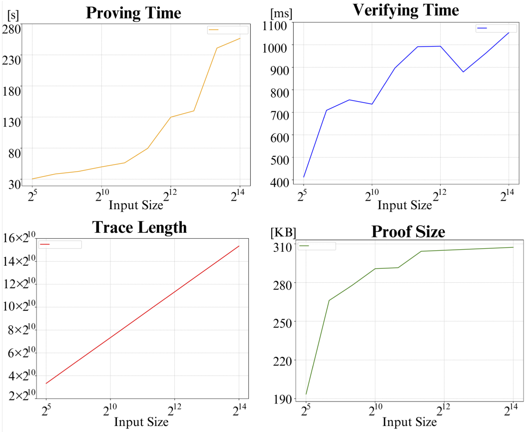
$$W_{t+1} = W_t + \frac{1}{|C_t|} \sum_{i \in C_t} \Delta W_i.$$

Evaluation Setup

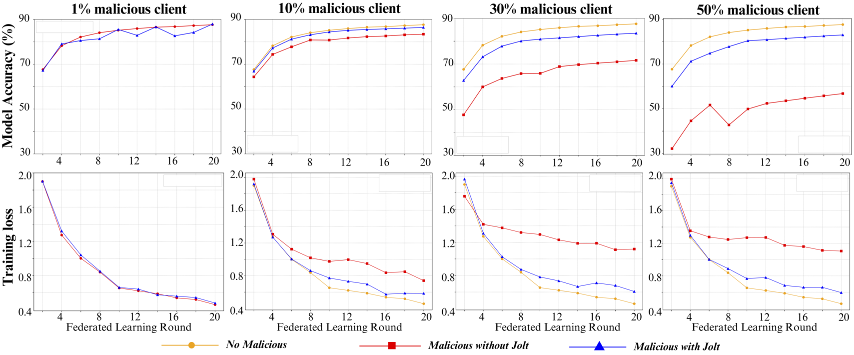
- Platform: Apple M3 MacBook Pro, 8-core CPU, 16 GB RAM.
- Dataset: **MNIST** handwritten digits.
- Model: lightweight CNN
  - Two 3×3 convolutional layers (4 filters), 2×2 max pooling, ReLU, fully connected output layer.
- FL configuration:
  - 10 clients, each with 6,000 local images.
  - 20 FL rounds; mini-batches of 50 images.
  - Learning rate $\eta = 0.02$.
- To reduce overhead we also explore **partial proofs** over selected batches/weights.

Performance Results: Proving / Verification / Size

- Proving per client: **tens of seconds** (~ 10–$15\times$ slower than native).
- Verification: $<$ **1 s** per proof.
- Proof size: **190–310 KB**, comparable to model update.
- **Proving is the main bottleneck.**
- **Verification and proof size are modest.**

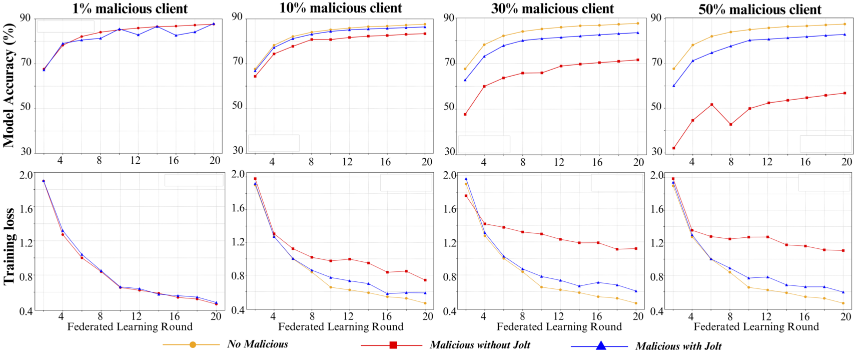


Accuracy and Robustness: Convergence (1/2)



- **Convergence:** Jolt-FL **matches** standard FL accuracy on MNIST.
- 20 rounds: similar test accuracy; fixed-point arithmetic does **not** hurt learning.

Accuracy and Robustness: Malicious Clients (2/2)



- **Scenario:** 10%, 30%, 50% clients send adversarial updates.
- Standard FL loses about **40–60%** accuracy.
- Jolt-FL **rejects** invalid updates, keeping roughly **80–90%** accuracy even at 50% malicious clients.

Jolt vs Jolt-FL & the zkML Roadmap

- **Jolt (a16z zkVM):**
  - General-purpose zkVM for RISC-V programs.
  - Designed for programmable ZK and efficient recursion/accumulation.
- **Jolt-FL (our work):**
  - First end-to-end verifiable FL framework built directly on a zkVM.
  - No custom circuits; we reuse the same VM for ML code and protocol logic.
  - Demonstrates feasibility of proving realistic CNN training inside FL.
- **Towards future zkML:**
  - Larger models and more complex workloads.
  - Partial proofs, better arithmetization, hardware acceleration.
  - Combining zkVMs with folding / accumulation schemes for scalability.

Jolt vs Jolt-FL & the zkML Roadmap

- **Jolt (a16z zkVM):**
  - General-purpose zkVM for RISC-V programs.
  - Designed for programmable ZK and efficient recursion/accumulation.
- **Jolt-FL (our work):**
  - First end-to-end verifiable FL framework built directly on a zkVM.
  - No custom circuits; we reuse the same VM for ML code and protocol logic.
  - Demonstrates feasibility of proving realistic CNN training inside FL.
- **Towards future zkML:**
  - Larger models and more complex workloads.
  - Partial proofs, better arithmetization, hardware acceleration.
  - Combining zkVMs with folding / accumulation schemes for scalability.

Jolt vs Jolt-FL & the zkML Roadmap

- **Jolt (a16z zkVM):**
  - General-purpose zkVM for RISC-V programs.
  - Designed for programmable ZK and efficient recursion/accumulation.
- **Jolt-FL (our work):**
  - First end-to-end verifiable FL framework built directly on a zkVM.
  - No custom circuits; we reuse the same VM for ML code and protocol logic.
  - Demonstrates feasibility of proving realistic CNN training inside FL.
- **Towards future zkML:**
  - Larger models and more complex workloads.
  - Partial proofs, better arithmetization, hardware acceleration.
  - Combining zkVMs with folding / accumulation schemes for scalability.

Limitations and Future Work

- **Prover cost:** still too heavy for many real deployments.
- **Threat model:**
  - We guarantee *correct execution*, not *good data*.
  - Semantic poisoning or label manipulation are still possible.
- **Future directions:**
  - Optimize Jolt for ML primitives and leverage parallel hardware.
  - Integrate robust aggregation or anomaly detection with verifiable proofs.
  - Explore recursive aggregation of many Jolt-FL proofs using folding schemes.
  - Closer integration between zkVM toolchains and mainstream ML frameworks.

Takeaways & Q&A

- Jolt-FL shows that **general-purpose zkVMs** can support verifiable FL, not just hand-crafted circuits.
- We provide **strong integrity guarantees**: every accepted update is backed by a ZK proof of correct training.
- Our prototype on MNIST demonstrates:
  - Modest proof sizes and fast verification.
  - 10–15$\times$ prover overhead with preserved accuracy.
  - Robustness against a high fraction of malicious clients.
- **Big picture:** zkVM-based zkML is a promising direction for trustworthy, privacy-preserving distributed learning.

Thank you!

Questions and discussions are very welcome.

`m5281033@u-aizu.ac.jp`