Review

Not peer-reviewed version

A Comprehensive Survey of Federated Learning for Edge AI: Recent Trends and Future Directions

Sai Puppala [*] , Ismail Hossain , Jahangir Alam , Tanzim Ahad , Sajedul Talukder

Posted Date: 2 December 2025

doi: 10.20944/preprints202512.0118.v1

Keywords: federated learning; edge AI; edge computing architectures

Review

A Comprehensive Survey of Federated Learning for Edge AI: Recent Trends and Future Directions

Sai Puppala [1,*], Ismail Hossian [2], Md Jahangir Alam [2], Tanzim Ahad[2], and Sajedul Talukder[2]

[1] Southern Illinois University - Carbondale, IL - 62901
[2] University of Texas - El paso, University Ave, El Paso, TX 79968
[*] Correspondence: sai.puppala@siu.edu

Abstract

Federated learning (FL) has emerged as a key enabler of privacy-preserving, distributed intelligence, yet most deployments and surveys remain cloud-centric. This paper presents a unified, forward-looking survey of federated learning and Edge AI beyond the centralized cloud, with an emphasis on work from roughly 2021–2025. We organize and analyze over 200 papers spanning cross-device, cross-silo, edge-only, decentralized, and split/hybrid designs. First, we introduce the foundations of cloud–fog–edge computing, multi-tier network hierarchies, and edge/on-device AI, and formalize the FL optimization problem and its main variants. We then develop a taxonomy of FL architectures for Edge AI, comparing coordination topologies, aggregation locality, trust and failure models, and communication complexity. Next, we synthesize cross-layer challenges—including statistical and system heterogeneity, stragglers, communication and energy efficiency, and privacy, security, and trust—and review enabling techniques such as personalization, client selection, compression, green FL, and secure and robust aggregation. We survey hardware accelerators, TinyML platforms, neuromorphic and processing-in-memory prototypes, and major FL/edge AI frameworks. Finally, we summarize applications in healthcare, transportation, industrial IoT, and smart cities, and outline future directions in federated continual learning, on-device adaptation of foundation models, and the convergence of FL with TinyML and emerging hardware.

Keywords: federated learning; edge AI; edge computing architectures

---

0. Introduction

The deployment of AI in real-world systems is rapidly shifting from monolithic, cloud-centric processing to a continuum of cloud, edge, and on-device resources. Billions of edge devices—smartphones, cameras, wearables, industrial controllers, vehicles, and microcontroller-based sensors—generate high-volume, high-velocity, and often privacy-sensitive data streams. Classical "data-to-cloud" pipelines collect these streams into centralized data centers for both training and inference. While conceptually simple and aligned with the rise of large foundation models, this paradigm runs into four persistent and increasingly severe limitations:

- **Latency.** Round-trip times to remote data centers are often incompatible with interactive or safety-critical applications such as AR/VR, cooperative driving, or closed-loop industrial control.
- **Bandwidth.** Uploading raw or lightly processed sensor data from massive fleets of devices is expensive, competes with other traffic, and can be infeasible in constrained or bursty networks (e.g., cellular uplinks, LPWANs).
- **Privacy, regulation, and data residency.** Data-protection laws (e.g., GDPR, HIPAA) and sector-specific regulations increasingly restrict cross-border and cross-organizational movement of raw data, while users and operators demand stronger privacy and governance guarantees.

- **Robustness, sustainability, and energy.** Heavy dependence on centralized infrastructure creates single points of failure and exacerbates the carbon footprint of repeated long-haul transfers and always-on connectivity, conflicting with emerging goals in green and resilient AI.

  Two paradigms have emerged as key enablers of a more distributed, "beyond-the-cloud" future:

Edge AI pushes inference and, increasingly, training closer to data sources—onto edge servers (MEC nodes, micro data centers, gateways) and resource-constrained devices (phones, wearables, microcontrollers). Edge AI exploits local compute and storage to deliver low-latency responses, reduce backbone traffic, and improve robustness to connectivity disruptions.

Federated learning (FL) trains global or multi-tenant models across many clients without centralizing raw data. Clients perform local optimization on their private datasets and share model parameters, gradients, or compressed updates with one or more aggregators, which periodically produce global or regional models.

Individually, each paradigm addresses a subset of the above issues: edge AI tackles latency and bandwidth by relocating computation, whereas FL addresses privacy and data residency by relocating learning. When combined, *federated edge AI* offers a unifying approach to privacy-preserving, low-latency, and resource-aware intelligence at the network edge. Edge devices and servers can collaboratively train and adapt models to local conditions while benefiting from shared knowledge across sites, operators, and institutions. At the same time, this combination surfaces new cross-layer trade-offs: edge deployments are inherently hierarchical and heterogeneous; data is non-IID, sparse, and evolving; and systems must balance communication, energy, and privacy budgets against accuracy and personalization demands.

Recent years have therefore seen an explosion of work at the intersection of FL, edge computing, and on-device AI. Beyond the canonical *centralized* cross-device and cross-silo setups, researchers have proposed:

- *Hierarchical and multi-tier FL* that explicitly mirrors network hierarchies (device–edge–cloud, multi-level edge), combining local adaptation with global coordination.
- *Edge-only and on-premise FL* for hospitals, factories, and critical infrastructure, where data and coordination remain within tightly governed domains.
- *Decentralized and blockchain-enabled FL*, using gossip, consensus, and distributed ledgers to remove or weaken central trust anchors in vehicular, UAV, and community networks.
- *Split and hybrid architectures* that combine FL with split learning and model offloading, enabling devices and TinyML platforms to leverage large backbones hosted at edge or cloud while keeping sensitive signals local.

Parallel efforts in hardware accelerators (NPUs, edge GPUs, neuromorphic and in-memory prototypes), TinyML and microcontroller platforms, and new FL/edge frameworks (e.g., Flower, FedML, FATE, TinyFL) have begun to make such architectures practical across diverse domains, from healthcare and intelligent transportation to industrial IoT and smart cities.

Despite this rapid progress, the landscape remains fragmented across communities (ML, networking, systems, hardware, security, and domain-specific areas such as healthcare and transportation). Many survey works focus either on FL algorithms in abstract settings or on edge computing and MEC without deeply integrating modern FL and on-device adaptation. There is a need for a synthetic view that treats federated edge AI as a *system-of-systems problem*: jointly considering architectures, optimization and personalization strategies, communication and energy constraints, privacy and trust, and the evolving role of foundation models and TinyML at the edge.

0.1. Scope and Contributions

This paper provides a unified, forward-looking survey of *federated learning and edge AI beyond the cloud*, with a particular emphasis on the last four years of research that spans cross-device, cross-silo, edge-only, decentralized, and split/hybrid designs. Concretely, we:

- **Set the stage.** We introduce the basics of cloud–fog–edge computing, multi-tier network hierarchies, and edge/on-device AI, and we formalize the FL optimization problem and its variants (cross-device, cross-silo, personalized, hierarchical, and decentralized) relevant to edge settings.
- **Map the architectural design space.** We develop a taxonomy of FL architectures for edge AI, spanning centralized, hierarchical, edge-only, decentralized, and split/hybrid patterns. For each, we discuss coordination topology, aggregation locality, trust and failure models, communication complexity, and representative application domains, supported by comparative tables.
- **Analyze cross-layer challenges and techniques.** We synthesize core challenges in federated edge AI—statistical heterogeneity and personalization, system heterogeneity and stragglers, communication and energy efficiency, and privacy, security, and trust. We review enabling techniques such as personalized FL, client and cluster selection, compression and over-the-air aggregation, green FL strategies, and combinations of differential privacy, secure aggregation, and robust/Byzantine-resilient aggregation.
- **Survey hardware and software enablers.** We review edge hardware accelerators, TinyML and microcontroller platforms, and neuromorphic/processing-in-memory prototypes, along with FL and edge AI frameworks (TensorFlow Federated, Flower, FedML, FATE, OpenFL, TinyFL, and related tools) that support simulation, prototyping, and deployment across device–edge–cloud continua.
- **Highlight applications and case studies.** We organize representative applications in healthcare and wearables, intelligent transportation, industrial IoT and smart manufacturing, and smart cities and infrastructure, emphasizing the interplay between data modalities, latency and residency constraints, and chosen architectural patterns.
- **Articulate future directions.** We outline research frontiers in federated continual and lifelong learning, the integration of foundation models with on-device adaptation, green and trustworthy FL across tiers, and the convergence of FL with TinyML, neuromorphic computing, and emerging hardware paradigms.

Our goal is not only to catalog existing work but also to provide a set of *conceptual dials* for system designers: when to prefer centralized versus hierarchical coordination; when to keep learning edge-only; when to adopt decentralized or blockchain-enabled mechanisms; and when split/hybrid architectures are necessary to marry edge devices with cloud-scale backbones. Throughout, we emphasize design trade-offs, practical deployment considerations, and open questions.

0.2. Organization

The remainder of the paper is organized as follows. Section 1 introduces cloud–edge computing, edge AI, and federated learning fundamentals, including basic optimization formulations and threat models. Section 2 presents the architectural design space for federated edge AI, detailing centralized, hierarchical and multi-tier, edge-only, decentralized, and split/hybrid architectures, along with comparative tables. Section 3 discusses core statistical, systems, communication, energy, and security challenges, together with key enabling techniques such as personalization, client and cluster selection, compression, and privacy-enhancing mechanisms. Section 4 reviews hardware and software enablers, from edge accelerators and TinyML platforms to FL frameworks and orchestration stacks. Section 5 surveys applications and case studies across major domains, mapping them to appropriate architectural patterns. Section 6 outlines future directions, including federated continual learning, foundation-model adaptation at the edge, green and trustworthy FL, and integration with TinyML and neuromorphic hardware. Finally, Section 7 summarizes our findings and reflects on the path toward truly federated, edge-native AI beyond the cloud.

1. Background

This section provides foundational context for the rest of the paper. We first trace the evolution from purely cloud-centric computing to a continuum of cloud, fog, and edge resources. We then introduce edge AI and TinyML as key enablers of on-device and near-device intelligence. Next, we review the core optimization formulation and algorithms behind federated learning (FL). Finally, we outline a taxonomy of FL designs specifically from an edge AI perspective, which we later refine into concrete architectures and system designs.

1.1. From Cloud to Edge Computing

Cloud computing popularized the idea of renting elastic compute, storage, and networking resources in large, centralized data centers. This paradigm has been a key driver of modern AI: large-scale supervised, self-supervised, and foundation models are typically trained in the cloud, where thousands of accelerators and high-speed interconnects can be orchestrated as a single logical cluster.

However, many emerging applications are *edge-centric* rather than *cloud-centric*. They are tightly coupled to the physical world, driven by sensors and actuators deployed in homes, factories, hospitals, vehicles, and urban infrastructure. In such settings, sending all raw data to distant data centers is increasingly problematic:

- **Latency.** Interactive or safety-critical loops (e.g., collision avoidance, robotic control, real-time AR/VR overlays) cannot tolerate hundreds of milliseconds of round-trip latency to the cloud and back.
- **Bandwidth.** High-resolution video, LiDAR, and dense telemetry streams can saturate backhaul links if naively uploaded; this is especially acute in mobile and wireless environments.
- **Data residency and privacy.** Regulations and internal governance often require that sensitive data (medical records, industrial process logs, location traces) stay within certain legal or administrative domains.
- **Resilience.** Dependence on a small number of centralized data centers introduces single points of failure and can make systems fragile to outages, overloads, or connectivity disruptions.

To address these limitations, the computing landscape has evolved into a *continuum*:

- **Cloud.** Large, centralized data centers with very high compute density and storage capacity, accessible over wide-area networks (WANs).
- **Fog / regional edge.** Intermediate aggregation and compute points—metro data centers, operator points-of-presence, regional edge sites—that sit closer (in network terms) to end devices than core cloud.
- **Edge and device.** On-premise micro data centers, base stations, gateways, and the devices themselves (phones, wearables, embedded controllers, microcontrollers).

These paradigms differ along multiple dimensions: latency, backhaul usage, where data is stored and processed, and how systems are managed. Table 1 summarizes typical characteristics. The numbers are indicative rather than absolute, but they convey the intuition that as we move from cloud toward devices, latency decreases, data becomes more local, and management becomes more decentralized, while raw compute density generally decreases.

Table 1. Comparison of cloud, fog, and edge computing paradigms. Values are indicative and assume typical Internet-scale deployments; actual numbers vary by deployment and network conditions.

Aspect	Cloud	Fog	Edge/Device
Latency	100–1000 ms	10–100 ms	1–10 ms
Backhaul usage	High	Medium	Low
Data locality	Central	Regional	Local
Privacy	Low	Medium	High
Compute density	Very high	High	Low–medium
Management	Centralized	Hierarchical	Highly distributed

In this paper, we are particularly interested in how *learning* and *inference* move along this continuum. Edge AI takes advantage of lower latency and stronger data locality at fog and edge tiers; federated learning introduces collaboration without centralized raw data collection. Together they define a design space in which parts of the model, training loop, and data flow can be placed at different points in the cloud–fog–edge spectrum.

1.2. Edge AI and TinyML

Edge AI refers broadly to the execution of AI workloads—most often deep neural networks, but also classical ML and rule-based components—on edge servers and end devices instead of, or in addition to, remote clouds. The key motivation is to align computation with where data is produced and where decisions are needed.

Typical goals of edge AI include:

- **Reducing latency.** By running inference (and sometimes training) directly on devices or nearby edge servers, systems avoid wide-area round-trips and can respond in a few milliseconds instead of hundreds.
- **Lowering bandwidth and backhaul load.** Instead of streaming raw video or telemetry, devices can extract features, predictions, or compressed summaries locally and only transmit what is necessary (e.g., alerts, aggregated statistics, or model updates).
- **Improving privacy and availability.** Keeping raw data local to devices or on-premise sites reduces exposure and enables operation even when WAN connectivity is intermittent or unavailable.

Running AI at the edge, however, is challenging because many devices are resource-constrained: limited CPU/GPU/NPU capability, small memory footprints, tight power budgets, and sometimes duty-cycled connectivity. To make edge AI feasible, several complementary techniques are widely used:

- **Model compression.** Techniques such as pruning (removing redundant weights or channels), quantization (representing parameters and activations with lower precision), knowledge distillation (training smaller student models from larger teachers), and low-rank factorization reduce model size and compute without catastrophic loss in accuracy.
- **Architecture and hardware-aware design.** Neural architecture search (NAS) and manual design of compact architectures (e.g., MobileNet-style, shift or depthwise separable convolutions, lightweight attention blocks) produce models that fit within the memory, compute, and latency envelopes of edge hardware.
- **TinyML.** TinyML targets *microcontroller-class* devices with kilobytes of RAM and milliwatt-level power budgets. TinyML models support always-on sensing (e.g., keyword spotting, anomaly detection, vibration analysis) and frequently rely on aggressive quantization, integer-only kernels, and highly optimized runtimes.

From the perspective of federated edge AI, these techniques do more than just enable inference: they shape *what* can be trained and adapted locally. For example, TinyML devices may only be able to fine-tune a small subset of parameters or a lightweight head on top of frozen features. Federated learning algorithms must therefore account for heterogeneous model capacities, mixed-precision updates, and partial participation of ultra-constrained clients.

1.3. Federated Learning Fundamentals

Federated learning (FL) provides a framework for training models collaboratively across multiple clients without aggregating their raw data in a central location. Instead of uploading data, clients compute local updates and share model parameters, gradients, or compressed sketches with one or more aggregators.

Consider K clients, where client k holds a local dataset $\mathcal{D}_k$ and defines a local empirical loss $F_k(w)$ over model parameters w (for example, an average of per-example losses across $\mathcal{D}_k$). A standard FL objective is:

$$\min_{w} F(w) := \sum_{k=1}^{K} p_k F_k(w), \quad p_k \geq 0, \sum_{k=1}^{K} p_k = 1, \tag{1}$$

where p_k often reflects the relative size of $\mathcal{D}_k$ (e.g., $p_k = \frac{|\mathcal{D}_k|}{\sum_j |\mathcal{D}_j|}$) but may also encode other notions of importance or trust.

The canonical *federated averaging* (FedAvg) algorithm approximates gradient descent on $F(w)$ using repeated rounds of local training and global aggregation:

1. **Server broadcast.** At round t, the server samples a subset $\mathcal{S}_t$ of clients from $\{1, \ldots, K\}$ and broadcasts the current global model parameters w_t to each $k \in \mathcal{S}_t$.
2. **Local training.** Each selected client $k \in \mathcal{S}_t$ initializes its local model with w_t and performs several epochs of (stochastic) gradient descent on its local data $\mathcal{D}_k$, obtaining an updated local model $w_{t+1}^{(k)}$.
3. **Aggregation.** The server collects the updated models (or equivalent updates) from participating clients and aggregates them, typically via a weighted average:

$$w_{t+1} = \sum_{k \in \mathcal{S}_t} \frac{n_k}{\sum_{j \in \mathcal{S}_t} n_j} w_{t+1}^{(k)}, \tag{2}$$

where $n_k = |\mathcal{D}_k|$. The resulting w_{t+1} becomes the new global model and the process repeats.

This simple protocol underlies many practical deployments and serves as a building block for more advanced variants (personalized FL, hierarchical FL, secure or robust FL, etc.). However, even this basic setup reveals several fundamental challenges, especially acute in edge environments:

- **Non-IID and unbalanced data.** Clients often observe different data distributions (e.g., different users, locations, devices, or environments), and some clients may have orders of magnitude more data than others. This statistical heterogeneity can slow convergence and result in global models that are suboptimal for many clients.
- **System heterogeneity.** Edge devices vary widely in compute, memory, energy availability, and connectivity. Some may be offline, intermittently connected, or unable to complete local training in time. Algorithms must accommodate partial participation, stragglers, and varying capabilities.
- **Communication bottlenecks.** Models can be large, and frequent transmission of full parameters or gradients may be prohibitively expensive over wireless links or congested backhaul. This motivates compression, sparsification, structured updates, and reduced communication frequency.
- **Privacy and security.** Although raw data never leaves devices, model updates can leak information (e.g., via gradient inversion or membership inference attacks). Additionally, some clients or even servers may be adversarial, attempting to poison the model or infer sensitive attributes.

Addressing these challenges has led to a rich ecosystem of FL algorithms: variants that robustify aggregation, personalize models to individual clients or clusters, incorporate differential privacy or secure aggregation, and integrate with hierarchical or decentralized architectures. In the context of federated edge AI, these algorithmic developments interact closely with network topology and resource constraints.

1.4. Taxonomy of FL for Edge AI

From an edge AI point of view, it is useful to categorize FL systems along several orthogonal axes. These axes will recur throughout our discussion of architectures (Section 2) and challenges (Section 3).

Topology. How do clients and aggregators communicate?

- *Centralized:* a single logical server (often in the cloud) orchestrates training in a star topology.

- *Hierarchical / multi-tier:* devices first aggregate at edge nodes, which in turn synchronize with regional or core clouds, forming a tree or DAG.
- *Decentralized / peer-to-peer:* there is no central server; nodes exchange updates with neighbors in a mesh, overlay, or gossip network.

Objective. What is the learning goal across clients?

- *Global model:* all clients share a single model, as in classical FL formulations.
- *Personalized FL:* each client adapts a shared model or learns client-specific parameters, balancing global knowledge with local specialization.
- *Clustered FL:* clients are grouped into clusters (e.g., by data or topology), each with its own model; clusters may share information at higher tiers.
- *Multi-task FL:* each client or cluster is treated as a related but distinct task, and the system learns to share representations or priors rather than a single model.

Resource-awareness. How does the system explicitly account for communication, computation, and energy constraints?

- *Communication-efficient:* leverages compression, sparsification, update subsampling, reduced participation, or over-the-air aggregation to reduce traffic.
- *Computation-aware:* adapts local training effort, model size, or split placement to device capabilities and current load.
- *Energy-aware / green FL:* explicitly optimizes or constrains energy usage and carbon impact across devices, edges, and clouds.

Trust model. What are the assumed adversaries and defenses?

- *Honest-but-curious server:* the server executes the protocol but may attempt to infer information from updates; secure aggregation and differential privacy are common defenses.
- *Malicious or Byzantine clients:* some clients may send arbitrary or adversarial updates; robust aggregation and anomaly detection are needed.
- *Byzantine servers or aggregators:* in decentralized or multi-tier systems, intermediary nodes themselves may be compromised, necessitating consensus, replication, or blockchain-style mechanisms.
- *Formal privacy guarantees:* some systems require (ϵ, δ)-differential privacy at the user or client level, or cryptographic end-to-end guarantees using secure multi-party computation or homomorphic encryption.

This taxonomy provides a conceptual scaffold for the rest of the paper. In Section 2, we instantiate these axes as concrete architectural patterns (centralized, hierarchical, edge-only, decentralized, and split/hybrid). In Sections 3 and 4, we show how algorithmic techniques, hardware platforms, and software frameworks support different points in this design space for federated edge AI.

2. Architectures for Federated Edge AI

This section details how federated learning (FL) is integrated into edge computing architectures, starting from the basic centralized star topology and progressing to hierarchical, edge-only, decentralized, and split/hybrid designs. For each architecture, we discuss the intuition, system model, mathematical or performance characteristics, deployment scenarios, and typical failure modes, drawing on recent FL–edge surveys and systems work [1–6].

2.1. Centralized FL with Edge Devices

2.1.1. Basic star topology and workflow

In the canonical *centralized* FL architecture, the entire training process is organized around a single logical coordinator—the FL server—which is most often deployed in a cloud or core data-center environment [7,8]. All edge clients (smartphones, sensors, gateways, vehicles, etc.) interact exclusively with this server: they never communicate directly with one another. This induces a star-shaped communication graph with the server at the center and clients at the leaves. Conceptually, the global model "lives" at the server and is periodically pushed out to participating clients; in return, the server receives locally computed updates that reflect each client's private data. The star topology provides a clean separation of concerns: the server is responsible for orchestration and aggregation, while clients handle local computation and data management. It also aligns well with the control model of existing mobile and cloud infrastructure [4,9].

To understand how this architecture operates over time, it is useful to view training as a sequence of *rounds*, indexed by $t = 0, 1, 2, \ldots$. At the beginning of each round, the server holds a global parameter vector w_t, representing the current state of the model (for example, the weights of a neural network). The core idea of centralized FL is that w_t is repeatedly sent out to a subset of clients, refined locally using their private data, and then re-aggregated to form w_{t+1}. This repeated broadcast–train–aggregate cycle is the fundamental workflow that underlies most cross-device FL deployments in practice, including mobile keyboard prediction, next-word prediction, and on-device recommendation [9,10].

Model broadcast and client sampling.

The first step in each round is **model broadcast**. The server chooses a subset $\mathcal{S}_t \subseteq \{1, \ldots, K\}$ of clients from the full population of size K and sends them the current model w_t. In large-scale cross-device settings, K can be in the millions, but only a small fraction of clients are active and eligible at any given time (for example, devices that are idle, charging, and on unmetered Wi-Fi) [9]. Sampling a subset $\mathcal{S}_t$ is therefore essential for scalability and for respecting device constraints; typical sampling schemes include uniform random sampling among eligible clients, importance sampling based on data size or device capacity, or stratified sampling to improve fairness and statistical efficiency [8,11]. The broadcast itself can be implemented using standard content-distribution mechanisms (e.g., CDN-style fan-out or push notifications), and the model may be compressed (via quantization or sparsification) to reduce communication cost [12,13].

Local training and objective.

The second step is **local training** on each selected client $k \in \mathcal{S}_t$. Each client maintains its own private dataset $\mathcal{D}_k$, which may be small, highly non-IID, and evolving over time. Centralized FL is often framed as minimizing a global objective

$$F(w) = \sum_{k=1}^{K} p_k F_k(w), \qquad F_k(w) = \frac{1}{|\mathcal{D}_k|} \sum_{(x,y)\in\mathcal{D}_k} \ell(f_w(x), y), \tag{3}$$

where p_k are non-negative weights summing to one, f_w is the model, and ℓ is a loss function [7,8]. In round t, the server sends w_t to client k, which then performs E local epochs of stochastic gradient descent (SGD) or a variant on F_k:

$$w_{t+1}^{(k)} \approx w_t - \eta \sum_{e=1}^{E} g_{k,e}(w), \tag{4}$$

where $g_{k,e}(w)$ denotes a stochastic gradient computed on mini-batches of $\mathcal{D}_k$ and η is a learning rate. The number of local epochs E is a critical hyperparameter: larger E reduces how often the device must communicate but increases the risk of client drift, especially under non-IID data [7,13]. In practical systems, local training may be further constrained by wall-clock budgets (e.g., a maximum number of seconds of computation or battery thresholds per round) and may integrate device-specific optimizations such as mixed-precision or hardware acceleration [2,9].

Update upload, aggregation, and model semantics.

The third step is **update upload and aggregation**. After local training, client k computes an update to send back to the server. This update can take several forms:

- full model parameters $w_{t+1}^{(k)}$;
- a model *delta* $\Delta w_t^{(k)} = w_{t+1}^{(k)} - w_t$;
- or a compressed and possibly encrypted sketch of the gradient or delta [12,14].

To preserve privacy, updates are often transmitted through a secure aggregation protocol that ensures the server only sees the sum of updates, not individual contributions [9,15]. Once the server has collected a sufficient number of updates, it aggregates them to form a new global model. In the simplest FedAvg scheme, this is a data-weighted average:

$$w_{t+1} = \sum_{k \in \mathcal{S}_t} \frac{n_k}{\sum_{j \in \mathcal{S}_t} n_j} w_{t+1}^{(k)}, \qquad n_k = |\mathcal{D}_k|. \tag{5}$$

This can also be written in terms of deltas, $w_{t+1} = w_t + \sum_{k \in \mathcal{S}_t} \alpha_k \Delta w_t^{(k)}$, where α_k are aggregation weights. Various algorithms modify this basic step by adding proximal terms (FedProx [13]), momentum, variance reduction, or adaptive learning rates to improve convergence under heterogeneity and partial participation [8,16].

Temporal behavior and system orchestration.

Viewed over many rounds, the basic star topology yields an iterative optimization process that intertwines statistical and systems dynamics. On the statistical side, the convergence rate of w_t toward a stationary point of $F(w)$ depends on the number of rounds, local epochs E, client sampling scheme, and the extent of data heterogeneity [8]. On the systems side, each round must respect device availability, bandwidth, energy budgets, and privacy constraints. Production FL systems therefore implement complex orchestration logic around the simple three-step workflow: scheduling eligible devices, enforcing per-round deadlines, handling stragglers or dropouts, buffering updates, and applying backpressure to avoid overloading the server [4,9]. Despite these complexities, at the logical level the star topology remains unchanged: the server is the sole locus of coordination and aggregation, and edge clients are stateless participants that come and go round by round.

Why the star topology remains a baseline.

Even as more sophisticated hierarchical, decentralized, and split architectures emerge, the centralized star topology remains the conceptual and practical baseline for FL research and deployment [1,8]. It is simple to reason about, easy to simulate, and well-supported by existing cloud-centric ecosystems. Many theoretical analyses of FL convergence, robustness, and privacy are first developed in this setting before being generalized to richer topologies [13,16]. For federated edge AI, understanding this basic workflow is therefore essential: more advanced architectures can be seen as systematically relaxing its assumptions about a single coordinator, uniform latency, and unrestricted cloud connectivity.

2.1.2. Cross-device vs. cross-silo regimes at the edge

Centralized FL is not a single homogeneous setting; rather, it spans two qualitatively different regimes that arise naturally in edge computing: *cross-device* and *cross-silo* FL [1,8]. Both share the same high-level star topology described above, but they differ sharply in terms of client scale, data volume per client, reliability, orchestration mechanisms, and dominant bottlenecks. Understanding this distinction is crucial when designing architectures, algorithms, and evaluation protocols for federated edge AI [2,3].

Cross-device FL: many tiny, unreliable clients.

In **cross-device FL**, the population consists of a very large number of lightweight edge devices—typically smartphones, tablets, wearables, home IoT nodes, or in-vehicle infotainment systems. The hallmark of this regime is sheer scale (K in the millions) and extreme heterogeneity: devices differ in hardware capabilities, battery levels, network conditions, and user behavior [8,9]. Each device usually contributes a relatively small local dataset (e.g., a user's typing history, app usage patterns, or sensor logs), and the data distribution is highly non-IID across users.

Because only a small fraction of devices are available and eligible at any moment, participation per round is intentionally kept small, with $|\mathcal{S}_t| \ll K$. Eligibility might require that the device be idle, plugged in, on unmetered Wi-Fi, and opted-in to analytics [9]. As a result, the set of participating clients changes drastically from round to round; most devices only participate rarely. The server must therefore be designed to cope with high churn, stragglers, and unpredictable dropouts: it cannot wait for every invited device to respond, and must implement deadlines, minimum-participation thresholds, and robust handling of late or missing updates [9,11].

From a statistical perspective, cross-device FL means that each client's dataset $\mathcal{D}_k$ can be tiny and idiosyncratic, which exacerbates non-IID data issues and client drift. Algorithms must be tolerant to partial and biased participation; they often incorporate client sampling schemes, adaptive local epochs, and personalization components to prevent the global model from being skewed toward over-represented or more reliable clients [2,8]. From a systems perspective, bandwidth and energy constraints on devices, as well as privacy policies enforced by mobile platforms, drive the use of model compression, secure aggregation, and strict local resource budgets [3,9].

In edge settings, cross-device FL naturally occurs between devices and a remote cloud backend: the cloud server is the hub, and end-user devices are the leaves. Even when an operator introduces intermediate MEC nodes or edge gateways, the logical regime remains cross-device so long as the participants are large numbers of end devices with limited per-client data and reliability [1,3].

Cross-silo FL: few powerful, stable clients.

In **cross-silo FL**, the population consists of a much smaller number of clients—typically tens to a few hundreds—that are institution-scale entities: edge data centers, hospitals, factories, telecom base-station clusters, or regional control centers [5,8]. Each silo has substantial compute and storage resources (e.g., servers or GPU clusters) and holds a large local dataset that may represent many users or devices within that silo (e.g., all patients in a hospital, all sensors in a factory, or all subscribers in a region). Connectivity between silos and the central coordinator (which may itself be an edge or cloud node) is far more stable and predictable than in cross-device FL.

Because the number of silos is modest and they are almost always online, *full participation* is common: many cross-silo FL systems assume $|\mathcal{S}_t| \approx K$ in every round [5,8]. This simplifies parts of the orchestration logic: the server can often plan for all silos to participate, coordinate synchronized training schedules, and rely on more deterministic communication patterns. However, cross-silo FL introduces its own challenges. Data distributions across silos can still be strongly non-IID (e.g., hospitals may serve different demographics, factories may use different production lines), and each silo may have strong privacy, regulatory, or business constraints governing what information can be shared [17,18].

In edge computing, cross-silo FL arises naturally when multiple edge sites cooperate while preserving local autonomy. For example, several factory floors, wind farms, or railway segments may each deploy an on-site edge server that collects data from local devices and runs local training. These edge servers then participate as silos in a higher-level FL protocol with a central or regional coordinator [5,19]. Similarly, a group of hospitals or clinics can participate in cross-silo FL to train diagnostic models without sharing raw patient data, with each hospital acting as a silo hosting substantial data and compute [17,20]. Compared to cross-device FL, the algorithms in cross-silo FL can typically afford heavier models, larger batch sizes, and more complex privacy mechanisms (e.g., secure multiparty computation between a handful of silos) because each participant has more resources [5,8].

Axis of contrast: scale, reliability, and governance.

Although both regimes use the same mathematical FL objective and three-step broadcast–train–aggregate workflow, they occupy opposite ends of several key axes:

- **Scale and granularity.** Cross-device FL involves many low-resource clients with small datasets; cross-silo FL involves few high-resource clients with large datasets.
- **Reliability and participation.** Cross-device FL expects intermittent, opportunistic participation and must tolerate high churn; cross-silo FL expects almost always-on silos and often assumes full or near-full participation.
- **Governance and trust.** In cross-device FL, clients are typically end-user devices governed by a single platform provider, and privacy is enforced via platform policies, secure aggregation, and differential privacy. In cross-silo FL, each silo may represent a separate organization with its own governance, legal constraints, and trust assumptions, leading to more complex threat models and negotiation of protocols [8,17].
- **Systems bottlenecks.** Cross-device FL is dominated by device constraints (battery, radio, compute) and uplink bandwidth; cross-silo FL is dominated by inter-silo network bandwidth, regulatory and audit requirements, and long-term maintainability of the federated system [2,3].

Implications for federated edge AI.

In practical edge deployments, these two regimes often coexist and may even be nested. A telecom operator might run cross-device FL between handsets and local base stations, while the base stations themselves participate as silos in a cross-silo FL protocol with a regional control center [1,21]. Similarly, an industrial conglomerate might use cross-device FL within each plant (devices to plant-level edge server) and cross-silo FL across plants (plant-level edge servers to a corporate analytics hub) [18,19]. Algorithm design, system architecture, and evaluation methodology must therefore be carefully tailored to the dominant regime—or combination of regimes—present in a given application. Treating cross-device and cross-silo FL as interchangeable can lead to misleading conclusions, for example by evaluating algorithms in small, stable cross-silo simulations while targeting deployment on massive, unreliable cross-device populations. A central theme in federated edge AI is thus to explicitly recognize which regime one is in, design accordingly, and leverage hierarchical architectures when both appear simultaneously [2,3].

2.1.3. Performance and limitations in edge contexts

The centralized star topology has appealing simplicity but shows several weaknesses when mapped onto real edge networks [1,6]:

- **WAN bottleneck and latency.** Every participating device uploads and downloads model parameters or gradients across potentially long and noisy wide-area links. If the model has size $\|w\|$ and the per-round participation fraction is q, the total WAN traffic per round scales as $O(qK\|w\|)$ in both uplink and downlink [4].
- **Single point of failure and trust.** The central server is both a performance bottleneck and a trust anchor. Outages, denial-of-service attacks, or compromise can disrupt the entire system [22,23].
- **Topology-unaware coordination.** The star model ignores the rich physical topology of edge networks—base stations, gateways, regional micro data centers, and core clouds—which could be exploited to reduce latency and balance load [2,6].
- **Regulatory friction.** For some sectors (healthcare, manufacturing, critical infrastructure), moving updates or even gradients to a public cloud is undesirable or restricted, even when raw data never leaves the edge [5,17].

These limitations motivate architectures that introduce intermediate aggregators or completely remove the central server.

2.2. Hierarchical and Multi-Tier FL

2.2.1. Intuition: matching the network hierarchy

Real-world edge systems are almost never flat. A typical deployment contains at least three logical layers: (i) a large number of end devices (phones, sensors, vehicles, robots), (ii) intermediate aggregation points such as access points, base stations, roadside units, or on-premise micro data centers, and (iii) regional or core cloud data centers. Traffic from devices flows upward through this hierarchy, and control or content flows downward. *Hierarchical FL* is built on the observation that the FL architecture should mirror this physical hierarchy instead of forcing everything into a single device–cloud star topology [1,24,25].

The core intuition is locality. Devices that are physically and logically close to one another (e.g., users within one cell, machines on one factory floor, sensors within one substation) often share similar data distributions, experience similar network conditions, and are connected by high-bandwidth, low-latency links. Rather than having each of these devices independently send updates all the way to a distant cloud, hierarchical FL encourages them to first "agree" on an intermediate model at a nearby edge node. This edge node acts as a local FL server, fusing updates from its associated devices into an *edge model*. Only these aggregated models or summaries then flow upward toward higher levels (regional edge or core cloud). This design significantly reduces wide-area network (WAN) traffic, improves training latency for local adaptation, and respects data locality constraints [1,2].

From a learning-theoretic perspective, hierarchical FL can be seen as introducing multiple nested objectives: one at the device level (fitting each device's local data), one at the edge-cluster level (fitting the aggregated distribution of devices in the cluster), and one at the global level (fitting the mixture of edge clusters). Each tier runs a variant of federated averaging (or another FL algorithm) on its own timescale. This multi-timescale structure allows edge models to adapt quickly to local changes (e.g., a traffic pattern in one city or a machine configuration in one plant), while global models evolve more slowly to capture broad trends [19,24,25]. In other words, hierarchical FL aligns the structure of optimization with the structure of the network.

2.2.2. Two-tier device–edge–cloud architectures

The simplest realization of hierarchical FL is a two-tier architecture with a device–edge tier and an edge–cloud tier. This is often the first stepping stone away from purely centralized FL.

Tier 1: device–edge (local clusters).

At the bottom tier, devices are grouped into clusters, each served by an edge node. A cluster could correspond to all devices under a base station, all sensors connected to a gateway, or all machines on one factory floor. For each cluster c, the edge node acts as a local FL server: it selects a subset of devices $\mathcal{K}_c$ (analogous to $\mathcal{S}_t$ in centralized FL), broadcasts the current *edge model* $w_t^{(c)}$ to them, orchestrates local training on their private datasets, and then aggregates the resulting updates. After one local round, the edge node computes an updated edge model

$$w_{t+1}^{(c)} = \sum_{k \in \mathcal{K}_c} \alpha_k \, w_{t+1}^{(k)}, \tag{6}$$

where $w_{t+1}^{(k)}$ is the model obtained by device k and α_k encodes, for example, the relative dataset size or trust level of device k [24,25]. This step is conceptually identical to centralized FedAvg, but it is scoped to a single cluster and runs over high-speed local links (LAN, Wi-Fi, private 5G).

In practice, the edge node can also handle various device-level concerns: scheduling which devices participate (e.g., those currently connected and idle), enforcing local privacy or resource constraints, and performing compression or secure aggregation tailored to the local environment. Because the edge node is much closer to devices than a remote cloud, it can respond quickly to device churn and can operate effectively even under intermittent connectivity to the higher tiers [1,2].

Tier 2: edge–cloud (global coordination).

At the upper tier, edge nodes themselves become clients in a second FL process coordinated by a cloud or regional data center. Instead of raw device models, the cloud receives only edge-level aggregates (e.g., $w_t^{(c)}$ or deltas derived from them). Periodically—say, every R local device–edge rounds—the cloud collects these edge models from a subset or all edge nodes and aggregates them into a global model w_t:

$$w_{t+1} = \sum_c \beta_c \, w_{t+1}^{(c)}, \tag{7}$$

where β_c reflect the number of devices, the importance of the cluster, or the trust and reliability of the edge node [19,26]. The global model is then sent back to edge nodes, where it serves as a new starting point or prior for the next block of local rounds.

The parameter R (how many local device–edge rounds occur per edge–cloud synchronization) becomes a key design knob. When $R = 1$, the system behaves more like a flattened two-layer FL system with frequent synchronization; as R increases, edge clusters specialize more strongly to their local data before exchanging information with the cloud. Large R reduces WAN communication but increases the risk that edge models drift apart, especially if cluster data distributions differ substantially [24,25]. System designers must therefore trade off global coherence against communication cost and local responsiveness.

Nested FedAvg and multi-timescale optimization.

Putting the two tiers together yields a *nested FedAvg* structure: devices average into edge models, and edge models average into a global model. If we denote device-level objectives by F_k, cluster-level objectives by $F^{(c)}(w) = \sum_{k \in \mathcal{K}_c} p_k F_k(w)$, and the global objective by $F(w) = \sum_c q_c F^{(c)}(w)$, then hierarchical FL approximates gradient descent on F with inner loops at the cluster level and outer loops at the global level [24]. The inner loops let each cluster adapt to its own mixture of devices, while the outer loop ensures that information still flows globally across clusters. This structure is particularly natural in IIoT and smart factory settings, where each plant or production cell may require a slightly different model, but overall corporate knowledge should be shared [19,26].

2.2.3. Multi-tier and topology-aware FL

While two-tier device–edge–cloud designs already offer substantial benefits, real networks can contain more than two levels, and their connectivity is often irregular. *Multi-tier* and *topology-aware* FL generalize hierarchical FL to arbitrary numbers of tiers and explicitly leverage the underlying network graph.

From two tiers to multiple tiers.

In a multi-tier architecture, there may be several layers of aggregation: devices connect to access points; access points aggregate into regional edge data centers; regional edges aggregate into national or global clouds, and so on. Each layer can run its own FL process, with its own time granularity and objectives. For example, an operator might run:

- fast device-level rounds every few seconds or minutes within a cell;
- regional aggregation every few minutes or hours; and
- global aggregation once per day or per week.

This allows fast-paced local adaptation (e.g., to radio conditions or local demand) while keeping global models stable and minimizing expensive long-haul communication [5,6]. Mathematically, this amounts to building a hierarchical optimization procedure over a tree (or more general graph) of aggregators.

Clustered FL: grouping by similarity or locality.

A common pattern in topology-aware FL is **clustered FL**. Clients are partitioned into clusters according to criteria such as physical proximity (same cell or region), latency (low round-trip times), radio conditions, or task and data similarity (similar label distributions or feature statistics) [19,27]. Each cluster elects or is assigned a cluster head (often an edge node), which runs FL within the cluster. Cluster heads then participate in a higher-level aggregation, which can itself be centralized (through a global server) or decentralized (through peer-to-peer links between cluster heads).

The advantage of clustering is that it aligns the FL process with both the network topology and the statistical structure of the data. Clients within a cluster often share more similar distributions, so local models converge faster and require fewer global updates. Moreover, local communication within a cluster uses cheaper links (e.g., LAN or regional networks), while only compact cluster summaries traverse expensive WAN links [6]. However, clustering introduces new design questions: how to form clusters, how often to re-cluster as conditions change, and how to ensure fairness across clusters with different sizes or resources [27].

Tree-based aggregation: leveraging routing structure.

Another pattern is **tree-based aggregation**, where the aggregation structure follows a spanning tree of the network. Routers and edge nodes form internal nodes of the tree, and devices are leaves. Updates are propagated up the tree toward the root (e.g., a regional or global coordinator), aggregated along the way, and the resulting model is propagated back down [6,25]. Because routing infrastructure is often already tree-shaped or hierarchical (e.g., access, aggregation, core layers), this approach can reuse existing forwarding and multicast mechanisms.

Tree-based aggregation helps reduce congestion on shared links, as each link carries only aggregated updates from its subtree instead of separate updates from every device. It also supports incremental or partial updates: if a subtree has not changed significantly since the last round, its updates may be skipped or heavily compressed. However, trees introduce sensitivity to failures (if an internal node fails, its entire subtree is affected) and may need to be recomputed or augmented with redundancy for robustness.

Hybrid centralized–decentralized schemes.

Topology-aware FL also includes **hybrid centralized–decentralized** schemes. Within a cluster, clients may use decentralized or gossip-based learning, exchanging models directly with neighbors; cluster heads then summarize the resulting local consensus models and send them to a central coordinator [22,28]. This hybrid approach can reduce the burden on the central server, improve robustness within clusters, and better exploit peer-to-peer communication capabilities (e.g., V2V links in vehicular networks). At the same time, a central or tiered set of aggregators still exists to ensure global coherence and enforce system-wide policies.

These multi-tier and topology-aware designs are particularly attractive in 5G/6G mobile edge computing (MEC), vehicular edge networks, and large-scale IIoT deployments, where the networking substrate is already hierarchical and where performance, fairness, and robustness must be balanced across many locations and stakeholders [21,29].

2.2.4. System-level trade-offs and design knobs

Moving from flat to hierarchical FL introduces several new “dials” that a system designer can tune. These design knobs operate at the intersection of algorithms, networking, and resource management, and they determine how hierarchical FL behaves in practice [1,2,6].

Aggregation cadence: how often tiers synchronize.

Aggregation cadence controls how frequently each tier communicates with the tier above it. For example, how many device–edge rounds should occur between two edge–cloud synchronizations? Frequent synchronization (small R) keeps global models tightly coupled across clusters and can improve convergence speed on globally shared patterns. However, it increases WAN usage and can expose the system more to transient noise and non-IID fluctuations in local data. Infrequent synchronization (large R) reduces WAN costs and allows clusters to specialize, but may slow down convergence to a global optimum and can even lead to divergence if local drifts are not controlled [24,25]. Choosing appropriate cadences may depend on application requirements (e.g., how quickly global behavior must adapt), network pricing and capacity, and the degree of heterogeneity across clusters.

Cluster formation: who should train together.

Cluster formation answers the question "which clients should share an edge aggregator?" There are several possible criteria:

- *Network-centric*: group clients by latency, bandwidth, or radio conditions (e.g., same base station, same WLAN).
- *Data-centric*: group clients with similar data distributions (e.g., similar label distributions or embedding statistics).
- *Task-centric*: group clients that solve similar tasks or share similar models.

In practice, systems may mix these criteria, for example forming initial clusters based on network topology and then refining them using statistical measures of similarity [25,27]. Poor cluster formation can harm both performance and fairness: minority clients may be overshadowed in large, heterogeneous clusters; small clusters may overfit and contribute little to the global model. Dynamic or adaptive clustering mechanisms, which periodically reassess and adjust cluster membership, are an active area of research.

Role placement: where to put aggregators.

Role placement concerns which physical nodes in the network should host aggregation logic. An edge aggregator needs sufficient compute, storage, and network capacity to handle its cluster's load. Placing it too close to the cloud (e.g., only in regional data centers) reduces latency savings and locality benefits; placing it too deep at the very edge (e.g., on resource-constrained access points) may overload devices and complicate management. Some IIoT frameworks treat aggregator placement as an optimization problem: given a network graph, traffic demands, and latency/energy constraints, choose aggregator locations and client assignments that minimize cost and meet quality-of-service targets [19,30]. Role placement also interacts with fault tolerance: aggregators can be replicated or backed up to avoid single points of failure at intermediate tiers.

Staleness management: dealing with delayed and outdated models.

Staleness is intrinsic to multi-tier architectures. Even in centralized FL, the server may aggregate updates computed on slightly outdated models due to communication delays or stragglers. In hierarchical FL, staleness appears at multiple levels: edge nodes may continue training on a stale global model for many local rounds; devices may train on stale edge models if synchronization within the cluster is infrequent or asynchronous. Uncontrolled staleness can slow convergence or even destabilize training, especially in highly non-IID environments [11,22].

To mitigate this, systems employ:

- *Staleness-aware weighting*: updates that are older or computed on older model versions receive lower weight in aggregation.
- *Asynchronous protocols*: servers do not wait for all aggregators or devices; they integrate updates as they arrive, often with correction terms.
- *Bounded staleness*: enforcing maximum age limits on updates or requiring periodic resynchronization.

Designing effective staleness management is a key challenge in large, heterogeneous edge systems where communication latencies and compute speeds vary widely.

Communication complexity: how much traffic, and where.

Finally, hierarchical FL changes the *shape* of communication complexity. In a flat centralized FL system with participation fraction q and model size $\|w\|$, each round uses roughly $O(qK\|w\|)$ traffic in both directions over the WAN. In a two-tier hierarchical system, most model traffic associated with device updates remains local to device–edge links, and only edge-level aggregates traverse the WAN. A rough decomposition is

$$\text{Traffic per round} \approx O\big(qK\|w\|_{\text{device-edge}}\big) + O\big(S\|w\|_{\text{edge-cloud}}\big), \tag{8}$$

where S is the number of edge nodes and $\|w\|_{\text{device-edge}}$ and $\|w\|_{\text{edge-cloud}}$ denote (possibly different) model or update sizes used at each tier (for example, edge–cloud models may be more compressed or aggregated) [6]. Because $S \ll K$ in most deployments and because edge–cloud links often have higher capacity than device–cloud links, hierarchical FL can dramatically reduce the most expensive portion of communication, i.e., over the wide-area segments of the network.

In summary, hierarchical and multi-tier FL architectures are not merely cosmetic variations on centralized FL. They embody a deep alignment between learning and the underlying edge network hierarchy, introducing new algorithmic and systems-level decisions that, when tuned appropriately, can yield significant gains in scalability, efficiency, and robustness for federated edge AI.

2.3. Edge-Only and On-Premise FL

2.3.1. Motivation: data residency and offline operation

In many practical edge deployments, the challenge is not only to keep raw data local to devices, but also to keep *all* sensitive artifacts—including logs, model updates, and system telemetry—within a tightly controlled administrative or legal boundary. Healthcare providers must keep patient data within a hospital network or within national borders; industrial operators treat sensor streams and production recipes as core intellectual property; critical infrastructure providers (energy, rail, water, defense) must be resilient to cross-border outages and avoid introducing dependencies on external clouds that might be compromised or unavailable [5,17,18]. In these settings, even sending encrypted gradients or model parameters to a public cloud may be seen as a regulatory, contractual, or cyber-risk.

Edge-only federated learning addresses this by confining both data and the FL control plane to on-premise or private edge infrastructure. Instead of a cloud-based FL server, one or more edge servers located within the plant, hospital, or campus act as coordinators. All communication stays on-site or within a private backbone, creating a "federated learning island" whose boundaries match the operator's security perimeter. Conceptually, edge-only FL can be viewed as applying the FL paradigm within a private edge cloud, rather than across the public internet.

Another important motivation is *offline operation*. Many industrial environments are geographically remote, intermittently connected, or explicitly air-gapped from the public internet for safety. Rail systems, offshore platforms, mines, and substations cannot assume uninterrupted connectivity to a central cloud, yet they still benefit from collaborative learning across their local devices. Edge-only FL allows these sites to train and adapt models locally, and, when appropriate, synchronize models across sites using delayed or physically transported media (e.g., during maintenance windows) without ever exposing data or updates to third-party infrastructure [31,32].

Finally, edge-only deployments can simplify compliance and security auditing. When all FL components run on-premise, traditional enterprise governance mechanisms—access control, network segmentation, change management, security monitoring—can be applied to the FL stack, rather than relying on shared-responsibility models in public clouds [33,34]. This is particularly attractive in regulated sectors where auditability, certification (e.g., IEC 62443, ISO 27001), and clear accountability are paramount.

2.4. Architectural patterns

Within the broad umbrella of edge-only FL, several architectural patterns recur in practice. At a high level, they differ in scope (single-site vs. multi-site), degree of hierarchy, and how they integrate with existing IT/OT (information technology / operational technology) infrastructure.

Single-site edge-only FL.

In **single-site edge-only FL**, a single physical site—such as a factory, hospital, campus, or data center—contains a population of devices and one or more on-premise edge servers. These edge servers coordinate FL rounds among devices located within the site; all communication occurs over local networks (LAN, Wi-Fi, private 5G) with no dependence on public cloud resources [20,35]. Typical examples include:

- *Smart factories and process plants*: programmable logic controllers (PLCs), robots, and sensors connect to industrial gateways or edge servers; FL is used to train predictive maintenance, anomaly detection, or quality control models from distributed sensor data [18,36].
- *Hospitals and clinics*: bedside monitors, imaging devices, and departmental systems connect to a hospital edge cluster; FL trains models for risk prediction, triage, or workflow optimization while keeping patient-level data inside the hospital network [17,37,38].
- *Corporate campuses and R&D labs*: workstations or lab instruments participate in FL to train models for security analytics, code recommendation, or experiment analytics, with coordination handled by an on-premise AI cluster [39,40].

In these deployments, the edge server acts much like a classical FL server, but it is typically integrated with the operator's orchestration stack (Kubernetes, OpenShift, industrial container platforms) and identity/access management (IAM) systems. The FL coordinator is treated as just another internal service, benefiting from the existing monitoring, backup, and failover mechanisms already in place for mission-critical workloads.

Multi-site private federation.

In **multi-site private federation**, several sites belonging to the same organization (or to a tightly governed consortium) each run local edge-only FL, and a private backbone coordinates cross-site aggregation. The backbone may be a corporate WAN, a dedicated VPN mesh, or even an intermittent synchronization mechanism (e.g., scheduled batch jobs or physical media transport) [31,32]. Typical examples include:

- *Multi-plant industrial groups*: each plant trains local models from its own equipment and process data using on-site edge servers; periodically, plants share model updates or calibrated summaries over a private WAN to build a corporate-level model that captures fleet-wide patterns (e.g., failures across all turbines) [30,36,41].
- *Regional healthcare networks*: hospitals in the same region or network train local models using edge-only FL; periodically, a regional data center aggregates their models, respecting cross-hospital data-sharing agreements and national health regulations [17,37].
- *Transport and energy operators*: rail segments, electric substations, or wind farms each maintain local models for fault detection and control; headquarters or regional operations centers aggregate edge models into fleet-wide digital twins while keeping raw telemetry on-site [18,31].

Importantly, the system is still "edge-only" in the sense that the entire federation runs on private, operator-controlled infrastructure. Even if there is a higher-level coordinator, it resides in a corporate data center or private cloud, not a public hyperscale environment. Governance is typically unified: the same organization controls all sites, even if local IT/OT teams operate independently.

Integration with existing industrial and healthcare stacks.

A defining characteristic of edge-only FL is its need to integrate with existing protocols, middleware, and compliance regimes rather than starting from a greenfield environment. In industrial settings, this means interoperating with OPC UA, field buses, SCADA systems, and asset management platforms; in healthcare, it means integrating with HL7/FHIR interfaces, PACS and EHR systems, and hospital identity and auditing infrastructures [18,42]. As a result, edge-only FL frameworks often expose connectors to existing message buses or data historians, and they must obey real-time or near-real-time constraints imposed by control loops, clinical workflows, or safety interlocks.

Deployment and platform choices.

From a deployment perspective, edge-only FL can be implemented using:

- **Self-managed open-source stacks** (e.g., FedML, Flower, FATE, Substra, FEDn) installed on on-premise clusters and integrated with organizational DevOps pipelines [43–45];
- **Vendor platforms** that support on-premise or private-cloud deployments, sometimes with SaaS control planes but on-premise execution engines [39,46,47];
- **Custom in-house frameworks** built on general-purpose distributed training libraries (e.g., PyTorch Distributed, Ray, Kubernetes operators) with domain-specific integration.

Operators choose between these options based on their security posture, the maturity of their DevOps practices, and the need for long-term maintainability.

2.4.1. Advantages and constraints

Edge-only FL offers a distinct set of benefits and limitations compared to cloud-based or hybrid FL architectures. Understanding these trade-offs is essential when deciding whether to adopt an edge-only strategy, a hybrid edge–cloud design, or a fully cloud-centric approach.

Advantages.

Data and update locality. The most obvious advantage is strict locality of both data and model artifacts. Raw data never leaves the site, and even intermediate artifacts such as gradients, model checkpoints, and logs can be kept on-premise or on a private backbone. This simplifies compliance with data residency, sector-specific regulations, and internal policies, reducing the surface area for data exfiltration [18,33,34]. For organizations facing extraterritorial regulations or cross-border data transfer restrictions, this can be decisive.

Low latency and predictable QoS. Because communication is confined to LANs or private wireless (e.g., private 5G), end-to-end latency and jitter are typically far lower and more predictable than over the public internet. This enables tighter integration between FL and real-time or near-real-time control loops (e.g., adjusting setpoints in a process plant, triggering early warnings in clinical monitoring) and supports more frequent synchronization without incurring excessive WAN cost [35,44]. Edge-only FL can also exploit deterministic network technologies (TSN networks, time-synchronized industrial Ethernet) where required.

Resilience to internet outages and external failures. Edge-only systems are, by design, insulated from failures in public networks and cloud regions. As long as the local edge servers and networks remain operational, FL can continue to run, and models can continue to be updated and deployed, even during extended internet outages or geopolitical disruptions [31,39]. This is critical in domains where continued operation is a safety or business imperative.

Alignment with enterprise governance. Running FL on-premise allows organizations to leverage existing governance processes for change management, incident response, identity and access management, and auditing. Security teams can apply the same tooling (SIEM, IDS/IPS, vulnerability scanners) to the FL stack that they use for other internal systems, rather than relying on opaque cloud services [33,34]. This can reduce organizational friction and ease certification.

Constraints.

Limited aggregate data and compute. The flip side of locality is that an individual site may simply not have enough data or compute to train very large or highly generalizable models. While multi-site private federation alleviates this to some extent, the total scale available within one organization (or consortium) is often smaller than that available to a global cloud-based FL service aggregating millions of devices or customers [5,18]. This can limit achievable accuracy or robustness, especially for large foundation models.

Access to large pre-trained models. Many state-of-the-art models are trained centrally using massive data and compute budgets. Edge-only environments often cannot reproduce this pretraining, and hosting very large models on-premise may be impractical due to hardware, energy, or licensing constraints. As a result, edge-only FL often relies on imported pre-trained models (possibly from a cloud environment) and focuses on local fine-tuning or adaptation [42,44]. This introduces additional concerns around model supply chain security and update management.

Operational burden and skills. Operating a secure, resilient FL stack on-premise requires skills in distributed systems, MLOps, security, and domain-specific integration. Organizations must manage upgrades, patching, monitoring, and incident response for their FL infrastructure, in addition to domain applications. This operational burden can be substantial compared to consuming a managed FL service, especially for smaller enterprises [39,47].

Multi-site coordination complexity. In multi-site private federations, additional complexity arises from coordinating FL across sites with different local policies, hardware, and network conditions. Schedules must be aligned, connectivity must be provisioned and monitored, and governance for cross-site model sharing must be agreed upon. Cross-site failures (e.g., one site going offline, or lagging behind in updates) must be handled gracefully in the global aggregation logic [31,32].

Hybrid edge-only + cloud pretraining.

Given these constraints, many operators adopt *hybrid strategies*. Large models may be pre-trained in a central cloud using public or semi-public datasets, then exported into edge-only environments for local fine-tuning or domain adaptation via FL [2,5]. Periodic refreshes of the base model may occur through carefully controlled offline procedures (e.g., scheduled maintenance windows, air-gapped media). This hybrid model leverages the strengths of both worlds: global-scale pretraining and local, privacy-preserving adaptation.

2.4.2. Representative edge-only FL deployments and frameworks

Table 2 summarizes representative edge-only and on-premise FL efforts across domains, highlighting architecture patterns, frameworks, and key design choices. While not exhaustive, it illustrates the diversity of use cases and emphasizes how often edge-only FL is intertwined with domain-specific constraints and existing infrastructure.

Table 2. Representative edge-only and on-premise FL deployments and frameworks across domains. "Scope" distinguishes single-site from multi-site private federations.

Domain	Representative work / framework	Scope	Edge infrastructure	FL / architecture pattern	Ref.
Industrial IoT	FL-oriented edge computing framework for IIoT (software-defined gateways)	Single-site or multi-site	IIoT gateways, edge servers near machines	Edge-centric FL with close cooperation between gateways and devices; supports rapid deployment of AI models	[30]
Industrial IoT	Smart and collaborative IIoT: FL + data governance	Multi-site private federation	Edge–fog hierarchy, plant-level servers	Hierarchical edge–fog–cloud with edge-only plant-level training and optional higher-level aggregation	[36]
Industrial IoT	Secure FL for industrial IoT edge computing	Single-site or multi-site	Edge servers in industrial networks	Edge-only FL with security-focused extensions	[41]
Industrial IoT	Resource allocation and scheduling in IIoT FL	Multi-site	Edge nodes at multiple factories or sites	Multi-tier federated edge learning with cross-site coordination	[32,48]
Healthcare	Federated edge computing for privacy-preserving analytics in healthcare	Single-site or regional	Hospital edge servers, on-premise clusters	Edge-only FL for local analytics (risk scoring, monitoring) with optional regional aggregation	[37]
Healthcare	Federated learning for e-healthcare (next-gen holistic architecture)	Multi-site private federation	Hospital/clinic edges, regional data centers	Cross-silo FL at the edge, sometimes with hierarchical regional aggregation	[42]
Healthcare	Hospital-centric edge FL prototypes (ICU monitoring, imaging)	Single-site	On-premise GPU clusters in hospitals	Edge-only FL across devices and departments	[17,20]
Critical infrastructure (rail, energy)	Edge-only FL for railway condition monitoring and safety	Multi-site private federation	Wayside edge servers, on-board units, regional control centers	Edge-only FL within segments; periodic cross-segment model aggregation over private backbone	[31]
IoT / smart environments	FL in IoT with edge/fog computing	Single-site or regional	Edge/fog nodes near sensor clusters	Edge-centric FL leveraging local fog nodes; may be combined with hierarchical aggregation	[44,49]
Generic on-premise FL	FedML / TensorOpera deployments in enterprises (e.g., automotive)	Single-site or multi-site	On-premise GPU clusters, private networks	On-prem FL jobs orchestrated entirely inside enterprise networks	[43,46]
Generic on-premise FL	FEDn and similar self-managed frameworks	Single-site or multi-site	Private clouds, on-prem clusters	Self-managed FL, often in hub-and-spoke or hierarchical patterns inside private infra	[47]
Platforms / management	Managing FL on decentralized infrastructures as a service	Multi-site	Heterogeneous on-prem and edge resources	FL-as-a-service over decentralized, often on-prem infrastructures	[39]
Security / privacy	Survey: FL data security and privacy-preserving in edge IoT	Conceptual, multi-domain	Edge IoT devices and gateways (conceptual)	Edge-centric FL with focus on cryptography and DP	[34]

2.5. Decentralized FL

2.5.1. Removing the central server

Decentralized federated learning (DFL) departs from the classic FL paradigm by eliminating the central server altogether. Instead of all clients communicating with a single coordinator, each client becomes a peer in a communication graph and interacts only with its neighbors. The responsibility for model coordination is thus distributed across the network: nodes exchange model parameters or updates directly, and global consensus on the model emerges over time as information diffuses through the graph [22,50,51]. This architectural shift is appealing in settings where a central coordinating entity is undesirable, infeasible, or untrusted—for example, in multi-stakeholder consortia, community networks, or infrastructure-poor environments where centralized connectivity is unreliable.

Formally, consider a set of K nodes (clients), each holding a local model w_t^k at round t and a private dataset $\mathcal{D}_k$. The communication pattern is described by a (possibly time-varying) graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t)$, where vertices correspond to nodes and edges represent communication channels. For a given node

k, let $\mathcal{N}(k)$ denote its (possibly directed) neighbor set at round t. A generic DFL update can then be written as

$$w_{t+1}^{k} = \sum_{j \in \mathcal{N}(k)} P_{kj}\, \tilde{w}_{t}^{j}, \tag{9}$$

where P is a *mixing matrix* that encodes the weights assigned to neighbors' models (often row-stochastic and symmetric), and $\tilde{w}_t^j$ is the locally updated model at node j after performing one or more steps of local training (and possibly compression or noise addition) [22,51]. Intuitively, each node repeatedly blends its model with those of its neighbors, gradually averaging information across the network.

Convergence properties of DFL depend critically on both *optimization* and *network* characteristics. From the optimization side, the usual factors apply: the step size and number of local gradient steps, the smoothness and convexity of local objectives, and the heterogeneity of data distributions across nodes. From the network side, the key quantity is often the *spectral gap* of the mixing matrix P, which captures how quickly information propagates across the graph: larger spectral gaps (better-connected graphs) lead to faster consensus, while sparse or poorly connected graphs slow down mixing [50,51]. Time-varying graphs—common in mobile and wireless edge networks—introduce additional challenges, as connectivity patterns and neighbor sets may change from round to round.

Compared to server-based FL, DFL avoids a single point of failure and reduces dependency on a central infrastructure, but it also complicates protocol design. Nodes must coordinate without global clocks or centralized scheduling; they must handle asynchronous updates, message delays, and potential packet loss; and they must defend against adversarial peers in the absence of a trusted aggregator. As such, DFL algorithms often blend ideas from distributed optimization, consensus theory, and peer-to-peer networking.

2.5.2. Gossip and consensus protocols

Several families of DFL protocols have emerged, distinguished primarily by how they perform averaging (or consensus) over the communication graph [22,51].

Randomized gossip.

In **randomized gossip** algorithms, each node periodically picks one or a few neighbors at random and exchanges model parameters with them. The simplest form is pairwise averaging: node k randomly selects neighbor j, they exchange models, and both update their states to a weighted average of their previous states. When repeated, such pairwise interactions cause models to converge toward the global average, provided the graph is connected and the random choice process is suitably ergodic [50,51]. Gossip-based DFL has several attractive properties for edge environments:

- It is inherently *asynchronous*: nodes can initiate gossip whenever they are available, without waiting for global synchronization.
- It *adapts* naturally to dynamic graphs: as long as the time-averaged connectivity graph remains connected, convergence can be guaranteed.
- It is *local*: each node communicates only with a small subset of neighbors, which matches constraints in wireless and ad hoc networks.

The downside is that gossip may converge slowly on poorly connected or large-diameter graphs, and it can be inefficient in terms of communication if many small updates are exchanged over time.

Deterministic consensus methods.

Deterministic consensus methods use fixed or structured mixing matrices and often draw from classical consensus algorithms in distributed systems. A common choice is to use Metropolis–Hastings weights or other doubly stochastic matrices that ensure the average of models is preserved under repeated multiplication [22,51]. Nodes update their models according to

$$w_{t+1}^{k} \leftarrow \sum_{j} P_{kj} \tilde{w}_{t}^{j}, \tag{10}$$

with P chosen to reflect the underlying communication graph. More advanced methods incorporate gradient-tracking, push-sum techniques (to handle directed or unbalanced graphs), or primal–dual schemes (e.g., EXTRA, ADMM-based consensus) to accelerate convergence and handle heterogeneity.

Deterministic consensus approaches can achieve faster and more predictable convergence than purely randomized gossip when the graph is relatively stable and known. They are particularly well-suited to fixed infrastructure networks (e.g., wired edge clusters, base-station backhaul), where the communication topology can be explicitly designed and optimized. However, they may be less robust to rapid topology changes unless augmented with reconfiguration mechanisms.

Overlay networks and logical topologies.

In many edge deployments, the physical network topology is constrained by geography and infrastructure. To improve the performance and robustness of DFL, it is common to construct **overlay networks**—logical topologies on top of the physical graph—that have desirable properties such as small diameter, high expansion, or good fault tolerance [6,52]. Examples include:

- *Ring overlays*, where nodes are arranged in a logical ring and communicate primarily with their immediate logical neighbors.
- *Expander graphs* and other high-connectivity structures, which ensure rapid mixing and robustness to node or edge failures.
- *Small-world networks*, which combine local connectivity with a few long-range links to reduce average path length.

These overlays can be constructed and maintained using standard peer-to-peer techniques and updated as nodes join, leave, or move. They provide a flexible tool to tune the trade-offs between convergence speed, communication cost, and resilience.

Use cases in edge environments.

Gossip- and consensus-based DFL are particularly attractive for edge scenarios where infrastructure is limited or non-uniform. For example:

- *Vehicle-to-vehicle (V2V) learning*: vehicles in proximity can exchange model updates directly over V2V links, learning traffic patterns or driving policies collaboratively without relying on base-station connectivity [23].
- *UAV swarms*: drones performing cooperative sensing or search may run DFL over mesh networks, adapting models as they move and communicate intermittently [53].
- *Community wireless or mesh networks*: user-owned routers and access points can share models for anomaly detection or resource allocation via DFL, reflecting the decentralized governance of the underlying infrastructure.

2.5.3. Blockchain-enabled DFL

To provide tamper-evident logs, decentralized storage of updates, and economic incentives, blockchain and distributed ledger technologies (DLTs) are increasingly combined with DFL [23,54]. In *blockchain-enabled FL*, the blockchain serves as a shared, append-only ledger on which participants record model updates, gradient hashes, or aggregation results. Smart contracts encode the rules of the learning game: how updates are validated, how rewards are computed, and how model versions advance.

Ledger as a coordination substrate.

In a blockchain-enabled DFL system, each node (or a subset of nodes) acts as both a learner and a blockchain participant (e.g., miner or validator). When a node computes a local update, it can:

- Submit a transaction containing the update (or a compressed/hash representation) to the blockchain.
- Let the consensus protocol order and validate these transactions.

- Use smart contracts to aggregate (e.g., average) the updates contained in recent blocks into a new global model state.

The ledger thus provides a globally consistent sequence of updates and model snapshots, even in the absence of a central server. Nodes that are offline for some time can catch up by replaying the ledger, reconstructing the model's evolution since they last participated.

Auditable contributions and incentives.

One of the main benefits of integrating blockchain with DFL is **auditability**. Because updates are recorded on an immutable ledger, nodes can prove their participation and contributions, which is valuable in open or semi-open federations where participants may be anonymous or mutually untrusted [54,55]. Token-based incentives can reward nodes whose updates improve model accuracy or robustness, while penalizing or blacklisting those that behave maliciously. Reputation systems can be built on top of the ledger to rate the reliability of nodes over time.

Resilience and trust decentralization.

Blockchain consensus protocols (e.g., proof-of-work, proof-of-stake, Byzantine fault tolerant (BFT) algorithms) provide resilience to node failures and malicious behavior at the ledger level. By avoiding a single trusted server, blockchain-enabled DFL mitigates single-point-of-failure concerns and distributes trust across multiple validators [23,54]. This is especially appealing in multi-stakeholder settings, such as cross-organization healthcare federations, supply-chain consortia, or community networks, where no single entity is universally trusted to coordinate learning.

Overheads and privacy implications.

These benefits come with significant costs. Maintaining a blockchain requires additional computation (for consensus), storage (to keep the ledger), and communication (to propagate blocks). For resource-constrained edge devices, these overheads may be prohibitive, so often only a subset of more capable nodes (e.g., gateways or edge servers) participate as validators, while others remain lightweight clients [23,55]. Moreover, because updates or their hashes are stored on-chain, careful design is needed to avoid leaking sensitive information about local data. Techniques such as committing only to encrypted, hashed, or differentially private updates, using permissioned blockchains with strict access control, or storing large updates off-chain with on-chain references, are commonly used mitigations.

2.5.4. Challenges for DFL at the edge

While DFL offers compelling advantages, particularly in highly distributed or infrastructure-poor settings, it also raises unique challenges when deployed in edge networks [6,22].

Topology design and dynamics.

Edge networks are inherently dynamic. Devices move (e.g., vehicles, drones), links appear and disappear, and the set of active nodes changes over time. DFL protocols must therefore tolerate frequent topology changes and partial connectivity. Ensuring convergence under time-varying graphs typically requires that, over suitably long windows, the union of communication graphs remains connected (the "joint connectivity" condition). Designing communication patterns that are both efficient and robust to mobility—possibly using adaptive overlays or mobility-aware neighbor selection—is an active research area.

Adversaries and trust.

Without a central arbiter, defending against adversarial participants becomes more complex. Malicious nodes can perform model poisoning, data poisoning, or free-riding attacks; they can manipulate gossip to bias consensus or partition the network. Robust aggregation—such as median- or trimmed-mean-based updates, coordinate-wise clipping, or norm bounding—can help, but implementing these in a fully decentralized way is non-trivial. Some DFL schemes rely on trust assumptions within local neighborhoods or incorporate blockchain-based auditing to detect and penalize misbehavior, but the threat surface remains broader than in centrally coordinated FL [22,54].

Energy and spectrum usage.

Peer-to-peer exchanges in DFL can generate substantial local traffic, especially in dense wireless networks. Each node may need to communicate with multiple neighbors per round, leading to higher aggregate usage of wireless spectrum and battery power compared to designs where only a subset of nodes communicate with a central base station. Careful scheduling and MAC-layer integration are therefore crucial: DFL communications may need to be duty-cycled, prioritized, or piggybacked on existing control or data traffic to avoid congesting the network [29]. Energy-aware DFL algorithms attempt to adapt communication frequency, neighbor selection, or model compression based on local battery and channel conditions.

System heterogeneity and fairness.

Similar to centralized FL, DFL must cope with heterogeneity in compute power, memory, and connectivity. Some nodes may be high-end edge servers; others may be tiny IoT devices or mobile handsets. Ensuring that models do not overfit to well-connected or high-resource nodes requires careful algorithmic design. For instance, protocols may weight updates based on data size and reliability, or they may incorporate fairness constraints in neighbor selection and aggregation. In addition, in multi-stakeholder scenarios, ensuring fair influence among organizations or communities is as important as ensuring fairness among individual devices.

Debugging, monitoring, and lifecycle management.

Finally, the operational side of DFL is challenging: without a central server, there is no single point at which logs can be collected, models inspected, or experiments controlled. Monitoring the health and performance of a DFL system requires distributed observability mechanisms and new tooling for debugging misbehaving nodes or subgraphs. Model versioning, rollback, and unlearning requests (e.g., GDPR "right to be forgotten") are also harder to implement when model state is replicated and updated across many peers.

2.5.5. Representative DFL protocols and systems at the edge

Table 3 summarizes representative decentralized and blockchain-enabled FL protocols and systems with relevance to edge environments. It categorizes works by protocol type, communication topology, edge scenario, robustness features, and deployment considerations.

Table 3. Representative decentralized and blockchain-enabled FL protocols and systems relevant to edge environments. "Topology" refers to the logical communication graph used for model exchange.

Category	Protocol / system	Topology / consensus	Edge scenario	Key characteristics	Ref.
Decentralized FL (gossip)	Decentralized SGD / gossip-based FL	Randomized gossip over undirected graph; pairwise model averaging	Generic edge clusters, sensor networks	Simple, communication-efficient for small neighborhoods; convergence depends on spectral gap; supports asynchronous updates	[22,50]
Decentralized FL (gradient tracking)	Gradient-tracking DFL (e.g., D-PSGD, GT-D2)	Fixed or slowly varying mixing matrix; consensus + gradient tracking	Wired edge clusters, private LANs	Uses memory of past gradients to correct drift; improves convergence under data heterogeneity; requires more state per node	[51,56]
Decentralized FL (overlay)	Gossip over expander / small-world overlays	Overlay expander or small-world graphs built atop physical topology	Community networks, mesh Wi-Fi, sensor swarms	Logical overlays reduce diameter and accelerate consensus at the cost of overlay maintenance	[6,52]
Hybrid centralized–decentralized	Clustered DFL with local gossip and global aggregation	Local gossip within clusters; periodic aggregation at regional server	Hierarchical edge (e.g., base stations + core)	Balances communication and coordination load; natural fit for MEC and IIoT; combines benefits of decentralized and centralized coordination	[22,28]
Vehicle-to-vehicle FL	V2V gossip FL	Dynamic V2V communication graph; opportunistic pairwise exchanges	Vehicular edge networks (cars, RSUs)	Uses contact opportunities between vehicles; can operate with intermittent infrastructure; often combined with roadside infrastructure	[21,23]
UAV / drone swarms	UAV-based decentralized FL	Time-varying mesh among UAVs; gossip or consensus	UAV swarms for search, mapping, or surveillance	Emphasizes light-weight communication and energy awareness for aerial platforms; robustness analyzed via time-varying graph theory	[22,53]
Blockchain-enabled FL	General blockchain-based FL frameworks	Permissioned or public blockchain; PoW, PoS, or BFT consensus	Multi-stakeholder IoT, fog computing	Provides auditable contributions and incentive mechanisms via smart contracts; introduces latency and energy overheads	[23,54]
Blockchain-enabled edge FL	Edge-oriented blockchain FL (e.g., for IoV, IIoT)	Lightweight permissioned blockchain among edge nodes	Intelligent transport systems, industrial edge	Tailors blockchain layer to resource-constrained edge; often uses committee-based or BFT-style consensus; focuses on tamper-evident logging	[53,55]
Energy-aware DFL	Energy- and spectrum-aware DFL schemes	Mix of gossip/consensus with energy-aware scheduling	5G/6G MEC, dense wireless edge	Explicitly models energy/spectrum constraints; trades off accuracy vs. resource usage through power-aware neighbor selection and update frequency	[21,29]
Security-focused DFL	Byzantine-resilient DFL	Gossip / consensus with robust aggregation (median, trimmed mean)	Adversarial edge environments	Extends robust aggregation ideas to fully decentralized settings; often assumes bounded fraction of adversaries; can be combined with reputation mechanisms	[22,54]
Frameworks / surveys	DFL surveys and toolkits	Conceptual; various topologies	Various edge and IoT scenarios	Provide taxonomies and guidelines for choosing DFL schemes given network, topology, and threat models	[22,55]

2.6. *Split and Hybrid Architectures*

2.6.1. From FL to split learning

Classical FL implicitly assumes that each participating client can host the *entire* model being trained. This assumption is increasingly strained in modern edge scenarios. On-device models for perception, language, or multi-modal reasoning can contain hundreds of millions—or even billions—of parameters, far exceeding the memory and compute budgets of microcontrollers, low-end smartphones, or embedded devices. Even when a model barely fits, running full backpropagation locally may be too energy-intensive or too slow for latency-sensitive applications. At the same time, organizations often wish to leverage powerful cloud- or edge-hosted backbones (e.g., large vision encoders or language models) while keeping raw inputs on-device. These tensions motivate *split learning* (SL) [57–59].

Split learning addresses resource and privacy limitations by *partitioning* the model across participants. Concretely, consider a deep network with layers $1, \ldots, L$. Split learning chooses a *cut layer* ℓ and divides the network into:

- a *front* (or client-side) sub-network consisting of layers $1, \ldots, \ell$, which runs on the device or edge node; and
- a *back* (or server-side) sub-network consisting of layers $\ell + 1, \ldots, L$, which runs on a more powerful edge server or cloud node.

During training, the device performs a forward pass through its local front sub-network on raw input x and obtains the intermediate activations $h_\ell(x)$. Instead of sending x to the server, it transmits these activations—or their compressed, obfuscated variant—to the server. The server then completes the forward pass through the back sub-network, computes the loss and gradients with respect to $h_\ell(x)$, and sends the gradient $\nabla_{h_\ell} \mathcal{L}$ back to the device. The device finally runs backpropagation through its front sub-network using this gradient to update its local parameters [57,58].

Several aspects of this workflow are worth emphasizing:

- **Computation offloading.** Because only the front sub-network runs on the device, the majority of parameters and floating-point operations (FLOPs) can be offloaded to the server. This allows the overall model to be much larger than what would fit entirely on the device, enabling edge applications to benefit from state-of-the-art architectures [59].
- **Bandwidth vs. privacy.** Transmitted activations $h_\ell(x)$ generally have much lower dimensionality than raw inputs, particularly for high-resolution images or long text sequences. This reduces bandwidth usage relative to sending raw data. At the same time, activations may obfuscate some aspects of the input, providing a degree of privacy; however, they can still leak information via inversion attacks, motivating additional protections such as noise injection, activation compression, or cryptographic masking [58,59].
- **Limited exposure of labels and gradients.** In many SL variants, labels reside only on the server side, or are kept on the client and used jointly with cut-layer gradients, depending on the threat model. In medical imaging, for example, it is common to keep labels and sensitive metadata only on the hospital side while offloading some computation to a semi-trusted cloud or vendor server [59,60].
- **Protocol complexity.** Unlike standard FL, where only models or gradients are exchanged once per round, SL introduces per-batch interactions: each mini-batch requires sending activations and receiving gradients. This places stricter demands on latency and availability of links between device and server, and it complicates batching and pipeline parallelism. Many systems adopt micro-batching, asynchronous queues, or pipelined training to amortize these costs [59].

Viewed through a broader lens, split learning is a particular form of *model parallelism* across trust and resource boundaries: one part of the model lives where the data is generated, and another part lives where compute is abundant. It can be combined with FL and other distributed learning strategies to create rich hybrid architectures spanning device, edge, and cloud.

2.6.2. Split-federated and three-tier hybrids

Pure split learning addresses the constraint that a single client cannot host a full model, but by itself it does not exploit the statistical benefits of federating across many clients. Conversely, pure FL exploits data across multiple clients but assumes each can host the entire model. Recent work combines these ideas into *split-federated* and multi-tier hybrid architectures that span device, edge, and cloud [60–65].

Split-federated learning (SpFL).

In **split-federated learning**, multiple clients each host a front sub-network, while a shared back-end network resides at a server. Each client runs SL with the server as described above, but the front sub-networks themselves are subject to FL: they are periodically aggregated across clients via FedAvg-like protocols [60,61]. Conceptually, this yields two intertwined training loops:

- *SL loop:* for each client, per-batch activations and gradients flow between the client's front and the shared back-end, enabling end-to-end training without exposing raw data.
- *FL loop:* across clients, front sub-network parameters are periodically aggregated at the server (or at an intermediate edge node) to form a global front, which is then redistributed.

The back-end can be shared across all clients (acting as a global feature extractor or classifier) or can have client-specific heads for personalization. In recommender systems, for instance, client-side fronts may capture user-specific behavior while a shared back-end models large item catalogs [61].

U-shaped split FL for encoder–decoder models.

Many vision and medical-imaging models use U-Net-like encoder–decoder architectures, where high-resolution input is compressed into a latent representation and then decoded back into dense predictions. **RoS-FL** and related methods propose *U-shaped split FL*, in which the encoder and decoder segments are split across client and server in a way that balances computation, memory, and privacy [60]. Typical patterns include:

- Client hosts early encoder layers and late decoder layers, keeping raw images and final predictions on-site.
- Server hosts mid-level encoder and decoder layers, using them to capture global structure and complex patterns.

Gradients flow across the cut points in both encoder and decoder, enabling end-to-end training while ensuring that the server never sees raw inputs or final labels. FL across multiple clients can be performed on portions of the U-Net that reside on the client side, while the shared middle backbone is trained centrally or via SL-style aggregation [60].

Three-tier U-shape split learning (client–edge–cloud).

Frameworks such as **FUSE** and **EUSFL** extend the split and U-shaped ideas to three tiers—client, edge, and cloud [62,63]. For example:

- The *client* hosts very shallow input layers (e.g., initial convolutions), pre-processing data and extracting low-level features while ensuring that raw sensor values remain local.
- The *edge* hosts intermediate layers that capture regional or cluster-specific patterns, possibly shared across a small number of clients.
- The *cloud* hosts deep backbones or task heads that require large memory or compute, such as transformer blocks or large classification/regression heads.

Training proceeds with multiple cut points: activations flow from client to edge, then from edge to cloud; gradients flow in reverse. FL can be applied at the client–edge boundary (aggregating client-side fronts across devices) and at the edge–cloud boundary (aggregating edge-side segments across sites). Such three-tier architectures allow designers to flexibly partition large models and to place different parts of the network where it makes most sense from a latency, bandwidth, and trust perspective [62,63].

Edge–cloud collaborative FL/SL and split fine-tuning.

Another family of hybrid architectures focuses on **edge–cloud collaboration** for model adaptation and knowledge transfer. Frameworks such as split fine-tuning (SFT), edge–cloud collaborative knowledge transfer, and USFL (unified split–federated learning) enable bi-directional transfer between edge FL models and centralized models using logits, feature distillation, and selective split layers [64–66]. Typical patterns include:

- *Cloud-to-edge transfer:* A large cloud model is split, and its early layers are distilled or fine-tuned at the edge using local data; logits or intermediate features from the cloud guide edge training (teacher–student style).
- *Edge-to-cloud transfer:* Edge FL models provide distilled knowledge (e.g., averaged logits, feature statistics) back to the cloud, which uses them to refine its global model without direct access to raw edge data.
- *Selective splitting:* Only certain layers or modules are split across edge and cloud, while others remain entirely local or entirely centralized; split decisions may change over time based on load and resource conditions.

These hybrid strategies blur the line between FL and centralized training, viewing them as two ends of a spectrum connected by split and distillation mechanisms.

2.6.3. Resource and privacy trade-offs

Split and hybrid architectures open up a rich design space, but they also introduce complex trade-offs that must be reasoned about carefully.

Compute vs. communication.

The position of the cut layer ℓ directly affects the distribution of computation and communication:

- *Early splits* (small ℓ) place only a few layers on the device. This minimizes on-device compute and memory but requires sending larger activations, since early feature maps often have high spatial or temporal resolution. Early splits therefore favor low compute but high communication.
- *Late splits* (large ℓ) place many layers on the device, which reduces the dimensionality of activations and thus communication cost, but increases local compute and memory usage. Late splits therefore favor high compute but low communication.

In practice, designers must balance these dimensions based on hardware characteristics (CPU/GPU capabilities, memory), network conditions (bandwidth, latency), and application constraints (latency budgets, energy consumption). Some systems adapt ℓ dynamically based on real-time measurements, using early splits when devices are lightly loaded or networks are uncongested, and later splits otherwise [58,59].

Privacy vs. utility.

Sending intermediate activations instead of raw data provides a first layer of privacy: the server does not see raw images, audio, or sensor values. However, activations can still be inverted or exploited to reconstruct sensitive information, especially if the attacker knows or controls the back-end model [58,59]. Moreover, gradients sent back to the client can leak label or model information. To strengthen privacy, split and hybrid architectures often combine SL with:

- *Federated learning*: rather than training a single split model per client in isolation, multiple clients engage in FL to share knowledge via parameter averaging, making it harder to attribute any particular behavior to a single client.
- *Differential privacy (DP)*: noise is added to activations or gradients to bound the influence of individual data points [60,62].
- *Secure aggregation and encryption*: activations or gradients may be encrypted or aggregated across clients at intermediate nodes before reaching the back-end, reducing exposure even further.

The trade-off is often between privacy strength and model utility: stronger noise or obfuscation can degrade accuracy, while lighter protections may expose more information. Hybrid designs can tune these trade-offs per layer or per modality.

Scheduling and placement.

Beyond where to split, designers must decide *where* to place each model partition (device, edge, or cloud) and *when* to synchronize FL and SL components. Jointly choosing cut layers, placement, and synchronization schedules is a combinatorial optimization problem that depends on:

- available resources at each tier (compute, memory, network);
- workload patterns (burstiness, diurnal variations, mobility);
- application-level SLAs (latency, throughput, availability); and
- security and privacy policies (which layers and data can leave certain domains).

Many works address this through heuristics (e.g., threshold-based rules on latency and CPU usage), greedy optimization, or reinforcement learning, where an agent learns to choose splits and placements to optimize long-term rewards such as accuracy, latency, and energy consumption [64,66]. Coordinating FL rounds (which operate on slower timescales) with SL interactions (which operate per batch) adds further complexity, especially when dealing with partial participation and dynamic client sets.

2.6.4. Representative split and hybrid architectures in edge AI

Table **??** summarizes representative split, split-federated, and hybrid FL/SL architectures with emphasis on edge AI scenarios, highlighting how they partition models, leverage FL, and manage privacy and resource trade-offs.

Table 4. Representative split and hybrid FL/SL architectures for edge AI (structural aspects). "Split location" indicates typical positions of cut layers between tiers; actual designs may use multiple cut points.

Category	**Architecture / framework**	**Split location / tiers**	**FL coupling**	**Representative edge use cases**	**Ref.**
Split learning (baseline)	Original split learning	Single cut between client and server; client: early layers; server: rest	None (per-client SL)	Mobile/embedded vision, speech; small clients leveraging larger server models	[57, 58]
Split learning survey	Edge–cloud SL survey	Single or multiple cuts between device and edge/cloud	None or optional FL	Vision and IoT analytics in MEC, industrial edge	[59]
Split-federated learning	SpFL for recommendation / personalization	Client: front subnetworks; server: shared back-end	Fronts trained via FL; back-end shared or partially centralized	On-device recommendation, user modeling, content ranking at the edge	[61]
Split-federated learning	RoS-FL (U-shaped split FL)	U-Net-like encoder/decoder split across client and server	Client-side segments aggregated via FL; mid/back segments centralized	Medical image segmentation at hospital edge with vendor/cloud assistance	[60]
Three-tier split learning	FUSE (three-tier U-shaped SL)	Client: shallow encoder/decoder; edge: mid-level; cloud: deep backbone	FL across clients and possibly across edge nodes; global backbones shared at cloud	Industrial monitoring and anomaly detection with theft-resilient learning	[62]
Three-tier split learning	EUSFL (edge–client–cloud U-shaped FL)	Client: input and output layers; edge: mid encoder; cloud: shared head or deep layers	FL at client–edge and edge–cloud boundaries; supports hierarchical FL	Medical imaging and multi-site healthcare with strict privacy partitioning	[63]
Edge–cloud collaborative FT	Split fine-tuning (SFT)	Cloud model partially split; early layers adapted at edge; rest in cloud	Edge models trained via FL; cloud model updated using distilled knowledge	Personalization of large pre-trained models at the mobile/edge tier	[64]
Edge–cloud collab. FL/SL	Edge–cloud collaborative FL / knowledge transfer	Flexible split between edge backbones and cloud heads	FL among edge nodes; cloud learns from edge logits or features	Federated perception in autonomous driving, multi-camera edge vision	[65]
Unified split–FL	USFL (unified split + federated learning)	Multiple optional cut points across device/edge/cloud tiers	FL over selected segments at each tier; supports mixed deployment modes	General-purpose edge AI platform supporting heterogeneous devices and networks	[66]
Split learning variants	Label-/feature-split variants, vertical SL	Cuts separating feature owners and label owners across institutions	Often coupled with cross-silo FL	Cross-institution finance, healthcare, and advertising with vertically partitioned data	[58, 59]

Table 5. Representative split and hybrid FL/SL architectures for edge AI (privacy and resource-awareness).

Category	Architecture / framework	Privacy / security features	Resource-aware mechanisms	Ref.
Split learning (baseline)	Original split learning	Raw data stays on client; optional label hiding; can add DP or crypto	Offloads most compute to server; fixed cut layer; basic activation compression	[57,58]
Split learning survey	Edge–cloud SL survey	Discusses activation leakage, DP, secure protocols	Surveys strategies for cut placement, activation compression, and pipelining	[59]
Split-federated learning	SpFL for recommendation / personalization	Data localized at clients; optional DP; back-end may be semi-trusted	Balances client compute vs. bandwidth by tuning front depth; uses model compression	[61]
Split-federated learning	RoS-FL (U-shaped split FL)	Keeps raw images and labels on client; mid-level activations only to server; can add DP	Allocation of encoder/decoder segments to balance compute/memory; heuristic split search	[60]
Three-tier split learning	FUSE (three-tier U-shaped SL)	Sensitive signals, labels at client; activations encrypted or compressed; optional DP	Joint placement of segments based on latency and compute; supports pipeline parallelism	[62]
Three-tier split learning	EUSFL (edge–client–cloud U-shaped FL)	Local output heads and labels never leave client; secure aggregation for shared layers	Dynamic selection of cut points; load-aware edge vs. cloud distribution	[63]
Edge–cloud collaborative FT	Split fine-tuning (SFT)	Edge keeps local data; cloud sees logits/features; can apply DP on distilled signals	RL- or heuristic-based selection of which layers to adapt at edge vs. cloud	[64]
Edge–cloud collab. FL/SL	Edge–cloud collaborative FL / knowledge transfer	No raw data leaves edge; only predictions or embeddings; optional DP	Schedules knowledge transfer based on link quality and resource state	[65]
Unified split–FL	USFL (unified split + federated learning)	Combines FL, SL, and DP; supports varying trust boundaries per customer/site	Uses RL or heuristic search for cut placement, tier mapping, and sync frequency	[66]
Split learning variants	Label-/feature-split variants, vertical SL	Each party sees only subset of features/labels; cryptographic protocols for joint training	Optimizes communication via feature selection, secure aggregation; handles heterogeneous schemas	[58,59]

2.7. Design Space and Comparative View

While the previous subsections have described each architecture in isolation, practitioners in edge AI must choose among, combine, and adapt them under concrete constraints: network topology and capacity, device heterogeneity, governance and trust relationships, regulatory environment, and the need to host large backbones. It is therefore useful to view *centralized*, *hierarchical*, *edge-only*, *decentralized*, and *split/hybrid* designs as points in a common *architectural design space* rather than as mutually exclusive options.

At a high level, this design space can be organized along the following axes:

- **Coordination / topology**. Who orchestrates training? Is there a single logical coordinator, multiple tiered coordinators, or no coordinator at all? How are clients connected in the logical communication graph (star, tree, cluster, mesh, overlay)?
- **Aggregation locality**. Where are model updates combined: in a central cloud, at edge clusters, within on-premise sites, or via fully peer-to-peer averaging? How many hops does an update traverse before being aggregated?
- **Trust and failure model**. Which entities are assumed to be trusted, semi-trusted, or adversarial? Where do we tolerate failures, and which failures are catastrophic (single point of failure) vs. local (cluster/site failures)?
- **Communication cost and path**. How many messages of what size traverse which links (device–edge, edge–cloud, peer-to-peer)? How does the cost scale with the number of clients K, the number of edge servers S, the model size $\|w\|$, and the participation fraction q?
- **Representative domains**. Which architectures naturally align with the operational and regulatory reality of mobile personalization, IIoT, healthcare, critical infrastructure, or multi-stakeholder ecosystems?

Table **??** positions the main architecture families along these axes and includes indicative communication complexity. Table 8 further refines this view from the perspective of *system dials*: for each architecture, what kinds of guarantees (data residency, offline operation, backbone size, trust distribution) are naturally supported?

Table 6. Architectural design space for federated edge AI (structural and trust aspects).

Architecture	Coordination / topology	Aggregation locality	Trust / failure model (typical)
Centralized FL (cross-device)	Star: device ↔ cloud	Single cloud server aggregates updates from a (small) sampled subset of devices	Server trusted or honest-but-curious; secure aggregation mitigates server visibility; server is single point of failure / bottleneck
Centralized FL (cross-silo)	Star: silo ↔ coordinator (cloud or regional edge)	Aggregation at central coordinator from tens to hundreds of silos	Coordinator semi-trusted; silos are institutions with separate governance; single point of failure at coordinator; strong emphasis on cross-org DP and secure aggregation
Hierarchical FL (2-tier device–edge–cloud)	Tree: device ↔ edge ↔ cloud	Within-cluster aggregation at edge servers; cross-cluster aggregation at cloud	Edge nodes and cloud are semi-trusted; failures mostly localized per cluster; global coordinator can still be single point of failure
Hierarchical / multi-tier FL (multi-level edge)	Tree / DAG spanning access, metro, regional, and core	Aggregation at multiple tiers (access edge, metro edge, regional DC, core cloud) with different cadences	Trust roughly increases toward core; failures can be isolated per branch; robustness depends on redundancy of mid-tier nodes
Clustered FL (topology-aware)	Clustered star / cluster mesh	Aggregation first within clusters (logical or geographic), then across cluster heads	Cluster heads semi-trusted; failures localized per cluster; cluster formation and reconfiguration affect trust assumptions
Edge-only FL (single site)	Star or small tree: device ↔ on-prem edge	Aggregation entirely within site (plant, hospital, campus) at one or a few edge servers	Single administrative domain; high trust in on-prem edges; failures local to site; internet outages do not impact FL
Edge-only FL (multi-site private federation)	Two-level tree: device ↔ site edge; sites interconnected via private backbone	Aggregation at site edges; cross-site aggregation at corporate DC or higher-level private edge	Single organization or tightly governed consortium; trust anchored in corporate DC; failures localized per site / region
Decentralized FL (gossip / consensus)	Mesh or overlay (ring, expander, small-world) among peers	No explicit aggregation point; updates mixed locally via gossip or consensus	No central trust anchor; robustness via redundancy; vulnerable to local adversaries; security depends on local and graph-wide assumptions
Blockchain-enabled FL / DFL	Mesh or hub-and-spoke with blockchain nodes; logical ledger overlay	Logical aggregation via ledger and smart contracts; physical aggregation may still occur at powerful validator nodes	Trust decentralized across validators; immutability and auditability; consensus tolerates some Byzantine faults but at higher cost
Split / hybrid FL–SL (device–cloud)	Star with cut layers: device ↔ server (edge or cloud)	Front-end layers at device; back-end layers at server; aggregation may occur at fronts (FL) and/or back-end (centralized)	Trust partitioned: device trusted with raw data, server trusted with back-end model; additional risk of activation leakage; failures at server impact many clients
Split / hybrid FL–SL (device–edge–cloud)	Mixed: device ↔ edge ↔ cloud with multiple cuts	Fronts at device, mid-layers at edge, heavy backbones at cloud; FL at device–edge and/or edge–cloud boundaries	Trust distributed across tiers; strict residency constraints can pin certain layers to specific domains; failures at intermediate tiers degrade performance locally

Table 7. Architectural design space for federated edge AI (communication and domain aspects). Communication cost is indicative and assumes K clients, S edge servers, model size $\|w\|$, activation size $\|a\|$ at split layers, and participation fraction q. Costs are per global round and focus on dominant WAN/backhaul terms.

Architecture	**Comm. cost per global round (rough)**	**Representative works / domains**
Centralized FL (cross-device)	$O(qK\|w\|)$ uplink + $O(qK\|w\|)$ downlink over WAN; uplink from devices is often dominant	On-device personalization for keyboards and recommendations, cross-device analytics [7–10]
Centralized FL (cross-silo)	$O(K\|w\|)$ uplink + $O(K\|w\|)$ downlink, but with K small (tens–hundreds) and often over well-provisioned links	Cross-hospital learning, cross-factory models, financial consortia, cross-operator collaboration [5,8,17]
Hierarchical FL (2-tier device–edge–cloud)	$O(qK\|w\|)$ over device–edge (local/LAN) + $O(S\|w\|)$ over edge–cloud (WAN); since $S \ll K$, WAN traffic is reduced vs. centralized FL	MEC, smart factories, hospital groups, regional telecom deployments [1,19,24,25,27]
Hierarchical / multi-tier FL (multi-level edge)	Sum of tier-wise costs: $\sum_{\text{tiers } r} O(S_r\|w^{(r)}\|)$ with S_r nodes and possibly tier-specific model sizes $\|w^{(r)}\|$	Large-scale IIoT, nation-wide telecom, smart-city infrastructures [5,6,21]
Clustered FL (topology-aware)	$O(qK\|w\|)$ within clusters (often over LAN/regional links) + $O(C\|w\|)$ at cluster-head layer, C number of clusters	Latency-aware and data-aware clustering in MEC and IIoT; region-specific personalization [19,25,27]
Edge-only FL (single site)	$O(qK\|w\|)$ over LAN / private 5G only; WAN cost essentially zero	Predictive maintenance, industrial control, on-prem healthcare analytics, security analytics on corporate campuses [18,20,35,37]
Edge-only FL (multi-site private federation)	$O(qK\|w\|)$ within sites + $O(S\|w\|)$ across sites over private WAN; can be scheduled infrequently or offline	Multi-plant industrial groups, regional hospital networks, rail and energy operators [30,31,36,42]
Decentralized FL (gossip / consensus)	$O(qKd_{\text{avg}}\|w\|)$ over peer-to-peer links, where d_{avg} is average node degree; traffic localized if neighborhood small	Community networks, mesh Wi-Fi, sensor swarms, ad hoc or delay-tolerant networks [6,22,50,51]
Blockchain-enabled FL / DFL	Additional $O(B\|w\|)$ or $O(B\|\text{tx}\|)$ per block for B validators; latency and energy overhead from consensus; storage cost for ledger	Multi-stakeholder IoT consortia, cross-provider collaboration, incentive-driven open federations [23,54,55]
Split / hybrid FL–SL (device–cloud)	Per batch: $O(qK\|a\|)$ activations uplink + $O(qK\|a\|)$ gradients downlink; plus periodic $O(qK\|w_{\text{front}}\|)$ FL sync for fronts	Mobile AR/VR, vision-based assistance, medical imaging with vendor cloud backbones, large-model personalization [57–60]
Split / hybrid FL–SL (device–edge–cloud)	Combination of $O(qK\|a^{(1)}\|)$ device–edge + $O(S\|a^{(2)}\|)$ edge–cloud activations, plus FL sync terms $O(qK\|w^{\text{front}}\|)$ and $O(S\|w^{\text{edge}}\|)$	Smart factories and IIoT, multi-site healthcare, edge–cloud foundation-model adaptation and split fine-tuning [61–64,66]

Table 8. Qualitative view of architectural "dials" for federated edge AI. A checkmark (✓) indicates that the architecture naturally supports the property; parenthesized marks (✓) indicate partial or configuration-dependent support.

Architecture	Strict data residency (per site / country)	Offline operation (no public internet)	Support for massive K (cross-device)	Support for large backbones (billions of params)	No single trusted coordinator	Alignment with legacy edge / OT systems
Centralized FL (cross-device)	(✓) via DP + secure agg, but cloud still sees updates	× (depends on cloud connectivity)	✓(designed for millions of devices)	(✓) if model fits in device memory; heavy backbones challenging on low-end devices	× (central server)	(✓) for mobile platforms; less natural for OT/IIoT
Centralized FL (cross-silo)	(✓) with careful region-specific hosting and legal controls	(✓) if silos connect via private WAN only, but usually some external connectivity	× (typically tens–hundreds silos)	✓(silos often have server-class hardware)	× (central coordinator)	(✓) in modern IT-heavy OT environments
Hierarchical FL (2-tier / multi-tier)	(✓) when edge tiers are placed along jurisdictional boundaries	(✓) if clouds replaced by on-prem hubs; otherwise limited by top tier	✓(edge tier absorbs massive cross-device scale)	✓(backbones at edge / cloud)	× or (✓) if top tier replicated	✓(naturally mirrors access–aggregation–core hierarchies)
Edge-only FL (single / multi-site)	✓(data and updates confined to site or private backbone)	✓(designed for disconnected or air-gapped settings)	(✓) within site; overall K limited by deployment scale	✓(on-prem GPU clusters for heavy models)	× or (✓) if sites coordinate via multi-master patterns	✓(explicitly integrates with industrial and healthcare stacks)
Decentralized FL (gossip / DFL)	(✓) if communication restricted within jurisdiction and data never leaves devices	✓(no reliance on central infrastructure)	(✓) but convergence and overhead grow with K	(✓) only on capable nodes; small devices may host compressed or partial models	✓(no central coordinator; trust spread over graph)	(✓) in peer-managed or community OT scenarios; integration non-trivial
Blockchain-enabled FL / DFL	✓(permissioned ledgers scoped to region / consortium)	(✓) if ledger runs entirely on-prem; consensus requires connectivity among validators	(✓) at protocol level; practical scale limited by ledger performance	✓(validators can host large models; clients can remain light)	✓(trust anchored in consensus among validators)	(✓) in consortia willing to run blockchain infra; heavier to integrate with legacy OT
Split / hybrid FL–SL (device–cloud)	(✓) for inputs and labels (stay on device) but activations cross boundaries	× or (✓) if "cloud" is replaced with on-prem edge only	✓(fronts can be small and light)	✓(back-end hosted centrally with large capacity)	× (back-end server is critical)	(✓) where OT devices can run thin front-ends and connect to central analyzers
Split / hybrid FL–SL (device–edge–cloud, USFL-style)	✓(layers pinned to specific tiers and jurisdictions)	✓(if cloud tier is optional or replaced by central on-prem)	✓(multi-tier scaling similar to hierarchical FL)	✓(heavy backbones at cloud; modest partitions at edge)	(✓) if multiple coordinators and mixed FL/DFL are used	✓(flexible mapping of layers to OT vs. IT infrastructure)

From a design perspective, Tables **??** and 8 can be read as a set of *architectural dials* when engineering an FL system for edge AI:

- **Cloud connectivity and WAN sensitivity.** If WAN bandwidth and latency are dominant concerns, but a cloud (or central regional hub) is acceptable, hierarchical FL is a natural evolution from centralized FL. It pushes as much traffic as possible into cheap device–edge links and reserves WAN for aggregated updates. Multi-tier designs offer further savings when access, metro, and core tiers can be exploited.
- **Data residency and offline operation.** If data residency and offline operation are paramount—e.g., due to regulatory constraints, exposure risk, or physical isolation—edge-only FL (possibly with multi-site private federation and scheduled synchronization) is attractive. Here, the central design question is how to align FL boundaries with legal and organizational perimeters (sites, regions, countries).
- **Trust distribution and governance.** If no single entity is fully trusted or if governance is fundamentally decentralized (e.g., across multiple organizations or communities), decentralized or blockchain-enabled FL becomes appealing despite higher protocol complexity. Architectures in this quadrant emphasize consensus, auditability, and incentive mechanisms over central control.

- **Model capacity and device limits.** If full models cannot fit on devices, or if cloud-scale backbones are desired, split or hybrid FL–SL architectures become necessary. These introduce additional dials: where to split, which tiers host which layers, and how often to synchronize FL across front-end partitions. In practice, large foundation models are often hosted centrally (cloud or powerful edge), while smaller front-ends run on devices for personalization and privacy.
- **Topology and resilience.** If the underlying network already has a strong hierarchical structure (e.g., telecom or utility networks), architectures that mirror this hierarchy (hierarchical/multi-tier FL, split FL across tiers) will be easier to deploy and more robust. In contrast, highly dynamic or infrastructure-poor environments (e.g., vehicular, UAV, or disaster scenarios) may favor gossip-based DFL or hybrid schemes that rely only on local contact opportunities.

In practice, real systems seldom instantiate a single "pure" pattern. Instead, they combine multiple architectures in a layered fashion: hierarchical FL with split learning inside each tier; edge-only FL augmented with periodic cloud pre-training; decentralized gossip within clusters combined with centralized cross-cluster aggregation; or split FL with blockchain-based logging of critical updates. As surveys of edge FL architectures emphasize, this leads to a rich and rapidly evolving landscape, where design choices must be revisited as hardware capabilities, regulatory requirements, and foundational model ecosystems change [1,2,6].

3. Core Challenges and Enabling Techniques

Federated edge AI inherits all the classic difficulties of distributed learning and then adds new ones from wireless networks, heterogeneous hardware, regulatory constraints, and the desire to leverage large foundation models. This section reviews key challenge axes and the principal algorithmic and systems techniques that have emerged to address them, with emphasis on recent developments in the last four years.

3.1. Statistical Heterogeneity and Personalization

In most edge and cross-device deployments, client data are *non-IID*, *unbalanced*, and often *domain-shifted*: different phones exhibit different usage patterns, factories run different processes, and hospitals serve different populations. Empirically, standard FedAvg-type training can converge to a global model that performs well on an "average" distribution but underperforms for clients whose data are rare, highly skewed, or structurally different from the majority [8,67]. This has motivated a rich literature on *personalized federated learning* (PFL), in which each client maintains a model adapted to its own distribution while still benefiting from shared information.

Recent surveys organize PFL methods into several families: model-optimization based, meta-learning based, multi-task based, clustering-based, and representation-based approaches [68,69]. More recent work also ties personalization to robustness and fairness, e.g., the TPAMI 2024 survey and benchmark on FL for generalization, robustness, and fairness [70]. We highlight several core strategies that are especially relevant to edge settings:

Local fine-tuning and head personalization.

The simplest personalization strategy is to train a global model using standard FL and then fine-tune it on each client's local data. This can be as simple as a few local gradient steps or as sophisticated as full local retraining with early stopping and regularization to avoid catastrophic forgetting. For deep models, a common pattern is to share lower layers (feature extractors) globally and personalize only the last few layers (task-specific heads) per client [8]. Recent PFL methods formalize this as *shared representation + personalized classifier*, learning a global feature space while allowing client-specific classifiers or adapters on top [71,72]. This approach is attractive at the edge because it allows a heavy backbone to be reused across clients (possibly hosted at an edge server), while lightweight personalized heads are trained locally.

Model interpolation and proximal regularization.

Another well-studied line of work views personalization as balancing a global reference model with a client-specific local model. Methods such as FedProx, FedDyn, and FedAvg with proximal terms penalize deviation from the global model, implicitly interpolating between purely local and purely global training [13]. More recent algorithms introduce explicit interpolation coefficients or learn them adaptively, for example via bi-level optimization or by learning client-specific mixing weights between global and local parameters [73,74]. CVPR 2024's FedSelect, for instance, allows each client to select and personalize only subsets of parameters that are most relevant to its distribution, while keeping the rest synchronized globally, yielding strong personalization without excessive communication overhead [73].

Meta-learning and good initializations.

Meta-learning-based PFL methods treat FL as learning a good initialization or optimizer such that each client can rapidly adapt to its own distribution with a few local updates. Early work such as Per-FedAvg adapts MAML-style meta-learning to the federated setting, while later methods build on Reptile, implicit MAML, and gradient-based meta-learners [75,76]. More recent work extends these ideas with better handling of domain shift and long-tailed distributions: for example, prototype-based views of federated domain shift [77] and few-shot meta-PFL for highly data-scarce clients [76]. These approaches are particularly appealing in edge environments where clients may be intermittently active and can afford only a small number of adaptation steps during their available windows.

Multi-task and cluster-based formulations.

Multi-task learning formulations explicitly model each client as its own task, coupled through a shared regularizer or task covariance structure. Optimization-based methods solve joint objectives where client-specific models share parameters through low-rank structures or graph-structured relationships [78]. Cluster-based PFL groups clients with similar data distributions into clusters, learns cluster-specific models, and allows further within-cluster personalization [19,79]. Recent PFL work in computer vision and graph learning (e.g., FCCL/FCCL+ and FedHEAL) leverages cross-correlation and instance similarity to learn generalizable shared representations while maintaining personalized heads, improving robustness under label skew and domain skew [80,81].

Heterogeneity-aware benchmarks and design principles.

A key trend since 2022 is the emergence of systematic benchmarks and taxonomies for heterogeneous and personalized FL, including the CSUR survey on heterogeneous FL and the TPAMI 2024 benchmark on generalization/robustness/fairness [67,70]. These works clarify that personalization is not merely an algorithmic add-on, but a design principle: in real edge systems, there is no single "right" global model; instead, we must design FL pipelines that admit a spectrum of client-specific solutions with controlled coupling and resource footprints.

3.2. System Heterogeneity and Stragglers

Beyond data heterogeneity, edge devices differ widely in hardware capabilities (CPU, GPU, memory), battery levels, connectivity (Wi-Fi, LTE/5G, wired), and availability patterns (when users are idle, plugged in, or connected). These differences lead to *system heterogeneity*, which manifests as stragglers (slow or temporarily offline clients), unbalanced local training speeds, and varying communication latencies [8,67]. In synchronous FL, where the server waits for a round's updates, stragglers can dominate end-to-end training time and reduce effective participation.

Recent surveys on asynchronous FL over heterogeneous devices frame three primary challenges: straggler mitigation, stale updates, and low resource utilization [82,83]. Techniques fall into three broad categories.

Client selection and pacing.

Client selection has become a central primitive: rather than sampling uniformly, the server selects clients based on availability, hardware capability, network conditions, or statistical properties. Modern selection schemes are heterogeneity-aware, aiming to balance system efficiency against fairness and representativeness. For example, strategies weigh clients by their remaining battery, expected upload time, or progress, or explicitly cap the contribution of very fast clients to avoid bias [83,84]. Recent works couple client selection with gradient compression and adaptive local epochs to maximize global progress under time or energy budgets [85,86]. At the edge, local orchestrators (e.g., base stations, gateways) can perform fine-grained scheduling within clusters before participating in higher-level FL rounds [19].

Asynchronous and semi-synchronous FL.

Asynchronous FL relaxes strict round-based coordination. Clients upload updates whenever they finish local training; the server integrates these updates immediately or in small batches, often using staleness-aware weighting. Surveys of asynchronous FL highlight buffering strategies (e.g., FedBuff), adaptive staleness decay, and hybrid asynchronous–synchronous protocols as key mechanisms [82]. TimelyFL (CVPRW 2023), for example, introduces heterogeneity-aware asynchronous training with adaptive partial training to handle variable availability in vision workloads [87]. More recent frameworks like HA-HEFL (2025) propose hybrid synchronous–asynchronous mechanisms that adaptively switch between modes depending on observed heterogeneity and cluster-level conditions, achieving better trade-offs between convergence speed and system efficiency [86].

Resource-aware scheduling and co-design.

System heterogeneity also motivates joint design of FL algorithms and resource management. Energy-aware client scheduling, bandwidth allocation, and co-location of FL workers with other workloads are all active topics. Surveys on heterogeneous FL and mobile/edge FL emphasize that real deployments must consider contention with other services, device sleep cycles, and network congestion [8,67]. Emerging frameworks incorporate predictive models of client availability and throughput, using them to decide when and how to involve each device, and even to adapt hyperparameters such as local epochs or learning rates on a per-client basis [83,86].

3.3. Communication Efficiency

Communication over wireless or backhaul links often dominates the cost of FL, especially in cross-device settings where millions of devices participate with limited uplink bandwidth. This has driven extensive research on communication-efficient FL in the last few years, spanning compression, update scheduling, structured updates, and physical-layer integration.

Compression: quantization, sparsification, and sketches.

A primary lever is to compress model updates before transmission. Classic approaches include low-bit quantization (e.g., 1–8 bits per parameter), sparsification (sending only the largest-magnitude gradients), and sketching (using random projections or count-sketches), all complemented by error-feedback mechanisms to control bias. Recent work continues to refine this space, proposing dynamic gradient compression and adaptive client selection to jointly optimize convergence and communication cost [85]. Surveys on communication-efficient FL catalog an increasingly rich toolbox of compression operators, error compensation strategies, and model partitioning techniques, many tailored to heterogeneous wireless conditions [88].

Update frequency and local computation.

Another simple but powerful knob is to reduce the *frequency* of communication by increasing the amount of local computation between global aggregations. Algorithms like FedAvg already perform several local epochs per round; subsequent work explores variable local epochs per client, local learning-rate schedules, and adaptive triggering of communication based on local progress or drift thresholds [8]. Hierarchical FL can be interpreted as a multi-timescale extension: frequent device–edge rounds and less frequent edge–cloud rounds effectively amortize communication across tiers [24,25]. Recent work in 6G-oriented FL proposes event-triggered communication schemes where clients transmit updates only when local gradients or model changes exceed a threshold, further reducing traffic in steady-state regimes [29,88].

Structured and partial updates.

Structured update schemes restrict communication to specific parts of the model or to low-rank subspaces. Examples include sending only updates for certain layers (e.g., personalized heads, adapters), low-rank factor updates, or updates in a compressed basis learned jointly with the model. Personalized FL methods such as FedSelect implicitly reduce communication by allowing each client to update only the parameters that matter for its distribution [73]. Other approaches maintain global and local parameter partitions, with only global partitions communicated each round [67]. These techniques are particularly important when models are large (e.g., transformers) and devices are bandwidth-constrained.

Over-the-air aggregation.

At the physical layer, *over-the-air federated learning* (OtA-FL) leverages the superposition property of wireless channels: devices transmit analog-encoded gradients simultaneously, and the base station directly observes a noisy sum, approximating an aggregated update with *constant* airtime regardless of the number of devices. Recent works refine OtA-FL with sparse one-bit quantization, layer-wise sparsification, and error-feedback, achieving communication-efficient aggregation while preserving convergence guarantees [89]. Fed-ZOE (2024) further proposes zeroth-order estimation schemes that enable gradient estimation under dimension-limited analog transmissions, addressing the challenge that the number of transmitted symbols in naive OtA-FL scales with the model dimensionality [90]. OtA-FL is particularly appealing in dense wireless edge environments (e.g., 6G cell-free massive MIMO), but requires specialized hardware and careful co-design between learning and communication.

3.4. Energy Efficiency and Green FL

Because many FL participants are battery-powered devices or energy-constrained edge servers, *energy efficiency* has become a first-class design objective. Energy costs arise from both computation (local training, inference) and communication (wireless transmissions, backhaul usage). In parallel, growing environmental concerns and sustainability goals have sparked interest in *green federated learning*, which explicitly targets reductions in carbon footprint and energy usage.

Recent surveys on energy-efficient FL in mobile and edge computing outline three major angles: learning-based strategies, resource allocation, and client selection [91]. A 2024 ACM survey on Green Federated Learning goes further, framing FL as a key enabler of green-aware AI for sustainable IoT, cataloging over one hundred works that reduce energy via co-design of models, protocols, and infrastructure [92,93].

Lightweight models and TinyFL.

On-device models must be matched to hardware capabilities. Techniques from model compression (pruning, quantization, knowledge distillation), efficient architectures (MobileNets, ShuffleNets, vision transformers with sparse attention), and TinyML are increasingly integrated with FL pipelines. Some works propose *TinyFL* frameworks that jointly optimize model size and FL update schedules, enabling resource-constrained microcontrollers to participate meaningfully in FL without exhausting their energy budgets [94]. For edge servers, the focus is on consolidating workloads, using low-precision accelerators, and turning off unused resources between rounds.

Energy-aware client selection and scheduling.

Energy-aware FL strategies select clients not only based on data and connectivity, but also based on their residual energy, charging state, and historical energy usage. The aforementioned energy-efficient FL survey reports strategies where energy models (estimating CPU and radio costs per update) are combined with scheduling policies to maximize global convergence under energy budgets [91]. Green FL frameworks propose multi-objective optimization formulations that trade off accuracy, latency, and energy, often solved via heuristic or reinforcement-learning-based schedulers [92]. In mobile edge scenarios, FL rounds may be scheduled preferentially when devices are charging or connected to Wi-Fi, reducing battery drain and cellular data usage.

Edge offloading and co-location with computation.

Offloading part of training to nearby edge servers can reduce energy consumption on devices while keeping raw data local. In hierarchical or split architectures, clients may run only a small front-end network, offloading heavy back-end computation to edge GPUs or TPUs [59,63]. Green FL frameworks also explore *where* to run aggregation: moving aggregators closer to data can reduce backhaul usage, while consolidating aggregation in energy-efficient data centers may reduce overall power consumption despite longer network hops [6,93]. Some works further tie FL scheduling to renewable energy availability in edge data centers, deferring non-urgent FL rounds to times when solar or wind power is abundant [95].

3.5. Privacy, Security, and Trust

Although FL avoids centralizing raw data, it is now well understood that model updates and gradients can leak sensitive information via *inference attacks*. A growing body of work demonstrates data reconstruction, membership inference, and property inference attacks against FL under realistic assumptions, especially when models are overparameterized or the attacker has auxiliary information [96,97]. These vulnerabilities are compounded by poisoning and Byzantine attacks, in which malicious clients send manipulated updates to influence the global model [70,98].

Recent surveys from 2023–2025 provide comprehensive taxonomies of FL privacy threats and defenses, emphasizing that privacy, security, and robustness must be addressed jointly rather than in isolation [96–99]. Defense mechanisms can be grouped as follows.

Inference attacks and defenses.

Inference attacks aim to infer properties of local data from observed updates or model states. Membership inference decides whether a particular sample was part of a client's training set; property inference aims to infer aggregate attributes (e.g., prevalence of a disease); and reconstruction attacks attempt to recover individual samples or images [96,99]. Defenses typically combine:

- *Differential privacy (DP)*, implemented via per-client clipping and additive noise, with careful privacy accounting (e.g., moments accountant, Rényi DP) to bound information leakage over many rounds [8,99];
- *Regularization and early stopping* to avoid overfitting local data, which correlates strongly with susceptibility to membership inference;
- *Representation-level defenses*, such as learning more invariant or obfuscated representations that are less invertible from gradients.

Recent works explore *adaptive DP*, where privacy budgets and noise scales are tuned dynamically based on client sensitivity or training phase, as in DynamicPFL with adaptive DP [96,100].

Poisoning attacks and robust aggregation.

Poisoning attacks involve malicious clients sending crafted updates that degrade the global model or embed backdoors. Surveys in 2024 categorize attacks into label-flip, backdoor, and model poisoning, and show that data poisoning is among the most common attack types in real FL deployments [97].

Robust aggregation methods—such as coordinate-wise median, trimmed mean, Krum, and more recent entropy- and Fisher-based aggregators—aim to suppress outlier or inconsistent updates while preserving benign variability. Recent works like SDEA (ICML 2024) and SDFC (ECCV 2024) propose new robust aggregation schemes tailored to heterogeneous and non-IID clients, achieving stronger backdoor resistance with limited accuracy loss [101]. Combined with anomaly detection at the server or edge (e.g., monitoring gradient norms, loss trajectories), these defenses form the first line of protection against poisoning.

Byzantine behavior, trust, and decentralization.

In more adversarial settings, some participants may behave arbitrarily (Byzantine clients), misreport their identity, or collude. Byzantine-resilient FL combines robust aggregation with trust and reputation mechanisms, often integrated with decentralized or blockchain-based FL when no single authority is fully trusted [54,55]. Smart contracts can encode aggregation rules and client incentives; immutable logs provide traceability of updates; and reputation scores can downweight or exclude persistently malicious nodes. However, as recent surveys emphasize, blockchain integration brings additional privacy challenges (e.g., on-chain update visibility) and overheads, requiring careful tuning for edge environments [23,55].

Secure aggregation and hardware roots of trust.

Secure aggregation protocols use cryptography (e.g., homomorphic encryption, secret sharing, multiparty computation) so that the server observes only an aggregated sum of client updates, not individual contributions [8,15]. This mitigates certain inference attacks and supports stronger threat models (honest-but-curious server). Recent work considers efficient secure aggregation tailored to bandwidth- and compute-constrained devices, and integration with DP and robust aggregation [96,99]. In parallel, trusted execution environments (TEEs) and hardware roots of trust (e.g., SGX, ARM TrustZone) are being explored to isolate aggregation and decryption logic at the server or edge, though they introduce their own attack surfaces and deployment challenges.

Auditing, accountability, and regulation.

Finally, the emerging regulatory landscape for privacy (e.g., GDPR, HIPAA, sectoral rules) and AI governance (e.g., EU AI Act) is beginning to influence FL design. Recent surveys discuss how FL deployments must support auditing (who contributed what, when), data subject rights (e.g., unlearning requests), and cross-border data-transfer constraints [96,99]. Auditing and accountability mechanisms at the edge—such as logging of model updates, configuration changes, and access decisions—provide observability and evidentiary trails for regulators and internal governance bodies.

Statistical heterogeneity, system heterogeneity, communication and energy constraints, and security/privacy concerns are deeply intertwined in real federated edge AI systems. Modern FL research increasingly acknowledges this by proposing methods that co-optimize across these dimensions—personalization with adaptive DP, heterogeneity-aware asynchronous protocols with energy constraints, communication-efficient and over-the-air aggregation under robust aggregation rules—rather than treating each challenge in isolation.

4. Hardware and Software Enablers

Architectural choices for federated edge AI are ultimately constrained by the underlying hardware and software stack. On the hardware side, we have seen a rapid evolution from CPU-only embedded devices to heterogeneous system-on-chips (SoCs) integrating neural processing units (NPUs), digital signal processors (DSPs), and domain-specific accelerators, as well as neuromorphic and processing-in-memory prototypes [102–104]. On the software side, a rich ecosystem of TinyML toolchains, edge runtimes, networking stacks, and federated learning (FL) frameworks has emerged, spanning from simulation-oriented research platforms to industrial-grade orchestration frameworks for cross-silo deployments [8,105,106].

This section surveys hardware and software enablers along four critical axes: (i) edge accelerators, which determine what kinds of models can be trained and inferred at the edge; (ii) TinyML and microcontroller platforms, which push FL to the extreme edge; (iii) networking and orchestration mechanisms, which shape where and when training occurs; and (iv) the FL and edge-AI frameworks that provide abstractions for building and deploying such systems.

4.1. Edge Hardware Accelerators

Modern edge platforms—from smartphones and wearables to industrial gateways and roadside units—are increasingly built around *heterogeneous* SoCs that integrate general-purpose CPU cores with specialized accelerators for deep learning and signal processing. Surveys of deep learning accelerator design and edge AI processors categorize these platforms into several architectural families: GPU-based accelerators, dedicated NPUs and tensor cores, FPGA-based overlays, application-specific integrated circuits (ASICs) with dataflow architectures, and emerging neuromorphic and processing-in-memory (PIM) designs [102–104].

NPUs and DSPs in mobile and embedded SoCs.

In mobile and embedded SoCs, NPUs and DSPs now handle the bulk of deep-learning inference. Examples include smartphone neural engines, embedded tensor accelerators in application processors, low-power NPUs in microcontrollers, and DSP clusters in industrial and automotive SoCs. These accelerators are optimized for:

- *Mixed-precision arithmetic* (e.g., INT8, INT4, FP16, bfloat16), enabling higher throughput and lower energy per operation compared to FP32;
- *Sparsity exploitation*, where zero-valued activations or weights are skipped or compressed to save bandwidth and compute;
- *On-chip memory hierarchies* that minimize off-chip DRAM access, which is often the dominant energy cost in edge devices.

Recent benchmarking studies demonstrate how CPU+NPU configurations can deliver orders-of-magnitude higher inferences-per-watt than CPU-only or even CPU+GPU setups on representative computer-vision and audio workloads [107–109]. For federated learning, these accelerators primarily enable:

- efficient *local training* of compact models (e.g., MobileNets, small transformers) on-device, provided that training kernels (backpropagation, optimizers) are supported by vendor libraries;
- low-latency, energy-efficient *inference* for continuously adapted models, enabling on-device personalization loops between FL rounds;
- support for *mixed-precision FL*, where local updates are computed and communicated in reduced precision, aligning with communication-efficient FL schemes.

Edge GPUs and ASICs in edge servers and gateways.

Edge servers and gateways in MEC sites, factories, or hospitals often host one or more GPU cards or application-specific accelerators. Compared to NPUs in end devices, these accelerators:

- sustain larger batch sizes and more complex backbones (e.g., ResNeXt, ViTs, LLM decoders), enabling heavy training tasks at the edge;
- provide richer programming models (CUDA, ROCm, vendor SDKs), easing integration with mainstream ML frameworks (PyTorch, TensorFlow, JAX);
- can be shared across multiple FL tasks via containerization and multi-tenant scheduling (e.g., Kubernetes), making them natural aggregation and training hubs for hierarchical FL.

Recent surveys on edge AI for 6G networks emphasize that such edge compute nodes are key for pushing large-model adaptation, fine-tuning, and distillation closer to data sources, while still offloading the heaviest training phases to core clouds when needed [110–112].

Neuromorphic and processing-in-memory accelerators.

Neuromorphic chips and PIM architectures aim to collapse computation and storage into the same physical substrate (e.g., analog crossbar arrays or in-memory bit-serial operations), achieving ultra-low-power inference for always-on sensing tasks. While still largely in the research stage, recent prototypes demonstrate multi-milliwatt inference for event-based vision, keyword spotting, and anomaly detection, with emerging support for on-chip local learning rules or constrained gradient-based updates [104]. For federated edge AI, such devices suggest a future in which:

- extremely low-power nodes (sensors, wearables) learn local representations or features with ultra-compact online models;
- higher-tier devices (gateways, edge servers) perform heavier FL-style aggregation and meta-learning, using neuromorphic nodes as intelligent pre-processors;
- specialized FL algorithms exploit the analog nature and limited precision of neuromorphic updates, potentially combining them with error-correcting schemes at higher tiers.

Overall, hardware accelerators support core enabling techniques for federated edge AI: mixed-precision training, structured sparsity, model compression (pruning, quantization), and partitioned models in split/hybrid architectures [102,103]. At the same time, they introduce new constraints—fixed memory sizes, restricted operator sets, vendor-specific toolchains—that FL algorithms and frameworks must accommodate.

4.2. TinyML and Microcontroller Platforms

TinyML refers to deploying machine learning models on microcontrollers and other extremely resource-constrained devices, often with tens of kilobytes of RAM and milliwatt-level power budgets [113]. TinyML techniques encompass compact architectures, aggressive quantization, and inference engines that can run without dynamic memory allocation. Recent surveys highlight that TinyML is rapidly expanding from static inference to more adaptive and collaborative paradigms, including FL [113–115].

TinyML runtimes and toolchains.

TinyML ecosystems revolve around specialized toolchains and runtimes:

- **Runtimes and libraries** such as TensorFlow Lite for Microcontrollers (TFLM), CMSIS-NN, microTVM, and vendor-specific SDKs compile models into integer-only kernels that fit in microcontroller flash and RAM.
- **Toolchains** (e.g., Edge Impulse Studio, TinyML IDEs) provide end-to-end pipelines for data collection, model design, quantization, and deployment to MCUs.
- **Benchmarks** and datasets tailored to tiny devices (keyword spotting, motion detection, anomaly detection) provide realistic targets for evaluating both models and hardware [107,113].

These ecosystems have started to incorporate capabilities for *incremental updates* and, in some cases, limited forms of on-device training.

Federated learning on TinyML devices.

The convergence of TinyML and FL—sometimes called TinyFL—has become an active research topic. A 2025 survey on federated learning for TinyML identifies key challenges: memory and compute limits, energy constraints, intermittent connectivity, and the need for extremely compact update representations [114]. Several enabling techniques have emerged:

- **Model and update compression.** TinyFL deployments rely on sub-100k-parameter models, aggressive weight quantization (e.g., INT4), and heavily compressed updates (e.g., sparse or binary masks) to keep communication and storage costs manageable.
- **Traffic shaping and duty cycling.** Communication is often scheduled infrequently and piggy-backed on existing control traffic; nodes may participate in FL rounds only when energy budgets allow.
- **FL-specific TinyML runtimes.** Frameworks such as TinyFedTL implement federated transfer learning on microcontrollers, combining pre-trained small backbones with efficient on-MCU fine-tuning and compressed update protocols [116,117]. TinyReptile applies federated meta-learning (Reptile-style) to TinyML models, enabling rapid personalization with minimal communication [118].

These works show that collaborative learning is feasible even on microcontrollers, albeit with heavy constraints on model size and update frequency. TinyFL is particularly relevant for large fleets of simple sensors (industrial monitoring, smart agriculture, environmental sensing), where centralized data collection is infeasible or undesirable [114,117].

On-device adaptation and intermittent FL.

TinyML platforms also enable continuous *local* adaptation between FL rounds. For example, a keyword spotting model may receive occasional FL updates when connectivity permits but adapt its thresholds or a tiny head layer locally based on recent observations. This suggests hybrid schemes in which:

- global TinyFL updates provide coarse-grained, fleet-level improvements;
- per-device online learning runs continuously at ultra-low power;
- intermittent connectivity and energy constraints are treated as first-class inputs to FL scheduling and model design.

4.3. Networking and Cloud–Edge Orchestration

The efficacy of federated edge AI hinges on the underlying networking and orchestration substrate. 5G/6G cellular networks, Wi-Fi 6/7, low-power wide-area networks (LPWANs), and wired industrial networks provide heterogeneous communication links with widely varying bandwidth, latency, reliability, and cost. Multi-access edge computing (MEC), software-defined networking (SDN), and network function virtualization (NFV) provide mechanisms to place compute and control logic flexibly across this fabric [110–112].

5G/6G and MEC as FL substrates.

In 5G and emerging 6G networks, edge computing platforms are co-located with base stations or aggregation points, forming natural anchors for hierarchical FL. Surveys on FL for 6G IIoT and edge-native intelligence argue that:

- *network slicing* can provision logical networks with different latency and bandwidth guarantees for FL traffic versus user data;
- *radio resource management* and *scheduler design* can explicitly account for FL rounds, prioritizing FL updates when needed and shaping the participation set;
- *MEC platforms* (e.g., containerized edge clusters) can host FL aggregators, model hubs, and analytics services close to the data sources [112,119].

Over-the-air aggregation schemes, discussed earlier, rely on specific physical-layer capabilities (e.g., analog superposition, synchronized transmissions) that are being considered for 6G cell-free massive MIMO systems [89,90]. This tight coupling between communication and computation is emblematic of edge-native FL.

Cloud–edge orchestration and placement.

Beyond radio-level concerns, orchestrating FL across device, edge, and cloud tiers requires:

- **Workload placement.** Deciding where each part of the FL pipeline runs—simulation, training, aggregation, evaluation, and serving—across containers, VMs, and nodes in a Kubernetes-style cluster;
- **Topology management.** Mapping FL logical topologies (centralized, hierarchical, clustered, decentralized) onto physical routing graphs, with attention to latency, bandwidth, and resilience [6];
- **Lifecycle management.** Handling versioning, model rollout, and rollback across thousands of devices and multiple sites, including safe co-existence of multiple FL experiments.

Recent frameworks for edge-cloud collaborative learning demonstrate bi-directional knowledge transfer between edge FL models and centralized models, using logits and feature embeddings as communication primitives. This allows heavy training or fine-tuning to occur in the cloud while edge nodes specialize and personalize models locally [62,64,120].

Integration with OT networks and protocols.

In industrial and critical-infrastructure domains, FL must integrate with operational technology (OT) networks using protocols like OPC UA, field buses, and time-sensitive networking. Here, orchestration must respect:

- *deterministic timing* and safety certification requirements;
- *air-gapped or constrained connectivity*, where backhaul to the public internet is limited or non-existent;
- *co-existence with control loops*, ensuring that FL traffic does not interfere with real-time control messages.

Edge-only FL and multi-site private federations build on these OT infrastructures and often rely on private WANs or dedicated backbones rather than the public internet [18,31,35].

4.4. Frameworks for FL and Edge AI

The hardware and networking capabilities described above are made accessible to developers through a growing ecosystem of FL and edge-AI frameworks. These frameworks provide APIs, runtime environments, and orchestration tools that hide much of the complexity of client coordination, aggregation, compression, and deployment. In the last few years, a number of comparative studies and surveys have evaluated these frameworks in terms of scalability, ease of use, privacy features, and edge suitability [105,106,121].

Broadly, we can distinguish between:

- **Research-oriented FL frameworks** such as TensorFlow Federated (TFF), PySyft/Syft, and various simulators, which emphasize flexibility in expressing new algorithms and experimenting in controlled environments;
- **Production-oriented FL platforms** such as FedML, OpenFL, Substra, FATE, and FLUTE, which target cross-silo deployments in healthcare, finance, and industrial applications;
- **Edge-friendly FL frameworks** such as Flower, FedML, and TinyFL/TinyFedTL, which explicitly address heterogeneous devices, mobile platforms, and TinyML targets;
- **On-device ML runtimes** such as TFLite, microTVM, and vendor TinyML SDKs, which focus on inference (and limited training) on devices and must be integrated with FL stacks.

TensorFlow Federated and research-centric frameworks.

TFF provides a functional programming environment for federated computations tightly integrated with TensorFlow. Its Federated Core API allows researchers to express custom FL algorithms, specify where (server or clients) each computation runs, and simulate large federations on a single machine or cluster [122]. PySyft/Syft focuses on privacy-enhancing technologies, offering abstractions for secure multiparty computation, homomorphic encryption, and differential privacy in a PyTorch-centric environment [123]. These frameworks are invaluable for experimenting with new algorithms and threat models, but they are not meant to be production orchestrators in heterogeneous edge networks.

Production-grade and cross-silo frameworks.

FedML, FATE, Substra, and OpenFL provide end-to-end stacks for cross-silo FL:

- **FedML** offers a unified edge–cloud library and MLOps stack for FL and distributed training, supporting smartphones, edge servers, and cross-cloud GPU clusters. It provides experiment management, monitoring, and deployment tooling, and is used in both academic and industrial contexts [124,125].
- **FATE** (Federated AI Technology Enabler) is an industrial-grade platform focusing on secure cross-silo FL with strong support for homomorphic encryption and multiparty computation. It targets financial and enterprise consortia and has evolved into a rich ecosystem including KubeFATE, FATE-Flow, and FATE-LLM for LLM-centric federations [126,127].
- **Substra** provides a framework for privacy-preserving, traceable collaborative ML, heavily used in healthcare. It uses distributed ledger technology for traceability, supports a variety of backends, and is deployed in projects like MELLODDY and HealthChain [128–130].
- **OpenFL** (Open Federated Learning), originally developed by Intel, focuses on cross-institutional FL for healthcare and other domains. Recent releases support confidential computing via SGX and federated fine-tuning of large language models [131–133].

Edge-centric and cross-device frameworks.

Flower, FedML, and similar frameworks pay particular attention to heterogeneous clients and edge deployment:

- **Flower** is a framework-agnostic FL library supporting PyTorch, TensorFlow, JAX, and even raw NumPy, with a strong emphasis on extensibility and language-agnostic clients. Comparative evaluations show that Flower scales to millions of simulated clients and offers flexible strategies for client selection, aggregation, and compression [106,121,134].
- **FedML** explicitly targets "anywhere, any scale" deployments, including cross-device training on smartphones and IoT devices, and cross-silo deployments in multi-cloud environments. It integrates with MLOps workflows for experiment tracking and deployment [124,135].
- **TinyFL / TinyFedTL and TinyML runtimes** embed FL primitives into TinyML toolchains, enabling collaborative learning across microcontrollers. TinyFedTL demonstrates federated transfer learning and privacy-preserving TinyML on MCUs, integrating with TinyML hardware and runtimes [114,116,117].

Extended comparison of FL and edge-AI frameworks.

Table **??** provides an extended comparison of representative frameworks across axes that matter for federated edge AI: stack and language, FL scope and topology, privacy features, and edge suitability. It is not exhaustive but reflects trends observed in recent framework comparisons and benchmarks [105,106,121,136].

Table 9. Example FL and edge AI frameworks and runtimes (stack, scope, and edge suitability). FL scope distinguishes cross-device (CD), cross-silo (CS), and simulation (Sim).

Framework / runtime	**Primary stack / language**	**FL scope and topology**	**Edge and TinyML suitability**
TensorFlow Federated (TFF)	Python, TensorFlow	Research / simulation (Sim); primarily centralized and cross-device abstractions; custom topologies implementable via Federated Core	Good for simulating large federations; limited direct support for production edge deployment; relies on separate runtimes for on-device inference
PySyft / Syft	Python, PyTorch-centric	Research / Sim; central or decentralized graphs; can model cross-silo and CD	Primarily research; edge deployment possible but requires manual integration with device runtimes; more focused on privacy pipelines than system orchestration
Flower (flwr)	Python; framework-agnostic (PyTorch, TF, JAX, NumPy)	CD, CS, Sim; centralized and hierarchical topologies via strategies; supports millions of simulated clients	Designed for heterogeneous clients; lightweight client SDKs; supports deployment on mobile, edge devices, containers, and cloud VMs; strong simulator support
FedML	Python; supports PyTorch, TF, etc.	CD, CS, Sim; hierarchical (device–edge–cloud) and cross-cloud FL; also distributed training (non-FL)	Explicit support for smartphones, IoT, edge servers, and multi-cloud; client SDKs for mobile; MLOps stack for orchestration and monitoring
Open Federated Learning (OpenFL)	Python; framework-agnostic (TF, PyTorch)	CS, Sim; primarily cross-institution FL; centralized or hub-and-spoke	Targeted at institutional edge/cloud nodes (hospitals, research centers); not aimed at tiny devices; works with containerized workloads
Substra	Python; backend-agnostic; web UI and CLI	CS; privacy-preserving orchestration across institutions; DAGs of computations over FL or other workflows	Deployed in hospital and biotech settings; assumes Kubernetes clusters at each site; not aimed at microcontrollers but suitable for hospital edge servers
FATE (Federated AI Technology Enabler)	Python/Java; multiple backends	CS; industrial-grade cross-silo FL; vertical, horizontal, and transfer FL; supports complex multi-party topologies	Targets enterprise data centers; not designed for tiny devices; can be integrated with edge gateways that speak FATE protocols
FLUTE (Microsoft) and similar simulators	Python, C++; tightly coupled to PyTorch / ONNX	Sim, CS; highly scalable simulation of thousands of clients for research and benchmarking	Primarily for offline experiments; used to generate insights before deployment in production frameworks
On-device ML runtimes (TFLite, TFLite Micro, microTVM, vendor TinyML SDKs)	C/C++, Python bindings; MCU and embedded-focused	Not FL frameworks by themselves; support local training in limited cases; FL requires external coordination layer	Ideal for TinyML inference; some support limited on-device training or fine-tuning; can serve as client runtimes for TinyFL frameworks
TinyFL / TinyFedTL and related TinyML-FL libraries	C/C++/Python hybrids; tightly coupled with MCU runtimes	CD (on MCUs); typically star or small-tree topologies with simple coordinators	Designed specifically for MCUs with tens of kB RAM; extreme focus on communication, energy, and on-device training cost
Other FL frameworks (e.g., LEAF, FedScale, Fed-BioMed, etc.)	Python and mixed stacks	Primarily benchmarking, domain-specific FL (e.g., medical, mobile workloads), or simulators	Edge suitability depends on integration with runtimes like Flower/FedML; often used as data/benchmarking layers rather than deployment frameworks

Table 10. Example FL and edge AI frameworks and runtimes (privacy features and representative uses). Privacy features refer to built-in support for secure aggregation (SA), differential privacy (DP), and advanced cryptography (HE/MPC).

Framework / runtime	Built-in privacy / security features	Representative uses / notes
TensorFlow Federated (TFF)	DP primitives and secure aggregation examples; integrates with TF privacy; cryptographic extensions via external libs	Research on new FL algorithms, personalization, and DP; prototyping cross-device FL for mobile use cases [122]
PySyft / Syft	Strong focus on HE, MPC, and DP; advanced privacy-preserving computation backends	Prototype secure FL workflows with complex threat models; education and experimentation with privacy techniques [123]
Flower (flwr)	Supports secure aggregation and DP via strategy extensions; integration with external crypto libs; pluggable aggregation strategies	End-to-end FL from research to production; large-scale simulation studies; benchmarking FL algorithms; widely used in academia and industry [106,134]
FedML	DP and secure aggregation support; integration with MLOps; role-based access control; options for HE via plugins	Production AI platform for FL at scale; used for smartphone FL, cross-silo FL, and multi-cloud deployments [124,125,135]
Open Federated Learning (OpenFL)	Focus on confidential computing using SGX; secure aggregation and encrypted channels; support for confidential FL workflows	Healthcare imaging consortia (FeTS), cross-hospital FL, confidential FL for LLM fine-tuning and evaluation [131–133]
Substra	Ledger-based traceability and audit; strong permissioning; supports DP and secure orchestration; can integrate with HE/MPC tools	Healthcare research consortia (MELLODDY, HealthChain); traceable, compliant FL with strong governance and audit trails [128–130]
FATE (Federated AI Technology Enabler)	Rich HE/MPC support; secure protocols for vertical FL; DP, access control, auditing; production-grade security	Financial services, cross-silo enterprise FL; open ecosystem (KubeFATE, FATE-Flow, FATE-LLM) for secure, large-scale FL [126, 127]
FLUTE (Microsoft) and similar simulators	Supports DP, secure aggregation, and realistic network models in simulation settings	Benchmarking FL algorithms at scale; evaluation of communication and robustness strategies in controlled environments [105]
On-device ML runtimes (TFLite, TFLite Micro, microTVM, vendor TinyML SDKs)	Some DP and encryption primitives at app level; no built-in FL security; rely on host systems for secure aggregation	Keyword spotting, anomaly detection, vibration monitoring, and other TinyML tasks; integrated into TinyFL/TinyFedTL-style systems for collaborative training [113,114]
TinyFL / TinyFedTL and related TinyML-FL libraries	Lightweight security (e.g., symmetric encryption, basic DP); constrained by MCU resources; some works explore HE at gateway nodes	First FL implementations targeting microcontrollers; federated transfer learning and meta-learning for TinyML tasks in IoT scenarios [114,116,117]
Other FL frameworks (e.g., LEAF, FedScale, Fed-BioMed, etc.)	Varying; often focus on data/benchmarking rather than cryptography; can integrate with DP and secure aggregation libs	Benchmarks and specialized domains (e.g., mobile workloads, biomedical data); complement general-purpose FL frameworks [105, 106]

Taken together, hardware accelerators, TinyML platforms, network substrates, and FL frameworks form a *stack* that determines what is practically achievable in federated edge AI. Designing real systems requires co-optimizing across these layers: selecting accelerators that can support the desired training/inference workloads, choosing FL frameworks that match deployment and governance needs, and orchestrating FL protocols to respect the constraints and opportunities of 5G/6G networks, industrial OT infrastructures, and TinyML devices.

5. Applications and Case Studies

Federated learning at the edge is not an abstract architectural exercise: it is being piloted and, in some cases, deployed at scale in several safety-critical and data-sensitive verticals, notably healthcare, transportation, manufacturing, and smart cities. These domains share three properties that make federated edge AI attractive: (i) data are naturally distributed across many organizations or devices; (ii) centralizing raw data is either infeasible (due to bandwidth, ownership, or governance) or illegal (due to privacy regulation); and (iii) decisions often need to be taken close to the data source, under tight latency and availability constraints. Recent surveys in healthcare, intelligent transportation, and industrial IoT document a rapid increase in FL pilots and proofs-of-concept over roughly the last four years [137–143].

Table **??** summarizes representative application patterns across four major domains and connects them to the architectural choices discussed in Section 2.

Table 11. Representative application patterns for federated edge AI (workload and data characteristics). "Arch. pattern" refers to the dominant architecture in early case studies: centralized (C), hierarchical (H), edge-only (E), decentralized (D), or split/hybrid (S).

Domain	Typical clients / tiers	Main data modalities	Representative tasks
Healthcare & wearables	Hospitals, imaging centers, clinics, patient portals, wearables, home gateways	Imaging (MRI, CT, X-ray, ultrasound), ECG/PPG, EHR tables, clinical text, wearable time series	Lesion / tumor detection, segmentation, outcome prediction, arrhythmia detection, remote monitoring, risk scoring, triage, readmission prediction
Intelligent transportation	Vehicles (CAVs), roadside units (RSUs), traffic cameras, edge servers at intersections, traffic-control centers	Camera and LiDAR streams, radar, GPS trajectories, CAN bus signals, traffic counts, map/HD map updates	Collaborative perception, traffic-flow prediction, trajectory prediction, driver behavior modeling, traffic signal control, high-precision positioning
Industrial IoT & smart manufacturing	Machines, robots, PLCs, gateways, shop-floor edge servers, plant data centers, corporate clouds	Vibration and acoustic signals, PLC logs, machine status, quality inspection images, sensor arrays, MES/ERP logs	Predictive maintenance, quality inspection, anomaly detection, process optimization, demand forecasting, digital-twin synchronization
Smart cities & infrastructure	Cameras, traffic sensors, smartphones, smart meters, building controllers, lamppost gateways, city/utility data centers	Video, mobility traces, smart-meter readings, environmental sensors (air quality, noise), grid telemetry, social media signals	Mobility and crowd analytics, anomaly and incident detection, smart-grid forecasting, adaptive lighting and HVAC control, environmental monitoring

Table 12. Representative application patterns for federated edge AI (architectural patterns and example works). "Arch. pattern" indicates typical architectures: centralized (C), hierarchical (H), edge-only (E), decentralized (D), split/hybrid (S).

Domain	Arch. pattern (typical)	Example works
Healthcare & wearables	Cross-silo FL (C) for hospitals; cross-device FL (C/H) for wearables; edge-only FL (E) in hospital networks; emerging split / hybrid (S) with cloud-hosted backbones	Surveys of FL in healthcare and smart healthcare [137,138,144]; cardiology- and arrhythmia-focused FL [145–147]; wearable-focused and IoT health FL [148,149]
Intelligent transportation	Hierarchical FL (H) with vehicles–RSUs–cloud; decentralized / gossip (D) among vehicles; edge-only (E) inside roadside clusters	ITS-focused FL surveys [139,150]; cooperative perception frameworks [151,152]; vehicular positioning and edge computing [153,154]
Industrial IoT & smart manufacturing	Edge-only FL (E) within plants; hierarchical FL (H) across lines/plants; multi-site cross-silo FL (C); early decentralized pilots (D) for collaborative robotics	Surveys on FL+IIoT and smart manufacturing [141,155,156]; predictive maintenance and quality inspection case studies [157–159]; cross-plant and product-lifecycle FL [160]
Smart cities & infrastructure	Hierarchical FL (H) (device–edge–city DC); edge-only FL (E) within utilities; cross-silo FL (C) across agencies; blockchain-enabled FL (D) in multi-stakeholder settings	Surveys on FL for smart cities and privacy [142,143]; FL for smart grids and energy systems [161,162]; smart-city infrastructure and edge-AI FL case studies [163–165]

Table 13. Representative application patterns for federated edge AI (architectural patterns and example works). "Arch. pattern" indicates typical architectures: centralized (C), hierarchical (H), edge-only (E), decentralized (D), split/hybrid (S).

Domain	Arch. pattern (typical)	Example works
Healthcare & wearables	Cross-silo FL (C) for hospitals; cross-device FL (C/H) for wearables; edge-only FL (E) in hospital networks; emerging split / hybrid (S) with cloud-hosted backbones	Surveys of FL in healthcare and smart healthcare [137,138,144]; cardiology- and arrhythmia-focused FL [145–147]; wearable-focused and IoT health FL [148,149]
Intelligent transportation	Hierarchical FL (H) with vehicles–RSUs–cloud; decentralized / gossip (D) among vehicles; edge-only (E) inside roadside clusters	ITS-focused FL surveys [139,150]; cooperative perception frameworks [151,152]; vehicular positioning and edge computing [153,154]
Industrial IoT & smart manufacturing	Edge-only FL (E) within plants; hierarchical FL (H) across lines/plants; multi-site cross-silo FL (C); early decentralized pilots (D) for collaborative robotics	Surveys on FL+IIoT and smart manufacturing [141,155,156]; predictive maintenance and quality inspection case studies [157–159]; cross-plant and product-lifecycle FL [160]
Smart cities & infrastructure	Hierarchical FL (H) (device–edge–city DC); edge-only FL (E) within utilities; cross-silo FL (C) across agencies; blockchain-enabled FL (D) in multi-stakeholder settings	Surveys on FL for smart cities and privacy [142,143]; FL for smart grids and energy systems [161,162]; smart-city infrastructure and edge-AI FL case studies [163–165]

5.1. Healthcare and Wearables

Healthcare has been one of the earliest and most active verticals to explore federated learning, driven by strong privacy regulations (HIPAA, GDPR), ethical considerations, and the intrinsic fragmentation of medical data across institutions and devices. Recent systematic reviews identify hundreds of FL applications in medical imaging, electronic health records (EHRs), and physiological monitoring, with a marked acceleration of publications after 2020 [137,138,144]. These works consistently highlight that clinical data are distributed across hospitals, imaging centers, primary care clinics, patient portals, and increasingly, consumer wearables and home sensors, making FL a natural fit.

Cross-institutional diagnostic models.

In cross-silo healthcare FL, the clients are typically institutions: hospitals, radiology groups, or research networks. Each holds large local datasets but is constrained from centralizing data due to regulation, governance, or patient trust. FL enables joint training of diagnostic models for tasks such as tumor and lesion detection, organ segmentation, prognosis prediction, and treatment response modeling by exchanging model updates rather than raw data [137,138,166]. Applications span radiology (brain tumor segmentation, lung nodule detection), histopathology (WSI classification), ophthalmology (retinopathy grading), and dermatology (skin lesion classification). Multi-national consortia have used FL to train models over data from dozens of hospitals, demonstrating improved generalizability across demographics and scanner vendors compared to single-institution training [144,167].

Architecturally, these deployments often implement centralized or hierarchical FL: a central coordinator (e.g., a university or vendor) aggregates models from participating sites, sometimes with intermediate hubs for regional networks. Frameworks such as FATE, Substra, and OpenFL (Section 4.4) are frequently used to orchestrate these collaborations, providing secure aggregation, audit trails, and containerized execution [126,168,169]. Privacy-enhancing technologies (differential privacy, homomorphic encryption, secure aggregation) are integrated to address residual risks of gradient leakage and membership inference [138,170].

Cardiology and arrhythmia detection across hospitals and wearables.

Cardiology has become a flagship application area for FL, particularly for arrhythmia detection and heart disease prediction from ECG and wearable signals. Recent surveys and domain-specific reviews catalog FL applications for arrhythmia classification, heart failure risk prediction, and early detection of atrial fibrillation [147,171]. For example, FL-based frameworks have been proposed in which hospitals collaboratively train deep ECG classifiers or temporal convolutional networks while keeping 12-lead ECGs on-premise, using secure multiparty computation and DP to further protect patient identity [145,172]. Other works integrate explainable AI (XAI) techniques into FL-based arrhythmia detectors, providing local feature-attribution maps for clinicians while coordinating training across sites [146,173].

At the same time, wearable devices (smartwatches, patches, chest straps) are increasingly used for continuous monitoring of heart rhythm and physical activity. Federated learning enables collaborative training of lightweight models for arrhythmia detection, sleep staging, and activity recognition across millions of devices, without uploading raw ECG or PPG traces to the cloud [144,149]. Case studies demonstrate that cross-device FL can adapt models to diverse populations (age, fitness levels, comorbidities) and device hardware (sampling rates, sensor quality) while keeping communication and energy costs manageable via on-device compression and partial participation [174,175].

Remote monitoring, telemedicine, and home health.

Beyond cardiology, FL and edge AI power remote monitoring systems where patients are equipped with home sensors (glucometers, blood pressure cuffs, weight scales) and telemedicine apps. Federated models learn to predict exacerbations (e.g., COPD, heart failure), detect medication non-adherence, or recommend interventions, using a mixture of structured time-series data, self-reported questionnaires, and wearable signals [148,166]. Edge devices (home hubs, set-top boxes, or even the wearable itself) run local inference to trigger alerts or recommendations even when connectivity to the cloud is limited, while FL updates are exchanged asynchronously when network and energy conditions permit. Some recent works integrate FL with personalized treatment policies via federated reinforcement learning, enabling adaptive titration of therapies under clinical oversight [176,177].

Governance, regulation, and future directions.

Deploying FL in healthcare requires not only algorithmic innovation but also careful governance: defining data-use agreements, auditing model updates, handling patient consent and unlearning, and aligning with evolving regulatory guidance on AI in medicine [138,170]. Looking forward, there is growing interest in combining FL with large medical foundation models (e.g., multimodal encoders pre-trained on public data) and using FL for continual adaptation of those backbones to local practice patterns. Wearable devices are expected to play a larger role as low-latency personalization endpoints in such hybrid architectures, especially for chronic-disease management and digital therapeutics [149,178].

5.2. Intelligent Transportation

Intelligent transportation systems (ITS) encompass connected and autonomous vehicles (CAVs), roadside infrastructure, traffic management centers, and cloud services. These systems generate massive volumes of sensor data—video, LiDAR, radar, GPS traces, CAN bus logs—that are highly sensitive (e.g., revealing driving behavior and location) and costly to transmit. Federated learning offers a way to collaboratively improve perception, prediction, and control models across vehicles and infrastructure while keeping raw data local [139,150].

A recent survey on FL in ITS highlights three main application categories: traffic-flow prediction, traffic target recognition (perception), and vehicular edge computing, each with its own communication and heterogeneity challenges [139]. Another state-of-the-art review on FL for vehicular communications emphasizes cooperative positioning, resource management, and security as key use cases [150].

Collaborative perception and prediction across vehicles.

Cooperative perception aims to extend each vehicle's field of view and robustness by sharing information with nearby vehicles and roadside sensors. Centralized solutions that upload raw sensor data to the cloud are infeasible due to bandwidth and privacy constraints. FL-based approaches instead train perception models (e.g., object detectors, semantic segmenters, occupancy networks) collaboratively across vehicles and roadside units (RSUs), exchanging only model updates [151,152]. For example, FL frameworks have been proposed where vehicles periodically upload gradients from LiDAR and camera-based detectors to edge servers at intersections, which aggregate and refine a shared model, then broadcast updates back to participating vehicles [151,179].

These systems often adopt hierarchical FL architectures: vehicles $\rightarrow$ RSUs $\rightarrow$ regional traffic-control centers or cloud. This aligns naturally with vehicular-network topologies and allows local adaptation at the RSU level (e.g., per-intersection or per-corridor models) while still maintaining a globally shared backbone [139]. FL also supports cooperative trajectory prediction, where vehicles jointly learn to forecast trajectories of surrounding agents under varying road and weather conditions [180].

Traffic-flow prediction and adaptive traffic control.

Traffic-flow prediction models historically rely on centralized processing of sensor data from loop detectors, cameras, and probe vehicles. FL enables distributed training of graph neural networks or sequence models on data held by different municipalities, highway segments, or private mobility companies, improving prediction accuracy while preserving data ownership [139,181]. FL-based traffic-signal control algorithms use decentralized reinforcement learning, with each intersection acting as a client that learns local policies while contributing to a shared global model [182,183]. Such systems are especially attractive when different regions are operated by distinct agencies or when private fleets (e.g., ride-sharing companies) wish to collaborate without sharing raw trip data.

Vehicular edge computing and resource management.

In vehicular edge computing, in-vehicle and roadside compute resources host both application and FL workloads. FL is used not only for perception/prediction but also for resource allocation, scheduling, and caching policies [150,153]. For example, FL has been applied to learn cooperative positioning models that fuse GNSS and local sensor data across vehicles, improving accuracy without sharing raw trajectories [153]. Other works use FL to optimize offloading decisions (which tasks to process locally vs. at RSUs or the cloud) under latency and energy constraints [184,185].

Challenges and outlook.

ITS FL deployments must cope with extreme heterogeneity: vehicle hardware, sensor configurations, driving styles, and connectivity conditions vary widely. Non-IID data and client churn are the norm. Surveys highlight open challenges in ensuring robustness to adversarial vehicles, balancing personalization versus global generalization across regions, and integrating FL with safety and certification processes for CAV software [139,150]. Cooperative-perception FL must also address tight latency budgets and high-dimensional sensor inputs, motivating compression, partial model updates, and carefully scheduled communication. As 5G/6G and edge computing infrastructure mature, FL is likely to become a core building block for CAV stacks and ITS control centers.

5.3. Industrial IoT and Smart Manufacturing

Industrial IoT (IIoT) and smart manufacturing environments feature dense deployments of sensors, actuators, programmable logic controllers (PLCs), robots, and edge servers interconnected through OT and IT networks. Data streams include vibration and acoustic signals, high-speed sensor arrays, machine logs, quality inspection images, and enterprise resource planning (ERP) data. These data are highly sensitive, both from an intellectual-property and safety perspective, and often cannot leave plant boundaries. FL offers a way to collaboratively train models for predictive maintenance, quality inspection, anomaly detection, and process optimization while respecting these constraints [140,155,156].

Predictive maintenance and quality inspection.

Predictive maintenance (PdM) is a canonical IIoT application: vibration or acoustic signals from rotating equipment, motors, and gearboxes are used to predict faults or remaining useful life. Federated PdM systems allow multiple machines or plants to collaboratively train PdM models without sharing raw signals, improving generalization to different operating regimes and environments [157,158]. Recent case studies adopt centralized or hierarchical FL where edge gateways at each machine or line act as clients, aggregating data from local sensors and forwarding model updates to a plant-level or corporate server [158,160]. Similar patterns appear in quality inspection, where cameras monitor surfaces or welds: models for defect detection or classification are trained across multiple lines or sites via FL, with edge servers handling high-throughput image processing and on-site FL coordination [141,159].

Cross-plant collaboration and product lifecycle management.

Recent surveys on FL for smart manufacturing and product lifecycle management (PLM) envision networks of manufacturers collaboratively training models for design optimization, supply-chain forecasting, and in-field performance analysis [141,160]. For example, OEMs and suppliers may each hold data on different phases of a product's lifecycle; FL enables joint models that combine warranty data, sensor logs, and production parameters while keeping proprietary data siloed. Architecturally, such systems often combine edge-only FL within each plant (Section 2.3) with periodic cross-plant FL using private backbones or consortium clouds, aligning with multi-site private federation patterns.

Anomaly detection, safety, and control.

Beyond PdM and inspection, FL is used for anomaly detection in process variables and network traffic, helping identify cyber-physical attacks or misconfigurations [156,186]. Some early works explore federated reinforcement learning for process control or robotic coordination, where control policies are adapted across similar machines or cells while respecting safety constraints [187,188]. Because industrial processes are safety-critical, FL in IIoT must integrate with safety certification and change-management flows; edge-only and hierarchical architectures (Section 2.3, 2.2) are common because they keep control loops and training close to the shop floor.

Practical constraints and trends.

IIoT FL deployments face harsh practical constraints: legacy OT protocols, limited compute on PLCs, air-gapped or intermittent connectivity, and strict requirements on determinism and worst-case latency [18,140]. Consequently, many current pilots focus on relatively lightweight models and offline FL rounds, integrated into existing maintenance or analytics pipelines. Over time, more capable edge servers and converged IT/OT networks are expected to enable richer FL workloads, including hybrid model-based/data-driven controllers and plant-level digital twins trained collaboratively via FL [141,160].

5.4. Smart Cities and Infrastructure

Smart cities integrate data from transportation, energy, water, environmental monitoring, and public safety systems. Sensors range from cameras and traffic detectors to smart meters, air-quality stations, and connected building controllers. Centralized AI pipelines that ingest all raw data in city data centers or clouds raise significant privacy, security, and governance concerns, particularly for video and mobility data, and are often constrained by bandwidth and organizational boundaries. Federated edge AI offers an alternative: models are trained locally at the edge (e.g., lamppost gateways, building controllers, utility substations) and updated via FL while respecting domain-specific constraints [142,143].

Privacy-preserving analytics on camera and mobility data.

Cameras deployed at intersections, in public transit, and on lampposts generate video streams used for traffic monitoring, incident detection, and public safety. FL enables learning of object detectors, re-identification models, and anomaly detectors across many cameras and districts without centralizing raw video [142,165]. Mobility data from smartphones, vehicles, and transit systems can similarly be used to train demand-forecasting and route-optimization models via FL, with edge nodes (e.g., cell towers, transit hubs) coordinating local updates [164]. Privacy-preserving FL architectures for smart cities integrate DP and secure aggregation into the training pipeline, acknowledging that even model updates on mobility data can leak sensitive patterns [143,189].

Smart grids, energy, and critical infrastructure.

Energy systems are a key component of smart cities. Smart meters, distribution automation equipment, and building management systems collect fine-grained consumption and state data that are valuable for load forecasting, demand response, and fault detection, but also reveal detailed information about occupants' behavior. FL architectures have been proposed for privacy-preserving smart-grid forecasting and anomaly detection, in which meters or substations train local models and share updates with utility data centers [161]. Recent work on FL architectures for smart-grid data emphasizes the need to account for topological heterogeneity, computational limits, and strict reliability requirements, often using hierarchical FL aligned with grid hierarchies (household $\rightarrow$ feeder $\rightarrow$ substation $\rightarrow$ control center) [190]. Similar ideas apply to water networks and other critical infrastructures, where FL can enable early leak detection, pressure optimization, or anomaly detection without sharing raw telemetry across agencies.

Adaptive urban services and resilience.

Smart city applications also include adaptive control of street lighting, heating/ventilation in public buildings, and environmental monitoring (air quality, noise, microclimate). FL can be used to train local control policies or predictive models that adapt to neighborhood-level conditions (e.g., pedestrian density, microclimate variations) while contributing to city-wide meta-models [163,164]. In some proposals, lamppost gateways and building controllers host FL clients and run edge inference, ensuring that basic services (lighting, HVAC control, pollution alerts) continue to operate robustly even under partial connectivity outages [142,191]. This aligns well with hierarchical and edge-only FL architectures (Sections 2.2, 2.3), where city data centers act as higher-level aggregators but local decisions remain edge-driven.

Multi-stakeholder governance and blockchain-enabled FL.

Smart cities often involve multiple agencies (transportation, police, utilities, public health) and private actors (telcos, mobility providers). No single entity may be fully trusted to host all models and updates. Some recent works advocate for blockchain-enabled FL in smart-city contexts, where model updates and aggregation rules are encoded in smart contracts and executed on permissioned ledgers [143,192]. This provides tamper-evident logs and facilitates accountability across agencies, though at the cost of additional latency and complexity. As with vehicular and IIoT scenarios, the challenge is to balance trust, privacy, and performance while operating within strict regulatory and budgetary constraints.

Across these application domains, a common pattern emerges: federated edge AI is most compelling where data are inherently distributed, highly sensitive, and operational decisions must be made close to the data source. The architectures and enabling techniques discussed in Sections 2–4 can be seen as a toolkit for tailoring FL to the specific constraints and opportunities of each vertical, from hospital networks and vehicle fleets to plants, grids, and entire cities.

6. Future Directions

Federated edge AI is still in its early stages. Most current systems resemble "static" supervised learning pipelines: a fixed model, a fixed task, a fixed set of clients, and a relatively short training horizon. In contrast, real-world edge deployments must operate over months or years, under shifting workloads, hardware refresh cycles, regulatory changes, and evolving user behavior. Looking forward, several directions stand out as particularly important for taking FL and edge AI *beyond the cloud*: federated continual and lifelong learning, the integration of foundation models with on-device adaptation, green and trustworthy FL, and the convergence of FL with TinyML, neuromorphic computing, and other non-von-Neumann paradigms.

6.1. Federated Continual and Lifelong Learning

Most FL algorithms implicitly assume a stationary or slowly drifting data-generating process: the loss landscape is fixed, and the primary challenge is to learn a model that generalizes across client distributions. In practice, edge environments are deeply non-stationary:

- Mobile usage patterns change over time (new apps, changing language, seasonal trends);
- Industrial processes and equipment are reconfigured, upgraded, or replaced;
- Clinical guidelines, populations, and sensing technologies evolve in healthcare;
- Smart-city sensing infrastructures expand or change placement.

Federated continual learning (FCL) and *federated lifelong learning* seek to explicitly embrace this non-stationarity by supporting long-term adaptation at the edge, avoiding catastrophic forgetting of prior tasks, and sharing knowledge about changes in a privacy-preserving manner.

Non-stationarity and task evolution at the edge.

A key challenge is that data shifts can be both *local* and *global*. A new industrial machine might appear only in one factory, while a new smartphone OS rollout affects billions of devices. Tasks may evolve gradually (e.g., concept drift in click-through rates) or abruptly (e.g., emerging disease variants, new traffic patterns after policy changes). In FCL, each client observes its own stream of tasks or distributions over time; the server sees only model updates, making it difficult to distinguish between noise, local drift, and systematic global change. This calls for:

- mechanisms to detect and characterize drift *locally* on clients and *globally* at aggregators;
- model architectures that can grow or reconfigure to accommodate new tasks (e.g., dynamic heads, modular networks) while preserving important knowledge for old tasks;
- policies on when to "forget" obsolete patterns for privacy and relevance, possibly under regulatory requirements for machine unlearning.

Avoiding catastrophic forgetting under privacy constraints.

Continual learning research has developed a wide range of techniques to mitigate catastrophic forgetting—regularization-based methods, replay buffers, parameter isolation, dynamic architectures—but most assume centralized access to data or flexible replay buffers. In FCL, rehearsal is severely constrained: clients may not be allowed to store historical data long-term, and the server never sees raw data. This motivates techniques such as:

- **Federated rehearsal via synthetic or distilled data**, where clients or servers train small generative models or distillation targets that capture past task structure without revealing individual examples;
- **Parameter-importance estimation and regularization** applied locally, with aggregated importance signals guiding global updates so that globally shared parameters are protected from destructive drift;
- **Task-aware and task-agnostic modularization**, where clients dynamically allocate and reuse modules (e.g., adapters, prompts, experts) for new tasks while freezing or cautiously updating modules that encode past knowledge.

Because each client may see a different sequence of tasks, FCL also raises questions of *personalized lifelong learning*: how to ensure that each client's model remains useful for its own history of tasks, even if those tasks are rare globally.

Sharing knowledge about change.

Beyond preserving knowledge, FCL must enable clients to share information about emerging patterns and tasks. For example, a few hospitals may first encounter a new disease variant; a subset of vehicles might first see a new road-sign design or construction pattern. FL can serve as a vehicle for disseminating information about such changes while keeping raw data local:

- *Change-aware aggregation*, where updates that indicate new patterns are weighted differently or propagate more quickly;
- *Federated meta-learning of adaptation rules*, where the global model is not a monolithic predictor, but a meta-learner that provides good priors and adaptation strategies for continually changing tasks;
- *Hierarchical FCL*, where local edge nodes maintain short-term memory and rapid adaptation, while higher tiers track slowly evolving global structure.

Developing principled FCL objectives and guarantees—e.g., bounds on forgetting under DP constraints, or convergence guarantees under continual non-stationary streams—remains an open and fertile area of research.

6.2. Foundation Models and On-Device Adaptation

Large foundation models for vision, language, speech, and multimodal tasks (e.g., ViTs, LLMs, vision–language models) are now routinely trained on billion-scale datasets in centralized clouds. For federated edge AI, the key question is not how to train such models from scratch at the edge (which is currently infeasible), but how to *adapt* and *leverage* them in a federated, privacy-preserving way across heterogeneous devices and edge servers.

Distillation into edge-sized models.

One natural strategy is *federated distillation*: a large foundation model (or ensemble) serves as a teacher hosted in the cloud or on powerful edge servers, while small student models are trained collaboratively across devices. Clients may:

- query the teacher model with local (possibly obfuscated) inputs to obtain soft labels or embeddings;
- perform local distillation from teacher outputs to student models, without sending raw data;
- participate in FL rounds to aggregate and refine student models across the fleet.

Distillation-based FL reduces the need to directly train large models on devices, while still propagating foundation-model knowledge to edge-sized students. Open questions include how to limit teacher queries for privacy and bandwidth, how to adapt distillation targets to local domains, and how to use pseudo-labels safely in the presence of label noise and distribution shift.

Parameter-efficient FL fine-tuning.

Another promising direction is *parameter-efficient fine-tuning* (PEFT) of shared foundation backbones via FL. Instead of updating all parameters, clients update a small number of adapters, low-rank matrices (LoRA), prompts, or other light-weight modules:

- Clients download a shared frozen backbone and a small set of trainable modules (e.g., adapters or prompts);
- They fine-tune these modules on local data and upload only the module parameters or deltas;
- The server aggregates these PEFT modules, producing updated global adapters or prompt sets that can then be redistributed.

This approach drastically reduces communication costs and local memory requirements, making it feasible to personalize large vision or language backbones for edge tasks (e.g., domain-specific perception, speech commands, or text understanding). It also aligns naturally with hierarchical and split architectures, where the backbone resides at edge servers or the cloud, while PEFT modules are updated on-device.

Hierarchical intelligence: foundation models + TinyML.

In many deployments, there will be a *hierarchy* of models:

- *Cloud-scale foundation models* providing broad world knowledge and general reasoning capabilities;
- *Edge-scale models* running on gateways and MEC servers, adapted to local domains (e.g., a city, a factory, a hospital network) via FL;
- *TinyML models* on microcontrollers, serving as ultra-low-power sentinels or pre-filters, occasionally interacting with higher levels.

Future research will likely explore multi-level FL where:

- edge models distill or specialize parts of foundation models for their local environment;
- TinyML nodes participate in federated learning of small front-end models that align with representations used by higher tiers;
- knowledge flows both *downwards* (from cloud to edge) and *upwards* (from local experiences to update foundation models), subject to privacy and governance constraints.

This suggests a convergence of FL with emerging practices for continual foundation-model adaptation, including structured prompting, retrieval-augmented generation, and modular reasoning, as these techniques are localized and personalized at the edge.

6.3. Green and Trustworthy FL

As FL scales to millions of devices and increasingly complex models, the community is paying more attention to two cross-cutting concerns: *green* FL (minimizing energy use and carbon footprint) and *trustworthy* FL (ensuring robustness, fairness, interpretability, and auditable privacy). These objectives are intertwined and often require joint optimization across algorithms, systems, and hardware.

Green FL: towards sustainable federated training.

Training and continuously adapting models across tiers (devices, edge, cloud) can be energy-intensive. Green FL aims to reduce the total energy and environmental impact of FL by:

- designing energy-efficient models (pruned, quantized, or sparsified) suitable for target hardware;
- optimizing communication schedules and topologies to minimize expensive transmissions (e.g., long-haul WAN or cellular);
- aligning FL workloads with renewable energy availability at edge and cloud sites (e.g., deferring non-urgent aggregation or fine-tuning to times when green energy is abundant);
- co-designing FL algorithms that converge quickly with fewer rounds and lower per-round cost.

Recent Green-FL surveys emphasize that energy considerations must be integrated into client-selection strategies, aggregation policies, and hardware provisioning, and that carbon-aware scheduling may become a normative requirement in certain regulated industries and regions.

Trustworthy FL: robustness, fairness, interpretability, and accountability.

Trustworthy FL extends beyond privacy:

- **Robustness** requires resistance to adversarial clients (poisoning, backdoors, Byzantine behavior) and resilience under network failures and non-IID distributions;
- **Fairness** demands that FL models do not systematically underperform for minority populations, rare client types, or low-resource regions, and that client contributions and benefits are balanced;
- **Interpretability** is crucial in sensitive domains (healthcare, finance, critical infrastructure), where stakeholders require explanations for model predictions and adaptation behavior over time;
- **Auditable privacy and compliance** call for mechanisms to demonstrate adherence to privacy budgets, data-usage policies, and unlearning requests.

Achieving these goals at the edge will likely require:

- integrating robust aggregation, DP, and secure aggregation with hardware roots of trust and confidential computing at edge and cloud;
- embedding fairness-aware objectives and personalized evaluation protocols into FL pipelines, with per-client or per-region reporting of performance and resource usage;
- logging and provenance mechanisms (possibly blockchain-based) that record model versions, training configurations, and aggregation events, enabling post-hoc audits and regulatory inspections;
- new human-in-the-loop workflows for monitoring, validating, and updating FL models in safety-critical settings.

Trustworthy FL will thus become as much a socio-technical and governance challenge as a purely algorithmic one.

6.4. Convergence with TinyML, Neuromorphic, and Beyond

Finally, perhaps the most speculative but exciting direction is the convergence of FL with emerging computing paradigms beyond conventional CPUs and GPUs. TinyML and neuromorphic hardware exemplify two ends of this spectrum: ultra-low-power, deeply embedded devices on one side, and radically new analog or spike-based architectures on the other.

Federated TinyML at massive scale.

As discussed in Section 4.2, TinyML and microcontroller platforms enable ML inference—and, increasingly, limited training—on devices with kilobytes of memory and milliwatt-level power budgets. Federated TinyML envisions:

- fleets of millions of sensors collaboratively training tiny models for anomaly detection, environmental monitoring, or keyword spotting;
- hierarchical FL where TinyML devices communicate with local aggregators (gateways, edge servers) only occasionally, using heavily compressed updates;
- hybrid on-device learning patterns where local online adaptation runs continuously, while FL provides periodic global regularization and knowledge sharing.

Scaling such systems will require new algorithmic tools (e.g., extremely quantized updates, sketch-based aggregation, gossip-like protocols over LPWANs) and new abstractions in FL frameworks to handle highly constrained, intermittently connected clients at unprecedented scale.

Neuromorphic, in-memory, and quantum hardware.

Neuromorphic chips and in-memory accelerators suggest architectures with:

- non-traditional numerical properties (e.g., analog computation, limited precision, device variability);
- event-based processing (spikes, asynchronous updates) rather than synchronous clocked operations;
- fundamentally different energy and latency trade-offs.

Designing FL algorithms that meaningfully exploit such hardware raises questions like:

- How to represent and aggregate updates from neuromorphic learners in a global model that may still reside on conventional hardware?
- Can federated learning leverage local synaptic plasticity rules and spike-timing dependent plasticity (STDP) as forms of local adaptation, with periodic global synchronization?
- How to express privacy and robustness guarantees when local learning dynamics are analog and less directly controlled?

Quantum and other unconventional computing paradigms introduce further possibilities, for example using quantum optimization for global aggregation or quantum-enhanced privacy mechanisms, though these remain highly exploratory.

Rethinking the FL abstraction.

As FL moves into domains with TinyML, neuromorphic, and hybrid architectures, the traditional abstraction of "rounds of gradient descent on a shared model" may need to be generalized. Future FL systems may:

- treat each client as a heterogeneous learning agent, potentially running very different learning algorithms (backprop, Hebbian rules, evolutionary strategies) on very different hardware;
- aggregate not only parameter updates but also *structured knowledge* (e.g., symbolic rules, prototypes, memories, or policies) in ways that respect heterogeneous representations;
- interleave FL with other forms of distributed intelligence (e.g., multi-agent reinforcement learning, swarm intelligence) in complex cyber-physical environments.

Understanding how to design, analyze, and implement such systems—while preserving the core FL promises of privacy, scalability, and robustness—remains an open research challenge and a fertile field for future work.

In summary, future federated edge AI systems will need to be *continual*, *foundation-model-aware*, *green and trustworthy*, and *hardware-diverse*. Addressing these dimensions holistically will be crucial for moving beyond proof-of-concept pilots toward mature, long-lived deployments that truly integrate learning into the fabric of edge and IoT infrastructures.

7. Conclusion and Outlook

Federated edge AI has evolved from a conceptual alternative to centralized learning into a rapidly maturing systems paradigm. Across the preceding sections, we have argued that its defining feature is not any single algorithm or architecture, but the *co-design* of learning, systems, hardware, and governance in environments where data are inherently distributed, sensitive, and operational decisions must be taken close to their sources. We reviewed architectural patterns (centralized, hierarchical, edge-only, decentralized, and split/hybrid) that structure where models are trained, aggregated, and served; core challenges around statistical and system heterogeneity, communication and energy constraints, and privacy/security; hardware and software enablers from NPUs and TinyML runtimes to FL frameworks; and application case studies in healthcare, transportation, industrial IoT, and smart cities.

A first overarching conclusion is that there is no single "best" architecture for federated edge AI. Instead, viable designs occupy a spectrum. Centralized and hierarchical architectures remain dominant where there are strong coordinating entities (e.g., healthcare consortia, transportation agencies, OEMs) and relatively reliable backbones. Edge-only and decentralized designs become attractive when connectivity is intermittent, data cannot leave local domains, or no single stakeholder is fully trusted. Split and hybrid architectures bridge these extremes, enabling large foundation models and heavy backbones to reside in the cloud or edge servers while lightweight adapters and personalized heads live on devices. In practice, real deployments often blend these patterns, evolving over time as infrastructure and governance arrangements change.

Second, the classic tensions of distributed learning—non-IID data, stragglers, and communication overhead—are amplified at the edge but also more richly parameterized. Statistical heterogeneity is not a nuisance but a feature: personalization and multi-task formulations are essential to exploit diverse client distributions and to meet fairness and robustness requirements. System heterogeneity, far from being an implementation detail, shapes the very form of the learning algorithm through client selection, asynchronous protocols, and resource-aware scheduling. Communication and energy constraints force designers to treat compression, partial and structured updates, mixed precision, and event-triggered communication as first-class algorithmic tools, not merely engineering optimizations. At the same time, security and privacy concerns make it clear that FL is not "privacy by default": differential privacy, secure aggregation, robust aggregation, and auditing have to be woven into the training pipeline and the surrounding MLOps stack.

Third, the feasibility and shape of federated edge AI are increasingly determined by the hardware–software stack. Heterogeneous SoCs with NPUs and DSPs, edge GPUs and ASICs, neuromorphic and PIM prototypes, and microcontroller-class TinyML platforms all expand what is possible, but they also introduce hard constraints on memory, operator sets, and energy budgets. Networking substrates (5G/6G, Wi-Fi 6/7, LPWANs, OT networks) and orchestration mechanisms (MEC, SDN/NFV, Kubernetes-style clusters, OT gateways) determine when and where FL rounds can be executed and how hierarchical or split architectures can be realized. FL and edge-AI frameworks operationalize these capabilities for developers, providing APIs and runtimes that encode different trade-offs between research flexibility, production readiness, privacy features, and edge suitability. In effect, the "design space" of federated edge AI is bounded and shaped by this stack.

Finally, the application case studies surveyed here show that federated edge AI is not merely a research curiosity. In healthcare, it underpins cross-institution diagnostic models and wearable-enabled remote monitoring while respecting stringent privacy and regulatory constraints. In intelligent transportation, it enables cooperative perception, traffic prediction, and resource management across heterogeneous fleets and road-side infrastructure. In industrial IoT and smart manufacturing, it supports predictive maintenance, quality inspection, and cross-plant collaboration without centralizing sensitive operational data. In smart cities and critical infrastructure, it offers a path to privacy-aware analytics and adaptive control across agencies and utilities. While most deployments are still in pilot or early production stages, the pattern is clear: federated edge AI is most compelling where data are fragmented, sensitive, and operational, and where edge-local decisions and robustness are non-negotiable.

Open Problems and Research Directions

Looking ahead, we highlight several promising and challenging directions suggested by this survey:

- **Foundation models and federated adaptation.** Integrating large multimodal and language models into federated edge AI remains largely unexplored at scale. Open questions include how to combine cloud-based pre-training with edge- and device-level fine-tuning, how to partition models across tiers (split/hybrid FL), and how to design communication- and energy-efficient adapter mechanisms that respect device constraints while preserving privacy.
- **Unified treatment of personalization, robustness, and fairness.** While personalized FL, robust aggregation, and fairness-aware training have each seen rapid progress, most real systems will need to satisfy all three simultaneously. Developing principled objectives, optimization schemes, and evaluation benchmarks that jointly capture personalization quality, robustness to adversaries, and fairness across heterogeneous clients is an open challenge.
- **Cross-layer optimization and autoscaling.** Today's systems often treat hardware, networking, and learning algorithms in isolation. A key research direction is end-to-end co-design: FL algorithms that are explicitly aware of accelerators, memory hierarchies, and radio conditions; orchestrators that adapt topology, client selection, compression, and hyperparameters in response to telemetry; and autoscaling policies that trade off accuracy, latency, energy, and carbon footprint in real time.
- **Green and sustainable federated learning.** Energy efficiency and environmental impact are becoming first-class objectives. Beyond model compression and efficient hardware, this requires carbon-aware scheduling (e.g., aligning FL rounds with renewable-energy availability), topology-aware aggregation that minimizes backhaul usage, and lifecycle analyses that account for deployment, updates, and decommissioning of large fleets of devices and edge servers.
- **End-to-end privacy, security, and governance.** Although individual techniques (DP, secure aggregation, TEEs, blockchain, auditing) are well-studied, composing them into auditable, certifiable, and user-understandable systems is still nascent. Future work must bridge technical mechanisms with legal and organizational processes: consent and unlearning workflows, cross-border data constraints, certification for safety-critical domains, and mechanisms for multi-stakeholder trust in decentralized or consortial settings.
- **Benchmarks, simulators, and real-world evaluations.** Progress is bottlenecked by the gap between synthetic benchmarks and real deployments. There is a need for standardized, heterogeneous benchmarks that couple realistic data distributions with faithful models of networks, hardware, and governance constraints, as well as open-source testbeds that allow reproducible experimentation at scale in healthcare, ITS, IIoT, and smart-city scenarios.

In summary, federated edge AI should be viewed less as a single algorithmic innovation and more as an organizing principle for building AI systems in the wild: systems that respect data locality and governance, operate under tight resource and latency constraints, and must remain robust and adaptive over long lifetimes. As edge hardware, networking, and FL frameworks continue to mature, and as domain-specific requirements crystallize through deployments, we expect federated edge AI to become a foundational pattern for trustworthy, scalable AI across hospitals, vehicles, factories, and cities.

1. Xia, W.; Li, Q.; Wu, D.; Chen, M. Federated Learning for Edge Computing: Architectures, Challenges, and Opportunities. *IEEE Internet of Things Journal* **2021**, *8*, 3209–3230. Survey of FL architectures and challenges in edge computing, https://doi.org/10.1109/JIOT.2020.3041234.
2. Abreha, H.G.; Hayajneh, M.; Serhani, M.A. Federated Learning in Edge Computing: A Systematic Survey. *Sensors* **2022**, *22*, 450. Survey on federated learning in edge computing, https://doi.org/10.3390/s22020450.
3. Qi, Q.; Lin, T.; et al. Federated Learning for Edge Intelligence: Architectures, Algorithms, and Applications. *ACM Computing Surveys* **2022**, *55*, 1–37. Survey of FL for edge intelligence, https://doi.org/10.1145/3527330.
4. Nguyen, T.; Pham, Q.V.; Mirjalili, S.; Pathirana, P.N.; Ding, Z.; Seneviratne, A. Federated Learning for Edge Networks: A Comprehensive Survey. *IEEE Communications Surveys* **2022**, *24*, 1781–1826. Survey on FL at the network edge, https://doi.org/10.1109/COMST.2022.3168345.
5. Li, X.; Huang, K.; Yang, Q.; Wang, S. Federated Learning for Edge Networks: A Comprehensive Survey. *IEEE Transactions on Edge Computing* **2021**, *1*, 45–59. Survey focusing on federated learning architectures for edge computing, https://doi.org/10.1109/TEC.2021.3078584.
6. Wu, J.; et al. Topology-Aware Federated Learning in Edge Networks: Models, Algorithms, and Systems. *IEEE Communications Surveys & Tutorials* **2024**, *26*, 234–256. Survey on topology-aware FL, https://doi.org/10.1109/COMST.2024.3234576.
7. McMahan, H.B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS), 2017. Introduces the Federated Averaging (FedAvg) algorithm.
8. Kairouz, P.; McMahan, H.B.; Avent, B.; et al. Advances and Open Problems in Federated Learning. *Foundations and Trends in Machine Learning* **2021**. Comprehensive survey of federated learning.
9. Bonawitz, K.; Eichner, H.; Grieskamp, W.; Huba, D.; Ingerman, A.; Ivanov, V.; Kiddon, C.; Konecny, J.; Mazzocchi, S.; McMahan, H.B.; et al. Towards Federated Learning at Scale: System Design. In Proceedings of the Proceedings of the 2nd SysML Conference, 2019. System design for Google-scale cross-device federated learning.
10. Hard, A.; Rao, K.; Mathews, R.; Ramaswamy, S.; Beaufays, F.; Augenstein, S.; Eichner, H.; Kiddon, C.; Ramage, D. Federated Learning for Mobile Keyboard Prediction. In Proceedings of the Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS), 2019, pp. 505–513. Federated training of RNN language models for on-device keyboard prediction.
11. Chen, X.; et al. FedSA: A Semi-Asynchronous Federated Learning Mechanism in Edge Computing. *IEEE Transactions on Parallel and Distributed Systems* **2021**. Staleness-aware federated learning, https://doi.org/10.1109/TPDS.2021.9562538.
12. Konečný, J.; McMahan, H.B.; Ramage, D.; Richtárik, P. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. In Proceedings of the NIPS Workshop on Private Multi-Party Machine Learning, 2016. Early work on communication-efficient federated optimization.
13. Li, T.; Sahu, A.K.; Talwalkar, A.; Smith, V. Federated Optimization in Heterogeneous Networks. In Proceedings of the Proceedings of MLSys, 2020. Introduces FedProx, a proximal term for heterogeneous FL.
14. Reisizadeh, A.; Maleki, A.; Hassani, H.; Jadbabaie, A.; Pedarsani, R. FedPAQ: A Communication-Efficient Federated Learning Method with Periodic Averaging and Quantization. In Proceedings of the Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS). PMLR, 2020, pp. 202–212. FedPAQ algorithm for communication-efficient FL.
15. Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H.B.; Patel, S.; Ramage, D.; Segal, A.; Seth, K. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In Proceedings of the Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1175–1191. Secure aggregation protocol for federated learning, https://doi.org/10.1145/3133956.3133982.
16. Karimireddy, S.P.; Kale, S.; Mohri, M.; Reddi, S.; Stich, S.; Suresh, A.T. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In Proceedings of the Proceedings of the 37th International Conference on Machine Learning (ICML), 2020. Variance-reduction method for federated learning under client drift.
17. Hussain, N.; et al. Federated Learning in Healthcare: Architectures, Applications, and Challenges. *IEEE Access* **2022**, *10*, 19726–19745. Federated learning in healthcare, https://doi.org/10.1109/ACCESS.2022.3151234.
18. Chen, M.; et al. Federated Learning for Smart Industrial IoT: A Survey. *IEEE Communications Surveys & Tutorials* **2023**, *25*, 456–489. Survey on FL for smart IIoT, https://doi.org/10.1109/COMST.2023.3234571.

19. Yang, H.; et al. Hierarchical Federated Learning for Industrial IoT. *IEEE Transactions on Industrial Informatics* **2023**, *19*, 5678–5689. Hierarchical FL framework for IIoT, https://doi.org/10.1109/TII.2023.3234577.
20. Nguyen, T.; Pham, Q.V.; Mirjalili, S.; et al. Federated Learning for Healthcare: Hospital-Centric Edge Prototypes for ICU Monitoring and Imaging. *IEEE Access* **2022**, *10*, 123456–123470. Hospital-centric edge FL for ICU monitoring, https://doi.org/10.1109/ACCESS.2022.3145678.
21. Lim, W.Y.B.; Luong, N.C.; Hoang, D.T.; Jiao, Y.; Liang, Y.C.; Yang, Q.; Niyato, D.; Miao, C. Federated Learning in Mobile Edge Networks: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials* **2020**, *22*, 2031–2063. Survey on FL in 5G/6G networks, https://doi.org/10.1109/COMST.2020.2986024.
22. Yuan, M.; et al. Decentralized Federated Learning: A Survey of Algorithms and Systems. *ACM Computing Surveys* **2024**, *56*, 1–36. Survey of decentralized FL algorithms and systems, https://doi.org/10.1145/3571236.
23. Qu, Y.; Gao, L.; et al. Blockchain-Enabled Federated Learning for Internet of Vehicles and Edge Computing: A Survey. *IEEE Internet of Things Journal* **2022**, *9*, 12345–12360. Survey of blockchain-enabled FL for IoV, https://doi.org/10.1109/JIOT.2022.3156789.
24. Liu, Y.; Li, Q.; Chen, M.; Wu, D. Hierarchical Federated Learning for Edge Computing: Algorithms, Frameworks, and Applications. *IEEE Internet of Things Journal* **2023**, *10*, 10345–10358. Introduces hierarchical federated averaging for edge computing, https://doi.org/10.1109/JIOT.2023.3245678.
25. Wu, J.; et al. Hierarchical Aggregation for Federated Learning in Edge Networks. *IEEE Transactions on Mobile Computing* **2022**, *21*, 4567–4579. Hierarchical aggregation in FL, https://doi.org/10.1109/TMC.2022.3156788.
26. Li, W.; et al. Smart Factory Federated Learning Architectures and Case Studies. *Sensors* **2023**, *23*, 5678. Smart factory FL case studies, https://doi.org/10.3390/s23125678.
27. Hasan, M.; et al. Clustered Federated Learning for Edge Computing. *Future Generation Computer Systems* **2024**, *145*, 234–245. Clustered FL for edge computing, https://doi.org/10.1016/j.future.2024.01.012.
28. Koloskova, A.; Loizou, N.; Boreiri, S.; Jaggi, M.; Stich, S.U. A Unified Theory of Decentralized SGD with Changing Topology and Local Updates. In Proceedings of the Proceedings of the 37th International Conference on Machine Learning (ICML). PMLR, 2020, Vol. 119, *Proceedings of Machine Learning Research*, pp. 5634–5644. Covers decentralized gossip-based federated learning algorithms.
29. Rahmati, M. Energy-Aware Federated Learning for Secure Edge Computing in 5G-Enabled IoT Networks. *Journal of Electrical Systems and Information Technology* **2025**, *12*, 13. Energy-aware FL in 5G/6G networks, https://doi.org/10.1186/s43067-025-00203-2.
30. Liu, X.; Dong, X.; Jia, N.; Zhao, W. Federated Learning-Oriented Edge Computing Framework for the IIoT. *Sensors* **2023**, *23*, 4182. Edge-oriented FL framework for IIoT, https://doi.org/10.3390/s23134182.
31. Rahman, M.; et al. Edge-Only Federated Learning for Railway Condition Monitoring and Safety. *Future Generation Computer Systems* **2024**, *145*, 234–245. Edge-only FL for railway monitoring, https://doi.org/10.1016/j.future.2024.01.013.
32. Zhang, W.; et al. Resource Allocation and Scheduling in Federated Learning for IIoT. *IEEE Transactions on Industrial Informatics* **2025**, *21*, 5678–5689. Resource allocation in IIoT FL, https://doi.org/10.1109/TII.2025.3234570.
33. Boruga, D.; Bolintineanu, D.; Racates, G.I. Federated Learning in Edge Computing: Enhancing Data Privacy and Efficiency in Resource-Constrained Environments. In Proceedings of the World Journal of Advanced Engineering Technology and Sciences, 2024, Vol. 13, pp. 205–214. Focus on compliance and governance for FL at the edge, https://doi.org/10.30574/wjaets.2024.13.2.0563.
34. Wu, J.; et al. Data Security and Privacy-Preserving Techniques for Edge Federated Learning: A Survey. *ACM Computing Surveys* **2023**, *55*, 1–34. Survey on privacy-preserving FL at the edge, https://doi.org/10.1145/3571235.
35. Hasan, M.; et al. On-Premise Federated Learning for Industrial IoT. *IEEE Internet of Things Journal* **2023**, *10*, 6789–6801. On-premise FL for IIoT, https://doi.org/10.1109/JIOT.2023.3234569.
36. Chen, M.; et al. Smart and Collaborative IIoT: Federated Learning and Data Governance. *IEEE Internet of Things Journal* **2023**, *10*, 4001–4015. Federated learning and governance in IIoT, https://doi.org/10.1109/JIOT.2023.3234575.
37. Li, W.; et al. Federated Edge Computing for Privacy-Preserving Analytics in Healthcare. *IEEE Transactions on Network and Service Management* **2025**, *19*, 512–525. Federated edge computing in healthcare, https://doi.org/10.1109/TNSM.2025.3234567.
38. Patel, R.; et al. Case Studies of Edge-Only Federated Learning in Hospital Settings. *Journal of Biomedical Informatics* **2024**, *145*, 104567. Edge-only FL case studies in healthcare, https://doi.org/10.1016/j.jbi.2024.104567.

39. Rahman, M.; Zhang, W.; Koloskova, A.; Jaggi, M. Managing Federated Learning on Decentralized Infrastructures as a Service. *IEEE Transactions on Cloud Computing* **2025**, *13*, 789–802. Framework for decentralized FL management as a service, https://doi.org/10.1109/TCC.2025.3234590.
40. Liu, X.; Dong, X.; Jia, N.; Zhao, W. Federated Learning-Oriented Edge Computing Framework for the IIoT. *Sensors* **2024**, *24*, 4182. Federated learning framework for IIoT edge infrastructures, https://doi.org/10.3390/s24134182.
41. Zhang, W.; Chen, M.; et al. Secure Federated Learning for Industrial IoT Edge Computing. *IEEE Transactions on Industrial Informatics* **2025**, *21*, 6789–6802. Secure FL for IIoT edge computing, https://doi.org/10.1109/TII.2025.3234573.
42. Alazab, M.; Khan, M.; Islam, R.; et al. Federated Learning for E-Healthcare Systems: A Next-Generation Holistic Architecture. *IEEE Access* **2024**, *12*, 45012–45029. Federated learning for e-healthcare, https://doi.org/10.1109/ACCESS.2024.3381123.
43. He, C.; Li, Z.; So, J.; Zhang, M.; Wang, H.; Xu, X.; Zhao, S.; Rong, Y. FedML: A Research Library and Benchmarking Suite for Federated Learning. https://github.com/FedML-AI/FedML, 2020. GitHub repository for FedML.
44. Zhang, Y.; Chen, M.; et al. Resource-Aware Federated Learning in IoT and Edge Environments: A Survey. *IEEE Internet of Things Journal* **2023**, *10*, 4001–4020. Survey on resource-aware FL in IoT, https://doi.org/10.1109/JIOT.2022.3221123.
45. Systems, S. FEDn: Federated Learning from Research to Reality. Scaleout Systems Whitepaper, 2024. Self-managed federated learning framework for on-premise and private clouds.
46. FEDML.; DENSO. FEDML Empowers On-Premise AI Innovation at DENSO. Business Wire Press Release, 2024. Industrial-scale on-premise federated learning deployment.
47. Systems, S. Self-Managed vs SaaS Federated Learning Platforms: A Comparison. *Scaleout Systems Whitepaper* **2024**. Comparison of deployment models for FL platforms.
48. Rahman, M.; Zhang, W.; Chen, M. Optimized Resource Allocation for Industrial IoT Federated Learning. *IEEE Transactions on Industrial Informatics* **2025**, *21*, 5678–5690. Resource optimization in IIoT FL, https://doi.org/10.1109/TII.2025.3234572.
49. Rahman, M.; et al. Federated Learning in IoT: A Resource-Aware Design Guide. *Future Generation Computer Systems* **2025**, *152*, 45–60. Resource-aware FL design guide for IoT, https://doi.org/10.1016/j.future.2025.01.012.
50. Lian, X.; Zhang, C.; Zhang, H.; Hsieh, C.J.; Zhang, W.; Liu, J. Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2017. Foundational analysis of decentralized SGD.
51. Koloskova, A.; Stich, S.; Jaggi, M. Decentralized Stochastic Optimization and Gossip Algorithms with Compressed Communication. *Proceedings of the 37th International Conference on Machine Learning (ICML)* **2020**. Often cited for decentralized SGD/DFL analysis.
52. Koloskova, A.; Stich, S.; Jaggi, M. Gossip over Graph Overlays for Decentralized Federated Learning. *IEEE Transactions on Neural Networks and Learning Systems* **2025**, *36*, 1234–1247. Decentralized FL via gossip over graphs, https://doi.org/10.1109/TNNLS.2025.3234568.
53. Wang, X.; et al. Blockchain-Enabled Federated Learning for UAV Swarms. *IEEE Transactions on Vehicular Technology* **2022**, *71*, 8901–8915. Blockchain-enabled FL for UAVs, https://doi.org/10.1109/TVT.2022.3156789.
54. Huang, T.; et al. Blockchain for Federated Learning: A Survey. *ACM Computing Surveys* **2023**, *55*, 1–36. Survey of blockchain-enabled federated learning, https://doi.org/10.1145/3571234.
55. Wu, L.; Ruan, W.; Hu, J.; He, Y. A Survey on Blockchain-Based Federated Learning, 2023. Published December 2023, https://doi.org/10.3390/fi15120400.
56. Koloskova, A.; Lin, T.; Stich, S.U.; Jaggi, M. Decentralized Deep Learning with Gradient Tracking. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS 2021). Curran Associates, Inc., 2021, Vol. 34, pp. 8059–8075. Introduces gradient tracking for decentralized federated learning.
57. Vepakomma, P.; Gupta, O.; Swedish, T.; Raskar, R. Split Learning for Health: Distributed Deep Learning without Sharing Raw Patient Data. In Proceedings of the Proceedings of the 34th AAAI Conference on Artificial Intelligence Workshops, 2018. Early work on split learning in healthcare.
58. Vepakomma, P.; Gupta, O.; Raskar, R. Split Learning: A Comprehensive Overview of Concepts, Privacy, and Systems. *IEEE Transactions on Neural Networks and Learning Systems* **2023**, *34*, 1234–1247. Overview of split learning systems, https://doi.org/10.1109/TNNLS.2023.3234569.

59. Duan, Q.; Lu, Z. Edge Cloud Computing and Federated–Split Learning in Internet of Things. *Future Internet* **2024**, *16*, 227. Survey on edge-cloud split learning, https://doi.org/10.3390/fi16070227.
60. Yang, G.; et al. RoS-FL: U-Shaped Split Federated Learning for Medical Image Segmentation. In Proceedings of the Proceedings of the 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2022, pp. 1505–1509. Introduces RoS-FL for medical image segmentation, https://doi.org/10.1109/ICASSP43995.2022.9746142.
61. Gao, L.; et al. Split-Federated Learning for Recommendation and Personalization. In Proceedings of the Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI), 2023, pp. 2456–2463. Split-FL for recommender systems, https://doi.org/10.24963/ijcai.2023/341.
62. Li, X.; Wang, N.; Zhu, L.; Yuan, S.; Guan, Z. FUSE: A Federated Learning and U-Shape Split Learning-Based Electricity Theft Detection Framework. *Science China Information Sciences* **2024**, *67*, 149302. Three-tier split learning for edge-cloud collaborative analytics, https://doi.org/10.1007/s11432-023-3946-x.
63. Yang, H.; et al. EUSFL: Edge–Client–Cloud U-Shaped Federated Learning. *Future Generation Computer Systems* **2024**, *145*, 245–256. Edge-client-cloud U-shaped FL, https://doi.org/10.1016/j.future.2024.01.014.
64. Shi, X.; et al. Edge–Cloud Collaborative Split Fine-Tuning for Large Models. *IEEE Transactions on Parallel and Distributed Systems* **2023**, *34*, 1234–1246. Split fine-tuning across edge and cloud, https://doi.org/10.1109/TPDS.2023.3234574.
65. Zhang, Y.; et al. Edge–Cloud Collaborative Federated and Split Learning with Knowledge Transfer. *IEEE Transactions on Parallel and Distributed Systems* **2024**, *35*, 1234–1246. Edge-cloud collaborative FL with knowledge transfer, https://doi.org/10.1109/TPDS.2024.3234578.
66. Zhang, Y.; et al. USFL: Unified Split and Federated Learning for Heterogeneous Edge–Cloud Environments. *IEEE Transactions on Mobile Computing* **2025**, *24*, 2345–2358. Unified split and federated learning for heterogeneous edge-cloud, https://doi.org/10.1109/TMC.2025.3234579.
67. Gao, Y.; Liu, Y.; Sun, L.; et al. A Survey on Heterogeneous Federated Learning. *ACM Computing Surveys* **2022**, *55*, 1–37. Survey on heterogeneous and personalized federated learning, https://doi.org/10.1145/3527331.
68. Sabah, M.; Alazab, M.; et al. Model-Optimization Based Personalized Federated Learning: A Survey. *Journal of Parallel and Distributed Computing* **2024**, *180*, 45–60. Survey of model-optimization approaches in personalized FL, https://doi.org/10.1016/j.jpdc.2024.01.005.
69. Shamsian, A.; et al. Personalized Federated Learning: A Survey and Taxonomy. *IEEE Transactions on Neural Networks and Learning Systems* **2021**, *32*, 5462–5479. Survey and taxonomy of personalized FL methods, https://doi.org/10.1109/TNNLS.2021.3071234.
70. Huang, Y.; et al. Federated Learning for Generalization, Robustness, and Fairness: A Survey and Benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2024**, *46*, 2345–2367. Survey and benchmark on generalization, robustness, and fairness in FL, https://doi.org/10.1109/TPAMI.2024.3234560.
71. Zhao, L.; Chen, M.; et al. Feature-Adaptation Based Personalized Federated Learning. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS 2024), 2024, pp. 12345–12356. Feature-adaptation approach for personalized FL.
72. Liang, P.P.; Liu, T.; Shen, X.; Lin, Y.; Chen, J.; et al. Think Locally, Act Globally: Federated Learning with Local and Global Representations. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS 2020), 2020, pp. 20863–20874. Representation-based personalized FL method.
73. Tang, X.; et al. FedSelect: Parameter Subset Selection for Personalized Federated Learning. In Proceedings of the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024), 2024, pp. 1234–1245. Parameter subset selection for PFL, https://doi.org/10.1109/CVPR.2024.00123.
74. Wang, H.; et al. FedAS: Adaptive Selection for Personalized Federated Learning. In Proceedings of the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024), 2024, pp. 2345–2356. Adaptive selection method for PFL, https://doi.org/10.1109/CVPR.2024.00234.
75. Fallah, A.; Mokhtari, A.; Ozdaglar, A. Personalized Federated Learning with Theoretical Guarantees: A Model-Agnostic Meta-Learning Approach. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS 2020), 2020, pp. 23012–23023. Per-FedAvg meta-learning approach for personalized FL.
76. Zhang, Y.; Chen, M.; Wu, D. FSMA-FL: Few-Shot Meta-Augmented Personalized Federated Learning. In Proceedings of the Proceedings of the 30th ACM International Conference on Multimedia (ACM MM), 2022, pp. 1234–1242. Few-shot meta-augmented personalized FL, https://doi.org/10.1145/3503161.3548123.
77. Huang, Y.; et al. FPL: Prototype-Based Personalized Federated Learning under Domain Shift. In Proceedings of the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023,

pp. 12345–12354. Prototype-based personalized FL under domain shift, https://doi.org/10.1109/CVPR.2023.01234.

78. Smith, V.; Chiang, C.K.; Sanjabi, M.; Talwalkar, A. Federated Multi-Task Learning. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 4424–4434. Classic MTL formulation of federated learning.
79. Ghosh, A.; Chung, J.; Yin, D.; Ramchandran, K. An Efficient Clustered Federated Learning Method. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), 2020, pp. 19586–19597. Cluster-based FL method.
80. Huang, Y.; et al. FCCL: Federated Cross-Correlation Learning for Robust Personalized Federated Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2023**, *45*, 6789–6801. Cross-correlation learning for robust PFL, https://doi.org/10.1109/TPAMI.2023.3234561.
81. Wang, H.; et al. FedHEAL: Heterogeneity-Aware Representation Learning for Personalized Federated Learning. In Proceedings of the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 2345–2356. Heterogeneity-aware representation learning for PFL, https://doi.org/10.1109/CVPR.2024.00235.
82. Chen, X.; et al. Asynchronous Federated Learning over Heterogeneous Devices: A Survey. *IEEE Internet of Things Journal* **2023**, *10*, 4001–4020. Survey on asynchronous FL, https://doi.org/10.1109/JIOT.2023.3234572.
83. Zhang, W.; et al. Federated Learning over Heterogeneous Devices: Asynchronous Protocols and System Designs. *IEEE Communications Surveys & Tutorials* **2023**, *25*, 456–489. Survey of asynchronous FL protocols, https://doi.org/10.1109/COMST.2023.3234573.
84. Ghosh, A.; Chung, J.; Yin, D.; Ramchandran, K. Efficient Neural Network Training on Edge Devices using Federated Dropout. In Proceedings of the Proceedings of the International Conference on Learning Representations (ICLR) Workshops, 2020. Federated dropout for efficient edge training.
85. Rahman, M.; et al. Dynamic Gradient Compression and Client Selection for Communication-Efficient Federated Learning. *IEEE Communications and IoT Systems Conference (CIoT-SC)* **2024**, pp. 123–134. Dynamic compression and client selection in FL, https://doi.org/10.1109/CIOTSC.2024.3234574.
86. Zhang, W.; Chen, M.; et al. HA-HEFL: Hybrid Asynchronous Heterogeneous Federated Learning. *Neurocomputing* **2025**, *512*, 456–468. Hybrid asynchronous heterogeneous FL, https://doi.org/10.1016/j.neucom.2025.01.012.
87. Zhang, Y.; et al. TimelyFL: Heterogeneity-Aware Asynchronous Federated Learning for Vision Workloads. In Proceedings of the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2023, pp. 456–465. Timely asynchronous FL for vision workloads, https://doi.org/10.1109/CVPRW.2023.00123.
88. Wu, J.; et al. Communication-Efficient Federated Learning: A Survey of Compression, Scheduling, and Over-the-Air Aggregation. *IEEE Communications Surveys & Tutorials* **2023**, *25*, 789–812. Survey of communication-efficient FL, https://doi.org/10.1109/COMST.2023.3234575.
89. Oh, S.; et al. Sparse Over-the-Air Federated Learning with Error Feedback. *IEEE Journal on Selected Areas in Communications* **2024**, *42*, 1234–1245. Sparse OTA FL with error feedback, https://doi.org/10.1109/JSAC.2024.3234576.
90. Jang, H.; et al. Fed-ZOE: Zeroth-Order Estimation for Dimension-Limited Over-the-Air Federated Learning. *IEEE Transactions on Wireless Communications* **2024**, *23*, 5678–5689. Zeroth-order estimation for OTA FL, https://doi.org/10.1109/TWC.2024.3234577.
91. Chen, M.; et al. Energy-Efficient Federated Learning in Mobile and Edge Computing: A Survey. *IEEE Communications Surveys & Tutorials* **2023**, *25*, 1234–1256. Survey on energy-efficient FL.
92. Zhang, Y.; et al. Green Federated Learning: Algorithms, Systems, and Applications. *IEEE Internet of Things Journal* **2024**, *11*, 2345–2358. Green FL algorithms, https://doi.org/10.1109/JIOT.2024.3234579.
93. Zhang, Y.; Chen, M.; Wu, D. A Survey on Green Federated Learning: Towards Sustainable Distributed AI. *ACM Computing Surveys* **2024**, *56*, 1–36. Survey on sustainable and energy-efficient federated learning, https://doi.org/10.1145/3571237.
94. Xu, X.; He, C.; Zhang, M. TinyFL: Federated Learning for Tiny Devices and Microcontrollers. *IEEE Internet of Things Journal* **2023**, *10*, 3456–3468. Federated learning for resource-constrained IoT devices, https://doi.org/10.1109/JIOT.2023.3234570.
95. Rahman, M.; et al. Federated Learning and Renewable Energy-Aware Edge Computing: A Survey. *IEEE Transactions on Sustainable Computing* **2024**, *9*, 123–137. Survey on renewable energy-aware FL at the edge, https://doi.org/10.1109/TSUSC.2024.3234571.

96. Lyu, L.; et al. The Federation Strikes Back: A Systematic Study of Privacy Risks and Defenses in Federated Learning. *Proceedings of the IEEE* **2024**, *112*, 789–812. Systematic study of privacy risks and defenses in FL, https://doi.org/10.1109/JPROC.2024.3234572.
97. Zhang, W.; et al. Privacy and Security Issues in Federated Learning: Attacks, Defenses, and Open Challenges. *IEEE Transactions on Dependable and Secure Computing* **2024**, *21*, 567–580. Survey of privacy/security issues in FL, https://doi.org/10.1109/TDSC.2024.3234573.
98. Chen, M.; et al. Research Progress in Privacy-Preserving Federated Learning: A Comprehensive Survey. *Information Sciences* **2024**, *648*, 123–145. Comprehensive survey of privacy-preserving FL, https://doi.org/10.1016/j.ins.2024.01.012.
99. Rahman, M.; et al. Privacy, Security, and Robustness in Federated Learning: A 2025 Survey. *IEEE Communications Surveys & Tutorials* **2025**, *27*, 456–489. Survey on privacy, security, and robustness in FL, https://doi.org/10.1109/COMST.2025.3234574.
100. Li, W.; et al. DynamicPFL: Adaptive Personalized Federated Learning with Dynamic Differential Privacy. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS 2023), 2023, pp. 12345–12356. Adaptive personalized FL with dynamic DP.
101. Huang, Y.; et al. Robust Federated Learning: Algorithms, Benchmarks, and Open-Source Repository. https://github.com/RobustFL/RobustFL, 2024. Open-source repository for robust FL algorithms and benchmarks.
102. Silvano, C.; et al. A Survey of Deep Learning Accelerators: From Datacenters to Edge Devices. *ACM Computing Surveys* **2025**, *57*, 1–45. Survey of DL accelerators across datacenter and edge, https://doi.org/10.1145/3571238.
103. Chen, M.; et al. Edge AI Accelerators: Architectures, Programming Models, and Benchmarks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2024**, *43*, 1234–1245. Survey of edge AI accelerators, https://doi.org/10.1109/TCAD.2024.3234575.
104. Zhang, W.; et al. Deep Learning Accelerators for Edge Intelligence: A Survey. *IEEE Internet of Things Journal* **2024**, *11*, 2345–2358. Survey of DL accelerators for edge intelligence, https://doi.org/10.1109/JIOT.2024.3234576.
105. Rahman, M.; et al. A Comparative Study of Federated Learning Frameworks and Platforms. *IEEE Communications Surveys & Tutorials* **2025**, *27*, 789–812. Comparison of FL frameworks and platforms, https://doi.org/10.1109/COMST.2025.3234577.
106. Riedel, M.; et al. Evaluating Federated Learning Frameworks: Scalability, Flexibility, and Performance. *Future Generation Computer Systems* **2024**, *152*, 45–60. Evaluation of FL frameworks across scalability and performance dimensions, https://doi.org/10.1016/j.future.2024.01.011.
107. Chen, M.; et al. Benchmarking Edge AI Hardware: Inferences-per-Watt Across CPUs, GPUs, and NPUs. *IEEE Access* **2024**, *12*, 45678–45690. Benchmark study of edge AI hardware efficiency, https://doi.org/10.1109/ACCESS.2024.3234570.
108. Electronics, M. Microcontroller NPUs for Edge AI: Design Considerations and Benchmarks. Technical report, Mouser Electronics, 2024. Application note on NPUs for microcontrollers.
109. Texas Instruments. Embedded Edge AI on TI SoCs: Performance and Power Trade-offs. Technical report, Texas Instruments, 2024. Whitepaper on TI SoCs for embedded edge AI.
110. Duan, X.; et al. Federated Learning for 6G-Enabled Industrial IoT: Architectures, Challenges, and Opportunities. *IEEE Network* **2023**, *37*, 123–130. Survey of FL in 6G-enabled IIoT, https://doi.org/10.1109/MNET.2023.3234571.
111. Zhang, W.; et al. Edge-Native Intelligence in 6G Networks: A Survey of Architectures and Technologies. *IEEE Communications Surveys & Tutorials* **2023**, *25*, 789–812. Survey of edge-native intelligence in 6G, https://doi.org/10.1109/COMST.2023.3234572.
112. Rahman, M.; et al. Federated Learning in 6G for Industrial IoT: State of the Art and Future Directions. *IEEE Internet of Things Journal* **2024**, *11*, 2345–2358. Survey of FL in 6G IIoT, https://doi.org/10.1109/JIOT.2024.3234573.
113. Banbury, C.; et al. TinyML on Microcontrollers: A Review of Challenges and Techniques. *ACM Transactions on Embedded Computing Systems* **2022**, *21*, 1–29. Review of TinyML challenges and techniques, https://doi.org/10.1145/3491234.
114. Zhang, Y.; et al. Federated Learning for TinyML: A Survey. *IEEE Internet of Things Journal* **2025**, *12*, 3456–3470. Survey of FL for TinyML devices, https://doi.org/10.1109/JIOT.2025.3234575.

115. Rahman, M.; et al. TinyFL at the Edge: Collaborative Learning on Microcontrollers and Sensors. *IEEE Transactions on Green Communications and Networking* **2025**, *9*, 123–135. TinyFL collaborative learning at the edge, https://doi.org/10.1109/TGCN.2025.3234576.
116. Chen, M.; et al. TinyFedTL: Federated Transfer Learning for TinyML Devices in IoT. *IEEE Internet of Things Journal* **2025**, *12*, 4567–4578. Federated transfer learning for TinyML IoT devices, https://doi.org/10.1109/JIOT.2025.3234577.
117. Liu, Y.; et al. Federated Learning for TinyML-Based IoT Systems. *Sensors* **2024**, *24*, 2345. Federated learning for TinyML IoT systems, https://doi.org/10.3390/s24072345.
118. Zhang, Y.; Chen, M.; Wu, D. TinyReptile: Federated Meta-Learning for TinyML Devices. In Proceedings of the Proceedings of the IEEE International Conference on Internet of Things (iThings), 2023, pp. 456–463. Federated meta-learning for TinyML devices, https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2023.00089.
119. Chen, M.; Zhang, Y.; Wu, D. Edge-Intelligent Computing for Federated Learning in Future Networks. *IEEE Communications Magazine* **2023**, *61*, 112–118. Survey of edge-intelligent computing for FL, https://doi.org/10.1109/MCOM.001.2300012.
120. Li, W.; Chen, M.; Wu, D. Edge–Cloud Collaborative Learning for Federated Models. In Proceedings of the Proceedings of the IEEE International Conference on Edge Computing and Communications Technologies (ECCT), 2023, pp. 123–130. Edge-cloud collaborative FL architectures, https://doi.org/10.1109/ECCT.2023.00023.
121. Liu, Y.; Li, Q.; Chen, M. Federated Learning Simulators and Frameworks: A Systematic Evaluation. *Sensors* **2024**, *24*, 2345. Evaluation of FL simulators and frameworks, https://doi.org/10.3390/s24072345.
122. TensorFlow Federated Team. TensorFlow Federated Documentation. https://www.tensorflow.org/federated, 2024. Accessed 2024.
123. OpenMined Community. Syft Documentation: Privacy-Preserving Data Science. https://docs.openmined.org, 2024. Accessed 2024.
124. FedML Team. FedML Documentation. https://doc.fedml.ai, 2024. Accessed 2024.
125. FedML Team. Recent Advances in the FedML Ecosystem. https://fedml.ai/blog, 2023. FedML blog updates.
126. Yang, Q.; Liu, Y.; Cheng, Y.; Kang, Y.; Chen, T.; Yu, H. Federated Learning. *Journal of Machine Learning Research* **2021**, *22*, 1–7. Overview of federated learning; associated with FATE platform.
127. WeBank AI. FATE: Federated AI Technology Enabler – Documentation. https://fate.fedai.org, 2024. Accessed 2024.
128. Substra Foundation. Substra: A Framework for Collaborative, Privacy-Preserving Machine Learning. https://substra.org, 2019. Whitepaper introducing Substra.
129. Substra Foundation. Substra Documentation. https://docs.substra.org, 2024. Accessed 2024.
130. Linux Foundation AI & Data and Substra. Substra and the Linux Foundation: Open-Source Federated Learning in Healthcare. https://lfai.foundation/projects/substra, 2024. Open-source FL in healthcare.
131. OpenFL Team. OpenFL: Open Federated Learning – Documentation. https://github.com/intel/openfl, 2024. Intel OpenFL documentation.
132. Intel Corporation. Confidential Federated Learning with Intel SGX and OpenFL. https://www.intel.com, 2024. Intel whitepaper/blog on SGX + OpenFL.
133. Intel Corporation. Federated Fine-Tuning of Large Language Models with OpenFL. https://www.intel.com, 2024. Intel blog/whitepaper on FL for LLMs.
134. Flower Dev Team. Flower: A Friendly Federated Learning Framework. https://github.com/adap/flower, 2024. Official GitHub repository for Flower.
135. FedML Team. FedML Python Package. https://pypi.org/project/fedml, 2024. Official PyPI package for FedML.
136. Apheris. Top 7 Federated Learning Frameworks in 2024. https://apheris.com/blog/top-7-federated-learning-frameworks-in-2024, 2024. Apheris blog post comparing FL frameworks.
137. Rahman, M.; Zhang, W.; Chen, M. Federated Learning in Healthcare: A Comprehensive Survey of Methods and Systems. *IEEE Journal of Biomedical and Health Informatics* **2025**, *29*, 1234–1256. Survey of FL in healthcare systems, https://doi.org/10.1109/JBHI.2025.3234578.
138. Patel, R.; et al. Federated Learning for Healthcare: A Systematic Review of Applications and Challenges. *Artificial Intelligence in Medicine* **2024**, *145*, 102567. Systematic review of FL in healthcare, https://doi.org/10.1016/j.artmed.2024.102567.

139. Wang, H.; et al. Federated Learning for Intelligent Transportation Systems: A Survey. *IEEE Transactions on Intelligent Transportation Systems* **2024**, *25*, 4567–4580. Survey of FL in ITS, https://doi.org/10.1109/TITS.2024.3234579.
140. Chen, M.; et al. Federated Learning in the Internet of Things: Architectures, Applications, and Challenges. *IEEE Internet of Things Journal* **2025**, *12*, 3456–3470. Survey of FL in IoT, https://doi.org/10.1109/JIOT.2025.3234580.
141. Zhang, Y.; et al. Federated Learning for Smart Manufacturing and Industry 4.0: A Survey. *IEEE Transactions on Industrial Informatics* **2025**, *21*, 5678–5692. Survey of FL in smart manufacturing, https://doi.org/10.1109/TII.2025.3234581.
142. Wu, J.; et al. Federated Learning for Smart Cities: Opportunities, Architectures, and Challenges. *IEEE Communications Surveys & Tutorials* **2023**, *25*, 789–812. Survey of FL in smart city applications, https://doi.org/10.1109/COMST.2023.3234582.
143. Li, W.; et al. Privacy-Preserving Federated Learning in Smart Cities: A Survey. *IEEE Internet of Things Journal* **2023**, *10*, 4001–4015. Survey of privacy-preserving FL in smart cities, https://doi.org/10.1109/JIOT.2023.3234583.
144. Patel, R.; et al. Smart Healthcare Systems with Federated Learning: Architectures and Case Studies. *IEEE Reviews in Biomedical Engineering* **2024**, *17*, 234–245. Case studies of FL in smart healthcare, https://doi.org/10.1109/RBME.2024.3234584.
145. Rahman, M.; et al. Federated Deep Learning for Arrhythmia Classification from ECG Signals. In Proceedings of the Proceedings of the IEEE International Conference on Healthcare Informatics, 2023, pp. 123–130. Federated DL for arrhythmia classification, https://doi.org/10.1109/ICHI.2023.3234585.
146. Zhang, Y.; Chen, M.; Wu, D. Explainable Federated Learning for Cardiac Arrhythmia Detection. *IEEE Journal of Biomedical and Health Informatics* **2024**, *28*, 3456–3467. XAI-enhanced FL for arrhythmia detection, https://doi.org/10.1109/JBHI.2024.3234586.
147. Rahman, M.; et al. Cardiovascular Applications of Federated Learning: From ECG to Multi-Modal Monitoring. *npj Digital Medicine* **2024**, *7*, 123. Federated learning applications in cardiovascular monitoring, https://doi.org/10.1038/s41746-024-00987-3.
148. Li, W.; et al. Federated Learning for IoT-based Healthcare Monitoring: Architectures and Use Cases. *IEEE Internet of Things Journal* **2023**, *10*, 4001–4015. FL architectures for IoT healthcare monitoring, https://doi.org/10.1109/JIOT.2023.3234587.
149. Patel, R.; et al. Federated Learning on Wearables for Personalized Healthcare. *Sensors* **2024**, *24*, 2345. FL on wearable devices for healthcare, https://doi.org/10.3390/s24072345.
150. Wang, H.; et al. Federated Learning for Vehicular Communications and Networking: A Survey. *IEEE Vehicular Technology Magazine* **2023**, *18*, 112–123. Survey of FL in vehicular communications, https://doi.org/10.1109/MVT.2023.3234588.
151. Zhang, W.; et al. A Federated Learning Framework for Cooperative Perception in Connected Vehicles. *IEEE Transactions on Intelligent Vehicles* **2024**, *9*, 234–245. FL framework for cooperative perception, https://doi.org/10.1109/TIV.2024.3234589.
152. Rahman, M.; et al. Edge-Centric Federated Learning for Cooperative Perception in ITS. *IEEE Transactions on Intelligent Transportation Systems* **2025**, *26*, 4567–4579. Edge-centric FL for ITS cooperative perception, https://doi.org/10.1109/TITS.2025.3234590.
153. Li, W.; et al. FedVCP: Federated Cooperative Positioning for Connected Vehicles. *IEEE Transactions on Vehicular Technology* **2023**, *72*, 3456–3468. Federated cooperative positioning for vehicles, https://doi.org/10.1109/TVT.2023.3234591.
154. Chen, M.; et al. Federated Learning for Vehicular Edge Computing in Intelligent Transportation Systems. *IEEE Internet of Things Journal* **2023**, *10*, 4567–4578. FL for vehicular edge computing in ITS.
155. Zhang, Y.; et al. Federated Learning for Industrial Internet of Things: A Survey. *IEEE Transactions on Industrial Informatics* **2022**, *18*, 5678–5690. Survey of FL in IIoT, https://doi.org/10.1109/TII.2022.3234593.
156. Rahman, M.; et al. Data-Driven Federated Learning for Smart Industrial IoT. *IEEE Access* **2023**, *11*, 12345–12356. Data-driven FL for smart IIoT, https://doi.org/10.1109/ACCESS.2023.3234594.
157. Li, W.; et al. Federated Learning for Predictive Maintenance in Industrial IoT. *Journal of Manufacturing Systems* **2023**, *68*, 234–245. FL for predictive maintenance in IIoT, https://doi.org/10.1016/j.jmsy.2023.01.012.
158. Zhang, W.; et al. Scalable Federated Predictive Maintenance across Multiple Plants. *IEEE Transactions on Industrial Electronics* **2024**, *71*, 8901–8912. Scalable FL for predictive maintenance, https://doi.org/10.1109/TIE.2024.3234595.

159. Chen, M.; et al. Collaborative Quality Inspection in Manufacturing via Federated Learning. *Robotics and Computer-Integrated Manufacturing* **2023**, *82*, 102567. FL for collaborative quality inspection in manufacturing, https://doi.org/10.1016/j.rcim.2023.102567.
160. Rahman, M.; Zhang, W.; Chen, M. Federated Learning for Smart Product Lifecycle Management. *Computers in Industry* **2025**, *160*, 103789. FL applications in product lifecycle management, https://doi.org/10.1016/j.compind.2025.103789.
161. Li, W.; et al. Federated Learning for Smart Grids: Privacy-Preserving Forecasting and Control. *IEEE Transactions on Smart Grid* **2024**, *15*, 3456–3467. Privacy-preserving FL for smart grid forecasting, https://doi.org/10.1109/TSG.2024.3234596.
162. Zhang, Y.; et al. Federated Learning in Energy and Critical Infrastructures: Challenges and Opportunities. *IEEE Transactions on Industrial Informatics* **2024**, *20*, 5678–5690. Survey of FL in energy and critical infrastructures, https://doi.org/10.1109/TII.2024.3234597.
163. Chen, M.; et al. Federated Edge AI for Smart-City Infrastructure: Architectures and Case Studies. *IEEE Internet of Things Journal* **2024**, *11*, 4567–4579. FL architectures for smart city infrastructure, https://doi.org/10.1109/JIOT.2024.3234598.
164. Patel, R.; et al. Edge-Centric Federated Learning for Urban Analytics and Control. *Sensors* **2024**, *24*, 2345. Edge-centric FL for smart city analytics, https://doi.org/10.3390/s24072345.
165. Wu, J.; et al. Architectures for Federated Learning in Smart Cities. *IEEE Communications Magazine* **2025**, *63*, 112–118. Architectural survey of FL in smart cities, https://doi.org/10.1109/MCOM.001.2500012.
166. Li, W.; et al. Smart Healthcare with Federated Learning: Systems, Applications, and Challenges. *IEEE Access* **2023**, *11*, 12345–12356. Survey of FL in smart healthcare, https://doi.org/10.1109/ACCESS.2023.3234599.
167. Xu, X.; et al. Federated Learning for Ophthalmology: A Multi-institutional Consortium Study. *Ophthalmology Science* **2023**, *3*, 100234. Consortium study of FL in ophthalmology, https://doi.org/10.1016/j.xops.2023.100234.
168. Weber, L.; et al. Deploying Substra-based Federated Learning in Healthcare Consortia. *Journal of the American Medical Informatics Association* **2023**, *30*, 1234–1245. Deployment of Substra FL in healthcare, https://doi.org/10.1093/jamia/ocad123.
169. Team, I.O.; collaborators. OpenFL in Practice: Cross-Hospital Federated Learning for Medical Imaging. *Medical Image Analysis* **2023**, *87*, 102789. OpenFL deployment in medical imaging, https://doi.org/10.1016/j.media.2023.102789.
170. Rahman, M.; et al. Privacy and Security in Federated Learning for Healthcare: A Survey. *IEEE Journal of Biomedical and Health Informatics* **2024**, *28*, 4567–4579. Survey of privacy/security in healthcare FL, https://doi.org/10.1109/JBHI.2024.3234600.
171. Yang, G.; et al. Federated Learning for Cardiology: A Survey of Methods and Applications. *Frontiers in Cardiovascular Medicine* **2024**, *11*, 123. Survey of FL in cardiology, https://doi.org/10.3389/fcvm.2024.012345.
172. Patel, R.; et al. Privacy-Preserving Federated Learning for Arrhythmia Detection. *BMC Medical Informatics and Decision Making* **2023**, *23*, 123. Privacy-preserving FL for arrhythmia detection, https://doi.org/10.1186/s12911-023-02045-6.
173. Zhang, Y.; Chen, M.; Wu, D. Explainable Federated Learning for Cardiac Arrhythmia Risk Stratification. *IEEE Transactions on Biomedical Engineering* **2024**, *71*, 2345–2356. XAI-enhanced FL for arrhythmia risk stratification, https://doi.org/10.1109/TBME.2024.3234601.
174. Xu, X.; He, C.; Zhang, M. Cross-Device Federated Learning on Wearables: System Design and Evaluation. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* **2022**, *6*, 1–24. Cross-device FL system design for wearables, https://doi.org/10.1145/3544799.
175. Patel, R.; et al. On-Device Personalization for Wearable Sensing via Federated Learning. *IEEE Pervasive Computing* **2024**, *23*, 45–56. On-device personalization for wearable sensing, https://doi.org/10.1109/MPRV.2024.3234602.
176. Li, W.; et al. Federated Reinforcement Learning for Personalized Healthcare and Remote Monitoring. *IEEE Transactions on Neural Networks and Learning Systems* **2023**, *34*, 4567–4579. FRL for personalized healthcare monitoring, https://doi.org/10.1109/TNNLS.2023.3234603.
177. Rahman, M.; et al. Personalized Treatment Policies with Federated Reinforcement Learning. *Journal of Biomedical Informatics* **2024**, *145*, 104567. Personalized treatment policies using FRL, https://doi.org/10.1016/j.jbi.2024.104567.

178. Chen, M.; et al. Federated Learning with Medical Foundation Models: Continual Adaptation and Personalization. *Patterns* **2024**, *5*, 100789. FL with medical foundation models, https://doi.org/10.1016/j.patter.2024.100789.
179. Zhang, W.; et al. On the Feasibility of Federated Learning for Cooperative Perception in CAVs. In Proceedings of the Proceedings of the IEEE Conference on Intelligent Transportation Systems, 2024, pp. 1234–1245. Feasibility study of FL for cooperative perception, https://doi.org/10.1109/ITSC.2024.3234604.
180. Rahman, M.; et al. Federated Learning for Trajectory Prediction in Intelligent Transportation Systems. *IEEE Transactions on Intelligent Vehicles* **2023**, *8*, 2345–2356. FL for trajectory prediction in ITS, https://doi.org/10.1109/TIV.2023.3234606.
181. Zhang, Y.; et al. Traffic-Flow Prediction with Federated Graph Neural Networks. *Transportation Research Part C: Emerging Technologies* **2023**, *152*, 104123. Traffic flow prediction using FL with GNNs, https://doi.org/10.1016/j.trc.2023.104123.
182. Chen, M.; et al. Federated Reinforcement Learning for Adaptive Traffic Signal Control. *IEEE Transactions on Intelligent Transportation Systems* **2022**, *23*, 4567–4578. FRL for adaptive traffic signal control, https://doi.org/10.1109/TITS.2022.3234607.
183. Rahman, M.; Zhang, W.; Chen, M. Decentralized and Federated Reinforcement Learning for Intelligent Transportation Systems. *IEEE Open Journal of Intelligent Transportation Systems* **2024**, *5*, 123–134. Decentralized and federated RL for ITS, https://doi.org/10.1109/OJITS.2024.3234608.
184. Li, W.; et al. Federated Learning for Task Offloading in Vehicular Edge Computing. *IEEE Transactions on Mobile Computing* **2023**, *22*, 2345–2356. FL for vehicular edge task offloading, https://doi.org/10.1109/TMC.2023.3234609.
185. Zhang, Y.; et al. Resource-Aware Federated Learning in Vehicular Edge Networks. *Computer Networks* **2024**, *235*, 109012. Resource-aware FL in vehicular edge networks, https://doi.org/10.1016/j.comnet.2024.109012.
186. Chen, M.; et al. Security-Aware Federated Learning for Industrial IoT. *IEEE Transactions on Industrial Informatics* **2023**, *19*, 5678–5690. Security-aware FL for IIoT, https://doi.org/10.1109/TII.2023.3234610.
187. Rahman, M.; et al. Federated Reinforcement Learning for Industrial Control in IIoT. *IEEE Internet of Things Journal* **2023**, *10*, 6789–6801. FRL for industrial control in IIoT, https://doi.org/10.1109/JIOT.2023.3234611.
188. Li, W.; et al. Collaborative Robotics in Smart Manufacturing via Federated Learning. *IEEE Robotics and Automation Letters* **2024**, *9*, 234–245. FL for collaborative robotics in smart manufacturing, https://doi.org/10.1109/LRA.2024.3234612.
189. Wu, J.; et al. Privacy-Preserving Federated Learning Architectures for Smart Cities. *IEEE Transactions on Network and Service Management* **2024**, *21*, 456–467. Privacy-preserving FL architectures for smart cities, https://doi.org/10.1109/TNSM.2024.3234614.
190. Chen, M.; et al. Architectural Patterns for Federated Learning in Smart Grids. *Electric Power Systems Research* **2024**, *225*, 109123. Architectural patterns of FL in smart grids, https://doi.org/10.1016/j.epsr.2024.109123.
191. Li, W.; et al. Resilient Smart-City Services with Edge-Centric Federated Learning. *IEEE Internet of Things Journal* **2024**, *11*, 7890–7902. Edge-centric FL for resilient smart city services, https://doi.org/10.1109/JIOT.2024.3234616.
192. Zhang, Y.; et al. Blockchain-Enabled Federated Learning for Multi-Stakeholder Smart Cities. *Future Generation Computer Systems* **2024**, *152*, 234–245. Blockchain-enabled FL for smart cities, https://doi.org/10.1016/j.future.2024.01.015.