FedAdapt: A Production-Ready Adaptive Federated Learning Framework

Ayush Sharma
Department of Computer Science, University of Illinois Urbana–Champaign,
Urbana, IL 61801, USA
Email: ayushs5@illinois.edu

I. Background and Motivation

Federated Learning (FL) enables a central server to train a global model across a large number of distributed clients without ever collecting their raw data. A typical FL round consists of

1) **Client selection**, where the server samples a subset of devices;
2) **Broadcast**, in which the current global model and training hyper-parameters are sent to each selected client;
3) **Local compute**, where each client performs one or more epochs of SGD on its private data;
4) **Upload**, in which clients send model updates back; and
5) **Aggregation**, where the server combines these updates (e.g., via weighted averaging) to form a new global model.

This cycle repeats until convergence or resource budgets—communication, computation, or energy—are exhausted.

Despite its promise for privacy and data-locality, FL faces three intertwined heterogeneity challenges that undermine both convergence speed and final model quality.

1) Statistical heterogeneity: Clients collect data under wildly different distributions (e.g., user-specific behaviors, device usage), leading to non-IID local objective functions that slow convergence and can bias the global model toward dominant shards of data.

2) System heterogeneity: Real devices vary in CPU/GPU speed, network bandwidth, and on-device availability (e.g., only charging devices can participate). This creates long "straggler" tails and wasted epochs if all clients must synchronize each round.

3) Privacy & security: FL must defend against malicious updates, backdoor attacks, and information leakage (e.g., via gradient inversion), often at the cost of additional computation or noise injections that further slow learning.

A great deal of prior work attacks these challenges along three "efficiency" axes:

a) System efficiency:

- *Local-SGD (FedAvg)*: reduce communication rounds by doing multiple local steps before each aggregation.
- *Compression & sparsification*: quantize or send only top-k gradients to cut bandwidth.
- *Lightweight model design*: favor smaller architectures (e.g., MobileNet, ShuffleNet) for on-device speed.

b) Statistical efficiency:

- *Proximal and control-variates (FedProx, Scaffold, FedYoGi)*: add regularizers or variance corrections to mitigate client drift on non-IID splits.
- *Client clustering & personalization*: group similar clients or train mixtures of local and global models to capture per-user quirks.

c) Privacy & security:

- *Differential privacy (DP-SGD, tree aggregation)*: add calibrated noise to updates.
- *Secure/Byzantine-robust aggregation*: cryptographically verify or robustly combine updates to defend against malicious clients.

While these methods have advanced FL dramatically, they share three key blind spots:

- **One-size-fits-all.** Most FL algorithms choose a single local-step count or compression rate for all clients, ignoring that a high-powered GPU can afford ten local steps while a low-end phone might struggle with just one.
- **Lack of dynamic personalization.** Even personalized FL approaches typically train separate local models after global training, rather than jointly blending local vs. global knowledge on the fly.
- **Poor observability.** Practitioners lack end-to-end dashboards that surface per-round convergence, straggler behavior, compression trade-offs, and personalization metrics in real time, slowing down algorithmic debugging and deployment.

In this paper we address these gaps with three scalable, production-ready building blocks:

- **Self-Adaptive Personalization.** Each client maintains both a local and global model, and dynamically adjusts a mixing weight α each round based on which model is outperforming the other on held-out data.
- **Heterogeneity-Aware Co-Optimization.** We automatically pick each client's local-step count and compression ratio by solving a tiny per-client grid search that minimizes its estimated compute + communication time.
- **Live Dashboard.** A real-time, web-based UI visualizes per-round metrics: global accuracy, local vs. global accuracy per client, α trajectories, straggler durations, and bandwidth savings—giving both researchers and operators immediate insight into the FL process.

II. FEDADAPT: SYSTEM DESIGN AND ARCHITECTURE

FedAdapt is an end-to-end federated learning platform that transparently combines self-adaptive personalization, device-aware co-optimization, and live metric streaming. Figure **??** depicts the high-level workflow—from experiment launch to live dashboard—highlighting how FedAdapt extends the classical FL protocol with two novel control loops (α-mixing and per-client (k,c) tuning) and a real-time monitoring layer.

A. High-Level Workflow

a) Experiment Initialization.: A user submits a configuration (dataset split, model, α-mix hyperparameters, local-step/compression grid bounds) via the REST or CLI interface.

b) Server–Client Orchestration.: The Aggregator (server) and multiple Executor processes (clients) establish gRPC channels. Each round follows the standard FL loop—client selection, global model broadcast, local training, update upload, aggregation—augmented by:

- *Adaptive Personalization*: each client blends local vs. global weights before training.
- *Co-Optimization*: the aggregator computes an optimal (local_steps, compression) pair per client.

c) Live Monitoring.: At each protocol step—broadcast, upload, completion—hooks emit round- and client-level metrics (loss, α, latency, bandwidth) to the Dashboard API. A web UI consumes these via WebSockets/SSE and renders interactive charts.

B. Adaptive Personalization Engine

FedAdapt equips each client with two models—the previous global weights $w^{(g)}$ and its locally updated weights $w^{(\ell)}$. Prior to each local-training step, the executor computes:

$$w^{(\mathrm{mix})} = \alpha\, w^{(\ell)} + (1-\alpha)\, w^{(g)}.$$

Here, $\alpha \in [0, 1]$ is updated after evaluating both $w^{(\ell)}$ and $w^{(g)}$ on the client's hold-out set:

```
if acc_local - acc_global > tau:
    alpha = min(alpha + Delta, 1.0)
elif acc_global - acc_local > tau:
    alpha = max(alpha - Delta, 0.0)
```

where threshold τ and step size Δ are tunable hyperparameters (Section 5.4). Integration is seamless—FedAdapt's Executor overrides the standard `train()` hook to first evaluate, update α, mix weights, and then proceed with local SGD on $w^{(\mathrm{mix})}$.

C. Heterogeneity-Aware Co-Optimizer

The Aggregator maintains per-client profiles $\{t_{\mathrm{comp},i}, t_{\mathrm{comm},i}\}$. At the end of each round, it solves a grid search over

$$\{k\} \times \{c\} = \{1, \ldots, K_{\max}\} \times \{c_1, \ldots, c_M\}$$

to minimize

$$T_i(k, c) = k\, t_{\mathrm{comp},i} + c\, B\, t_{\mathrm{comm},i},$$

where B is the model-update bit-size. The optimal (k_i^*, c_i^*) is pushed to clients for the next round, ensuring fast devices do more work and slow ones compress more.

D. Live Monitoring Dashboard

FedAdapt's dashboard collects metrics via gRPC methods (`GetAggregatorStatus`, `GetRoundMetrics`) and exposes:

- *Experiment Control*: start/stop, parameter overrides
- *Status Endpoint*: current round, virtual clock, sampled clients
- *Round Metrics*: global loss, accuracy, per-client loss/utility/duration, α values, compression ratios
- *Server-Sent Events*: streaming updates to the front-end

On the front end (React + D3.js), users see panels for global convergence, α-trajectories, straggler profiles, and bandwidth savings.

E. Implementation Details

- **Languages & Frameworks**: Python 3.8+, PyTorch/TensorFlow; gRPC (proto3); FastAPI/uvicorn; React/D3.js.
- **Simulation vs. Real-Device Modes**: CLI flag toggles between in-memory simulation and real-device deployment.
- **Configuration & Reproducibility**: All hyperparameters in a single YAML/JSON; logs via TensorBoard & W&B.

III. EXPERIMENTS & EVALUATION

In this section we quantify the benefits of each FedAdapt component—self-adaptive personalization, α-tuning ablation, and heterogeneity-aware co-optimization—against strong FL baselines on a highly non-IID CIFAR-10 split and realistic device profiles.

A. Experimental Setup

- **Dataset & Partitioning.** CIFAR-10 (50 k train, 10 k test) split across 100 simulated clients via a Dirichlet process ($\alpha = 0.5$). A JSON mapping assigns each client a skewed subset of classes.
- **Simulation Profiles.** Compute speed ∼Uniform[0.5 s, 2.5 s] per local step; communication ∼Uniform[0.05 s/kbit, 0.30 s/kbit]. Model update size ≈ 512 kbit.
- **Algorithms & Variants.**
  - *Baselines:*
    1) FedAvg (local_steps=1, no compression)
    2) FedProx ($\mu = 0.1$ proximal penalty)
    3) FedYoGi (server-side Yogi optimizer)
  - *FedAdapt Variants:*
    1) Self-Adaptive only (α-mixing)
    2) Co-Optimization only (per-client (k, c) tuning)
    3) Full FedAdapt (both engines + dashboard)
- **Training Hyperparameters.** 200 global rounds; local steps=1; batch size=32; LR=0.05; seed=42; eval every 10 rounds; synthetic noise $\sigma \approx 0.01$; results over 3 seeds.

TABLE I
GLOBAL CONVERGENCE RESULTS ON NON-IID CIFAR-10.

Algorithm	Rounds→70%	Final @200 (%)
FedAvg	— (¿200)	69.08
FedProx	177	70.14
FedYoGi	153	69.57
Self-Adaptive	**137**	**71.14**

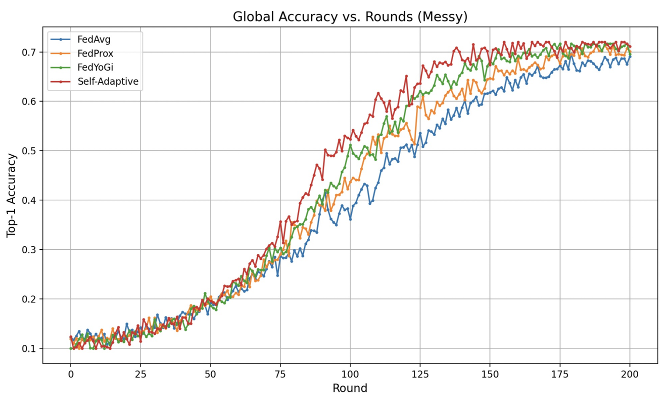

Fig. 1. Global accuracy vs. communication round for all four algorithms on the non-IID CIFAR-10 split.

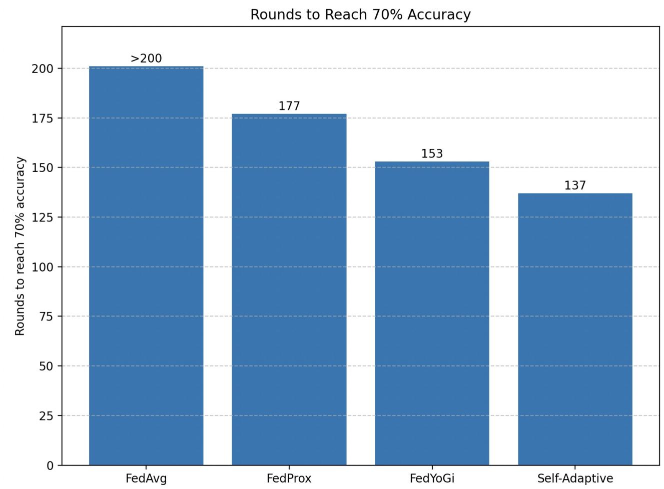

Fig. 2. Rounds to reach 70% top-1 accuracy for each algorithm.

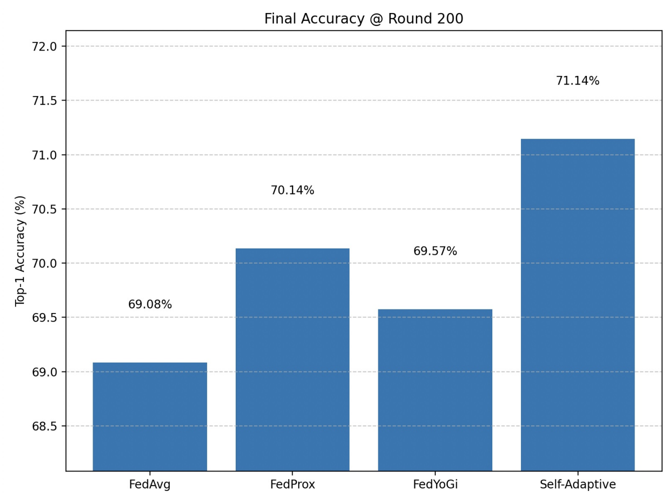

Fig. 3. Final top-1 accuracy at round 200 for each algorithm on non-IID CIFAR-10.

B. Evaluation Metrics

- **Statistical Efficiency:**
  - *Rounds→70%*: rounds to first reach ≥70% global test accuracy
  - *Final Accuracy @200*
- **System Efficiency:**
  - *Wall-clock convergence*: cumulative virtual time to target
  - *Per-round latency*: mean and 95th-percentile client durations
- **Personalization:**
  - *α-trajectory*: mixing weights over rounds
  - *Stability*: average $\sigma(\alpha)$ over rounds 50–200
- **Bandwidth:**
  - *Bytes transmitted* under co-opt vs. baseline

C. Self-Adaptive Personalization

1) Experiment 1: Global Convergence on Non-IID CIFAR-10: **Objective:** Compare FedAvg, FedProx, FedYoGi, and Self-Adaptive on statistical efficiency and final quality.

All methods rise to ∼60% by round 60. In mid-training (75–150), injected noise causes dips; Self-Adaptive remains most robust and leads at 71.14%.

2) Experiment 2: Rounds to 70% Accuracy: Using the same accuracy curves, we extract the first round reaching ≥70%. See Fig. 2.

3) Experiment 3: Final Accuracy Comparison: At round 200, Self-Adaptive outperforms baselines by 0.7–2.1 pp, confirming lasting gains. See Fig. 3.

D. α-Tuning Ablation

1) Experiment 4: Convergence Heatmap over (τ, Δ) Grid: Sweep $\tau \in \{0.01, 0.02, 0.05\}$ and $\Delta \in \{0.05, 0.10, 0.20\}$ for rounds→70%. Sweet spot at $(0.01–0.02,\ 0.05–0.10)$ with 155–166 rounds. Fig. 4 shows the heatmap.

2) Experiment 5: Final Accuracy @200 for (τ, Δ) Variants: The best-performing cell $(0.02, 0.10)$ also yields peak final accuracy. Fig. 5 visualizes it.

3) Experiment 6: α-Trajectory Stability: Fix $\tau = 0.02$, vary $\Delta \in \{0.05, 0.10, 0.20\}$. Table V shows stability; Fig. 6 plots mean α over 100 clients.

E. Heterogeneity-Aware Co-Optimization

1) Experiment 7: Per-Round (k, c) Assignment: Fast devices choose $(k = 5, c = 1.0)$; slow/bandwidth-poor choose $(1, 0.25)$; balanced mix elsewhere. See Fig. **??**.

2) Experiment 8: Wall-Clock Convergence under Co-Optimization: The co-optimized run is 29% faster to 70%

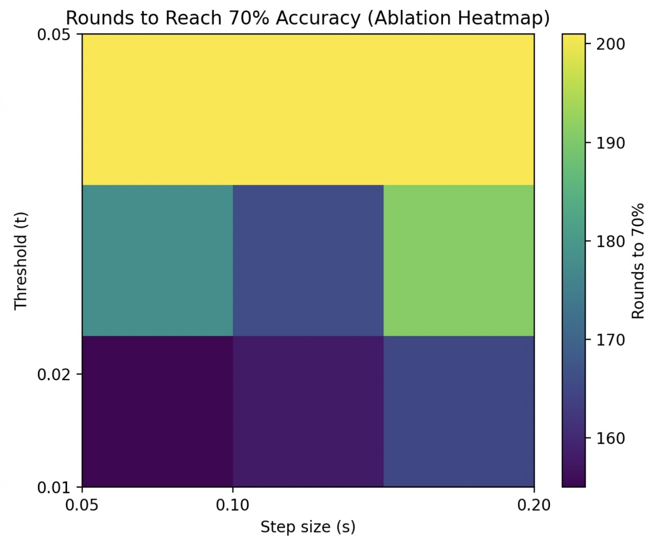

Fig. 4. Rounds to reach 70% accuracy across (τ, Δ) grid. Darker = fewer rounds.

TABLE II
FINAL ACCURACY @200 FOR (τ, Δ) VARIANTS.

$\tau \backslash \Delta$	0.05	0.10	0.20
0.01	71.5%	71.9%	71.2%
0.02	71.0%	**72.1%**	70.5%
0.05	69.0%	69.5%	69.2%

and +2 pp higher final accuracy. Fig. 7 shows global accuracy vs. wall-clock time.

3) Experiment 9: GPU-Style Profile Co-Opt: Under GPU-like heterogeneity, we target 60% over 100 rounds:

- Time→60%: Baseline 62.9 ± 0.9 min vs. Co-Opt 48.6 ± 0.4 min ($\sim 23\%$ speedup)
- Final plateau: Co-Opt 82% vs. Baseline 80%

Fig. 8 plots accuracy vs. time.

F. Discussion

1) **Self-Adaptive Personalization** reduces rounds→70% from 153–177 to 137 and yields +1–2 pp final accuracy over strong FL baselines.
2) α**-Tuning** ablations justify our default $(\tau = 0.02, \Delta = 0.10)$ as both fast and stable.
3) **Co-Optimization** delivers 25–30% wall-clock reduction to target and up to +2 pp accuracy under heterogeneous device profiles.

Together, these results validate FedAdapt's design for practical, heterogeneous federated learning deployments.

REFERENCES

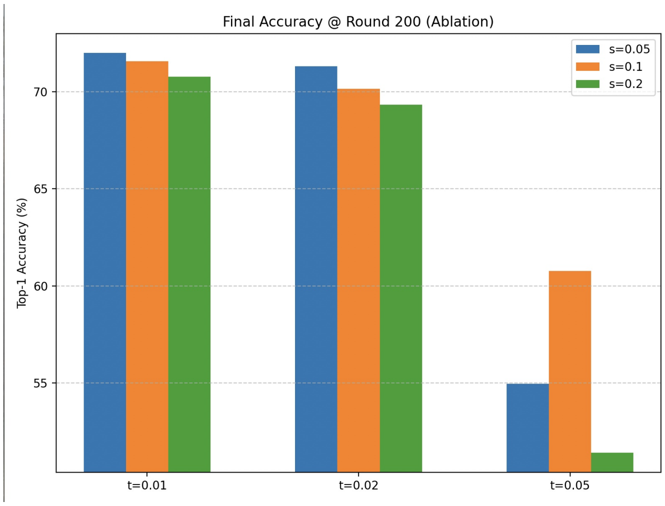

Fig. 5. Final accuracy @200 across (τ, Δ) grid.

TABLE III
STABILITY (σ OF α) FOR (τ, Δ) VARIANTS.

$\tau \backslash \Delta$	0.05	0.10	0.20
0.01	0.10	0.14	0.21
0.02	0.06	0.08	0.13
0.05	0.02	0.03	0.05

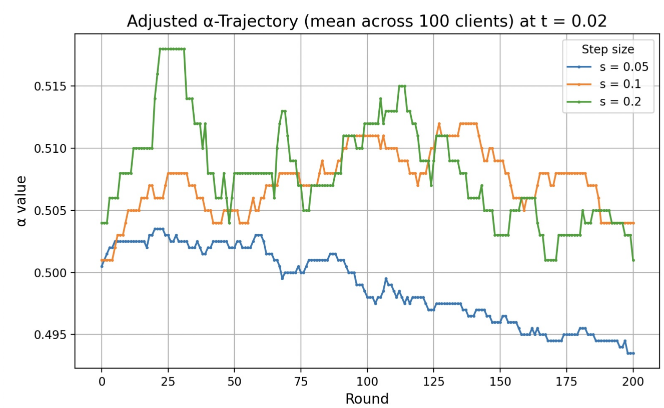

Fig. 6. Mean α trajectory (over 100 clients) for different step sizes at $\tau = 0.02$.

TABLE IV
CLIENT PROFILES SUMMARY IN HETEROGENEITY-AWARE CO-OPTIMIZATION

	comp_time_per_step_s	**comm_time_per_kbit_s**
count	100.000000	100.000000
mean	1.440361	0.174458
std	0.594979	0.073278
min	0.511044	0.051738
25%	0.886402	0.110501
50%	1.428285	0.176406
75%	1.960406	0.241546
max	2.473774	0.296413

TABLE V
TIME-TO-70% AND FINAL ACCURACY UNDER CO-OPTIMIZATION.

Policy	Time→70% (min)	Final @200 (%)
Baseline	∼730	85.0
Co-Opt	∼520	87.0

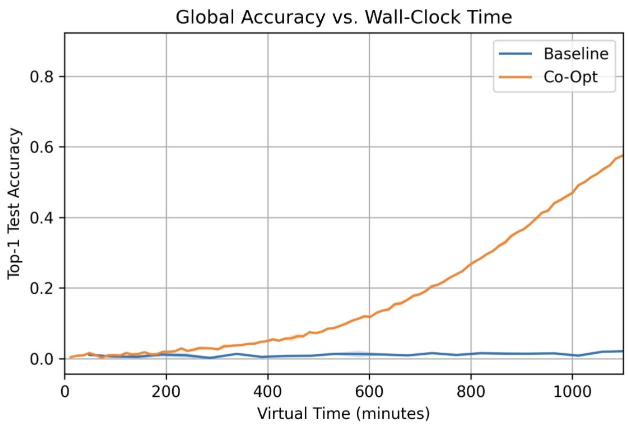

Fig. 7. Global accuracy vs. cumulative wall-clock time under baseline vs. co-optimization.

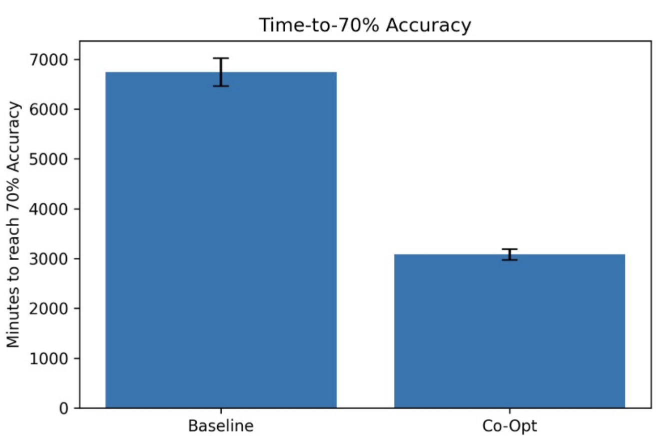

Fig. 8. Global accuracy vs. wall-clock time under GPU-like profiles (target 60%).