TrainTogether: Distributed Models and Free Software in Federated Learning

Linux Day — Federated Learning & Open Source with Flower

Fabio Turazza
PhD Student — DISMI, University of Modena and Reggio Emilia

October 25, 2025

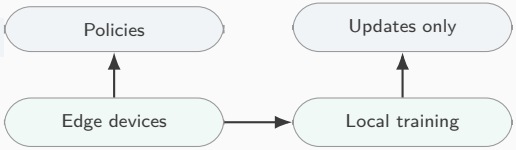
Why *not* always send data to the cloud?

Three real-world constraints

- **Privacy & Data sovereignty** (sensitive data remain on premises)
- **Bandwidth costs** (intermittent, constrained uplinks)
- **Operational latency** (faster local reactions)

Key idea

Train models where data are born and exchange only updates.



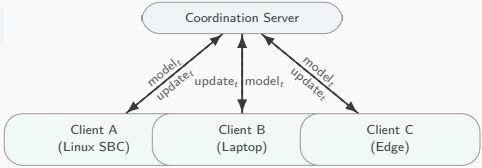
What is Federated Learning (FL)

High-level definition

A central *server* orchestrates multiple clients that:

1. train the model **locally**
2. send **updates** (weights/gradients)
3. receive an **aggregated model**

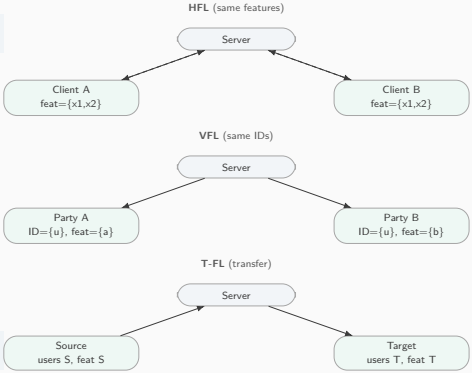
- Different from centralized training: data never leave devices
- Complementary to *edge inference*



Federated Learning: Horizontal vs Vertical vs Transfer

Taxonomy

- **Horizontal FL (HFL)** — same *feature space*, different *users/rows*.
  Ex: two hospitals share the EHR schema, different patients.
- **Vertical FL (VFL)** — same *users/IDs*, different *feature spaces/columns*.
  Ex: bank + telco, same customers, distinct attributes.
- **Transfer FL (T-FL)** — different *users* and *features*; transfer via pretrain/KD/adapters.
  Ex: image source → sensor target with little/no overlap.



Operational notes

- **HFL**: FedAvg-like aggregation; no ID join.
- **VFL**: privacy-preserving ID alignment & feature fusion.
- **T-FL**: pretrain on source, adapt on target (KD/LoRA/heads).

Horizontal vs Vertical vs Transfer — toy datasets

HFL — same features

A	f1	f2	y
u101	0.2	3.1	0
u102	0.5	2.7	1
u103	0.4	3.5	0

B	f1	f2	y
u201	0.3	3.2	1
u202	0.6	2.4	0
u203	0.7	2.9	1

VFL — same IDs

Party A		
ID	a1	a2
u301	1.2	0.4
u302	0.9	0.7
u303	1.1	0.5

Party B		
ID	b1	b2
u301	3.4	5.1
u302	2.8	4.7
u303	3.0	5.0

T-FL — transfer

Source A	s1	s2	y
s401	0.1	7.1	0
s402	0.3	6.8	1
s403	0.5	6.5	1

Target B	t1	t2
t501	2.4	0.9
t502	2.1	1.2
t503	2.0	1.1

Legend: HFL = rows differ; VFL = IDs match, columns differ; T-FL = transfer across users/features

Federated Learning: key challenges (compact)

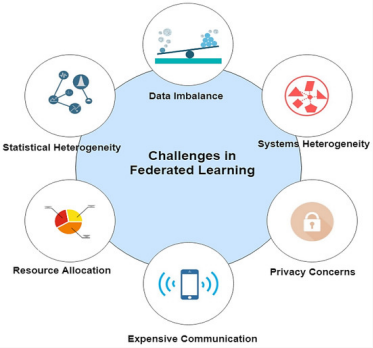


Illustration of clients, server, and noisy/robust updates

Data & learning

- **Non-IID & drift**: label/feature/quantity skew; changes over time.
- **Personalization vs global**: align local and global objectives.
- **Sound evaluation**: per-client metrics, reproducibility.

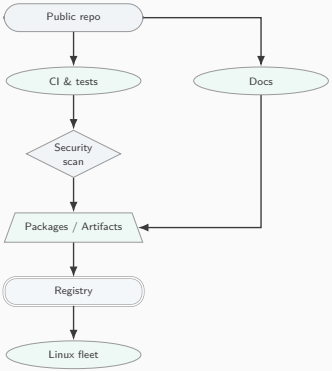
Systems, privacy & ops

- **Multi-round cost**: latency, bandwidth, stragglers/partial participation.
- **On-device limits**: CPU/GPU, RAM/energy; heterogeneous HW/OS.
- **Privacy leakage**: gradient/weight inversion, membership inference.
- **Attacks**: poisoning/backdoors, Sybil; need robust aggregation.
- **Trade-offs & governance**: DP noise vs utility, SA/HE overheads; observability, GDPR/residency.

Why Open Source & Linux-centric

Values

- **Auditability**: inspectable code (security & trust)
- **Reproducibility**: repeatable experiments
- **Portability**: x86/ARM, containers, Kubernetes
- **No lock-in**: open standards, freedom to fork



FOSS for Federated Learning: practical advantages

Aspect	Free/Open-Source (Linux/Flower)	Proprietary Platform
Auditability that matters	Code and update handling are inspectable; real risk reviews ✓	Black-box paths; trust delegated; limited evidence ✗
Cost control & exit	No per-device licenses; self-hosting; easy to migrate/fork ✓	Per-node/user fees; contract lock-in; exit is costly ✗
Data locality & compliance	Self-managed deployment; data stay on-prem/country ✓	SaaS regions/metadata constraints; opaque paths ✗
Heterogeneous hardware	Runs the same on x86/ARM, containers, K8s; SBC → DC ✓	SDK/runtime limits; ARM often second-class ✗
Extensibility you'll use	Plug DP/SA/HE; custom strategies/filters; quick patches ✓	Features gated/paywalled; slow vendor roadmap ✗
Observability & debugging	Prometheus/Logs/tracing; reproduce issues anywhere ✓	Closed telemetry; hard to extract signals ✗
Offline/edge reality	Works with flaky links; simulate many clients locally ✓	Tied to cloud control plane; poor offline story ✗

DP=Differential Privacy, *SA*=Secure Aggregation, *HE*=Homomorphic Encryption

Flower vs Proprietary & Other OSS

Flower

Framework-agnostic FL

- **Multi-framework**: PyTorch/TF/JAX
- **Minimal API**: NumPyClient, Strategy
- **Edge-first**: simulate → edge/K8s
- **Privacy-ready**: DP/LDP, SA hooks

Area	Proprietary FL	Other OSS (TFF/FedML/OpenFL...)	Flower (FOSS)
Setup & learning curve	Vendor SDKs; steep	Academic/verbose APIs	**In minutes**: NumPyClient, Strategy
Framework support	Tied to runtime/stack	Often single-stack	**Multi-framework**: PyTorch/TF/-JAX
Deploy (edge ↔ DC)	Cloud control-plane bias	Good for research; production gap	**Local sim** → edge/K8s, same code
Privacy & robustness	Opaque/limited	Variable focus (HE/SMPC)	**Pluggable**: DP/LDP, SA, robust rules
Heterogeneous HW	Licenses/regions; ARM second-class	Mixed support	**x86/ARM, Docker/K8s, SBC → DC**

DP = Differential Privacy, LDP = Local DP, SA = Secure Aggregation

Flower in 5 minutes — Server

Pillars

- **Framework-agnostic**: PyTorch/TF/JAX
- **Simple** client/server APIs
- **Extensible** strategies
- Built-ins: FedAvg, FedProx, etc.
- Runs on: bare metal, Docker, K8s

```python
import flwr as fl

strategy = fl.server.strategy.FedAvg(
    min_fit_clients=2, min_available_clients
    =2
)
fl.server.start_server(
    server_address="0.0.0.0:8080",
    strategy=strategy,
    config=fl.server.ServerConfig(num_rounds
    =5)
)
```

Listing 1: Minimal Flower server (FedAvg)

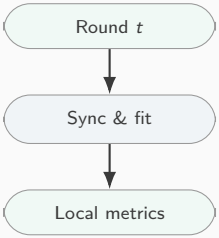
Live Demo: “FL in your pocket” — Client (Part 1/2)

```
import flwr as fl, torch

class Client(fl.client.NumPyClient):
    def get_parameters(self, cfg):
        return [p.detach().cpu().numpy()
                for p in model.parameters()]

    def fit(self, params, cfg):
        for p, npv in zip(model.parameters(), params):
            p.data = torch.tensor(npv)
        train_one_epoch(model, loader, device)
        return self.get_parameters({}), len(loader.
     dataset), {}
```

Listing 2: Client — init, params, fit



Live Demo: “FL in your pocket” — Client (Part 2/2)

```python
def evaluate(self, params, cfg):
    for p, npv in zip(model.parameters(), params):
        p.data = torch.tensor(npv)
    loss, acc = evaluate(model, val_loader, device)
    return float(loss), len(val_loader.dataset), {"acc":
     float(acc)}

fl.client.start_numpy_client("SERVER_IP:8080", Client())
```

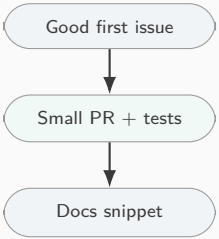
Listing 3: Client — evaluate, run

Fallback plan

Simulated clients on a single machine (`-num_clients`).

Contributing to the ecosystem (licenses & community)

Area	High-impact contributions
Packaging	Multi-arch Dockerfiles, Helm charts, systemd units
CI & QA	GitHub Actions, ARM integration tests
Strategies	Robust aggregation, FedProx variants
Observability	Structured logging, Prometheus exporters
Docs & DevRel	Tutorials, examples, quickstarts for SBCs



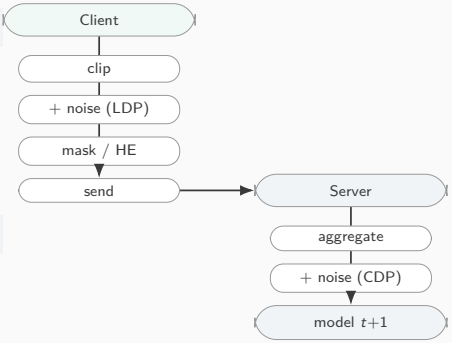
Privacy in Federated Learning

Differential Privacy (DP)

- **Local DP**: noise *on client* before sending (privacy↑, utility↓).
- **Central DP**: noise *on server* after aggregation (utility↑, needs trust/SA/HE).

Protecting updates

- **Secure Aggregation (SA)**: additive masks → server sees only the sum.
- **Homomorphic Encryption (HE)**: add/aggregate on encrypted vectors (Paillier/CKKS) — CPU/latency/bandwidth↑.



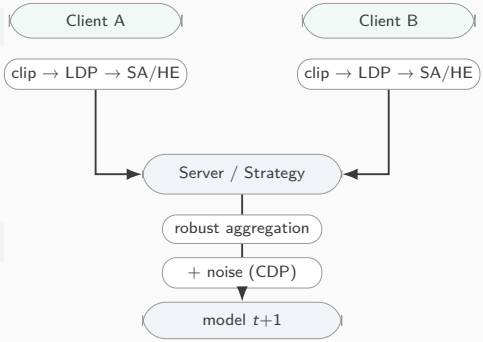
Robustness & How it's done in Flower

Robustness tools

- **Gradient clipping** & **outlier filters**.
- **Robust aggregation**: median, trimmed mean, Krum.
- **Heterogeneity**: sampling, timeouts, data-weighted updates.

Where to plug in Flower

- **Client** (`fit/evaluate`): clip → local DP → SA/HE before returning updates.
- **Server / Strategy** (`aggregate_fit`): robust rule → central DP → model update (optional unmask/decrypt).



Key messages & research landscape

Key messages

Local data
privacy, compliance, cost

Simple orchestration
Flower lowers time-to-FL

Free software
autonomy, community, reuse

Key takeaways so far

FL mental model
local-only data; send updates

Flower minimal API
fit/evaluate, Strategy, start_server/client

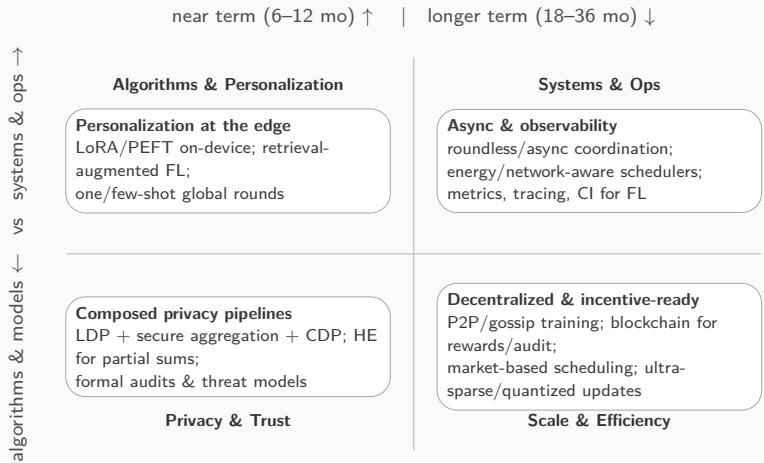
Open Source & Linux
portability, reproducibility, no lock

Privacy & robustness
LDP/CDP, SA, HE; robust rules

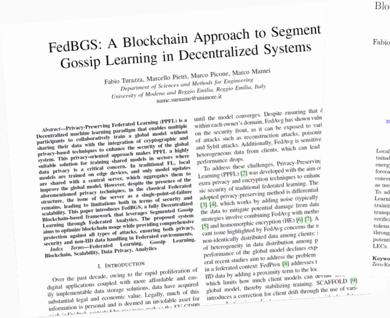
Live demo recap
hetero clients; rounds ↑ accuracy

Ops quick tips
venv; –num_clients

FL future directions — roadmap (6–36 months)



Our contributions



FedBGS: A Blockchain Approach to Segment Gossip Learning in Decentralized Systems

Fabio Turazza, Marcello Pietri, Marco Picone, Marco Mamei
Department of Sciences and Methods for Engineering
University of Modena and Reggio Emilia, Reggio Emilia, Italy
name.surname@unimore.it

Abstract—Privacy-Preserving Federated Learning (PPFL) is a Decentralized machine learning paradigm that enables multiple participants to collaboratively train a global model without sharing their data with the integration of cryptographic and privacy-based techniques to enhance the security of the global system. This privacy-oriented approach makes PPFL a highly suitable solution for training shared models in sectors where data privacy is a critical concern. In traditional FL, local models are trained on edge devices, and only model updates are shared with a central server, which aggregates them to improve the global model. However, despite the presence of the aforementioned privacy techniques, in the classical Federated structure, the issue of the server as a single-point-of-failure remains, leading to limitations both in terms of security and scalability. This paper introduces FedBGS, a fully Decentralized Blockchain-based framework that leverages Segmented Gossip Learning through Federated Analytics. The proposed system aims to optimize blockchain usage while providing comprehensive protection against all types of attacks, ensuring both privacy, security and non-IID data handling in Federated environments.

Index Terms—Federated Learning, Gossip Learning, Blockchain, Scalability, Data Privacy, Analytics

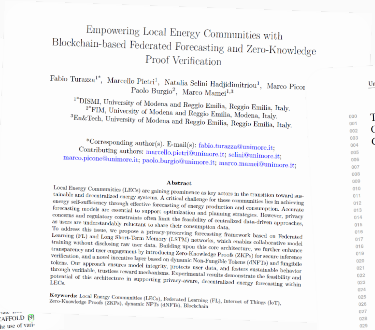
I. Introduction

Over the past decade, owing to the rapid proliferation of digital applications coupled with more affordable and easily implementable data storage solutions, data have acquired substantial legal and economic value. Legally, much of this information is personal and is deemed an inviolable asset for

until the model converges. Despite ensuring that d within each owner's domain, FedAvg has shown vuln on the security front, as it can be exposed to vari of attacks such as reconstruction attacks, poisonir and Sybil attacks. Additionally, FedAvg is sensitive heterogeneous data from clients, which can lead performance drops.

To address these challenges, Privacy-Preservin Learning (PPFL) [2] was developed with the aim o extra privacy and encryption techniques to enhanc sic security of traditional federated learning. The adopted privacy-preserving method is differential [3], [4], which works by adding noise (typically the data to mitigate potential damage from data strategies involve combining FedAvg with metho [5] and homomorphic encryption (HE) [6], [7]. A cant issue highlighted by FedAvg concerns the non-identically distributed data among clients; of heterogeneity in data distribution among p performance of the global model declines exp eral recent studies aim to address the problem in a federated context: FedProx [8] addresses t IID data by adding a proximity term to the loc which limits how much client models can deviate global model, thereby stabilizing training. SCAFFOLD [9] introduces a correction for client drift through the use of vari-

FedBGS
Blockchain + Segmented Gossip FL



Empowering Local Energy Communities with Blockchain-based Federated Forecasting and Zero-Knowledge Proof Verification

Fabio Turazza[1*], Marcello Pietri[1], Natalia Selini Hadjidimitriou[1], Marco Picon Paolo Burgio[2], Marco Mamei[1,3]
[1*]DISMI, University of Modena and Reggio Emilia, Reggio Emilia, Italy.
[2*]FIM, University of Modena and Reggio Emilia, Modena, Italy.
[3]En&Tech, University of Modena and Reggio Emilia, Reggio Emilia, Italy.

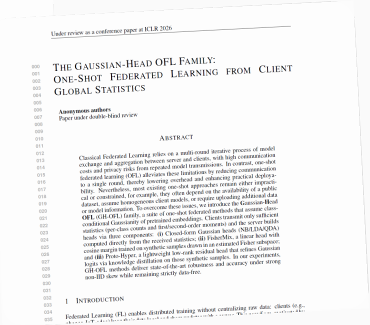
*Corresponding author(s). E-mail(s): fabio.turazza@unimore.it;
Contributing authors: marcello.pietri@unimore.it; selini@unimore.it; marco.picone@unimore.it; paolo.burgio@unimore.it; marco.mamei@unimore.it;

Abstract

Local Energy Communities (LECs) are gaining prominence as key actors in the transition toward sustainable and decentralized energy systems. A critical challenge for these communities lies in achieving energy self-sufficiency through effective forecasting of energy production and consumption. Accurate forecasting models are essential to support optimization and planning strategies. However, privacy concerns and regulatory constraints often limit the feasibility of centralized data-driven approaches, as users are understandably reluctant to share their consumption data. To address this issue, we propose a privacy-preserving forecasting framework based on Federated Learning (FL) and Long Short-Term Memory (LSTM) networks, which enables collaborative model training without disclosing raw user data. Building upon this core architecture, we further enhance transparency and user engagement by introducing Zero-Knowledge Proofs (ZKPs) for secure inference verification, and a novel incentive layer based on dynamic Non-Fungible Tokens (dNFTs) and fungible tokens. Our approach ensures model integrity, protects user data, and fosters sustainable behavior through verifiable, trustless reward mechanisms. Experimental results demonstrate the feasibility and potential of this architecture in supporting privacy-aware, decentralized energy forecasting within LECs.

Keywords: Local Energy Communities (LECs), Federated Learning (FL), Internet of Things (IoT), Zero-Knowledge Proofs (ZKPs), dynamic NFTs (dNFTs), Blockchain

LECs & FL + ZKPs
Forecasting with verifiable privacy



Under review as a conference paper at ICLR 2026

The Gaussian-Head OFL Family: One-Shot Federated Learning from Client Global Statistics

Anonymous authors
Paper under double-blind review

Abstract

Classical Federated Learning relies on a multi-round iterative process of model exchange and aggregation between server and clients, with high communication costs and privacy risks from repeated model transmissions. In contrast, one-shot federated learning (OFL) alleviates these limitations by reducing communication to a single round, thereby lowering overhead and enhancing practical deployability. Nevertheless, most existing one-shot approaches remain either impractical or constrained, for example, they often depend on the availability of a public dataset, assume homogeneous client models, or require uploading additional data or model information. To overcome these issues, we introduce the Gaussian-Head **OFL** (GH-OFL) family, a suite of one-shot federated methods that assume class-conditional Gaussianity of pretrained embeddings. Clients transmit only sufficient statistics (per-class counts and first/second-order moments) and the server builds heads via three components: **(i)** Closed-form Gaussian heads (NB/LDA/QDA) computed directly from the received statistics; **(ii)** FisherMix, a linear head with cosine margin trained on synthetic samples drawn in an estimated Fisher subspace; and **(iii)** Proto-Hyper, a lightweight low-rank residual head that refines Gaussian logits via knowledge distillation on those synthetic samples. In our experiments, GH-OFL methods deliver state-of-the-art robustness and accuracy under strong non-IID skew while remaining strictly data-free.

1 Introduction

Federated Learning (FL) enables distributed training without centralizing raw data: clients (e.g.,

GH–OFL Family
One-shot FL from client global stats

Code, artifacts, and extended results available upon request.

Q&A

Thank you!

fabio.turazza@unimore.it | @fabio_turazza |
linkedin.com/in/fabio-turazza-344521319

Appendix (quick setup)

Server

```
python -m venv .venv && source .venv/bin/
    activate
pip install flwr torch torchvision
python server.py
```

Client

```
source .venv/bin/activate
python client.py --data ./local_dataset
```