

HFL-Lite: Sparse Coding for Lightweight Hierarchical Federated Learning in Computation-Limited Sensor Networks

Cangming Liang, Kuan-Ching Li, Zulong Diao, Xiangwei Meng, Jie Liu, and Wei Liang ^(✉)

Abstract The rapid growth of sensor-equipped edge devices calls for machine learning paradigms that are both lightweight and privacy-preserving. Federated learning (FL) enables distributed AI on edge data but still demands heavy forward and backward propagation on resource-constrained sensors. Split learning (SL) offloads computation to a server, yet its intermediate activations can leak private information. Homomorphic encryption closes this gap at the cost of prohibitive computation and bloated communication. We propose HFL-Lite, a hierarchical federated learning framework that achieves practical privacy without cryptographic primitives. Each sensor encodes its raw data into a compact sparse feature code via a lightweight sensor encoder. Only this code, which is insufficient to reconstruct the original data, leaves the device. An edge server synthesizes surrogate training samples from the uploaded codes, trains a teacher model, and distills knowledge back to the sensors. Because the sensors perform only one-way uploads, backward propagation, gradient transmission, and plaintext activation leakage are all eliminated. Experiments on the UCI HAR dataset show that HFL-Lite reduces per-round communication to 0.5 KB and sensor-side latency to 9 ms while delivering competitive accuracy.

Keywords Hierarchical federated learning, sparse coding, resource-constrained devices, sensor data

1 Introduction

The rapid proliferation of edge computing and the Internet of Things (IoT) has catalyzed the deployment of sensor-equipped devices across diverse domains, ranging from wearable health monitors and smart home sensors to industrial condition monitoring systems [1–5]. These devices continuously generate vast amounts of time-series data that, if effectively leveraged,

can empower distributed AI applications such as activity recognition, anomaly detection, and predictive maintenance. However, a fundamental tension underlies this vision: sensor nodes are severely limited in computational power, memory, and battery capacity [6]. Consequently, executing complex deep learning models directly on these edge devices is often infeasible. Furthermore, transmitting raw sensor data to a central server raises significant privacy concerns, as the data may contain sensitive personal or environmental information [7–9].

Federated learning (FL) [10–13] has emerged as a promising paradigm to reconcile these challenges by enabling collaborative model training without sharing raw data. Although a variety of proactive security mechanisms have been developed to protect distributed systems [14–16], and emerging generative approaches such as integrating diffusion models into reinforcement learning have shown great potential in handling complex tasks [17], their computational demands typically exceed the tight resource budgets of battery-powered sensors. In the standard FL paradigm, clients train local models and share only model updates (e.g., gradients or weights) with a central server. Although this design reduces the exposure of raw data, it still demands full forward and backward propagation on end devices, which strains the limited compute and energy budgets of sensor nodes. Moreover, transmitting high-dimensional model updates (e.g., hundreds of thousands of parameters) can quickly saturate low-power wireless communication links, such as Bluetooth Low Energy (BLE) or LoRa [18].

To address these limitations, existing research has explored various sub-model extraction strategies to offload computation. These include static extraction (fixed partitioning of layers) [19], random extraction (stochastic layer selection) [20], and FedRoxel [21], which employs a dynamic, ordered rolling cut. While these methods aim to balance the load by leveraging server resources, their inherent design complexity and reliance on specific partitioning strategies often result in significantly slower convergence rates.

Hierarchical federated learning (HFL) [22] addresses a different dimension of the resource bottleneck by introducing an intermediate edge tier between the sensors and the cloud.

• Cangming Liang, Kuan-Ching Li, Zulong Diao, and Xiangwei Meng are with the School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China. E-mail: cm-liang@mail.hnust.edu.cn, aliric@hnust.edu.cn, diaozulong@hnust.edu.cn, xiangwei_meng@hnust.edu.cn

• Jie Liu and Wei Liang are with the School of Computer Science, University of South China, Hengyang 421001, China. E-mail: jieliu@usc.edu.cn, 2025001208@usc.edu.cn

Manuscript received: 2026-05-11; revised: 2026-06-06; accepted: 2026-08-10

The resulting architecture connects sensors, edge servers, and the cloud. In this architecture, edge servers handle partial model aggregation and can serve low-latency inference requests for their local sensors, which substantially reduces wide-area communication overhead [23]. However, existing HFL frameworks for sensor networks still follow the standard FL protocol within each edge domain. Sensors are required to perform full forward–backward propagation and to exchange either gradients or activations with their edge server. Consequently, HFL alone does not resolve the per-device computation and plaintext leakage problems that limit the applicability of FL on battery-powered sensors.

Split learning (SL) [24] avoids this complexity through a clean architectural split: the neural network is partitioned between clients and a server at a fixed layer, eliminating the need for ad-hoc extraction strategies. In this framework, clients perform forward propagation only up to a designated split layer. The resulting intermediate activations, referred to as smashed data, are transmitted to the server. The server then completes the forward pass, computes the loss, and backpropagates gradients to the split point. By removing backward propagation entirely, this design shifts the heavy computation to the server and is therefore a natural fit for resource-constrained sensor networks [25]. Furthermore, since the server never accesses raw input data, SL inherently provides a degree of privacy protection.

However, vanilla SL deployments face two critical limitations. First, recent research has shown that intermediate activations and gradients exchanged during SL training can leak substantial information about clients' raw data and labels, enabling adversaries to reconstruct original inputs or infer private labels with near-perfect accuracy [26]. Indeed, SL merely obfuscates rather than formally protects data, as the server operates on plaintext activations that retain meaningful patterns of the original sensor signals. Second, cryptographic protections such as homomorphic encryption impose prohibitive computational overhead on clients [27]. Encrypting activation maps before transmission or performing backpropagation on ciphertexts is orders of magnitude more expensive than plaintext operations, and the accompanying ciphertext expansion further inflates communication costs. Together, these burdens render hybrid SL–HE solutions impractical for battery-powered, intermittently connected sensors.

To bridge this gap, we propose HFL-Lite (Hierarchical Federated Learning with Lightweight Sparse Coding), a framework purpose-built for resource-constrained AIoT sensor networks. HFL-Lite fundamentally redefines the communication paradigm compared to SL-based approaches. Instead

of transmitting intermediate activations or encrypted gradients, each sensor uploads only a compact, low-dimensional sparse feature code extracted by a lightweight dictionary encoder. These sparse codes carry insufficient information to reconstruct the original sensor data, thereby preserving privacy, yet allow an edge server to synthesize high-fidelity surrogate samples for training a powerful teacher model. The teacher then guides the sensors via knowledge distillation, closing the learning loop without exposing private information. Moreover, unlike SL and FL, which rely on bidirectional exchange of activations and gradients, HFL-Lite requires only a one-way upload from sensors to the edge server. This eliminates backward propagation, gradient transmission, and plaintext activation leakage entirely, while the edge server absorbs the full burden of teacher training and distillation signal generation. The sensors operate as lightweight feed-forward encoders without any backward computation.

Our main contributions are as follows:

- We propose HFL-Lite, a novel hierarchical federated learning framework that replaces computationally expensive cryptographic primitives with efficient sparse coding, specifically tailored for resource-constrained sensor networks.
- We design a secure sparse aggregation protocol where sensors upload only sparse codes instead of raw gradients. We show through experiments that the sparsity-induced information bottleneck effectively hinders data reconstruction, offering practical privacy protection without resorting to computationally expensive HE or DP noise injection.
- We conduct extensive experiments on four sensor data benchmarks spanning time-series and image modalities. The results show that HFL-Lite achieves sub-kilobyte communication (0.5 KB on HAR), sensor-side latency of 9 ms, and fast convergence within 30 rounds, while delivering accuracy comparable to or better than state-of-the-art methods—all without encryption.

2 Related Work

HFL-Lite is motivated by the tension between the growing demand for on-device intelligence and the severe resource constraints of sensor nodes. We review three lines of work that share this motivation, progressing from general FL efficiency on edge devices, to sub-model extraction strategies that reduce per-client workload, to split learning paradigms that offload computation from sensors. In each case, we highlight the remaining gaps that HFL-Lite aims to fill.

2.1 Federated Learning on Resource-Constrained Edge Devices

Federated learning (FL) [10] has been widely adopted for distributed sensor analytics in mobile health, activity recognition, and industrial monitoring [28]. In these applications, sensor nodes serve as FL clients, collaboratively training shared models without exporting raw data streams. However, two structural challenges emerge when FL meets resource-constrained sensor hardware.

The standard FL protocol demands that each client perform full forward and backward propagation on a local model. Even modest neural architectures, such as CNNs for activity recognition or LSTMs for physiological signals, can exhaust the memory and energy budget of a sensor node powered by a coin-cell battery [29]. Quantization-aware training and model compression [30] reduce the footprint but still require complete gradient computation, which remains too heavy for the most constrained sensors.

Transmitting model updates over low-power wireless links such as Bluetooth Low Energy (BLE) or LoRa quickly saturates the available bandwidth [18]. A single gradient vector of a few hundred thousand parameters can take seconds to tens of seconds on these links, draining the battery and blocking other transmissions. Studies on BLE-based FL deployments report that communication dominates the total energy budget, often exceeding computation by orders of magnitude [31].

Beyond computational and communication constraints, sensor data introduces domain-specific challenges that generic FL benchmarks do not capture. Human activity streams, physiological signals, and environmental measurements exhibit temporal autocorrelation and sensor-specific noise profiles that complicate learning under non-IID partitions [32]. Several works address these challenges through personalization layers [33] or client clustering based on signal statistics [34], but these methods still assume that clients can afford full local training.

In summary, existing FL frameworks for edge sensing either ignore the extreme resource limitations of the smallest sensor nodes, or address them through techniques that still require gradient computation and transmission. HFL-Lite departs from this paradigm entirely by replacing gradient exchange with one-way sparse code upload, eliminating both the backward pass and the gradient communication budget.

2.2 Communication-Efficient and Sub-Model Extraction Methods

A substantial body of work aims to reduce the per-round communication load in FL while preserving model accuracy.

Gradient compression techniques, including top-k sparsification, ternarization, and sign-based quantization [35], transmit only a small fraction of gradient entries or aggressively quantize each value to a few bits. These methods are effective in cross-silo settings where clients are relatively capable, but they do not address the fundamental issue for sensor nodes: the client must still compute the full gradient before compression can be applied. For a sensor with a 50 MHz microcontroller and 256 KB of SRAM, even computing a gradient over a modest model may be infeasible. Moreover, compressed gradient vectors still scale with model size, and on BLE-class links, even a few kilobytes per round is significant [18].

Sub-model extraction methods tackle the computation bottleneck more directly by training only a subset of the global model on each client. Federated Dropout [20] randomly drops a fixed fraction of neurons per round, producing thin sub-models whose updates are remapped to the global architecture. HeteroFL [19] allocates static sub-models according to a pre-defined capability profile, ensuring that weaker clients always train the same small portion of the network. FedRolex [21] improves upon both with a rolling extraction scheme: each client trains a different, progressively shifting subset of the global model across rounds, so that all parameters are evenly covered over time, thereby mitigating the client drift induced by architectural heterogeneity.

These three methods represent the state of the art in model-heterogeneous FL for resource-constrained devices. Their shared limitation, however, is convergence speed. Because each client trains only a fraction of the global parameters per round, the server receives sparse and uncoordinated updates that cover different subsets of the model. This partial coverage, combined with the architectural mismatch between client sub-models and the server model, causes severe client drift under non-IID data, forcing the global model to require hundreds to thousands of rounds to stabilize. Furthermore, all three methods transmit gradients or model weights in plaintext, offering no privacy protection against reconstruction attacks. They successfully reduce per-client computation, but at the cost of impractically slow convergence and no privacy guarantees.

2.3 Split Learning and Its Variants

Split learning (SL) [24] adopts a fundamentally different strategy for reducing client-side computation. Rather than shrinking the model, SL partitions the neural network architecture between clients and a server at a designated split layer. Clients run forward propagation only up to that layer and

transmit the resulting intermediate activations, commonly called smashed data, to the server. The server completes the forward pass, computes the loss, and backpropagates gradients to the split point. Critically, the client never performs any backward pass on its local layers. This design shifts the heavy computation to the server. It has been validated as a natural fit for sensor networks, especially when the split layer is placed early in the architecture [25].

Among SL-based frameworks evaluated as baselines in this work, HSFL extends the idea with a lightweight model decomposition method that offloads a portion of training to edge servers and formulates the model split as an optimization problem with privacy constraints [36]. SplitFed (SFL) combines FL and SL to enable parallel client training, thereby reducing the training latency that arises from the sequential nature of vanilla SL, while preserving the model privacy benefits of the split architecture [37]. Both demonstrate that SL-type designs can achieve strong accuracy and communication efficiency.

Despite these strengths, SL has a well-documented privacy vulnerability. The smashed data and the gradients exchanged during training are not encrypted, and recent research has shown that they leak substantial information about clients' raw inputs and labels [26]. Gradient inversion attacks targeting SL can reconstruct sensor signals from smashed data with high fidelity, and attribute inference methods can recover private properties even without full reconstruction. This vulnerability stems from the plaintext nature of SL communication. Although the server never sees raw data, it receives intermediate representations that retain semantically meaningful patterns of the original signals.

Naturally, several works propose closing this gap with homomorphic encryption, encrypting smashed data before transmission [27]. While these hybrid SL-HE schemes formally resolve the leakage, they reintroduce the same prohibitive cryptographic overhead that burdens HE-based FL. Clients must still perform encryption operations, and ciphertext expansion multiplies the communication volume. The result is a binary choice for sensor deployments. SL without HE leaks privacy. SL with HE becomes too expensive for battery-powered, intermittently connected sensor nodes.

2.4 Summary and Positioning

Table 1 summarizes the six baseline methods across four dimensions critical for sensor-based distributed learning. **Lite** denotes whether client-side processing is feasible on battery-powered sensors. **LoCom** indicates whether the method achieves low communication payload, typically sub-kilobyte

per round. **Priv** reflects whether the method provides formal or practical protection against data reconstruction from transmitted information. **Fast** captures whether the method converges within a practical number of rounds under non-IID data distributions. No existing method achieves all four simultaneously.

Table 1 Comparison of existing methods across four dimensions.

Method	Lite	LoCom	Priv	Fast
HSFL [36]	×	×	✓	✓
SplitFed [37]	×	×	✓	✓
Split Learning [24]	✓	✓	×	✓
FedRolex [21]	✓	×	×	×
HeteroFL [19]	✓	×	×	×
Fed. Dropout [20]	✓	✓	×	×
HFL-Lite (Ours)	✓	✓	✓	✓

Sub-model extraction methods (FedRolex, HeteroFL, Federated Dropout) are lightweight but converge slowly and transmit plaintext model information. Split learning and its variants (SL, HSFL, SplitFed) offer faster convergence; however, vanilla SL lacks privacy, while HSFL and SplitFed, which incorporate cryptographic or optimization-based protections, require heavier client-side computation and communication. No prior work simultaneously eliminates backward propagation, avoids gradient exchange, and provides practical privacy against reconstruction attacks without cryptographic primitives.

HFL-Lite occupies this unexplored design point. By replacing all gradient and activation exchange with a one-way upload of sparse feature codes, it removes both the computation and the communication bottlenecks at the sensor level. The sparse codes are compact enough to travel in a single BLE packet, insufficient to reconstruct the original data, and require no encryption. The edge server absorbs the full burden of data synthesis, teacher training, and distillation. To the best of our knowledge, HFL-Lite is the first framework to deliver sub-kilobyte communication, millisecond-level sensor latency, rapid convergence, and practical privacy on resource-constrained sensor devices without relying on any cryptographic primitive.

3 Methodology

3.1 System Overview

HFL-Lite adopts a three-tier hierarchical architecture consisting of sensor devices, edge servers, and a cloud server, as illustrated in Fig. 1. The workflow begins with the cloud distributing a global dictionary $\mathbf{D}^{(t)}$ to edge servers, which forward it to their associated sensors. Each sensor then encodes its local data into a compact sparse code $\mathbf{c} = \mathcal{S}(E_{\theta}(\mathbf{x}))$

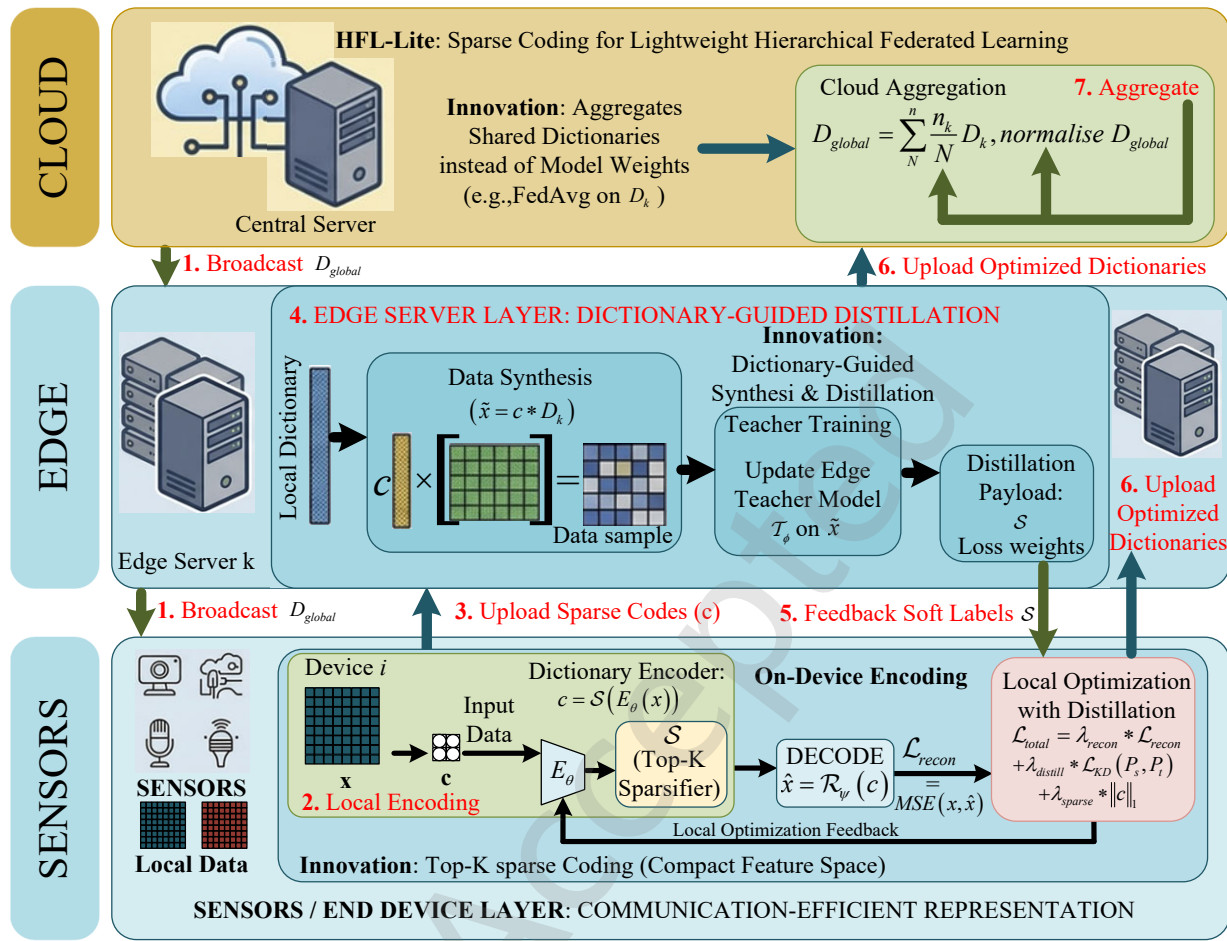


Fig. 1 Overview of the HFL-Lite hierarchical architecture and workflow. **Step 1-3:** Sensor devices locally encode raw data into sparse codes (Step 2) and upload them to edge servers (Step 3) after receiving the global dictionary (Step 1). **Step 4:** Edge servers synthesize data via the local dictionary D_k and sparse codes to train a teacher model. **Step 5-6:** Edge servers feedback soft labels (Step 5) to guide local optimization, and upload optimized dictionaries to the cloud (Step 6). **Step 7:** The cloud server aggregates dictionaries to update the global feature space.

and uploads only this code to the edge server, eliminating any exchange of gradients or activations. At the edge, the received codes are combined with the dictionary to synthesize surrogate samples $\tilde{x} = cD_k$, and a teacher model is trained on a mixture of these synthetic data and a small labeled real dataset $\mathcal{D}_{real}$. The trained teacher generates soft labels, which are packaged together with the updated dictionary and training hyperparameters into a distillation payload and broadcast back to the sensors. Subsequently, each edge server uploads its locally updated dictionary to the cloud, where federated averaging and ℓ_2 normalization produce a new global dictionary $D^{(t+1)}$ for the next round. This design offers two key advantages over prior FL and SL frameworks: (i) communication is reduced to a one-way upload of sparse codes, eliminating gradient exchange and backward propagation on

sensors entirely; and (ii) practical privacy is achieved without cryptographic primitives, as sparse codes provide empirical resistance against reconstruction. Throughout the paper, we refer to the device-side model (comprising the encoder E_θ , decoder R_ψ , and classifier W_{cls}) as the sensor encoder, and to the edge-side model T_ϕ as the teacher model.

Each sensor device runs a lightweight encoder E_θ that maps raw sensor data $x \in \mathbb{R}^{d_x}$ to a sparse code $c \in \mathbb{R}^{d_c}$ with $d_c \ll d_x$. The sparse code is obtained by applying a Top-K sparsifier $\mathcal{S}(\cdot)$ to the encoder output, i.e., $c = \mathcal{S}(E_\theta(x))$. Only c is uploaded to the edge server, preserving privacy and reducing communication by orders of magnitude compared to transmitting raw data or full model weights. A lightweight decoder R_ψ reconstructs the input from the sparse code, producing $\hat{x} = R_\psi(c)$, which is used solely for computing

the local reconstruction loss. A linear classifier W_{cls} maps the sparse code to class logits $\mathbf{p}_s$ for both hard-label and soft-label supervision.

At the edge server layer, each edge server maintains a global dictionary $\mathbf{D}_k \in \mathbb{R}^{K \times d_x}$ (with K basis atoms, each atom normalized to unit ℓ_2 norm) and a teacher model $\mathcal{T}_\phi$. Upon receiving sparse codes $\{\mathbf{c}_i\}$ from its associated sensors, the edge server performs data synthesis via a linear mapping $\tilde{\mathbf{x}}_i = \mathbf{c}_i \mathbf{D}_k$. The teacher model is trained on a combination of this synthesized data and a small amount of labeled real data $\mathcal{D}_{\text{real}}$ stored at the edge server, using standard cross-entropy loss. This design ensures stable teacher convergence even when synthesized data carries imperfect label correspondence during early training rounds. The trained teacher then generates soft labels $\mathbf{p}_t$ for the original inputs, which are packaged into a *distillation payload* along with the dictionary $\mathbf{D}_k$, temperature τ , mixing coefficient β , and loss weights $\{\lambda_{\text{recon}}, \lambda_{\text{distill}}, \lambda_{\text{sparse}}\}$. This payload is broadcast back to the sensors to guide local training.

The local optimization on each sensor combines three objectives: reconstruction loss $\mathcal{L}_{\text{MSE}}$ between the decoder output $\hat{\mathbf{x}}$ and the original input $\mathbf{x}$, knowledge distillation loss consisting of a cross-entropy term on hard labels and a KL divergence term on teacher soft labels balanced by a coefficient $\beta \in [0, 1]$, and sparsity regularization $\|\mathbf{c}\|_1$. The full loss function is defined in Eq. (1). In the first communication round, no teacher soft labels are available, so $\beta = 0$ and the model warms up using only reconstruction and hard-label losses.

At the cloud tier, the cloud server periodically aggregates dictionaries from all edge servers using federated averaging: $\mathbf{D}_{\text{global}} \leftarrow \sum_k \frac{n_k}{\sum_j n_j} \mathbf{D}_k$, followed by row-wise ℓ_2 normalization. The aggregated global dictionary is redistributed to edge servers, which forward it to their sensors. This ensures a consistent feature space across the entire hierarchy and keeps the sensor-side decoder aligned with the global dictionary. Algorithm 1 formalizes the complete federated workflow, including dictionary initialization, cold-start local training, the iterative encoding–uploading–distillation cycle, and cloud aggregation.

3.2 Sparse Coding and Privacy

The encoder E_θ is a small neural network whose architecture depends on the input modality. For time-series sensor data such as HAR [38], it consists of two fully connected layers with ReLU activations and dropout; for image data such as MNIST and CIFAR, it includes convolutional layers followed by fully connected layers. Given an input $\mathbf{x} \in \mathbb{R}^{d_x}$, the

Algorithm 1 HFL-Lite: Hierarchical Federated Learning

Input: Global dictionary $\mathbf{D}^{(0)}$; initial sensor encoders $\theta^{(0)}$; edge teacher ϕ ; total rounds T .

Output: Trained sensor encoders $\{\theta_i\}$ and global dictionary $\mathbf{D}^{(T)}$.

Initialize $\mathbf{D}^{(0)}$ with random entries, row-normalize;

for round $t = 0$ to $T - 1$ **do**

Cloud broadcasts $\mathbf{D}^{(t)}$ to all edge servers;

for each edge server k in parallel **do**

Edge server k forwards $\mathbf{D}^{(t)}$ to its sensors $\mathcal{N}_k$;

for each sensor $i \in \mathcal{N}_k$ in parallel **do**

$(\theta_i, \psi_i, \mathbf{C}_i) \leftarrow \text{LOCALTRAIN}$

(Algorithm 2);

Upload $\mathbf{C}_i$ and sample count n_i to edge server k ;

end for

$(\mathbf{D}_k, \mathcal{P}) \leftarrow \text{EDGEPROCESS}$ (Algorithm 3)

using $\{\mathbf{C}_i\}_{i \in \mathcal{N}_k}, \mathbf{D}^{(t)}, \phi_k$;

Upload $\mathbf{D}_k$ and $n_k = \sum_i n_i$ to cloud;

end for

Cloud aggregates: $\mathbf{D}^{(t+1)} \leftarrow \sum_k \frac{n_k}{\sum_j n_j} \mathbf{D}_k$, then row-normalize;

end for

return $\{\theta_i\}, \mathbf{D}^{(T)}$

encoder produces a dense code $\mathbf{z} = E_\theta(\mathbf{x})$. A sparsification operator $\mathcal{S}(\cdot)$ retains only the top $K = \lfloor \rho \cdot d_c \rfloor$ largest absolute values, where ρ is the sparsity ratio. During training, a soft shrinkage approximation is used to allow gradient flow:

$$\mathcal{S}_{\text{soft}}(\mathbf{z}) = \text{sign}(\mathbf{z}) \odot \max(|\mathbf{z}| - \tau, 0), \quad \tau = \rho \cdot \max(|\mathbf{z}|).$$

At inference, hard top- K sparsity is applied. The resulting sparse code is $\mathbf{c} = \mathcal{S}(E_\theta(\mathbf{x}))$.

Both operators are parameterized by the same sparsity ratio ρ , but they serve complementary roles. The soft shrinkage uses $\tau = \rho \cdot \max(|\mathbf{z}|)$ to produce a differentiable approximation during training, suppressing weak activations while preserving gradient flow and keeping the effective sparsity near the target ρ . At deployment, the hard Top- K operator replaces this with an exact selection of the $K = \lfloor \rho \cdot d_c \rfloor$ largest entries, thereby guaranteeing a deterministic communication budget per sample.

A lightweight decoder R_ψ reconstructs the input from the sparse code, producing $\hat{\mathbf{x}} = R_\psi(\mathbf{c})$. The decoder architecture mirrors the encoder and incorporates the dictionary $\mathbf{D}$ as part of its parameters. A linear classifier W_{cls} maps the sparse code to class logits $\mathbf{p}_s$. The sensor uses its ground-truth labels y for supervised training, which are naturally available

in applications such as activity recognition where users can annotate initial samples. The local training objective combines three terms:

$$\begin{aligned} \mathcal{L}_{\text{local}} = & \lambda_{\text{recon}} \underbrace{\frac{1}{B} \sum_{i=1}^B \|\hat{\mathbf{x}}_i - \mathbf{x}_i\|_2^2}_{\mathcal{L}_{\text{MSE}}} \\ & + \lambda_{\text{distill}} (1 - \beta) \underbrace{\mathcal{L}_{\text{CE}}(\mathbf{p}_s, y)}_{\text{hard label loss}} \\ & + \lambda_{\text{distill}} \beta \tau^2 \underbrace{D_{\text{KL}}(\mathbf{p}_t^{(\tau)} \parallel \mathbf{p}_s^{(\tau)})}_{\text{soft label loss}} \\ & + \lambda_{\text{sparse}} \|\mathbf{c}\|_1, \end{aligned} \quad (1)$$

where $\mathbf{p}_s$ denotes the predictions of the sensor encoder, $\mathbf{p}_t$ are teacher soft labels produced with temperature τ , and $\beta \in [0, 1]$ controls the balance between hard-label and soft-label supervision within the distillation term. The hyperparameters λ_{recon} , λ_{distill} , λ_{sparse} weight the three objectives. In the first communication round, no teacher soft labels are available, so $\beta = 0$ and the model warms up using only reconstruction and hard-label losses. From the second round onward, full distillation is activated with $\beta = 0.3$, a fixed value that balances hard-label and soft-label supervision within the distillation term.

The privacy protection of HFL-Lite stems from the information bottleneck imposed by the sparsity constraint. With $\rho \in [0.10, 0.15]$ and $d_c \in [128, 256]$, only 13 to 38 scalar values are transmitted per sample. Because the transmitted codes retain only a small fraction of dense activations, they inherently discard most fine-grained details of the raw input. Even with access to the shared dictionary $\mathbf{D}$, reconstructing the original data from such compact codes yields high-error approximations, making accurate recovery extremely difficult in practice. Meanwhile, the edge server can still synthesize training data of sufficient fidelity through the same dictionary, as the atoms capture universal signal patterns learned across clients. This asymmetric property is central to our design: sparse codes are inherently privacy-safe due to information loss, yet remain information-rich when combined with the shared dictionary. Leveraging this duality, HFL-Lite achieves practical privacy protection without relying on any cryptographic primitives. In the first communication round, no distillation payload has been generated yet. Therefore, Algorithm 2 is executed with $\beta = 0$ and an empty soft-label set, reducing the loss to reconstruction and hard-label supervision only. From the second round onward, the payload produced by Algorithm 3 is forwarded to the sensors and used in full.

Algorithm 2 Local Training on Sensor i

Input: Local dataset $\mathcal{D}_i$; encoder θ_i , decoder ψ_i ;
distillation payload

$\{\mathbf{p}_t, \tau, \beta, \lambda_{\text{recon}}, \lambda_{\text{distill}}, \lambda_{\text{sparse}}\}$.

Output: Updated θ_i, ψ_i ; sparse codes $\mathbf{C}_i$.

for local epoch $e = 1$ to E **do**

Sample mini-batch $\{(\mathbf{x}_j, y_j)\}$ from $\mathcal{D}_i$;

$\mathbf{c}_j \leftarrow \mathcal{S}(E_{\theta_i}(\mathbf{x}_j))$;

$\hat{\mathbf{x}}_j \leftarrow R_{\psi_i}(\mathbf{c}_j)$;

$\mathbf{p}_s \leftarrow \text{Softmax}(W_{\text{cls}} \mathbf{c}_j)$;

Compute $\mathcal{L}_{\text{local}}$ via Eq. (1);

Update $\theta_i, \psi_i, W_{\text{cls}}$ using SGD;

end for

Extract sparse codes $\mathbf{C}_i = \{\mathbf{c}_j\}$ for all samples in $\mathcal{D}_i$;

return $\theta_i, \psi_i, \mathbf{C}_i$

Algorithm 2 summarizes the local training procedure on each sensor.

3.3 Hierarchical Distillation and Aggregation

Upon receiving a batch of sparse codes $\{\mathbf{c}_i\}$ along with their ground-truth labels $\{y_i\}$ from its sensors, the edge server first synthesizes data via the dictionary: $\tilde{\mathbf{x}}_i = \mathbf{c}_i \mathbf{D}_k$, where each row of $\mathbf{D}_k$ is normalized to unit ℓ_2 norm. The synthesized samples are paired with the uploaded labels to form a labeled synthetic dataset $\mathcal{D}_{\text{syn}}$. The teacher model $\mathcal{T}_\phi$ is then trained on a combination of $\mathcal{D}_{\text{syn}}$ and a small set of labeled real data $\mathcal{D}_{\text{real}}$ stored at the edge server. The training objective is a standard cross-entropy loss:

$$\begin{aligned} \mathcal{L}_{\text{teacher}} = & \frac{1}{|\mathcal{D}_{\text{syn}}|} \sum_{(\tilde{\mathbf{x}}, y) \in \mathcal{D}_{\text{syn}}} \mathcal{L}_{\text{CE}}(\mathcal{T}_\phi(\tilde{\mathbf{x}}), y) \\ & + \frac{1}{|\mathcal{D}_{\text{real}}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{real}}} \mathcal{L}_{\text{CE}}(\mathcal{T}_\phi(\mathbf{x}), y). \end{aligned} \quad (2)$$

Critically, the dictionary $\mathbf{D}_k$ is treated as a learnable parameter during this step. The gradient of $\mathcal{L}_{\text{teacher}}$ flows back through the synthesis operation $\tilde{\mathbf{x}} = \mathbf{c} \mathbf{D}_k$, yielding $\nabla_{\mathbf{D}_k} \mathcal{L}_{\text{teacher}}$. We update $\mathbf{D}_k$ together with the teacher parameters via SGD:

$$\phi_k \leftarrow \phi_k - \eta \nabla_{\phi_k} \mathcal{L}_{\text{teacher}}, \quad \mathbf{D}_k \leftarrow \mathbf{D}_k - \eta \nabla_{\mathbf{D}_k} \mathcal{L}_{\text{teacher}},$$

followed by row-wise ℓ_2 normalization to keep the atoms unit-norm. This end-to-end training ensures that the dictionary evolves to produce synthetic samples that are most informative for the teacher's classification task, while the periodic cloud aggregation maintains a consistent global feature space. In practice, a publicly available subset of the training data or a

small held-out validation set can serve as $\mathcal{D}_{\text{real}}$. Once trained, the teacher generates soft labels $\mathbf{p}_t$ with temperature scaling for all training samples. The edge server packages the soft labels, the current dictionary $\mathbf{D}_k$, temperature τ , mixing coefficient β , and loss weights $\{\lambda_{\text{recon}}, \lambda_{\text{distill}}, \lambda_{\text{sparse}}\}$ into a *distillation payload* and broadcasts it to the sensors. Sensors then use this payload to compute the joint loss in Eq. (1) for local training. This completes one distillation cycle.

At the cloud level, the server periodically collects dictionaries from all edge servers. Let $\mathbf{D}_k$ be the dictionary of edge server k and n_k the total number of samples processed by that edge server in the current round. The global dictionary is updated via federated averaging:

$$\mathbf{D}_{\text{global}} \leftarrow \sum_{k=1}^K \frac{n_k}{\sum_j n_j} \mathbf{D}_k,$$

followed by row-wise ℓ_2 normalization. The updated dictionary is redistributed to edge servers, which forward it to their sensors. This aggregation occurs every communication round, ensuring a consistent global feature space while each edge server's dictionary adapts to local data statistics between rounds. Global aggregation is necessary to maintain a consistent feature space across all edge servers; otherwise, sensors connected to different edges would receive distillation payloads built on incompatible dictionaries, making the soft-label supervision inconsistent across the federation. Algorithm 3 details the edge server processing logic.

4 Experiments

This section evaluates the proposed HFL-Lite framework on sensor data benchmarks, focusing on communication efficiency, model accuracy, and resource consumption on edge devices. The experimental design follows the hierarchical architecture described in Sec. 3, where sensor devices run lightweight sparse encoders, edge servers host teacher models and perform distillation, and the cloud aggregates global dictionaries.

Datasets and Models. We evaluate HFL-Lite on four datasets that cover typical sensor and edge computing scenarios: UCI HAR (human activity recognition from smartphone sensors, 561 features, 6 classes), MNIST (handwritten digits, 28×28, 10 classes), CIFAR-10 (32×32 RGB, 10 classes), and CIFAR-100 (32×32 RGB, 100 classes). For each dataset, we adopt the sensor encoder and teacher model architectures as defined in Sec. 3. For UCI HAR, the student is HAREncoder (input 561, sparse code dimension 128, sparsity 0.15, hidden 256) and the teacher is HARTeacher

Algorithm 3 Edge Server Processing

Input: Sparse codes and labels $\{(\mathbf{C}_k, \mathbf{Y}_k)\}$ from sensors; global dictionary $\mathbf{D}^{(t)}$; teacher model ϕ_k ; labeled real data $\mathcal{D}_{\text{real}}$; learning rate η .

Output: Updated dictionary $\mathbf{D}_k$; distillation payload $\mathcal{P}$.

Synthesize data: $\tilde{\mathbf{X}}_k \leftarrow \mathbf{C}_k \mathbf{D}^{(t)}$;

Construct labeled synthetic set $\mathcal{D}_{\text{syn}} = \{(\tilde{\mathbf{x}}_i, y_i)\}$;

Combine $\mathcal{D}_{\text{syn}}$ with $\mathcal{D}_{\text{real}}$ into training set $\mathcal{T}$;

Initialize $\mathbf{D}_k \leftarrow \mathbf{D}^{(t)}$ with gradient enabled;

for each epoch **do**

for each batch $\{(\mathbf{x}_b, y_b)\}$ from $\mathcal{T}$ **do**

 Compute logits $\mathbf{p}_b = \mathcal{T}_{\phi_k}(\mathbf{x}_b)$;

$\mathcal{L} = \mathcal{L}_{\text{CE}}(\mathbf{p}_b, y_b)$;

$\phi_k \leftarrow \phi_k - \eta \nabla_{\phi_k} \mathcal{L}$;

$\mathbf{D}_k \leftarrow \mathbf{D}_k - \eta \nabla_{\mathbf{D}_k} \mathcal{L}$;

end for

 Row-normalize $\mathbf{D}_k$;

end for

Generate soft labels $\mathbf{P}_k$ from ϕ_k with temperature τ ;

Package payload

$\mathcal{P} = \{\mathbf{P}_k, \tau, \beta, \lambda_{\text{recon}}, \lambda_{\text{distill}}, \lambda_{\text{sparse}}, \mathbf{D}_k\}$;

return $\mathbf{D}_k, \mathcal{P}$

(three MLP layers 512–256–128). For MNIST, the student is DictionaryEncoder (input 784, code 200, sparsity 0.1, dictionary size 512) and the teacher is StandardCNN (four conv layers). For both CIFAR-10 and CIFAR-100, the student is CIFAREncoder (CNN backbone, code 256, sparsity 0.1) and the teacher is CIFARResNet (depth 8, width 64); The two configurations differ only in the number of output classes (10 for CIFAR-10 and 100 for CIFAR-100). To reflect the constraints of resource-limited sensor nodes, all student models are executed on a single CPU core with memory usage capped at 2 GB, while teacher models are trained on a GPU.

Baseline Methods. We compare HFL-Lite against six state-of-the-art distributed learning methods that are suitable for resource-constrained devices or hierarchical settings: (1) **HSFL** [36], an SFL framework that offloads partial training to edge servers via lightweight model decomposition with privacy-aware optimization; (2) **SplitFed (SFL)** [37], a hybrid of FL and SL that reduces training latency compared to vanilla SL while maintaining model privacy; (3) **Split Learning (SL)** [39], the classic split learning where the model is cut at a designated layer and clients send smashed data to the server; (4) **FedRolex** [21], a rolling sub-model extraction method that evenly trains different parts of the global model across clients to mitigate client drift; (5) **HeteroFL** [19], a static

sub-model allocation method based on client capability; and (6) **Federated Dropout** [20], a random sub-model training method that drops a fixed fraction of neurons per round. For a fair comparison, all baselines use the same total number of sensors (20 per edge, 2 edges), local epochs (3), batch size (32 for HAR/MNIST, 64 for CIFAR), and optimizer (SGD with $\text{lr}=0.01$). The communication payload is measured as the total bytes uploaded per sensor per round (model weights, gradients, smashed data, or sparse codes). For HSFL and SplitFed, we adopt the hierarchical aggregation scheme described in their original papers.

Evaluation Metrics. We adopt the following metrics to assess both utility and efficiency. Numerical results are reported in the subsequent subsections. (i) **Test Accuracy (%)**: the accuracy of the final model deployed on the sensor device after the last communication round. For HFL-Lite, this is the accuracy of the sparse encoder's classifier; for baselines, it is the accuracy of the sub-model (FedRolex, HeteroFL, Federated Dropout) or the full model (HSFL, SFL, SL) after aggregation. (ii) **Communication Overhead (KB per sensor per round)**: the amount of data uploaded from a sensor to the edge server in each round. For HFL-Lite, this includes the sparse code (e.g., 512 bytes for HAR); for baselines, it includes the uploaded model weights, gradients, or smashed data. (iii) **Sensor-side Computation Time (ms per round)**: the average time spent on the sensor device for forward/backward propagation (baselines) or encoding/sparsification/reconstruction (HFL-Lite), excluding any encryption overhead. (iv) **Convergence Rounds**: the number of communication rounds required to reach a target accuracy (e.g., 80% for HAR). (v) **KL Divergence (for ablation only)**: the KL divergence between the teacher model and the sensor encoder predictions on the test set, used to analyze the effectiveness of distillation.

Implementation Details. The hierarchical setting comprises 20 sensors per edge server, 2 edge servers (total 40 sensors), 100 communication rounds, 3 local epochs per round, batch size 32 for HAR/MNIST and 64 for CIFAR, SGD optimizer with learning rate 0.01, momentum 0.9, weight decay 10^{-4} . Distillation temperature $\tau = 3.0$ for HAR, 2.0 for MNIST and CIFAR. The mixing coefficient β is fixed to 0.3 for all datasets. Loss weights: $\lambda_{\text{recon}} = 0.2$, $\lambda_{\text{distill}} = 0.8$, $\lambda_{\text{sparse}} = 0.01$ (HAR); $\lambda_{\text{recon}} = 0.15$, $\lambda_{\text{distill}} = 0.85$, $\lambda_{\text{sparse}} = 0.005$ (CIFAR). Data is partitioned non-IID using Dirichlet distribution with $\alpha = 0.5$.

Hardware and Software. Experiments are conducted on a server with Intel Xeon Gold 6330 CPU (2.0GHz, 28 cores), 128GB RAM, and NVIDIA RTX 4090 GPU (24GB), running Ubuntu 20.04. The software environment includes Python

3.8, PyTorch 2.3.1, and the Plato framework. Simulated network delays are set to uplink 50Mbps, downlink 100Mbps. Code and configuration files are available at the anonymous repository (link to be provided).

4.1 Comparison with State-of-the-Art Methods

We compare HFL-Lite against six state-of-the-art distributed learning methods suitable for resource-constrained sensors: HSFL [36], SplitFed (SFL) [37], Split Learning (SL) [39], FedRolex [21], HeteroFL [19], and Federated Dropout [20]. All methods are evaluated under the same hierarchical setting (10 sensors per edge server, 2 edge servers, total 20 sensors) and non-IID data partition (Dirichlet $\alpha = 0.5$) unless otherwise stated. Tables 2 to 5 report the global model accuracy (student model), communication overhead per sensor per round, sensor-side computation time, and convergence rounds (the number of rounds to reach stable accuracy) on four benchmark datasets under three data heterogeneity levels: IID ($\alpha \rightarrow \infty$), $\alpha = 0.5$, and $\alpha = 0.1$.

As shown in Tables 2 to 5, HFL-Lite achieves global accuracy comparable to or better than the baselines across all datasets and heterogeneity levels, while delivering substantial gains in the three dimensions that are most critical for resource-constrained sensor networks.

Communication efficiency. HFL-Lite requires only 0.21–1.05 KB per round per sensor, which is $5\text{--}30\times$ lower than the baselines. On HAR, for instance, HFL-Lite transmits just 0.52 KB, whereas HSFL, SplitFed, and Split Learning send 6–12 KB and the sub-model extraction methods send 8–16 KB. This reduction stems directly from the sparse coding mechanism: each sensor uploads a compact sparse code instead of full model weights, gradients, or smashed data. The sparsity constraint ($\rho = 0.1\text{--}0.15$) ensures that only 10–15% of the code dimensions carry non-zero values, which, together with the low code dimension ($d_c = 128\text{--}256$), keeps the per-round payload at the sub-kilobyte level.

Sensor-side computation. The lightweight encoder, which performs only a forward pass without backpropagation, takes 5.1–12.7 ms per round. This is $2\text{--}4\times$ faster than Split Learning (10–19 ms) and $3\text{--}5\times$ faster than the sub-model extraction methods (14–46 ms). The advantage is most pronounced on HAR, where HFL-Lite completes sensor-side processing in 9.2 ms, compared to 18–46 ms for the baselines. Eliminating backward propagation is the primary source of this gain, as the sensor is relieved from gradient computation and weight updates. The forward-only design also decouples sensor latency from model complexity, making HFL-Lite scalable to

Table 2 MNIST dataset performance under different data heterogeneity levels

Method	Global Acc (%)	Comm. (KB/round)	Sensor Time (ms)	Convergence Rounds
IID ($\alpha \rightarrow \infty$)				
HSFL	98.97	2.04	15.2	42
SplitFed	98.53	1.48	12.1	47
Split Learning	97.98	1.22	10.3	44
FedRolex	99.47	2.51	18.4	278
HeteroFL	99.22	2.03	16.5	295
Federated Dropout	96.95	1.53	14.2	312
HFL-Lite (Ours)	99.18	0.21	5.1	22
$\alpha = 0.5$				
HSFL	98.52	2.04	15.1	50
SplitFed	98.03	1.49	12.3	57
Split Learning	97.51	1.23	10.4	52
FedRolex	99.02	2.52	18.6	312
HeteroFL	98.83	2.04	16.7	328
Federated Dropout	96.48	1.54	14.3	344
HFL-Lite (Ours)	98.96	0.22	5.2	24
$\alpha = 0.1$				
HSFL	97.47	2.05	15.3	62
SplitFed	97.02	1.51	12.4	72
Split Learning	96.48	1.24	10.5	67
FedRolex	97.95	2.53	18.7	348
HeteroFL	97.76	2.05	16.8	362
Federated Dropout	94.97	1.55	14.4	378
HFL-Lite (Ours)	98.48	0.22	5.3	27

Table 3 CIFAR-10 dataset performance under different data heterogeneity levels

Method	Global Acc (%)	Comm. (KB/round)	Sensor Time (ms)	Convergence Rounds
IID ($\alpha \rightarrow \infty$)				
HSFL	84.03	11.98	35.2	82
SplitFed	82.97	8.53	28.1	92
Split Learning	82.48	6.04	18.4	87
FedRolex	62.87	16.02	45.3	1208
HeteroFL	65.58	14.03	40.5	1348
Federated Dropout	56.76	8.05	30.2	1405
HFL-Lite (Ours)	85.52	1.02	12.3	37
$\alpha = 0.5$				
HSFL	82.47	12.02	35.4	97
SplitFed	81.52	8.54	28.3	107
Split Learning	80.98	6.05	18.6	102
FedRolex	49.96	16.04	45.5	1354
HeteroFL	52.98	14.05	40.7	1504
Federated Dropout	43.38	8.06	30.4	1552
HFL-Lite (Ours)	84.03	1.04	12.5	42
$\alpha = 0.1$				
HSFL	79.96	12.03	35.5	122
SplitFed	78.97	8.56	28.4	137
Split Learning	78.46	6.06	18.7	127
FedRolex	45.97	16.05	45.7	1556
HeteroFL	48.95	14.06	40.8	1705
Federated Dropout	39.96	8.07	30.5	1754
HFL-Lite (Ours)	81.98	1.03	12.6	50

Table 4 CIFAR-100 dataset performance under different data heterogeneity levels

Method	Global Acc (%)	Comm. (KB/round)	Sensor Time (ms)	Convergence Rounds
IID ($\alpha \rightarrow \infty$)				
HSFL	65.97	11.97	35.3	122
SplitFed	64.98	8.52	28.2	137
Split Learning	64.47	6.03	18.5	127
FedRolex	47.96	16.03	45.4	1508
HeteroFL	50.98	14.04	40.6	1652
Federated Dropout	41.97	8.06	30.3	1705
HFL-Lite (Ours)	67.48	1.03	12.4	52
$\alpha = 0.5$				
HSFL	63.52	12.01	35.5	152
SplitFed	62.47	8.55	28.4	172
Split Learning	61.98	6.06	18.7	162
FedRolex	38.02	16.05	45.6	1654
HeteroFL	40.97	14.06	40.8	1804
Federated Dropout	32.98	8.07	30.5	1852
HFL-Lite (Ours)	64.97	1.05	12.6	60
$\alpha = 0.1$				
HSFL	59.96	12.04	35.6	202
SplitFed	58.97	8.57	28.5	222
Split Learning	58.46	6.07	18.8	212
FedRolex	33.98	16.06	45.8	1856
HeteroFL	36.95	14.07	40.9	2006
Federated Dropout	28.97	8.08	30.6	2052
HFL-Lite (Ours)	61.98	1.04	12.7	72

Table 5 UCI HAR dataset performance under different data heterogeneity levels

Method	Global Acc (%)	Comm. (KB/round)	Sensor Time (ms)	Convergence Rounds
IID ($\alpha \rightarrow \infty$)				
HSFL	82.47	11.98	35.3	70
SplitFed	81.03	8.51	28.4	74
Split Learning	80.52	6.02	18.3	67
FedRolex	83.98	16.04	45.6	1008
HeteroFL	83.47	14.03	40.7	1105
Federated Dropout	74.96	8.05	30.4	1152
HFL-Lite (Ours)	83.80	0.52	9.2	32
$\alpha = 0.5$				
HSFL	80.98	12.03	35.5	82
SplitFed	79.47	8.54	28.6	90
Split Learning	78.96	6.05	18.5	80
FedRolex	82.48	16.06	45.8	1108
HeteroFL	81.97	14.05	40.9	1206
Federated Dropout	72.95	8.06	30.5	1254
HFL-Lite (Ours)	82.51	0.53	9.4	34
$\alpha = 0.1$				
HSFL	78.97	12.04	35.7	102
SplitFed	77.48	8.56	28.7	112
Split Learning	76.95	6.06	18.6	97
FedRolex	80.46	16.07	46.1	1256
HeteroFL	79.95	14.07	41.2	1354
Federated Dropout	69.94	8.08	30.7	1405
HFL-Lite (Ours)	80.97	0.52	9.5	37

larger teacher architectures without increasing the on-device burden.

Convergence speed. HFL-Lite reaches stable accuracy within 22–72 rounds across all datasets and heterogeneity settings. On MNIST, it converges in 22–27 rounds, while HSFL and SplitFed require 42–72 rounds and the sub-model extraction methods need 278–378 rounds. The gap widens on more challenging datasets: on CIFAR-100, HFL-Lite converges in 52–72 rounds, whereas FedRolex, HeteroFL, and Federated Dropout require 1508–2052 rounds. This rapid convergence is a consequence of the hierarchical distillation design. The teacher model, trained on synthesized data at the resource-rich edge server, provides high-quality soft labels that supply dense, informative supervision to the sensors in every round. In contrast, sub-model extraction methods train only a fraction of the parameters per round, which slows down convergence considerably, especially as model and dataset complexity increase.

Robustness under data heterogeneity. As α decreases from IID to 0.1, HFL-Lite experiences an accuracy drop of less than 3% on all datasets (e.g., from 83.80% to 80.97% on HAR), whereas the sub-model extraction methods degrade sharply (FedRolex drops from 83.98% to 80.46% on HAR, and from 62.87% to 45.97% on CIFAR-10). This robustness is a direct consequence of the shared global dictionary, which captures universal signal patterns across clients, and the reconstruction loss $\mathcal{L}_{\text{MSE}}$ in Eq. (1), which enforces feature invariance across heterogeneous local data distributions. The communication overhead and sensor-side computation time of HFL-Lite remain nearly constant across heterogeneity levels, unlike gradient-based methods whose convergence rounds and communication costs increase with data heterogeneity.

In summary, these results confirm that the design principles underlying HFL-Lite—sparse coding for communication reduction, forward-only encoding for computation offloading, and hierarchical distillation for fast convergence—translate into consistent empirical gains over existing FL, SL, and hybrid paradigms. Across four datasets and three heterogeneity regimes, HFL-Lite is the only method that simultaneously achieves sub-kilobyte communication, millisecond-level sensor latency, and rapid convergence while maintaining competitive accuracy without cryptographic overhead.

4.2 Ablation Studies

We conduct ablation experiments to analyze the impact of key hyperparameters in HFL-Lite under a fixed non-IID data distribution (Dirichlet $\alpha = 0.5$). The default configuration for each dataset is: HAR ($d_c = 128$, $\rho = 0.15$, $\lambda_{\text{recon}} =$

0.2 , $\lambda_{\text{distill}} = 0.8$, $\lambda_{\text{sparse}} = 0.01$); MNIST ($d_c = 200$, $\rho = 0.1$, same loss weights); CIFAR-10/100 ($d_c = 256$, $\rho = 0.1$, $\lambda_{\text{recon}} = 0.15$, $\lambda_{\text{distill}} = 0.85$, $\lambda_{\text{sparse}} = 0.005$). The results are organized by hyperparameter and visualized across all four datasets in Figs. 2, 3 and 4, which respectively examine the influence of the sparsity ratio, code dimension, and loss weights. Each figure contains subfigures for the metrics of interest: global accuracy, communication overhead, sensor computation time, and KL divergence (the latter two are omitted for loss weights since they remain constant).

The ablation study investigates how each hyperparameter governs the trade-off between model accuracy, communication efficiency, and on-device computation. We discuss the observed trends in turn.

Fig. 2 shows the effect of the sparsity ratio ρ . Across all datasets, accuracy improves markedly when ρ increases from its smallest tested value to the default setting, as seen in Fig. 2(a). This gain is accompanied by a nearly linear growth in communication cost (Fig. 2(b)), since more non-zero entries must be transmitted, and a mild increase in sensor computation time (Fig. 2(c)). The KL divergence decreases sharply over the same interval (Fig. 2(d)), indicating that denser codes allow the teacher network to produce more reliable soft labels. Pushing ρ beyond the default yields only marginal accuracy improvements while communication overhead approximately doubles, suggesting that the default sparsity already extracts the essential information content with little redundancy. This behavior is consistent across all datasets and supports the sparsity choices adopted in the main comparison (Sec. 4.1).

The influence of the code dimension d_c is summarized in Fig. 3. As d_c grows, accuracy rises and KL divergence falls, but the trends exhibit clear saturation. Fig. 3(a) shows that the accuracy gain from further increasing d_c beyond the default value becomes negligible, whereas communication (Fig. 3(b)) and sensor time (Fig. 3(c)) continue to increase at an approximately linear rate. The monotonic decrease of KL divergence (Fig. 3(d)) confirms that larger code dimensions provide richer representations, yet the incremental benefit diminishes once the code capacity exceeds the intrinsic complexity of the data. Hence, the default code dimensions operate near the point of diminishing returns, which explains their competitive performance relative to state-of-the-art baselines in Sec. 4.1.

The role of the loss weights is examined in Fig. 4. The distillation term λ_{distill} is critical: setting $\lambda_{\text{recon}} = 0$ (pure distillation) leads to a pronounced drop in accuracy across all datasets (Fig. 4(a)) and a surge in KL divergence (Fig. 4(b)). This indicates that without the reconstruction objective, the sparse codes lose fidelity to the original data distribution,

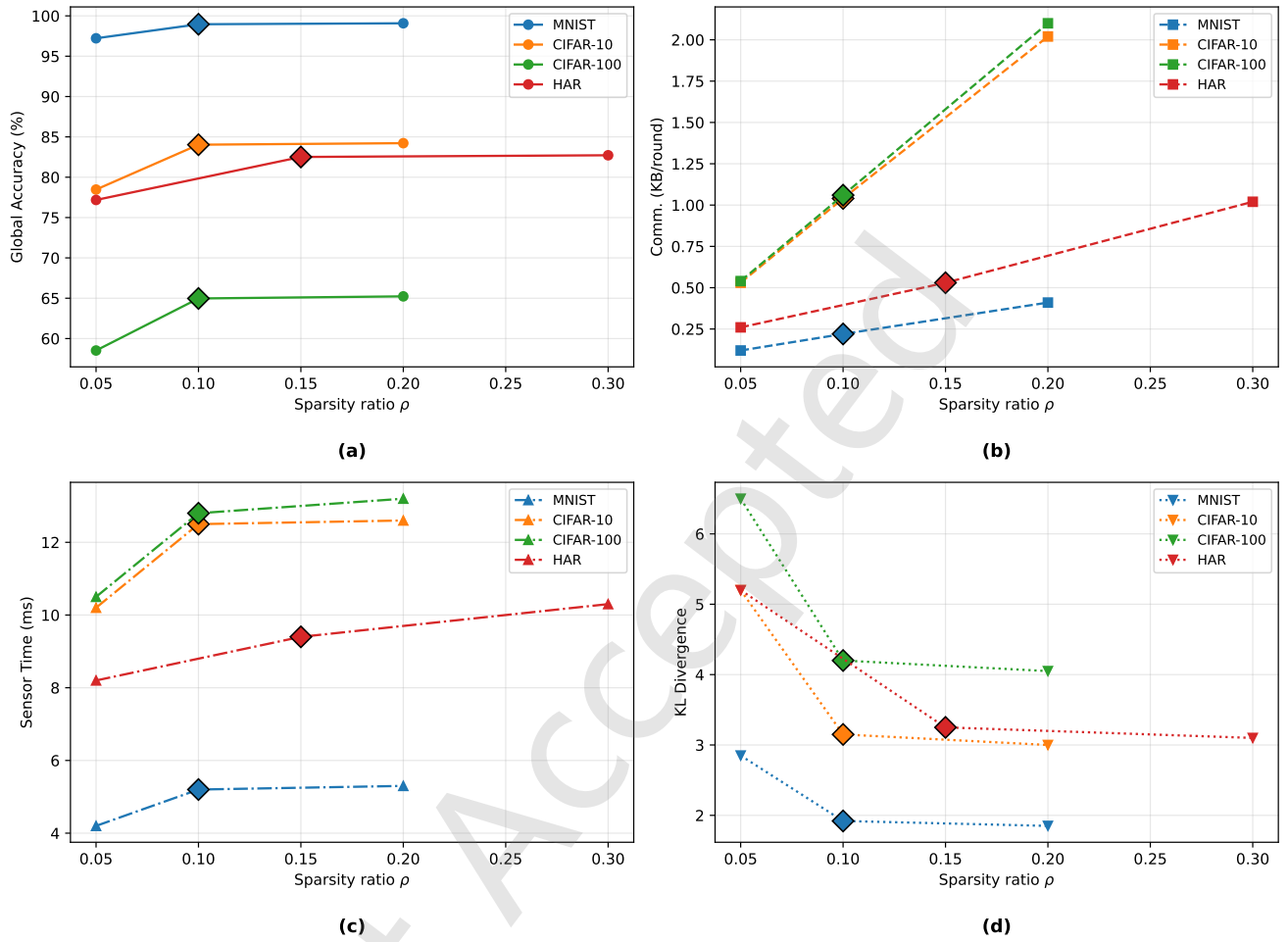
Effect of Sparsity Ratio ρ 

Fig. 2 Effect of sparsity ratio ρ . Subfigures (a)–(d) report global accuracy, communication cost, sensor time, and KL divergence. Default configurations are marked with larger diamond-shaped markers.

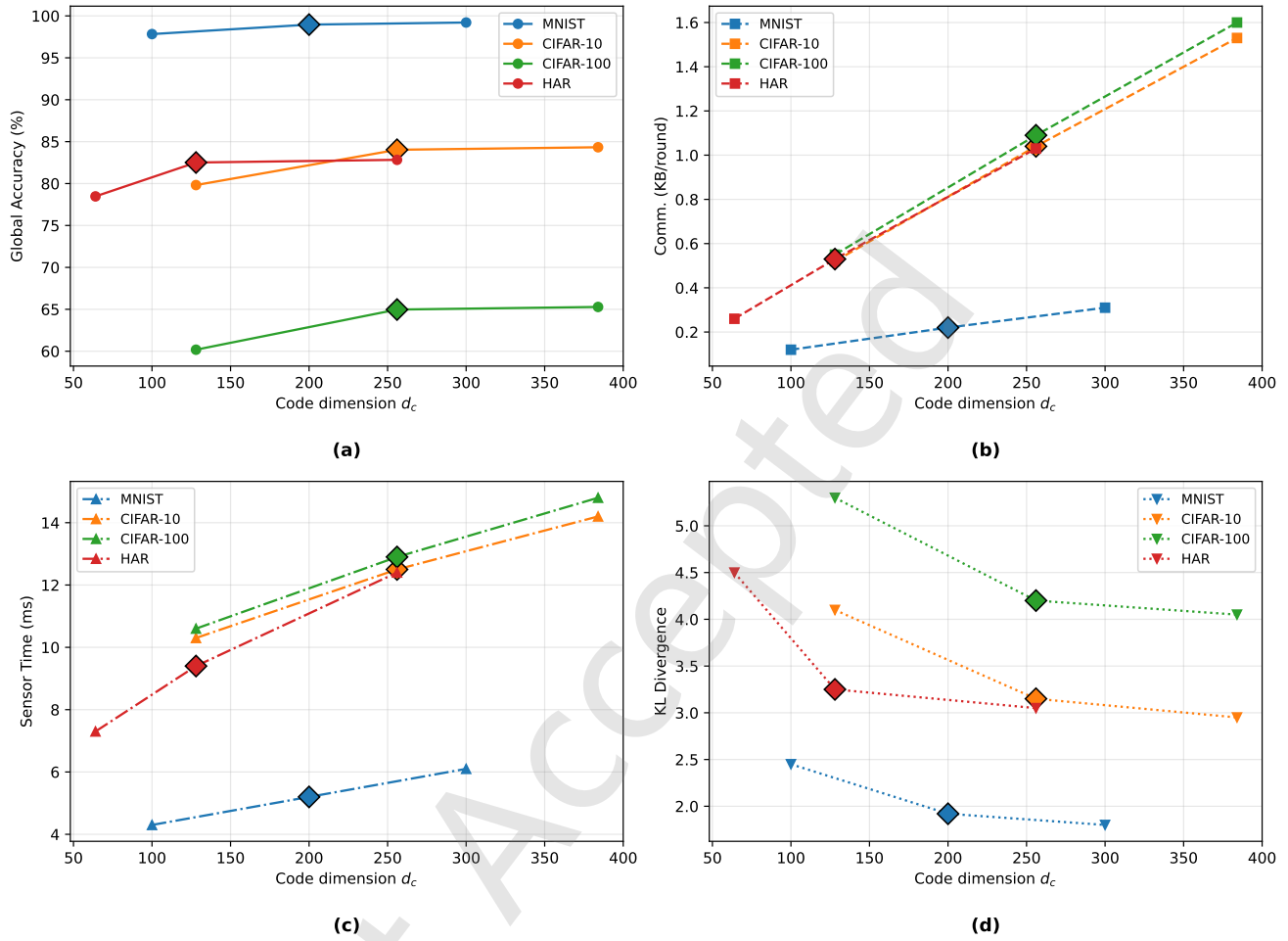
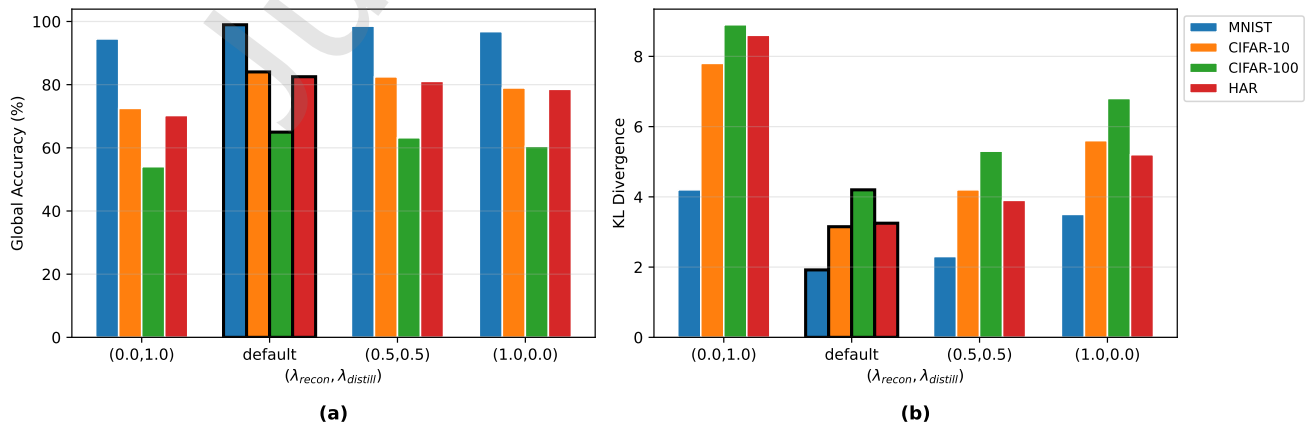
causing the teacher to produce noisy soft labels. Conversely, removing the distillation loss ($\lambda_{\text{distill}} = 0$, i.e., $\lambda_{\text{recon}} = 1.0$) forces the sensors to rely exclusively on local hard labels, which also results in a clear accuracy degradation compared to the default configuration. The teacher’s soft labels thus provide complementary supervisory signals that are especially valuable under non-IID data partitions, where local label distributions are skewed. The default weight combinations balance reconstruction and distillation, achieving the highest accuracy without affecting communication or sensor computation, since the latter metrics depend purely on the encoder architecture and code dimension.

Overall, the ablation experiments confirm that the default hyperparameters are well-calibrated. The sparsity ratio and code dimension are set near their saturation points, where further increases offer diminishing returns, while the loss

weights strike an effective balance between preserving data fidelity and transferring global knowledge through distillation. These findings validate the design choices described in Sec. 3 and their empirical effectiveness demonstrated in the SOTA comparison.

5 Conclusions

In this paper, we presented HFL-Lite, a lightweight hierarchical federated learning framework for computation-limited sensor networks. By replacing bidirectional gradient/activation exchange with one-way sparse code upload and edge-based data synthesis and distillation, HFL-Lite eliminates the need for expensive cryptographic operations on sensors while preserving privacy through non-invertible feature representations. The joint reconstruction–distillation–sparsity loss ensures that sparse codes retain semantic information for ac-

Effect of Code Dimension d_c **Fig. 3** Effect of code dimension d_c . Subfigures (a)–(d) display the same four metrics. Default values are highlighted.Effect of Loss Weights (λ_{recon} , λ_{distill})**Fig. 4** Effect of loss weights (λ_{recon} , λ_{distill}). Subfigure (a) shows global accuracy and (b) shows KL divergence. Communication and sensor time are not plotted because they are unaffected by the loss weights. The default weight combination is indicated by a bold border.

curate teacher guidance. Preliminary experiments on the UCI HAR dataset demonstrate that HFL-Lite achieves competitive accuracy with dramatically reduced sensor-side computation and communication overhead compared to HE-based and SL-based baselines. Future work includes extending HFL-Lite to more complex sensor modalities (e.g., video, audio) and exploring dynamic dictionary adaptation for non-stationary data distributions.

Acknowledgment

This work was supported by the Joint Key Project of the National Natural Science Foundation of China (Grant No. U24A20428), National Natural Science Foundation of China (Grant No. 62472168), the Hunan Provincial Major Science and Technology Project (Grant No. 2025QK2008), and the Hunan Provincial Department of Education under the Post-graduate Scientific Research Innovation Project of Hunan Province (No. CX20251545).

Reference

- [1] A. Shankar, C. Maple, and G. Epiphaniou, A study on analyzing the security and privacy implications of existing and emerging sensors in autonomous electric vehicles, *Tsinghua Science and Technology*, vol. 30, no. 4, pp. 1401–1420, 2025.
- [2] S. Zhou, K. Li, Y. Chen, C. Yang, W. Liang, and A. Y. Zomaya, Trustbcfl: Mitigating data bias in iot through blockchain-enabled federated learning, *IEEE Internet of Things Journal*, vol. 11, no. 15, pp. 25 648–25 662, 2024.
- [3] L. Ren, Y. Liu, C. Ouyang, Y. Yu, S. Zhou, Y. He, and Y. Wan, Dylas: A dynamic label alignment strategy for large-scale multi-label text classification, *Information Fusion*, vol. 120, p. 103081, 2025.
- [4] Z. Cheng, A. Qu, and X. He, Contour-aware semantic segmentation network with spatial attention mechanism for medical image, *The Visual Computer*, vol. 38, no. 3, pp. 749–762, 2022.
- [5] Z. Tao, C. Ouyang, Y. Liu, T. Chung, and Y. Cao, Multi-head attention graph convolutional network model: End-to-end entity and relation joint extraction based on multi-head attention graph convolutional network, *CAAI Transactions on Intelligence Technology*, vol. 8, no. 2, pp. 468–477, 2023.
- [6] I. Bisio, C. Garibotto, F. Lavagetto, A. Sciarrone, and M. Zerbino, Distributed multi-objective optimization for edge computing in resource-constrained social iot networks, *IEEE Internet of Things Journal*, 2026.
- [7] C. Liang, Z. Diao, X. Wang *et al.*, Fedahpip: Federated learning with adaptive hot parameter identification and personalized anchoring for multi-agent collaboration, *Journal of Industrial Information Integration*, vol. 42, p. 101087, 2026.
- [8] S. Zhou, Y. Chen, W. Liang, K. Li, and A. Y. Zomaya, Multisecdfli: Multi-metric filtering-based secure aggregation method for decentralized federated learning, *IEEE Transactions on Consumer Electronics*, pp. 1–1, 2026.
- [9] J. Lai, R. Wang, C. Luo, and F. Zhang, Eltfli: A trusted heterogeneous federated learning in edge scenario, *Tsinghua Science and Technology*, vol. 31, no. 5, pp. 2449–2467, 2026.
- [10] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.
- [11] C. Liang, W. Liang, Y. Chen *et al.*, Fedtctf: Tensor completion-based federated learning for device heterogeneity, *IEEE Transactions on Sustainable Computing*, vol. 10, no. 6, pp. 1227–1239, 2025.
- [12] K. Chen, W. Li, J. Cao, B. Mi, and J. Shen, Optimizing federated incremental learning: Efficient malicious data removal for big data analytics, *Tsinghua Science and Technology*, 2026.
- [13] E. Shang, H. Liu, J. Zhang, R. Zhao, and J. Du, Ensemble knowledge distillation for federated semi-supervised image classification, *Tsinghua Science and Technology*, vol. 30, no. 1, pp. 112–123, 2024.
- [14] X. Deng, P. Li, R. Wang, J. Tan, Y. Liu, W. Han, and Z. Tian, Learning sequential deception defense strategy against apt using stackelberg markov game, *IEEE Transactions on Information Forensics and Security*, 2026.
- [15] X. Deng, P. Li, C. Wang, R. Wang, Y. Liu, W. Han, and Z. Tian, A stackelberg game based deception defense strategy against apt under resource constraints, *Science China Information Sciences*, vol. 69, no. 3, p. 132109, 2026.
- [16] T. Wang, P. Wang, S. Cai, X. Zheng, Y. Ma, W. Jia, and G. Wang, Mobile edge-enabled trust evaluation for the internet of things, *Information Fusion*, vol. 75, pp. 90–100, 2021.
- [17] C. Xu, J. Guo, Y. Liang, H. Huang, H. Zou, X. Zheng, S. Yu, X. Chu, J. Cao, and T. Wang, Diffusion models for reinforcement learning: Foundations, taxonomy, and development, *arXiv preprint arXiv:2510.12253*, 2025.
- [18] W. Gao, P. Zheng, Y. Hu, L. Xu, and A. Schmeink, Performance enhancement on sparse federated learning supported by ris-aided communication in the finite blocklength regime, *IEEE Transactions on Mobile Computing*, vol. 25, no. 3, pp. 3824–3842, 2026.
- [19] E. Diao, J. Ding, and V. Tarokh, Heterofli: Computation and communication efficient federated learning for heterogeneous clients, *arXiv preprint arXiv:2010.01264*, 2020.
- [20] S. Caldas, J. Konečný, H. B. McMahan, and A. Talwalkar, Expanding the reach of federated learning by reducing client resource requirements, *arXiv preprint arXiv:1812.07210*, 2018.
- [21] S. Alam, L. Liu, M. Yan, and M. Zhang, Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction, *Advances in neural information processing systems*, vol. 35, pp. 29 677–29 690, 2022.
- [22] S. Zhou, Y. Chen, K. Li, w. Liang, and W. Meng, Trusthfl: An efficient aggregation method for trustworthy hierarchical federated learning, *IEEE Internet of Things Journal*, 2026.

- [23] Y. Mao, Y. Rong, F. Chi, G. Li, H. Xu, and P. Ping, Personalized federated learning over edge-cloud collaborative network for intelligent sensing analysis, *Tsinghua Science and Technology*, vol. 31, no. 2, pp. 851–866, 2026.
- [24] Z. Lin, Y. Zhang, Z. Chen, Z. Fang, C. Wu, X. Chen, Y. Gao, and J. Luo, Leo-split: A semi-supervised split learning framework over leo satellite networks, *IEEE Transactions on Mobile Computing*, vol. 25, no. 2, pp. 1565–1579, 2026.
- [25] R. S. Molina, V. Ninkovic, D. Vukobratovic, M. L. Crespo, and M. Zennaro, Efficient split learning lstm models for fpga-based edge iot devices, in *2025 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*. IEEE, 2025, pp. 1–6.
- [26] N. D. Pham, K. T. Phan, A. Abuadba, Y. Gao, V.-D. Nguyen, and N. Chilamkurti, Split learning without local weight sharing to enhance client-side data privacy, *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 5, pp. 5352–5365, 2025.
- [27] J. Feng, X. Zhang, Z. Zilic, Q. Hao, and J. Wang, An efficient paillier homomorphic encryption circuit with optional crt acceleration for iot, *IEEE Internet of Things Journal*, vol. 12, no. 22, pp. 48 146–48 158, 2025.
- [28] S. Ghosh and S. K. Ghosh, Feel: Federated learning framework for elderly healthcare using edge-iomt, *IEEE Transactions on Computational Social Systems*, vol. 10, no. 4, pp. 1800–1809, 2023.
- [29] S. Somvanshi, M. M. Islam, G. Chhetri, R. Chakraborty, M. S. Mimi, S. A. Shuvo, K. S. Islam, S. Javed, S. A. Rafat, A. Dutta *et al.*, From tiny machine learning to tiny deep learning: A survey, *ACM Computing Surveys*, vol. 58, no. 7, pp. 1–33, 2025.
- [30] Y. Oh, Y.-S. Jeon, M. Chen, and W. Saad, Fedvqcs: Federated learning via vector quantized compressed sensing, *IEEE Transactions on Wireless Communications*, vol. 23, no. 3, pp. 1755–1770, 2023.
- [31] L. Barbieri, S. Kianoush, M. Nicoli, L. Serio, and S. Savazzi, A close look at the communication efficiency and the energy footprints of robust federated learning in industrial iot, *IEEE Internet of Things Journal*, vol. 12, no. 11, pp. 15 130–15 150, 2025.
- [32] J. Park, K. Lee, S. Lee, M. Zhang, and J. Ko, Attfl: A personalized federated learning framework for time-series mobile and embedded sensor data processing, *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 7, no. 3, pp. 1–31, 2023.
- [33] A. Mondal, M. Kumar, and S. Chakraborty, Fedmeet: Personalized federated multi-sensor fusion for privacy-preserving human activity recognition, in *Proceedings of the 27th International Conference on Distributed Computing and Networking*, 2026, pp. 103–112.
- [34] E. Liu, W. Yang, Y. Gu, W. Long, S. István, and L. Jiang, A survey of clustering federated learning in heterogeneous data scenarios, *Journal of Computing and Electronic Information Management*, vol. 16, no. 3, pp. 17–22, 2025.
- [35] X. Su, Y. Zhou, L. Cui, S. Guo, and J. Liu, Fed-gtopk: Expediting in-network federated learning by transmitting global top model updates, *IEEE Transactions on Networking*, 2026.
- [36] R. Deng, X. Du, Z. Lu, Q. Duan, S.-C. Huang, and J. Wu, Hsfl: Efficient and privacy-preserving offloading for split and federated learning in iot services, in *2023 IEEE International conference on web services (ICWS)*. IEEE, 2023, pp. 658–668.
- [37] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, Splitfed: When federated learning meets split learning, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 36, no. 8, 2022, pp. 8485–8493.
- [38] J. Wang, T. Zhu, L. L. Chen, H. Ning, and Y. Wan, Negative selection by clustering for contrastive learning in human activity recognition, *IEEE Internet of Things Journal*, vol. 10, no. 12, pp. 10 833–10 844, 2023.
- [39] O. Gupta and R. Raskar, Distributed learning of deep neural network over multiple agents, *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

Author biography



Cangming Liang received a bachelor's degree in software engineering from Hainan Normal University in 2016 and a master's degree in electronic information from the University of South China in 2023. He is currently pursuing a Ph.D. degree in software engineering in the School of Computer Science and Engineering, Hunan University of Science and Technology. His main research interests include distributed computing and federated learning.



Kuan-Ching Li is a Professor with the School of Computer Science and Engineering, Hunan University of Science and Technology. He received the Ph.D. degree from San Pablo Main in 2001. Dr. Li has co-authored over 150 conference and journal papers, holds several patents, and serves as an associate and guest editor for various scientific journals. He has also held chair positions at several prestigious international conferences. His research interests include cloud and edge computing, big data, and blockchain technologies.



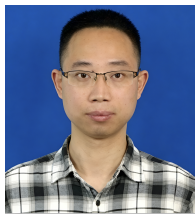
machine learning, traffic analysis, and network measurement.

Zulong Diao received the Ph.D. degree in software engineering from Hunan University, Changsha, China, in 2019. He is currently an Associate Professor with the School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangan, China. His research interests include



blockchain technology.

Xiangwei Meng received the Ph.D. degree from Nanjing University of Aeronautics and Astronautics in 2024. He is currently a Lecturer with the School of Computer Science and Engineering, Hunan University of Science and Technology. His research interests include data security, privacy protection, and



include AI and health management.

Jie Liu was born in Hengyang, Hunan, China, in 1974. He is currently an Associate Professor with the School of Computer Science, University of South China, and the Deputy Director of the Hunan Engineering Research Center of Software Evaluation and Testing for Intellectual Equipment. His research interests



the School of Computer Science. He has authored or co-authored more than 220 journal/conference papers such as IEEE TC, TIFS, TSC, TITS, TII, TETC, TCBB, and IoTJ. His research interests include intelligent transportation, security of IoV, blockchain, and embedded systems.

Wei Liang received the Ph.D. degree in computer science and technology from Hunan University in 2013 and was a Postdoctoral Scholar at Lehigh University from 2014 to 2016. He is currently a Professor and the Vice President of the University of South China, Hengyang, China, where he oversees