FLEAT: Energy-Accuracy Trade-off in Federated Learning over Heterogeneous Edge Devices

Javad Dogani, Reza Namvar, Masoumeh Khodarahmi, Farshad Khunjush, and Nikolaos Laoutaris

***Abstract*—Federated Learning (FL) enables collaborative model training across edge devices but faces challenges balancing energy consumption, heterogeneous resources, and accuracy in IoT ecosystems. Existing solutions often overlook adaptive coordination of computation and communication or depend on hardware-level adjustments unavailable on constrained devices. We propose FLEAT (Federated Learning Energy and Accuracy Tuning), a framework that jointly optimizes energy efficiency and model accuracy via dynamic local update adaptation and gradient-informed layer-wise pruning. FLEAT introduces a theoretical error bound for synchronous FL under these mechanisms and employs an energy-aware optimization loop to allocate per-device computation/communication time. By scaling local updates to device capabilities and pruning redundant layers based on parameter importance, FLEAT mitigates stragglers, reduces communication overhead, and stabilizes convergence via normalized gradient aggregation. We evaluate FLEAT in both real-world IoT deployments and simulated edge networks. At a fixed wall-clock budget, FLEAT achieves a higher convergence rate—reaching higher accuracy earlier—than FedAvg, FedProx, and FedNova; when all methods train to completion, it remains within 1.2–3.1% of FedAvg's final accuracy while cutting total energy by 16.2%. Relative to existing studies, FLEAT matches or surpasses their accuracy while strictly lowering energy, owing to its joint tuning of local steps and pruning under an energy-aware objective. This work bridges model- and system-level optimizations, offering a scalable solution for energy-accuracy equilibrium in heterogeneous FL deployments.**

***Index Terms*—Federated Learning (FL), Internet of Things (IoT), Energy-Accuracy Equilibrium**

I. Introduction

The proliferation of smart applications, from augmented reality to autonomous driving and digital twins, has driven a rapid growth of IoT devices [1], [2]. These devices generate massive volumes of data that are increasingly processed with machine learning (ML). However, cloud-centric training faces critical limitations: sending raw data to a central server adds latency, strains bandwidth, and creates privacy risks [3]. These concerns are acute in health monitoring and smart grids, where outsourcing user data to third-party clouds increases vulnerability [4], [5]. Federated learning (FL) helps address these challenges by training models directly on devices, avoiding raw-data transfer to a central server [6], [7]. By keeping sensitive information local, FL reduces latency and privacy risk [8]. In a typical round, devices update the model on local data and transmit model updates (not raw data) to an aggregation server. The server aggregates updates, refreshes the global model, and redistributes it until convergence [8]. FL is a key pillar of edge intelligence, integrating edge computing and AI. Processing data near end users improves reliability and reduces latency and network load.

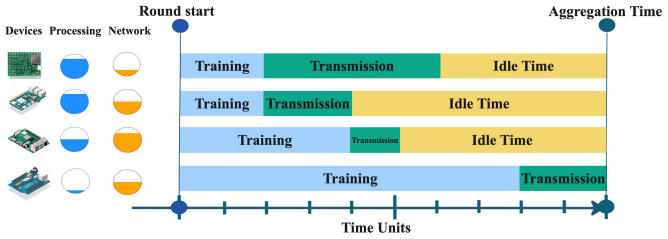

Fig. 1. Straggler in synchronous FL: round time is gated by the slowest client.

Challenges: Despite its promise, deploying FL on edge devices raises three interrelated challenges. First, device and data heterogeneity—uneven compute/network and non-independent and identically distributed (non-IID) data — leads to slower convergence and degraded accuracy. As shown in Fig. 1, synchronous FL is often bottlenecked by stragglers, devices with slower computation or communication speeds. For example, faster devices idle while waiting for slower peers (e.g., Device 2 in Fig. 1). Second, communication bottlenecks arise from coordinating thousands of devices. Synchronous methods like FedAvg [8], FedProx [9], FedNova [10], SCAFFOLD [11] are straggler-limited, while asynchronous methods like FedAsync [12] gain throughput at the cost of staleness and potential accuracy loss [13]. Third, battery-powered devices face tight energy budgets: prolonged computation and transmission cause client dropouts and lower participation [14]. Real-world initiatives such as Microsoft's Project FarmBeats [15] and Nvidia Clara [16] underscore the need for energy optimization in FL. While some studies address heterogeneity [17], [18] and others reduce energy via infrastructure controls (e.g., DVFS, transmit-power tuning) [19]–[21], such knobs are not universally available—especially in IoT deployments lacking access to radio or DVFS APIs [19]. Compression techniques (quantization and structured sparsity) reduce computation and shrink payloads [22]–[25]. However, they typically lack *energy-aware* adaptation to balance accuracy–energy trade-offs across heterogeneous devices. **Our Proposal:** To address these challenges, we propose Federated Learning Energy and Accuracy Tuning (FLEAT), which jointly optimizes energy, communication, and accuracy via

Javad Dogani and Nikolaos Laoutaris are with IMDEA Networks Institute, Madrid, Spain (e-mails: {javad.dogani, nikolaos.laoutaris}@networks.imdea.org).

Reza Namvar and Masoumeh Khodarahmi are with IMDEA Networks Institute, Madrid, Spain, and Universidad Carlos III de Madrid (UC3M), Madrid, Spain (e-mails: reza.namvar, masoumeh.khodarahmi@networks.imdea.org).

Farshad Khunjush is with Shiraz University, Shiraz, Iran (e-mail: khunjush@shirazu.ac.ir).

two adaptive mechanisms: (1) Dynamic local updates that adjust each device's number of local steps per round based on compute capacity and energy state. (2) *structured pruning* that omits a fraction of layers/parameters from transmission to reduce communication time/energy while retaining the full model locally for subsequent training. Each technique is well-studied independently—local updates balance computation and communication [26]–[28], and structured pruning reduces transmission [23]—but neither is typically integrated with explicit energy awareness.

Our Contribution. We propose FLEAT, a practical framework that *jointly* tunes the number of local steps and the upload pruning ratio under explicit energy–time constraints. To our knowledge, this is the first FL approach that couples these two algorithmic knobs inside an energy-aware optimization loop with a closed-form update for the local-steps parameter and a monotone, feasibility-preserving update for the pruning level, making per-round control lightweight and deployable. We derive an upper bound that links expected loss to local update frequency and pruning ratio, and we design an energy-aware optimizer that lets heterogeneous devices participate within a fixed round-time budget, mitigating stragglers. Beyond infrastructure-only and compression-only FL, FLEAT exposes a telemetry-driven controller that co-adapts computation and communication to device heterogeneity, aligning the training dynamics with energy budgets and round-time limits.

Our findings. We validate FLEAT by applying lightweight CNN and RNN classifiers to the Edge-IIoTset dataset in a real-world deployment across heterogeneous edge devices. By complementary large-scale simulations using EfficientNet-B0, MobileNetV2, ResNet-18, and GoogleNet on MNIST, SVHN, CIFAR-10, and Fashion-MNIST under non-IID label-skewed partitions, our findings are:

- *Tuning the accuracy–energy emphasis.* When we set the objective to balance accuracy and energy, accuracy stays within **3—5%** of the best we can get while total energy drops by 22–37% in our real-device runs.
- *Convergence vs. accuracy and energy.* At the common wall-clock time (mid-training), our method reaches higher accuracy than standard FL baselines such as FedAvg [8], FedProx [9], and FedNova [10]; when all are run to completion, we remain within 1.2–3.1% of FedAvg's final accuracy while using 16.2% less total energy.
- *Compute–communication rebalance.* By coordinating how much each device trains locally with how much it uploads, we shift time off the uplink and cut communication cost; compared to Local-Update-Only and Pruning-Only, accuracy increases by 15.6% and 23.0% while energy still falls by 4.1% and 1.2%.

II. Related Work

Infrastructure- and network-centric efficiency. A large body of work improves FL efficiency via hardware and network control, including CPU/DVFS and power tuning in ALTD [19], latency–energy co-optimization in FEDL [20], staleness-aware scheduling in wireless FL [18] and energy-aware worker selection (EaMC-FL) [29]. These methods primarily optimize infrastructure-level knobs (CPU, radio, placement), rather than jointly adapting model-level training depth and structure during learning.

Communication-efficient FL (compression/distillation). Communication can be reduced via sparsification, quantization, coding, and distillation, e.g., dynamic compression control [30], computation/communication co-adaptation [31], and probabilistic/side-information compression [32], [33]. These methods are effective for bandwidth reduction, but typically optimize proxy communication cost and do not explicitly optimize a per-device energy–accuracy objective.

Model-centric pruning/quantization. Model size and compute can be tailored to heterogeneous devices using structured pruning/quantization, e.g., FedPARL [22], PruneFL [24] and NISP [25]. Recent works extend this to split/federated hybrids and device-compatible sub-model generation (subMFL [34]). However, most do not jointly co-tune local training and model reduction in per-round closed loops.

Energy-aware and multi-objective FL optimization. Recent FL frameworks also explicitly optimize energy jointly with learning performance and/or delay, mainly through client selection and resource scheduling. Examples include FedAECS [35] and EAFL / EAFL+ [14] and time/energy-aware client-selection methods on heterogeneous resources [21]. These works strengthen the case for energy-aware FL, but mostly operate at the *client/scheduler/resource* layer—deciding which clients participate, when, and under what constraints—rather than using a control loop that adapts local training depth and model structure during training.

Joint tuning of local steps and compression. Fast FL (FFL) [36] jointly adapts local update depth and gradient sparsity to minimize error under a *wall-clock* budget; follow-ups similarly co-tune communication interval and compression [36]. These methods are closest in spirit to ours, but they primarily optimize time/cost proxies rather than a per-device energy–accuracy objective. On heterogeneous edge devices, equal round times correspond to different energy expenditures due to device- and phase-specific power draw.

Straggler mitigation and asynchrony. Scheduling with stale models [18], asynchronous FL protocols [12], and progressive compression (Snowball [37]) reduce synchronization overheads, but they typically decouple systems scheduling from model-structure adaptation and do not address aggregation correction under heterogeneous partial layer uploads.

Positioning and key distinctions. FLEAT is complementary to infrastructure- and scheduler-level methods, but targets a different control layer: it jointly adapts *local training depth* and *structured pruning* per device and per round using measured on-device telemetry (energy, accuracy and time). Unlike FFL-like methods, FLEAT optimizes an explicit measurable energy–accuracy objective. FLEAT also introduces aggregation renormalization to correct bias under partial layer uploads. Table I summarizes this positioning.

III. System Model

This section formalizes our FL setting, the heterogeneity-aware aggregation used by FLEAT, and the communication–time–energy models. We explicitly distinguish between

TABLE I
COMPACT POSITIONING SUMMARY. COLUMNS: **E** = DIRECT ENERGY OBJECTIVE, **A** = EXPLICIT ACCURACY-AWARE OBJECTIVE, **P** = PER-DEVICE/PER-ROUND ADAPTATION, **J** = JOINT MODEL-LEVEL CONTROL, **R** = AGGREGATION CORRECTION UNDER PARTIAL UPLOADS.

Family (repr.)	E	A	P	J	R
Infra./network [19], [20]	✓	✗	✓	✗	✗
Energy-aware client sel. [14], [21], [35]	✓	✓	✓	✗	✗
Compression/distill. [30], [32], [33]	✗	✗	✓	✗	✗
Model pruning/quant. [24], [25], [34]	✗	✗	✗	✗	✗
FFL-like [36]	✗	✓	✓	✗	✗
FLEAT (ours)	✓	✓	✓	✓	✓

(i) *simulation-based* and (ii) *real-world* profiling models for time and energy. A complete list of symbols is provided in Table IV in Appendix A.

A. Entities, Data, and Notation

We consider an edge network with a central server (unbounded compute) and K edge devices indexed by $k \in [K]$. Device k holds a private dataset $D_k = \{(x_i, y_i)\}_{i=1}^{|D_k|}$ with $x_i \in \mathbb{R}^d$ and label $y_i \in \{1, \dots, C\}$. The global dataset is $D = \bigcup_{k=1}^{K} D_k$ with $|D| = \sum_{k=1}^{K} |D_k|$. Let $\ell(w; x, y)$ be the per-sample loss and $\mathcal{L}_k(w) = \frac{1}{|D_k|} \sum_{(x,y)\in D_k} \ell(w; x, y)$ the local empirical risk. The global objective is

$$\min_{w} \ \mathcal{L}(w) = \sum_{k=1}^{K} \frac{|D_k|}{|D|} \mathcal{L}_k(w), \tag{1}$$

with generally non-IID data across clients ($D_i \not\sim D_j$), which complicates aggregation and may slow convergence.

B. Round Timeline and Canonical FL Protocol

Training proceeds in synchronous communication rounds $r = 0, 1, \dots, R-1$. In each round, the server broadcasts the current global model w_r to a participating set $S_r \subseteq [K]$ (for clarity, we assume $S_r = [K]$). Each device performs *local* stochastic updates, returns its update, and the server aggregates and redistributes. Concretely, device k runs $\tau_r^k \in \mathbb{N}$ local steps:

$$\begin{aligned} w_r^{k,0} &= w_r, \\ w_r^{k,t} &= w_r^{k,t-1} - \eta \nabla \ell\big(w_r^{k,t-1}; \xi_{k,t}\big), \\ & \qquad t = 1, \dots, \tau_r^k, \end{aligned} \tag{2}$$

$$w_r^k := w_r^{k,\tau_r^k}, \qquad \Delta_r^k := w_r^k - w_r, \tag{3}$$

where $\xi_{k,t} \subset D_k$ is a mini-batch. Standard FedAvg would aggregate full models, $w_{r+1} = \sum_k \frac{|D_k|}{|D|} w_r^k$ [8].

C. Heterogeneity-Aware Local Steps Tuning and Normalized Aggregation

Uniform local-step policies inflate end-to-end latency due to stragglers; faster devices idle while waiting, and naïve averaging can overweight devices that take many local steps under non-IID data. To mitigate this, FLEAT adapts the *per-device local depth* τ_r^k each round and aggregates *deltas* with local-step normalization:

$$w_{r+1} = w_r + \sum_{k=1}^{K} \frac{|D_k|}{|D|} \frac{\Delta_r^k}{\tau_r^k}. \tag{4}$$

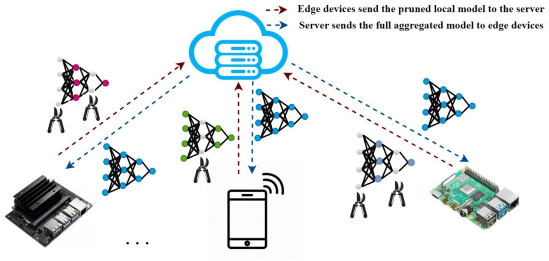

Fig. 2. Structured layer pruning: on upload, each client omits the least-important layers (gray) to reduce communication time, where importance is computed per layer. On download, all clients receive the full model.

How aggregation is used in our evaluation. In all experiments, the server uses Eq. (4) when full deltas are available and Eq. (6) when uploads are masked by pruning (partial layer updates). The similarity-aware server reweighting discussed in Appendix D-B is an optional extension and is disabled in the reported results (equivalently, its similarity term is set to 1).

This preserves the baseline w_r while scaling each contribution by its effective local depth, reducing bias and improving stability under heterogeneity. We assume $\tau_r^k > 0$, as devices with no local updates contribute zero to the aggregation.

Adaptive Pruning Ratio Adjustment While tailoring local update frequency addresses heterogeneous computational capabilities, communication overhead persists as a bottleneck in FL. Pruning offers a complementary approach to reduce communication costs by removing layers identified as non-critical. Conventional FL pruning methods, adapted from centralized ML, eliminate neuronal connections in multi-layer perceptrons (MLP) or filters in Convolutional Neural Networks (CNN) (unstructured pruning), which risk compromising intermediate feature dimensionality [23]. We use layer-level pruning, commonly termed horizontal pruning, as depicted in Fig. 2. We prune entire layers of low importance only upon uploading them to the server, which reduces communication overhead without significantly affecting accuracy. To avoid transmitting long contiguous segments of stale parameters, we never omit uploads from adjacent layers within the same stage. Importantly, our "layer pruning" is communication-side: the client keeps the full model for forward/backward passes and only suppresses selected *update blocks* on upload (i.e., sets the corresponding $\Delta_r^k(l)$ to zero). Therefore, the computation graph (including skip connections) is unchanged; the constraint is introduced to preserve update coverage in tightly-coupled layer groups. Because clients train with the full local model, pruning does not modify the local SGD trajectory. Its only effect is that the server aggregates a masked delta, which can be modeled as an additive aggregation perturbation bounded via the kept-coverage metric γ_r^k in Section IV-C. For skip-connected architectures (e.g., ResNet), we enforce a residual-block constraint in Algorithm 1 to prevent omitting multiple coupled layers within the shortcut group.

Definitions. Let the model have L layers $w = (w(1), \dots, w(L))$ and let b_l denote the byte size of layer l. Device k chooses a binary mask $m_r^k(l) \in \{0, 1\}$ indicating whether the update for layer l is uploaded in round r. The

masked delta is

$$\left(\Delta_r^k\right)_{\text{masked}}(l) = m_r^k(l)\,\Delta_r^k(l).$$

To preserve tensor shapes and stage interfaces (e.g., in ResNets/MobileNets), masks are restricted to a prunable set $\mathcal{S} \subseteq \{1, \ldots, L\}$ that excludes shape-critical layers (e.g., skip connections, normalizations), and we forbid omitting adjacent layers within the same stage. For networks with shortcut connections, we further define a skip-group (residual-block) map $\mathrm{block}(\cdot)$ that assigns layers to the same residual unit; within each block, we omit at most one layer update per round to avoid inconsistent/stale residual branches.

We define a *bytes-based* pruning ratio

$$p_r^k = 1 - \frac{\sum_{l=1}^{L} m_r^k(l)\, b_l}{\sum_{l=1}^{L} b_l} \in [0, 1). \tag{5}$$

Per-layer aggregation with renormalization. When only a subset of devices uploads layer l, we avoid bias by renormalizing over contributors:

$$w_{r+1}(l) = \begin{cases} w_r(l) + \dfrac{\sum_{k \in K_l^r} \frac{|D_k|}{|D|} \frac{\Delta_r^k(l)}{\tau_r^k}}{\sum_{k \in K_l^r} \frac{|D_k|}{|D|}}, & K_l^r \neq \varnothing, \\ w_r(l), & K_l^r = \varnothing, \end{cases} \tag{6}$$

where $K_l^r = \{k \in S_r : \; m_r^k(l) = 1\}$ is the set of contributors for layer l at round r. If no client uploads layer l, we retain $w_r(l)$. This makes each updated layer a convex combination of contributor updates and leaves $w_r(l)$ unchanged if no uploads arrive. Appendix D-B1 provides an analytical isolation showing that, without per-layer renormalization, missing uploads introduce a layer-dependent shrinkage factor that can bias the aggregated direction across layers when masks differ across clients. In this paradigm, clients remove entire low-impact layers before transmitting updates. We employ an importance score (IS) computed via the gradient's L2 norm (during local training) to quantify layer impact, and can therefore be evaluated after the local update phase. Let $w_r^k(l)$ represent the parameters of the l-th layer in device k's model at round r:

$$\mathrm{IS}_r^k(l) \;=\; \frac{\left\|\nabla \mathcal{L}_k(w_r^k(l); \xi_k)\right\|^2}{\left\|\nabla \mathcal{L}_k(w_r^k; \xi_k)\right\|^2}, \tag{7}$$

where $\nabla \mathcal{L}_k(w_r^k(l); \xi_k)$ is the gradient of the local loss $\mathcal{L}_k$ with respect to layer l's parameters, computed over mini-batch ξ_k, and $\nabla \mathcal{L}_k(w_r^k; \xi_k)$ is the global gradient across all layers of device k's model. Layers with lower $\mathrm{IS}_r^k(l)$ exhibit minimal loss reduction impact, prioritizing them for pruning [23], [38].

To avoid numerical issues when the total gradient energy is (near) zero, we use an ϵ-stabilized normalization in Eq. (7). Specifically, we compute

$$IS_k^r(l) = \frac{\left\|\nabla \mathcal{L}_{k,l}(w_r^k; \xi_k)\right\|^2}{\left\|\nabla \mathcal{L}_k(w_r^k; \xi_k)\right\|^2 + \epsilon_{\mathrm{IS}}},$$

where $\epsilon_{\mathrm{IS}} > 0$ is a small constant (e.g., 10^{-12}).

Choice of importance metric (plug-in design). We use gradient-energy importance because it is *data-adaptive* (depends on the current mini-batch/loss landscape) and incurs no extra cost beyond backpropagation. Nevertheless, FLEAT is *metric-agnostic*: any nonnegative layer score $s_r^k(l) \geq 0$ can be normalized as $IS_r^k(l) = s_r^k(l) / \sum_{j=1}^{L} s_r^k(j)$ and then used by Algorithm 1 without changing the protocol or aggregation logic. Common alternatives include (i) magnitude-based scores (e.g., $\|w(l)\|$ or $\|\Delta_r^k(l)\|$), which are cheap but less data-aware, and (ii) Fisher-based scores (diagonal/empirical Fisher), which are more second-order informed but add estimation overhead. We summarize their trade-offs in Appendix D-C.

To prevent structural instability, we prohibit pruning consecutive layers within the same network stage. After receiving pruned models, the server aggregates them, reconstructing the global model: Retained layers are updated via weighted averaging of parameters from contributing devices. In the rare event that *all* clients prune the same layer (suggesting negligible collective importance), its parameters are retained from the prior global model, preventing structural collapse.

The layer-wise pruning algorithm (Algorithm 1) optimizes FL communication by structured model compression. After training w_r^k (line 1), layers are scored ($IS_r^k(l)$, Eq. 7, lines 2-3) and sorted by ascending importance (line 4).

The algorithm initializes parameters (line 5) and iterates through layers from least to most important (lines 6-13). For each non-protected layer (line 7) that would not create consecutive pruning within the same stage (line 8), the layer is pruned if adding it does not exceed the target byte-pruning ratio p_r^k (lines 9-11). The loop terminates early once the next candidate would exceed the byte budget (line 12). As a result, the achieved ratio can remain below p_r^k when (i) many remaining layers are protected or infeasible due to NONCONTIGUOUS/NOBLOCKCLASH, or (ii) the remaining feasible layers are too large to add without exceeding p_r^k. In that case, the fill-up phase (lines 14–18) scans the remaining layers in ascending byte size and prunes the smallest *feasible* ones while still enforcing NONCONTIGUOUS and NOBLOCKCLASH. If no feasible layer remains, the algorithm stops and returns the closest achievable pruning ratio under the structural constraints. Finally, the binary mask m_r^k is applied element-wise to the local delta Δ_r^k (line 19), and both the mask and masked delta are returned (line 20).

D. Time, Communication, and Energy Models

We consider two complementary scenarios for quantifying per–round runtime and energy at the device level. In the *simulation* scenario, we estimate the same quantities using closed-form expressions parameterized by local steps/epochs, model size, and the upload pruning ratio. In the *real-world* scenario, we do not assume an analytical model: we rely on on-device instrumentation to measure wall-clock times and power/energy directly, segmenting compute and communication phases at FL round boundaries.

1) Simulation-Based Estimation (Closed-Form): When on-device instrumentation is unavailable or when we need large-scale what-if analysis, we estimate per–round time and energy using closed-form models. The policy considered here specifies *a fixed number of local steps per round* (no epoch-based variant). The goal is to capture how two main knobs affect

Algorithm 1 Layer-wise pruning (mask generation)

Input: w^k_{r,τ^k_r} (client k model after τ^k_r local steps at round r), w_r (global model), p^k_r (target *byte*-pruning ratio), $\{b_l\}_{l=1}^L$ (bytes per layer), $\mathrm{stage}(\cdot)$ (layer→stage map), $\mathrm{block}(\cdot)$ (layer→skip-group map)
Output: $m^k_r \in \{0,1\}^L$ (binary upload mask), $(\Delta^k_r)_{\text{masked}}$

$\Delta^k_r \leftarrow w^k_r - w_r$ ▷ full local delta (no local deletion)
for $l \in \{1,\dots,L\}$ **do**
  compute $IS^k_r(l)$ using (7) ▷ layer importance
$\mathcal{R} \leftarrow$ layers sorted by *ascending* $IS^k_r(l)$; tie-break by *ascending* b_l
$m^k_r \leftarrow \mathbf{1}_L$; $\mathscr{P}^k_r \leftarrow \varnothing$; $B \leftarrow 0$
$B_{\text{tot}} \leftarrow \sum_{j=1}^L b_j$
for $l \in \mathcal{R}$ **do**
  if not PROTECTED(l) **then**
    if NONCONTIGUOUS($l, \mathscr{P}^k_r$) **then**
      if NOBLOCKCLASH($l, \mathscr{P}^k_r$) **then**
        if $\dfrac{B + b_l}{B_{\text{tot}}} \le p^k_r$ **then**
          $m^k_r(l) \leftarrow 0$; $\mathscr{P}^k_r \leftarrow \mathscr{P}^k_r \cup \{l\}$
          $B \leftarrow B + b_l$
        else
          break ▷ remaining layers are more important
if $\dfrac{B}{B_{\text{tot}}} < p^k_r$ **then**
  $\Gamma \leftarrow \{\, l \in \mathcal{R} \setminus \mathscr{P}^k_r$: **not** PROTECTED(l) $\}$ sorted by *ascending* b_l
  while $\dfrac{B}{B_{\text{tot}}} < p^k_r$ **and** $\Gamma \neq \varnothing$ **do**
    $l^\star \leftarrow \varnothing$
    for $x \in \Gamma$ **do** ▷ Γ is scanned in ascending b_x
      if NONCONTIGUOUS($x, \mathscr{P}^k_r$) **then**
        if NOBLOCKCLASH($x, \mathscr{P}^k_r$) **then**
          $l^\star \leftarrow x$; **break**
    if $l^\star = \varnothing$ **then**
      break ▷ no feasible layer remains; p^k_r cannot be reached
    $l \leftarrow l^\star$
    $m^k_r(l) \leftarrow 0$; $\mathscr{P}^k_r \leftarrow \mathscr{P}^k_r \cup \{l\}$; $B \leftarrow B + b_l$
    $\Gamma \leftarrow \Gamma \setminus \{l\}$
$(\Delta^k_r)_{\text{masked}} \leftarrow m^k_r \odot \Delta^k_r$ ▷ element-wise mask
return m^k_r, $(\Delta^k_r)_{\text{masked}}$
function PROTECTED(l)
  return $(l = 1) \ \vee \ (l = L)$ ▷ protect first/last layers
function NONCONTIGUOUS($l, \mathscr{P}^k_r$)
  for $x \in \mathscr{P}^k_r$ **do**
    if $\mathrm{stage}(x) = \mathrm{stage}(l)$ **and** $|x - l| = 1$ **then**
      return false ▷ prevent consecutive prunes
  return true
function NOBLOCKCLASH($l, \mathscr{P}^k_r$)
  for $x \in \mathscr{P}^k_r$ **do**
    if $\mathrm{block}(x) = \mathrm{block}(l)$ **then**
      return false ▷ avoid omissions inside one residual/skip group
  return true

round time and energy: (i) the number of local steps τ^k_r and (ii) the upload-layer pruning pattern m^k_r.

Computation time. Let ϕ^k_r denote the average wall-clock time for *one* local training step (one forward–backward–update pass on a mini-batch) on device k during round r. We smooth raw step-time observations with an m-round moving average (default m=5) to reduce variability due to transient OS/network effects:

$$\phi^k_r \ \triangleq \ \frac{1}{m} \sum_{u=r-m}^{r-1} \widehat{\phi}^k_u, \qquad r \ge m$$

where $\widehat{\phi}^k_u$ is the measured per-step time in round u. If layer-level probes are available, we may decompose $\phi^k_r = \sum_{l=1}^L \phi^k_r(l)$ for diagnostics, where $\phi^k_r(l)$ is the mean contribution of layer l (e.g., convolution/attention/FC).

The controller chooses τ^k_r (how much local work to do); to turn that choice into a concrete time budget, we need the average time per step on that device and model. Given the *steps* policy, the compute time of device k in round r is

$$Y^k_r \ = \ \tau^k_r \, \phi^k_r. \tag{8}$$

$\tau^k_r \in \mathbb{N}$ is the device's local depth for the round; ϕ^k_r (seconds/step) reflects hardware (CPU/GPU, memory) and software (precision, kernel) factors; Y^k_r is the total compute time (seconds) for that device in the round. Each step repeats the same compute pattern on a mini-batch, scaling is linear.

Communication times. Let the model have L layers with per-layer byte sizes $\{b_l\}_{l=1}^L$ (including datatype and, if modeled, optimizer state). The device chooses an *uplink mask* $m^k_r(l) \in \{0,1\}$ to decide which layer deltas to upload. Then

$$b^k_r \ = \ \sum_{l=1}^L m^k_r(l)\, b_l, \qquad b_0 \ = \ \sum_{l=1}^L b_l. \tag{9}$$

b^k_r is the payload that determines the upload time, which depends directly on the pruning decision. By design, downloads always carry the full model (b_0), since clients resume full-capacity training next round. Given device-specific throughputs $(\mathrm{BW}^{\uparrow}_k, \mathrm{BW}^{\downarrow}_k)$ in Mbps, the communication times are:

$$C^k_r = \frac{b^k_r}{\mathrm{BW}^{\uparrow}_k}, \qquad C'_k = \frac{b_0}{\mathrm{BW}^{\downarrow}_k}. \tag{10}$$

C^k_r is the upload time (seconds) and shrinks as the pruning ratio increases (fewer layers uploaded); C'_k is the download time and remains constant for a node during the rounds because the server always sends the full model. Throughput comes from traces or scenario assumptions. In synchronous FL, the round completes when the slowest participant finishes:

$$T_r \ = \ \max_k \ \{\, Y^k_r \ + \ C^k_r \ + \ C'_k \,\}. \tag{11}$$

T_r is the wall-clock duration of the round; it determines end-to-end training time and exposes the straggler bottleneck. Increasing τ^k_r linearly increases Y^k_r, while increasing pruning ratio (fewer uploaded layers) reduces C^k_r. These two levers trade off within T_r.

Energy. We model energy by multiplying *phase durations* by *average powers* characteristic of each phase:

$$E^k_r \ = \ P^k_{\text{comp}}\, Y^k_r \ + \ P^k_{\text{send}}\, C^k_r \ + \ P^k_{\text{rec}}\, C'_k. \tag{12}$$

P^k_{comp} is the average device power (W) while training locally (CPU/GPU busy), P^k_{send} while uploading (radio or NIC active), and P^k_{rec} while downloading. E^k_r is energy (W_s). **DVFS and time-varying power.** Equation (12) uses *phase-average* powers. Under DVFS, instantaneous compute power varies over time; more generally,

$$E^k_{r,\text{comp}} = \int_0^{Y^k_r} P^k_{\text{comp}}(t)\, dt \quad \Rightarrow \quad E^k_{r,\text{comp}} = \bar{P}^k_{\text{comp},r}\, Y^k_r,$$

where $\bar{P}^k_{\text{comp},r} := \frac{1}{Y^k_r} \int_0^{Y^k_r} P^k_{\text{comp}}(t)\, dt$ is the round-r average during local training. Thus, DVFS is captured by allowing

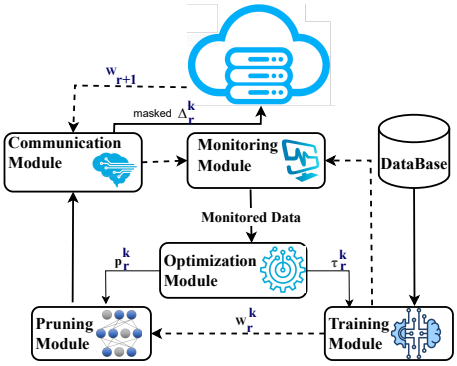

Fig. 3. FLEAT architecture and control loop

phase-average powers to be round-dependent (or by directly measuring E_r^k in the real-world scenario).

Interpretation and units. All times $(Y_r^k, C_r^k, C_k', T_r)$ are in seconds; bytes (b_r^k, b_0) must match the bandwidth units (Mbps); powers are in Watts; energies are in W_s. Equations (8)–(12) thus form a consistent algebra: increasing steps (τ_r^k) raises compute time and energy; increasing pruning ratio (fewer uploaded layers, smaller b_r^k) lowers upload time and energy, while download cost remains fixed because the full model is always redistributed. To avoid overreacting to transient spikes (e.g., Wi-Fi contention), we smooth ϕ_r^k and, if using traced throughputs, $(\mathrm{BW}_k^{\uparrow}, \mathrm{BW}_k^{\downarrow})$ with moving averages or exponential smoothing and clip extreme values (e.g., to the 10th/90th percentiles). This yields stable, reproducible estimates for large-scale simulations.

2) Real-World Measurement(No Analytical Model): **Principle.** Hardware DVFS, cache effects, and wireless contention make closed-form models unreliable on IoT platforms. By relying on synchronized markers, byte counts, and platform sensors, we obtain ground-truth times and energies that directly reflect the device and network conditions of each round. Each client emits phase markers for (`BEGIN` and `END`) of computation, upload, and download so that timestamps and power samples can be aligned to the FL round.

Wall-clock times. Compute time per round is obtained from high-resolution timers (e.g., `std::chrono`, or `time.perf_counter()` in Python). Upload/download durations are measured from the same timer using the phase markers, and are cross-checked with socket/NIC statistics (e.g., byte counters from `send`/`recv`, `ss`/`ethtool`, or TLS callbacks) to ensure the recorded interval covers the actual network transfer. We always record the *bytes* sent and received; uplink bytes reflect the current upload mask m_r^k (sum of per-layer sizes), while downlink bytes equal the full model size.

Algorithm 2 FLEAT Server Procedure

```
Broadcasts (w_0, F(w_0))
for r = 1 to R do
    Receives ((Δ_r^k)_masked, τ_r^k, γ_r^k, m_r^k) from clients
    Update the global model to find w_{r+1}
    Broadcasts w_{r+1} to all clients
```

Algorithm 3 FLEAT Client Procedure at Device k

```
Receive (w_0, F(w_0)); initialize τ_0^k, p_0^k
Measure C_k', and G_k^2 = E‖∇L_k‖^2
for r = 1 to R do
    Measure φ_r^k
    w_r^k ← w_r
    Compute IS_r^k(l)
    Normalize Σ_l IS_r^k(l) = 1
    Update (τ_r^k, p_r^k) via Eqs. (16), (18) and (19)
    for i = 1 to τ_r^k do
        Sample mini-batch ξ_k ⊂ D^k
        Compute gradient ∇L_k(w_r^k; ξ_k)
        Update local model w_r^k ← w_r^k − η∇L_k(w_r^k; ξ_k)
    Select layers to prune P_r^k based on p_r^k
    Keep top-mass layers by IS_r^k(l) to meet p_r^k
    Construct m_r^k
    Generate masked delta: (Δ_r^k)_masked
    Set γ_r^k = Σ_{l∉P_r^k} IS_r^k(l)
    Upload ((Δ_r^k)_masked, τ_r^k, γ_r^k, m_r^k)
```

IV. FLEAT

This section introduces the FLEAT framework, which formalizes energy-constrained federated optimization in heterogeneous IoT networks as a dual-objective problem. The goal is to trade off the global model loss and aggregate energy consumption while adhering to per-device resource limitations.

A. Overview and Structure

Existing adaptive FL methods such as fedAvg [8], FedNova [10], and FedProx [9] primarily mitigate data or update heterogeneity but assume fixed model structures and communication budgets, which are impractical in energy- and bandwidth-limited IoT environments. To address this limitation, FLEAT jointly adapts two critical parameters: (i) the local update frequency $\tau_r^k \in [\tau^{\min}, \tau^{\max}]$, which affects computation energy and local model freshness, and (ii) the pruning ratio $p_r^k \in [0, p^{\max}]$, which governs communication cost and model distortion. This dual adaptation leverages the complementary trade-off between computation and communication: increasing τ_r^k accelerates convergence but raises energy usage, while increasing p_r^k reduces transmission load but can impair accuracy. By dynamically balancing these effects across rounds, each device operates near its energy-efficient point without compromising global learning performance. Layer-wise pruning is adopted to achieve fine-grained adaptivity under heterogeneous architectures [23]. Furthermore, FLEAT derives a measurable per-round upper bound that explicitly accounts for pruning effects and provides closed-form or numerically simple updates for (τ_r^k, p_r^k).

The FLEAT structure, illustrated in Fig. 3, establishes an adaptive, resource-efficient FL paradigm tailored for heterogeneous IoT ecosystems. It integrates six core components that work in synergy to address straggler mitigation, resource optimization, and convergence. The Monitoring Module tracks real-time device latency and communication overhead, feeding metrics to the Optimization Module, which performs constrained hyperparameter optimization to compute local update frequencies τ_r^k and layer pruning ratios p_r^k. The Training Module implements localized SGD, while the Pruning Module applies structured sparsity using a *normalized gradient-norm*

importance. The Communication Module synchronizes pruned models; the Database Module stores snapshots and telemetry. Algorithms 2–3 summarize the protocol.

B. Optimization model

We formulate the FL control as a per-round computation–communication trade-off. The per-round completion time for device k is $T_r^k = Y_r^k + C_r^k + C_k'$, where $Y_r^k = \phi_r^k \tau_r^k$, $C_r^k = b_0(1-p_r^k)/\mathrm{BW}_k^{\uparrow}$, and $C_k' = b_0/\mathrm{BW}_k^{\downarrow}$. The corresponding per-round energy is $E_r^k = \bar{P}_{\mathrm{comp},r}^k Y_r^k + \bar{P}_{\mathrm{send},r}^k C_r^k + \bar{P}_{\mathrm{rec},r}^k C_k'$. The multi-objective optimization balances learning progress against energy consumption in each round:

$$\min_{\{\tau_r^k\}_{k=1}^K,\ \{p_r^k\}_{k=1}^K} \quad \alpha\,\mathbb{E}[F(w_{r+1}) \mid w_r] + (1-\alpha)\sum_{k=1}^{K} E_r^k \tag{13a}$$

$$\text{s.t.} \quad T_r^k \le T^{\max}, \quad \forall k, \tag{13b}$$

$$0 \le p_r^k \le p^{\max}, \quad \forall k, \tag{13c}$$

$$\tau_r^k \in [\tau^{\min}, \tau^{\max}] \cap \mathbb{N}, \quad \forall k \tag{13d}$$

where α balances accuracy versus energy consumption, $F(w_r; D)$ denotes the global loss, $T^{\max}$ and T specify the per-round and total time budgets, $p^{\max}$ constrains the maximum pruning ratio, and $\tau^{\min,\max}$ the local iteration counts. To address scale disparities between model loss and energy consumption, FLEAT employs client-side normalization to $[0, 1]$ using initial reference values. Direct optimization of (13) is intractable due to non-convexity and stochasticity [27], [36], [39]. Instead, we derive a per-round error bound in terms of τ_r^k and p_r^k under the time constraint T_r. In synchronous FL, the round duration is $T_r = \max_k T_r^k$. Note that T_r couples clients through the straggler term. A global wall-clock budget T is enforced by terminating after the largest number of rounds R such that $\sum_{r=1}^{R} T_r \le T$. To combine loss and energy in (13) without mixing units, we optimize a *dimensionless* objective by normalizing each term with a fixed reference scale (defined once for the experiment), i.e., $\widehat{F}_r := (F(w_r; D) - F_{\inf})/F_{\mathrm{ref}}$ and $\widehat{E}_r := \big(\sum_{k=1}^{K} E_r^k\big)/E_{\mathrm{ref}}$, and we replace $F(w_r; D)$ and $\sum_k E_r^k$ in (13) by $\widehat{F}_r$ and $\widehat{E}_r$. All terms in (13) are measurable at the client/server.

DVFS-aware interpretation. The terms $\bar{P}_{\mathrm{comp},r}^k$, $\bar{P}_{\mathrm{send},r}^k$, and $\bar{P}_{\mathrm{rec},r}^k$ represent *round-dependent phase-average* powers. This covers DVFS and workload-dependent variability: instantaneous power may fluctuate within a phase, but the objective only needs the corresponding phase energy, i.e., $\bar{P}\times$duration. In the real-world scenario, these phase energies can be obtained directly via on-device measurement; in simulation, they can be estimated from a DVFS model or calibrated averages.

Choosing α in practice. In deployment, α is an operator-set budget-driven policy knob: choose α so the average energy per round meets a target over a short calibration window. If conditions change, e.g., due to DVFS or bandwidth fluctuations, a controller can adjust α online using recent measurements: lower α when energy exceeds budget and raise it when accuracy drops (or loss rises) beyond a target. This keeps the optimization unchanged and shifts the operating point on the energy–accuracy trade-off curve.

C. Upper Bound on Learning Error

We begin by establishing the foundational assumptions and theoretical framework for analyzing the learning error in our FL system with coverage-driven pruning.

1) Standard Assumptions: We adopt the following standard assumptions from the FL literature [9], [11], [27], [28]:

Assumption 1 (L-smoothness). $F(\cdot)$ has L-Lipschitz gradients. $\|\nabla F(x; D) - \nabla F(y; D)\| \le L\|x - y\|$, with L denoting the Lipschitz constant and $F_{\inf}$ the infimum of F.

Assumption 2 (Unbiased local SGD). $\mathbb{E}_{\xi_k}\big[\nabla \mathcal{L}_k(w_r^k; \xi_k)\big] = \nabla F_k(w_r^k)$.

Assumption 3 (Bounded variance). $\mathbb{E}\|\nabla \mathcal{L}_k(w_r^k; \xi_k) - \nabla F_k(w_r^k)\|^2 \le \sigma_k^2$.

2) Coverage-Driven Pruning Model: In FLEAT, pruning is applied only to the *uploaded delta*:

the client performs local SGD on the full model to obtain $\Delta_r^k = w_r^k - w_r$, and then uploads $(\Delta_r^k)_{\text{masked}} = m_r^k \odot \Delta_r^k$. Therefore, pruning does not alter the local SGD steps; it only perturbs the *aggregated update* at the server. To analyze this effect, define the per-round average-gradient proxy

$\bar{g}_r^k \triangleq -\frac{\Delta_r^k}{\eta\,\tau_r^k} = \frac{1}{\tau_r^k}\sum_{t=0}^{\tau_r^k - 1} \nabla \mathcal{L}_k(w_{r,t}^k; \xi_{k,t})$,

and its masked counterpart induced by the uploaded delta, $\tilde{\bar{g}}_r^k \triangleq -\frac{(\Delta_r^k)_{\text{masked}}}{\eta\,\tau_r^k} = m_r^k \odot \bar{g}_r^k$.

The pruning-induced aggregation error is $e_r^k \triangleq \bar{g}_r^k - \tilde{\bar{g}}_r^k$. Pruning is applied only to the uploaded delta, so it does not change the local SGD trajectory; it only changes the vector aggregated by the server. Accordingly, e_r^k should be interpreted as an aggregation perturbation (missing update blocks), not as a distortion of the local gradient used for training. Note that e_r^k can be biased because it depends on the mask and the realized gradients; our analysis does not require $\mathbb{E}[e_r^k] = 0$. We use a conditional second-moment bound on $\|e_r^k\|^2$, yielding a conservative but valid convergence guarantee. We define the kept-coverage using the normalized layer-wise energy of $\bar{g}_r^k$:

$$IS_r^k(l) = \frac{\|\bar{g}_r^k(l)\|^2}{\|\bar{g}_r^k\|^2}, \qquad \gamma_r^k = \sum_{l \notin \mathscr{P}_r^k} IS_r^k(l) \in [0, 1].$$

In implementation, $IS_r^k(l)$ can be estimated using an accumulated-gradient/EMA variant of Eq. (7); the bound below uses $\bar{g}_r^k$ for clarity.

Lemma 1 (Communication-side pruning distortion)**.** *With pruning set $\mathscr{P}_r^k$ chosen by removing the lowest importance mass to meet target ratio p_r^k, the aggregation error satisfies*

$$\mathbb{E}\|e_r^k\|^2 \le (1 - \gamma_r^k)\, G_k^2,$$

where $G_k^2 \triangleq \mathbb{E}\|\bar{g}_r^k\|^2$ (and G_k^2 can be upper-bounded by a per-step gradient second moment).

The orthogonality used in Appendix B-A is with respect to disjoint *parameter blocks* (layer-wise coordinates), hence it is not violated by residual/attention couplings in the computation graph. A conservative relaxation for non-orthogonal decompositions (e.g., parameter tying) is also discussed there. If parameter tying or shared modules break the disjoint-block view, the bound should be read as a conservative guarantee with a larger constant, so it can be less tight even though

the convergence statement remains valid. We also analyze the structural properties of importance distributions across layers:

Lemma 2 (Properties of Importance Curves)**.** *Let $S_k(p)$ denote the* pruned-mass curve*, defined as the cumulative importance of the smallest fraction p of parameters [40]. Then $S_k : [0,1] \to [0,1]$ is increasing, convex, with $S_k(0) = 0$, $S_k(1) = 1$, and $S'_k(p)$ nondecreasing.*

Proof Sketch. The proof, detailed in Appendix B-B, relies on sorting parameters by increasing importance and analyzing the resulting Lorenz curve. Monotonicity follows from nonnegative importance scores, while convexity stems from the nondecreasing property of sorted importances. □

3) Theoretical Error Bound: We now present our main theoretical contribution, which bounds the learning error under coverage-driven pruning.

Theorem 1 (Per-round Upper Bound with Pruning)**.** *Let the learning rate η satisfy the stability condition [27]:*

$$\eta L + \eta^2 L^2 \tau_r^k (\tau_r^k - 1) \leq 1.$$

Under Assumption1–Assumption3, based on the Appendix D in [27], the expected optimality gap after round r satisfies:

$$\begin{aligned}\Psi_r &\triangleq \mathbb{E}[F(w_{r+1}) - F_{\inf}] \\ &\leq \frac{2\left[F(w_r) - F_{\inf}\right]}{\eta T_r} \sum_{k=1}^{K} \frac{|D^k|}{|D|} \left(\phi_r^k \tau_r^k + \frac{N_r^k}{\tau_r^k}\right) \\ &\quad + \eta L \sum_{k=1}^{K} \frac{|D^k|}{|D|} M_r^k + \eta^2 L^2 \sum_{k=1}^{K} \frac{|D^k|}{|D|} M_r^k (\tau_r^k - 1).\end{aligned} \tag{14}$$

where $M_r^k \triangleq \sigma_k^2 + (1 - \gamma_r^k) G_k^2$ is the combined noise term: σ_k^2 captures stochastic-gradient variance and $(1-\gamma_r^k)G_k^2$ upper-bounds the second moment of the pruning-induced aggregation error *in Lemma 1, $G_k^2 \triangleq \mathbb{E}\|\bar{g}_r^k\|^2$ denotes the expected squared gradient norm, and $N_r^k = b_0(1-p_r^k)/\mathrm{BW}_k^{\uparrow}$.*

Proof Sketch. The complete proof is provided in Appendix B-C. The key steps include: (1) applying the L-smoothness descent lemma across local steps, (2) incorporating pruning distortion via Lemma 1, (3) using Assumption 3 (bounded stochastic-gradient variance) and a conservative second-moment bound for the masking error. □

The bound in Theorem 1 reveals three key components affecting convergence: the progress from the previous round, the noise from stochastic gradients and pruning, and higher-order correction terms. The pruning effect appears primarily through the $(1 - \gamma_r^k)$ term in M_r^k, quantifying the trade-off between communication efficiency and learning accuracy.

Corollary 1 (Nonconvex Convergence Rate)**.** *Choose the stepsize $\eta = \min\{1/(2L), c/\sqrt{R}\}$ for any constant $c > 0$. Then the averaged gradient norm over R rounds satisfies:*

$$\frac{1}{R} \sum_{r=1}^{R} \mathbb{E}\|\nabla F(w_r)\|^2 = \mathcal{O}\left(\frac{1}{\sqrt{R}}\right),$$

with constants depending on $\max_{k,r} M_r^k$.

Proof Sketch. The proof (Appendix B-D) telescopes Theorem 1 over R rounds and rearranges terms. The $\mathcal{O}(1/\sqrt{R})$ rate follows from standard nonconvex optimization arguments, with constants inflated by pruning-induced variance. □

D. Solution to find τ_r^k and p_r^k

Combine (14) with the energy term in (13). The sum over clients is separable; we minimize each client's per-round objective function. Let

$$A_k = \frac{2[F(w_r) - F_{\inf}]}{\eta T_r} \frac{|D^k|}{|D|}, \quad B_k = \eta L \frac{|D^k|}{|D|}, \quad C_k = \eta^2 L^2 \frac{|D^k|}{|D|}.$$

Client k minimizes

$$\begin{aligned}J_k(\tau, p) \;=\; &\alpha \left[A_k \left(\phi_r^k \tau + \frac{N_r^k}{\tau} \right) + (B_k + C_k(\tau - 1))\, M_r^k \right] \\ &+ (1 - \alpha) \left[P_{\mathrm{comp}}^k \phi_r^k \tau + P_{\mathrm{send}}^k N_r^k + P_{\mathrm{rec}}^k C'_k \right].\end{aligned} \tag{15}$$

Proposition 1 (Bi-convexity)**.** *For fixed pruning ratio p, the objective function $J_k(\tau, p)$ is convex in τ. For fixed local steps τ, $J_k(\tau, p)$ is convex in p when the pruned-mass curve $S_k(p) \equiv 1 - \gamma_r^k(p)$ is convex, as established in Lemma 2.*

Proof Sketch. The complete proof is provided in Appendix C-A. For fixed p, convexity in τ follows from analyzing the second derivative of the τ-dependent terms. For fixed τ, convexity in p relies on the convexity of $S_k(p)$ and the affine structure of the remaining terms. □

This bi-convex structure enables alternating minimization approaches that converge to stationary points efficiently.

1) Optimal Local Steps τ_r^k: We first derive the optimal local computation steps for a given pruning ratio.

Theorem 2 (Closed-form Update for τ)**.** *For fixed pruning ratio p_r^k and coverage γ_r^k, the optimal local steps are given by:*

$$\tau_r^{k,*} = \sqrt{\frac{N_r^k}{\phi_r^k + \Gamma_r^k}}, \quad \Gamma_r^k = \frac{C_k M_r^k + \frac{1-\alpha}{\alpha} P_{\mathrm{comp}}^k \phi_r^k}{A_k}, \tag{16}$$

with feasibility projection:

$$\tau_r^{k,*} \leftarrow \min\left\{ \tau^{\max}, \left\lfloor \min\left(\sqrt{\frac{N_r^k}{\phi_r^k + \Gamma_r^k}}, \frac{T^{\max} - N_r^k}{\phi_r^k} \right) \right\rfloor \right\}, \tag{17}$$

where $N_r^k = b_0(1 - p_r^k)/\mathrm{BW}_k^{\uparrow}$.

Proof Sketch. The derivation, detailed in Appendix C-B, involves differentiating the objective function with respect to τ, setting the derivative to zero, and solving the resulting equation. The projection ensures compliance with maximum step count and round time constraints. □

We use the normalized, dimensionless objective defined in Section IV-B; hence all constants (e.g., A_k, B_k, C_k) are formed from normalized quantities, and the stationary point in (16) is a pure (dimensionless) number of local steps. the closed-form update in this theorem uses the ratio $(1 - \alpha)/\alpha$, so we set $\tilde{\alpha} := \max(\alpha, \epsilon_\alpha)$ with a small $\epsilon_\alpha > 0$ (e.g., 10^{-6}) to avoid

division by zero. Equivalently, we can rewrite the weighted sum as $\widehat{F}_r + \lambda \widehat{E}_r$ with $\lambda := (1-\alpha)/\tilde{\alpha}$, and for the energy-only extreme ($\alpha = 0$) we use $\lambda = (1-\alpha)/\epsilon_\alpha$ (optionally clipped to a large $\lambda_{\max}$ for numerical stability).

2) Optimal Pruning Ratio p_r^k: We now derive the optimal pruning ratio for fixed local steps.

Theorem 3 (Optimality Condition for p)**.** *For fixed local steps τ_r^k, the optimal pruning ratio satisfies:*

$$S_k'(p_r^{k,*}) = \frac{R_k}{D_k}, \quad p_r^{k,*} \in [0, p^{\max}], \tag{18}$$

where:

$$R_k = \alpha A_k \frac{b_0/\mathrm{BW}_k^{\uparrow}}{\tau_r^k} + (1-\alpha) P_{\mathrm{send}}^k (b_0/\mathrm{BW}_k^{\uparrow}),$$
$$D_k = \alpha \left(B_k + C_k(\tau_r^k - 1)\right) G_k^2,$$

and $S_k'(p)$ is the derivative of the pruned-mass curve.

Proof Sketch. Appendix C-C provides the complete derivation. The key insight is that the optimal pruning ratio balances the communication savings (numerator R_k) against the distortion cost (denominator D_k) through the importance distribution's characteristics. □

For scenarios requiring closed-form solutions, we provide a parametric approximation:

Corollary 2 (Closed-form Parametric Update)**.** *If the pruned-mass curve follows a power law $S_k(p) \approx \kappa_r^k p^{\beta_r^k}$, then:*

$$p_r^{k,*} = \left[\frac{R_k}{D_k \kappa_r^k \beta_r^k}\right]^{\frac{1}{\beta_r^k - 1}} \text{ clamped to } [0, p^{\max}]. \tag{19}$$

A robust initialization uses $\beta_r^k = 2$ and $\kappa_r^k = \frac{S_k(p_{r-1}^k)}{(p_{r-1}^k)^2}$ from previous rounds. $\kappa_r^k > 0$, $\beta_r^k \geq 1 \Rightarrow S_k'(p) = \kappa_r^k \beta_r^k p^{\beta_r^k - 1}$ is nondecreasing.

Theorem 4 (Alternating Minimization Convergence)**.** *Let $\{p^{(t)}, \tau^{(t)}\}$ be sequences generated by alternating updates of* (16) *and* (18) *(or* (19)*) with feasibility projection. Every limit point is a stationary point of the bi-convex program $\min_{\tau,p} J_k(\tau, p)$ under box and round-time constraints.*

Proof Sketch: This follows from alternating convex search results for bi-convex optimization. In particular, Theorem 4.9 and Corollary 4.10 in [41] imply that, under compact feasible sets and exact block minimization as in Theorems 2 and 3, every limit point is a stationary point of $J_k(\tau, p)$.

3) Convergence Behavior and Practical Insights: A detailed nonconvex convergence analysis and system-level design implications derived from Theorem 1 are provided in Appendix D-A. These include the formal convergence rate, pruning-computation trade-offs, and coverage guarantees ensuring bounded distortion.

4) Implementation Details and Convergence Guarantees: Algorithmic aspects such as the empirical estimation of the pruned-mass curve, the bisection-based solver for p_r^k, and the alternating minimization convergence analysis are presented in Appendix D-B. These also discuss integration strategies, adaptive server-side aggregation, and measurability considerations.

TABLE II
SPECIFICATION OF EDGE DEVICES

Device	Model	CPU	RAM	OS
	Pi 0	1.0 GHz single-core ARM1176JZF-S (ARM11)	512 MB	Raspberry Pi OS Lite (32-bit, Debian 12 "Bookworm")
	Pi 3	1.2 GHz quad-core ARM Cortex-A53 (ARMv8)	1 GB	Raspberry Pi OS Lite (32-bit, Debian 12 "Bookworm")
	Pi 4	1.5 GHz quad-core ARM Cortex-A72 (ARMv8)	2 GB	Raspberry Pi OS (64-bit, Debian 12 "Bookworm")
	Nano	1.43 GHz quad-core ARM Cortex-A57	4 GB	Ubuntu 18.04 LTS (NVIDIA JetPack 4.6 / L4T 32.7)[1]
	Xavier NX	6-core NVIDIA Carmel (ARMv8.2)	8 GB[2]	Ubuntu 20.04 LTS (NVIDIA JetPack 5.x / L4T 35.x)[1]
	Server	Dual Intel Xeon Gold 6248R (2×24 cores) @ 2.9 GHz	128 GB	Ubuntu Server 22.04 LTS

Importance-metric discussion. The pruning module admits alternative layer-importance metrics by replacing Eq. 7 with magnitude-based or Fisher-based scores (Appendix D-C) while keeping the same byte budget and non-consecutive constraint.

Effect of aggregation normalization under partial uploads. The normalization in Eq. (4) and the per-layer renormalization in Eq. (6) are designed to prevent aggregation bias when clients use different local steps and upload masks. We isolate this effect analytically in Appendix D-B1. In short, treating missing layers as zeros introduces a layer-dependent shrinkage factor that can distort the global update direction when pruning patterns differ across clients, while Eq. (6) removes this shrinkage by renormalizing over contributors.

E. Computational Complexity Analysis

FLEAT introduces negligible overhead compared with standard local training. The computation of the importance score $IS_r^k(l)$ for all L layers requires $O(L)$ operations per device, which is minor relative to the $O(|D_k| \cdot P)$ cost of backpropagation, where $|D_k|$ and P denote the local dataset size and the number of model parameters, respectively. Updating the adaptive variables (τ_r^k, p_r^k) involves closed-form evaluations and a simple bisection search, of $O(1)$ and $O(\log(1/\epsilon))$ complexities. So, the FLEAT overhead is asymptotically negligible and does not affect the scalability across large settings.

V. PERFORMANCE EVALUATION

A. Datasets and Models

1) **Real World Scenarios:** We conduct a practical deployment on actual IoT devices, training and testing CNN and RNN models using the Edge-IIoTset [42] dataset. This comprehensive IoT/IIoT cybersecurity dataset is generated on a realistic testbed with diverse devices, protocols, and cloud/edge configurations. It includes more than ten sensor types (e.g., temperature, ultrasonic, water-level). From 1176 features, the dataset provides 61 high-correlation features, supporting centralized and federated intrusion detection. The CNN has two convolutional layers (16 and 32 filters, 3×3 kernel, ReLU activation), a max-pooling layer, a dense layer

[2]A 16 GB Xavier NX module variant exists; use 8 GB for standard dev kit.

(128 neurons), and a softmax output for efficient spatial feature extraction from sensor data. The RNN, designed for time-series analysis, includes two GRU layers (64 and 32 units), a dropout layer for regularization, a dense layer (64 neurons), and a softmax classifier, enabling sequential pattern learning. Batch size, learning rate, and local updates per FL round are $64, 1 \times 10^{-3}, 1$ for CNN and $32, 5 \times 10^{-4}, 2$ for RNN.

2) **Simulation Scenarios***:* We evaluate FLEAT through four neural architectures spanning computational footprints relevant to IoT-edge deployments. We select EfficientNet-B0 (5.3M parameters) and MobileNetV2 (3.4M parameters) for lightweight edge inference, both optimized for resource-constrained devices. To assess scalability, we include moderate-scale models: ResNet-18 (11.7M parameters) and GoogleNet (6.8M parameters), the latter using 22-layer inception modules for architectural diversity. Experiments span four vision datasets to evaluate FLEAT's robustness to data heterogeneity. Efficient Net-B0, MobileNetV2, ResNet-18, and GoogleNet are trained on datasets MNIST, SVHN, CIFAR-10, and FashionMNIST (FMNIST). Batch size, learning rate, and local updates per FL round are $(32,\ 1 \times 10^{-3},\ 1)$ for EfficientNet-B0, $(64,\ 1 \times 10^{-3},\ 2)$ for MobileNetV2, $(64,\ 1 \times 10^{-3},\ 3)$ for ResNet-18, and $(32,\ 5 \times 10^{-4},\ 2)$ for GoogleNet. The MNIST dataset has 70,000 grayscale 28×28 images of handwritten digits (0-9), split into 60,000 training and 10,000 test images. SVHN contains 600,000+ color 32×32 images of house numbers (0-9), with 73,257 for training and 26,032 for testing. CIFAR-10 contains 60,000 color 32×32 images in 10 classes, with 50,000 for training and 10,000 for testing. FashionMNIST (FMNIST), similar to MNIST but for clothing, has 70,000 grayscale 28×28 images across 10 fashion categories.

B. Baselines and Metrics

1) **Baselines***:* FLEAT is compared against three model-centric baselines: (i) Local Update-only, dynamically tuning local steps without pruning; (ii) Pruning-only, applying structured sparsity with fixed steps; and (iii) FedAvg [8], FedProx [9], FedNova [10], and SCAFFOLD [11], the FL standard with the same batch size, optimizer, and rounds. (iv) Infrastructure-focused comparators include ALTD [19], optimizing CPU/transmission power, and FFL [36], allocating compute/communication resources. Additionally, we evaluate FLEAT across weighting factors α to analyze the accuracy-energy trade-off. Unless noted otherwise, $\alpha = 0.5$ is our default, and all results use the normalized server aggregation.

2) **Metrics***:* Performance is quantified via four metrics: (i) energy consumption, aggregating computation and communication costs per device; (ii) round-wise accuracy, tracking global test accuracy progression; (iii) time-dependent accuracy, measuring accuracy relative to cumulative training time.

C. Experimental Setup

1) ***Real-World Scenario****:* We deploy five heterogeneous edge devices and one high-performance edge server (Table II). Each device runs the FL client inside a Docker container (ARM64 on Raspberry Pi/Jetson; x86-64 on the server) for dependency isolation and resource control. Clients communicate with the coordinator over gRPC (HTTP/2 with Protocol Buffers; optional TLS and compression). Containers implement health checks, timeouts, and exponential-backoff retries to handle transient network issues. At the start of each round, the server makes the current global model available; clients fetch it via gRPC, perform local training, and upload their updates. For the real-world runs, we log per-phase power at ≈10 Hz and use median steady-state values per device. Measured application-layer throughputs (*uplink*/*downlink*) and phase powers (*compute*/*send*/*receive*) are: Raspberry Pi 0 (Wi-Fi 4): $\mathrm{BW}^{\uparrow}$≈35 Mbps, $\mathrm{BW}^{\downarrow}$≈70 Mbps; P_{comp}≈3.0 W, P_{send}≈2.0 W, P_{rec}≈1.8 W. Raspberry Pi 3 (Wi-Fi 4): 55/110 Mbps; 4.2/2.8/2.4 W. Raspberry Pi 4 (Wi-Fi 5): 70/140 Mbps; 6.5/4.2/3.7 W. Jetson Nano (1 GbE): 930/930 Mbps; 9.0/6.0/5.2 W. Jetson Xavier NX (1 GbE): 940/940 Mbps; 12.0/7.2/6.0 W. Edge Server (1 GbE): 940/940 Mbps; 85.0/20.0/15.0 W. These values drive per-round time/energy accounting for the *compute*, *send*, and *receive* phases and are also recorded to report end-to-end energy and time-to-accuracy metrics.

TABLE III
SIMULATION DEVICE TIERS: SAMPLING RANGES (UNIFORM) FOR PER-NODE THROUGHPUTS (MBPS) AND PHASE POWERS (W)

Tier	$\mathrm{BW}^{\uparrow}$	$\mathrm{BW}^{\downarrow}$	P_{comp}	P_{send}	P_{rec}
Wi-Fi 5 SBCs	$[40, 90]$	$[80, 180]$	$[5.5, 11]$	$[3.5, 6.5]$	$[3.0, 5.5]$
Wi-Fi 6 SBCs	$[120, 320]$	$[250, 700]$	$[8, 16]$	$[4.5, 8.5]$	$[3.5, 7.0]$
1GbE mini-PCs	$[800, 950]$	$[800, 950]$	$[10, 30]$	$[7, 12]$	$[5, 9]$

2) **Simulation Scenarios***:* We simulate N=50 edge nodes with $\tau^{\min}$=2, $\tau^{\max}$=10, $p^{\max}$=0.5. Training runs for 50 rounds (batch size 32) with RMSprop (init. LR 0.01, decay 0.995). To emulate realistic heterogeneity, each node is assigned at t=0 to one of three connectivity/compute tiers with probabilities $0.55/0.30/0.15$ (Wi-Fi 5 SBCs / Wi-Fi 6 SBCs / 1GbE mini-PCs). Per node, the application-layer throughputs and phase powers are sampled once per run (uniform ranges in Table III) and held fixed across rounds. We use the same (ϕ) weights as real-world runs and fix the RNG seed to 42.

We evaluate FLEAT in real-world and simulation settings using Dir(0.5) label-skew non-IID partitions [43], which represent moderate heterogeneity. With more extreme non-IID data (smaller Dirichlet concentration), gradients may vary more and per-round layer-importance scores may become noisier. FLEAT remains applicable since IS_r^k is computed locally and the controller uses measured ϕ_r^k and coverage γ_r^k, though in highly skewed settings IS_r^k can be stabilized via smoothing across rounds (e.g., exponential moving average) or by using gradients accumulated over the τ_r^k local steps. A systematic sweep of Dirichlet concentrations is left to the extended evaluation. We also vary α to study the energy–loss trade-off and compare FLEAT against baseline and state-of-the-art methods to validate our optimization strategy. Additional experiments, including heterogeneity sensitivity, layer-importance ablation, and aggregation-renormalization ablation, are reported in Appendix E.

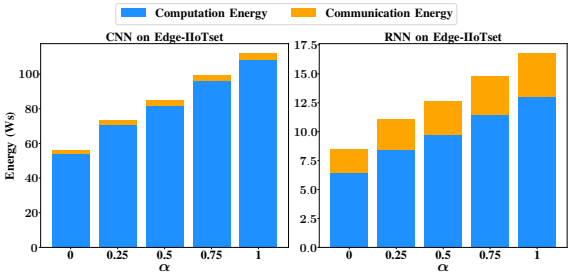

Fig. 4. Total training energy across models on Edge-IIoTset dataset for different values of the objective weight (α)

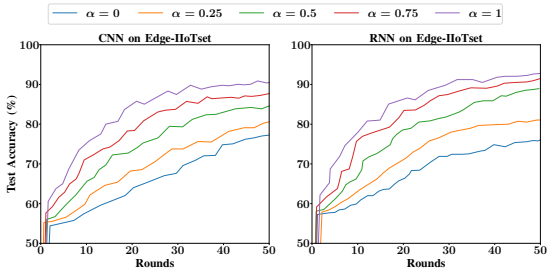

Fig. 5. Test accuracy vs. FL rounds for different values of the objective weight (α)

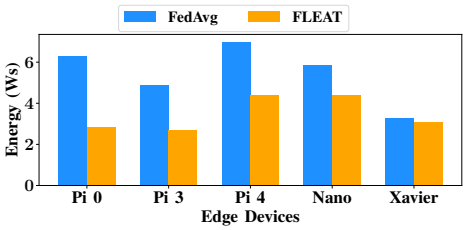

Fig. 6. Per-device energy breakdown on heterogeneous hardware under the same workload and network.

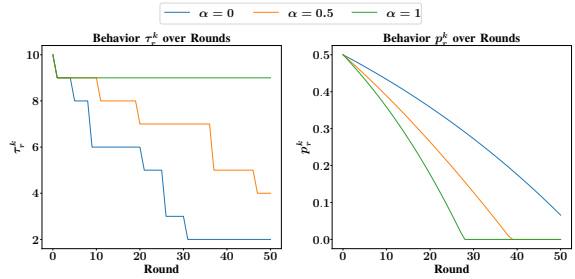

Fig. 7. Evolution of local steps τ_r^k and pruning ratio p_r^k over rounds for different values of α on a Raspberry Pi 3.

D. Real World Scenarios

*1) **Exploring the Impact of objective weight on Model Performance**:* Figure 4 illustrates how FLEAT tunes local updates and pruning ratio on Edge-IIoTset to balance energy and loss via α. At $\alpha = 0$, it prioritizes energy, using aggressive pruning (above 60–70%) and fewer local updates to reduce computation and communication. At $\alpha = 1$, it uses little or no pruning (below 10%) and more local updates, improving accuracy but increasing energy. For the CNN, total energy rises from about 60 to 100 W_s as α increases from 0 to 1; for the RNN, it rises from about 8 to 16 W_s. At $\alpha = 0.5$, energy is about 85 W_s for the CNN and 12 W_s for the RNN, giving a 24–25% reduction relative to $\alpha = 1$ while still improving accuracy over the energy-first setting ($\alpha = 0$).

Figure 5 reports accuracy over 50 FL rounds. For the CNN, accuracy increases from about 76% at $\alpha = 0$ to 90% at $\alpha = 1$, while $\alpha = 0.5$ reaches about 85–87% at lower energy cost. The RNN shows the same trend, rising from about 75% at $\alpha = 0$ to above 90% at $\alpha = 1$, with $\alpha = 0.5$ around 85%. These results confirm that adapting local updates and pruning via α yields a tunable accuracy–energy tradeoff. In particular, $\alpha = 0.5$ provides a good compromise, remaining within 4–6% of peak accuracy while lowering total energy consumption by up to 25%, which is especially valuable in heterogeneous edge environments, where devices face different energy constraints.

*2) **Energy Consumption Comparison Across Heterogeneous Edge Devices**:* We analyze FLEAT's ability to reduce energy disparity across heterogeneous edge devices. Figure 6 compares per-device energy under FedAvg (uniform settings, no pruning) and FLEAT ($\alpha = 0.5$). With FedAvg, hardware heterogeneity causes large energy differences; for example, Device 5 (Jetson Xavier) consumes 87.3% less energy than Device 1 (Raspberry Pi 0) for the same workload. FLEAT narrows this gap by adapting local updates and pruning ratios to device capability under fixed round-time limits. Thus, even though Device 2 performs more local iterations than Device 1 because of its higher compute capacity, its energy remains 34–48% lower than with FedAvg. In this way, constrained devices use more pruning and fewer updates, while stronger devices exploit extra iterations for refinement. Overall, FLEAT cuts the inter-device energy gap by 41–63% versus FedAvg, improving system-wide efficiency without sacrificing convergence.

Fig. 7 illustrates the round-wise evolution of local updates (τ_r^k) and pruning (p_r^k) on a Raspberry Pi 3 under different α values. For $\alpha = 0$, τ_r^k gradually falls to the minimum value of 2, while $\alpha = 1$ stabilizes at 9 to emphasize accuracy. With $\alpha = 0.5$, τ_r^k starts at 10 for faster early convergence and later drops to 4 to save energy. Pruning shows the same pattern: $\alpha = 0$ starts aggressively at $p_r^k = 0.5$ and decreases to 0.3, whereas $\alpha = 1$ quickly reduces pruning to 0. For $\alpha = 0.5$, pruning is moderate in early rounds (0.2–0.4) and then tapers to 0.0–0.1. This staged adjustment lowers energy use by 22–37% compared with $\alpha = 1$ while keeping accuracy within 4–6% of the loss-focused setting. Overall, by adapting τ_r^k and p_r^k across training phases, FLEAT achieves a strong balance between energy efficiency and model performance.

E. Simulation Scenarios

*1) **Exploring the Impact of objective weight on Model Performance**:* In the simulation setting, we similarly study the effect of the objective weight $\alpha \in 0, 0.25, 0.5, 0.75, 1$ on the energy–accuracy trade-off. Using the same methodology as in the real-world experiments, we evaluate how α affects computation and communication energy across models and datasets. The results are consistent with the real-world findings

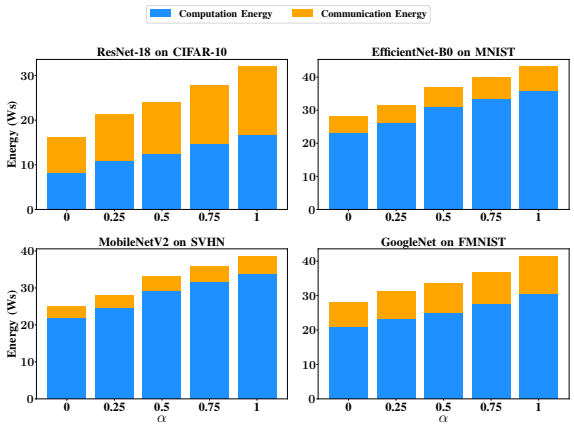

Fig. 8. Total energy (W_s) for values of the objective weight(α)

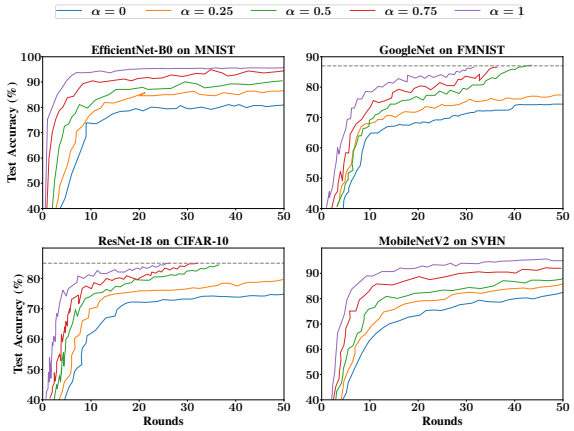

Fig. 9. Accuracy relative to FL rounds for values of objective weight (α)

and confirm the expected trade-off between energy efficiency and model performance. For example, with ResNet-18, energy at $\alpha = 0.5$ is about 47.2% and 12.1% higher than at $\alpha = 0$ and $\alpha = 0.25$, but 13.6% and 25.1% lower than at $\alpha = 0.75$ and $\alpha = 1$, respectively. Overall, increasing α by 25% raises computation and communication energy by averages of 13.1% and 12.6%, because the optimizer reduces pruning and increases local updates (τ_r^k) to improve model refinement. The larger rise in computation energy occurs because pruning affects only uplink transmission, while downlink model reception is unchanged, and computation grows directly with τ_r^k.

A clear energy–accuracy trade-off makes the choice of α important. As shown in Figure 9, $\alpha = 0$ maximizes energy savings through aggressive pruning and fewer local updates, but lowers test accuracy by 12–15% relative to $\alpha = 1$, especially in later rounds. By contrast, $\alpha = 1$ achieves the highest accuracy (e.g., 85% on CIFAR-10 with ResNet-18) but increases computation and communication energy by 100% and 91.3%, respectively, compared with $\alpha = 0$. A balanced setting, $\alpha = 0.5$, provides the best compromise, reaching 84.5% accuracy, only 0.5% below $\alpha = 1$, while using 47.2% less energy. These results highlight the importance of jointly optimizing accuracy and energy in resource-constrained edge settings. Across ResNet-18, EfficientNet-B0, MobileNetV2,

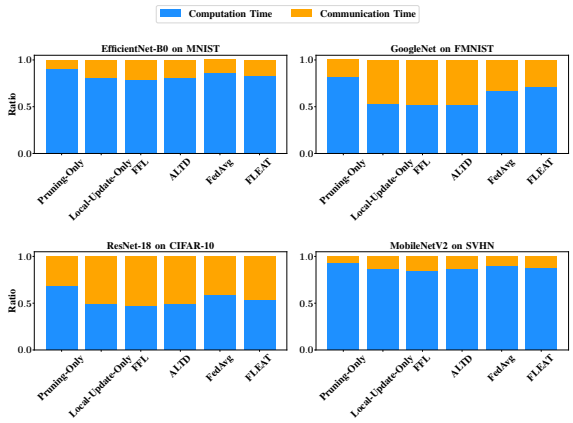

Fig. 10. Compute vs. communication time share in total training time

and GoogleNet on CIFAR-10, MNIST, SVHN, and FMNIST, FLEAT reveals consistent energy trade-offs. GoogleNet minimizes communication energy but incurs higher computation cost, whereas ResNet-18 is more balanced. EfficientNet-B0 and MobileNetV2 are the most energy-efficient, with MobileNetV2 yielding the lowest communication overhead. As α increases, energy consumption rises sharply, nearly doubling for ResNet-18 at $\alpha = 1$ versus $\alpha = 0$. FMNIST also requires 41% more computation energy than CIFAR-10 on average. Overall, $\alpha = 0.5$ offers the best accuracy–energy balance.

*2) **A Comparative Analysis with Baseline Methods and Prior Research**:* Figure 10 presents the percentage of computation and communication time in the total FL process across four model–dataset combinations, comparing different FL optimization strategies. FLEAT outperforms FedAvg by dynamically adjusting local update frequencies and pruning levels, reducing energy consumption. Unlike Local Update-only, which minimizes local updates, and Pruning-only, which applies aggressive pruning, FLEAT integrates both approaches to achieve a balanced trade-off. ALTD exhibits higher round times due to its reliance on infrastructure-level optimizations, such as lowering CPU frequency and transmission power, which slow data processing and communication. FFL demonstrates comparable round times to FLEAT, as it similarly optimizes energy and accuracy through local update frequency adjustments and compression, though we further analyze the impact of compression on accuracy degradation.

Figures 11 shows average communication and computation energy, while Figures 12 and 13 report test accuracy over FL rounds and wall-clock time across models and datasets. Across all settings, FLEAT achieves the strongest energy–accuracy trade-off. FedAvg is about 3% more accurate in two scenarios, but consumes 16.2% more total energy. FLEAT also surpasses the *single-knob* baselines, improving accuracy by 15.6% and 23.0% while lowering energy by 4.1% and 1.2% relative to *Local-Update-Only* and *Pruning-Only*. Against FedProx, FedNova, and SCAFFOLD, FLEAT offers a better Pareto frontier, achieving lower energy for similar accuracy and higher accuracy for similar energy. Unlike FFL and ALTD, which lack an explicit energy objective, FLEAT co-tunes local steps (τ) and sparsification (p) each round, increasing τ

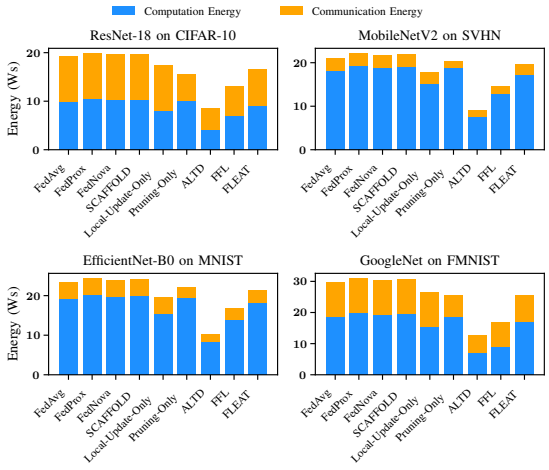

Fig. 11. Total training energy across methods.

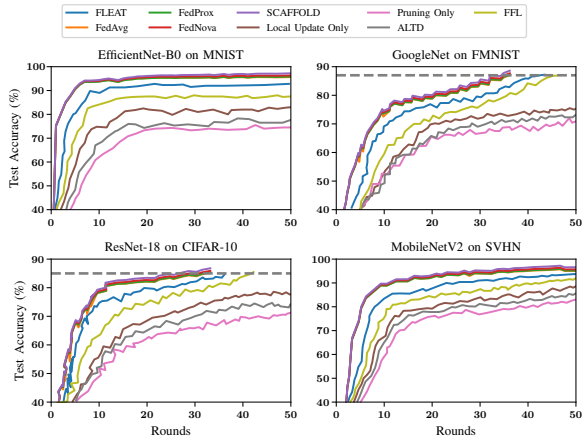

Fig. 12. Accuracy across FL rounds for different methods.

with compensating pruning early and relaxing pruning later to maintain accuracy without excess W_s.

Figures 12 and 13 highlight the slow practical convergence of FedAvg and limits of energy-only optimization. ALTD reduces energy through CPU-frequency and transmission-power control, and FFL uses local-update adaptation and compression, but neither jointly optimizes loss and energy. FLEAT instead combines both objectives, with layer-wise pruning reducing communication time while preserving accuracy. Although ALTD and FFL save more energy than FLEAT (-36%, -25%), they suffer larger accuracy drops (-19%, -8%).

VI. Conclusion

This paper introduces FLEAT, a FL framework designed to harmonize energy efficiency and model accuracy in heterogeneous edge environments. By dynamically adapting local computation and communication through layer-wise pruning and normalized gradient aggregation, FLEAT mitigates resource bottlenecks, reduces synchronization delays, and ensures stable convergence under non-IID data distributions. Experimental results demonstrate its superiority over existing methods

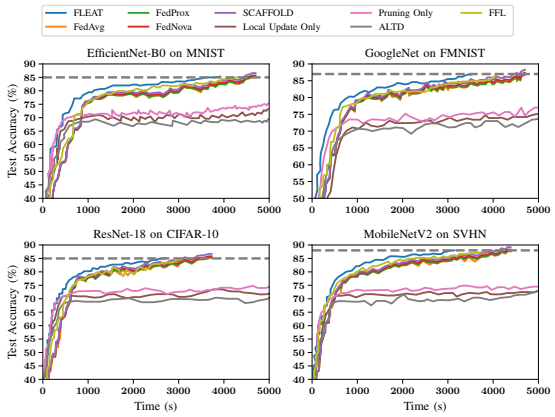

Fig. 13. Accuracy vs. wall-clock time (s) for all methods

in balancing energy consumption with accuracy preservation, offering a scalable solution for resource-constrained IoT deployments. Future work will extend FLEAT to decentralized FL, where staleness and the lack of central coordination introduce additional optimization challenges. We will adapt its mechanisms to enable energy-aware, consistent collaboration across large-scale edge networks.

References

[1] T. Wang, Y. Liang, X. Shen, X. Zheng, A. Mahmood, and Q. Sheng, "Edge computing and sensor-cloud: Overview, solutions, and directions," *ACM Computing Surveys*, 2023.

[2] D. Qiao, S. Guo, J. Zhao, J. Le, P. Zhou, M. Li, and X. Chen, "Asmafl: Adaptive staleness-aware momentum asynchronous federated learning in edge computing," *IEEE Transactions on Mobile Computing*, vol. 24, no. 4, pp. 3390–3406, 2025.

[3] C. Chen, T. Liao, X. Deng, Z. Wu, S. Huang, and Z. Zheng, "Advances in robust federated learning: A survey with heterogeneity considerations," *IEEE Transactions on Big Data*, vol. 11, no. 3, pp. 1548–1567, 2025.

[4] B. Rao, J. Zhang, D. Wu, C. Zhu, X. Sun, and B. Chen, "Privacy inference attack and defense in centralized and federated learning: A comprehensive survey," *IEEE Transactions on Artificial Intelligence*, vol. 6, pp. 333–353, 2025.

[5] Z. Chen, J. Du, X. Hou, K. Yu, J. Wang, and Z. Han, "Channel adaptive and sparsity personalized federated learning for privacy protection in smart healthcare systems," *IEEE Journal of Biomedical and Health Informatics*, vol. 28, no. 6, pp. 3248–3257, 2024.

[6] W. Hou, H. Wen, N. Zhang, W. Lei, H. Lin, Z. Han, and Q. Liu, "Adaptive training and aggregation for federated learning in multi-tier computing networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 4376–4388, 2024.

[7] H. Liu, Z. Zheng, F. Wu, and G. Chen, "From non-iid to iid: Mobility-aware hierarchical federated learning with client-edge association control," *IEEE Transactions on Mobile Computing*, vol. 24, no. 11, pp. 11 717–11 730, 2025.

[8] 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*, 2017, pp. 1273–1282.

[9] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 2, 2020, pp. 429–450.

[10] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *CoRR*, vol. abs/2007.07481, 2020. [Online]. Available: https://arxiv.org/abs/2007.07481

[11] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol.

119. PMLR, 2020, pp. 5132–5143. [Online]. Available: https://proceedings.mlr.press/v119/karimireddy20a.html
[12] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019. [Online]. Available: https://arxiv.org/abs/1903.03934
[13] L. You, Z. Tang, P. Wang, Z. Wang, H. Dai, and L. Fu, "Quick and reliable lora data aggregation through multi-packet reception," *IEEE/ACM Transactions on Networking*, 2023.
[14] A. Arouj and A. M. Abdelmoniem, "Towards energy-aware federated learning via collaborative computing approach," *Computer Communications*, vol. 221, pp. 131–141, 2024.
[15] M. Research, "Farmbeats: Ai, edge & iot for agriculture," 2023. [Online]. Available: https://www.microsoft.com/en-us/research/project/farmbeats-iot-agriculture/
[16] NVIDIA, "Nvidia clara." [Online]. Available: https://www.nvidia.com/en-us/clara/
[17] L. Luo, C. Zhang, H. Yu, G. Sun, S. Luo, and S. Dustdar, "Communication-efficient federated learning with adaptive aggregation for heterogeneous client-edge-cloud network," *IEEE Transactions on Services Computing*, vol. 17, pp. 3241–3255, 2024.
[18] H. Chen, S. Huang, D. Zhang, M. Xiao, M. Skoglund, and H. Poor, "Federated learning over wireless iot networks with optimized communication and resources," *IEEE Internet of Things Journal*, vol. 9, pp. 16 592–16 605, 2022.
[19] J. Yao and N. Ansari, "Enhancing federated learning in fog-aided iot by cpu frequency and wireless power control," *IEEE Internet of Things Journal*, vol. 8, pp. 3438–3445, 2020.
[20] N. Tran, W. Bao, A. Zomaya, M. Nguyen, and C. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, 2019, pp. 1387–1395.
[21] A. L. Nunes, C. Boeres, L. M. A. Drummond, and L. L. Pilla, "Optimal time and energy-aware client selection algorithms for federated learning on heterogeneous resources," in *36th IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2024)*. IEEE, 2024, pp. 148–158.
[22] A. Imteaj and M. Amini, "Fedparl: Client activity and resource-oriented lightweight federated learning model for resource-constrained heterogeneous iot environment," *Frontiers in Communications and Networks*, vol. 2, p. 657653, 2021.
[23] Z. Zhu, Y. Shi, J. Luo, F. Wang, C. Peng, P. Fan, and K. B. Letaief, "Fedlp: Layer-wise pruning mechanism for communication-computation efficient federated learning," in *ICC 2023-IEEE International Conference on Communications*. IEEE, 2023, pp. 1250–1255.
[24] Y. Jiang, S. Wang, V. Valls, B. Ko, W. Lee, K. Leung, and L. Tassiulas, "Model pruning enables efficient federated learning on edge devices," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.
[25] G. Kumar and D. Toshniwal, "Neuron specific pruning for communication efficient federated learning," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 4148–4152.
[26] T. Lin, S. Stich, K. Patel, and M. Jaggi, "Don't use large mini-batches, use local sgd," *arXiv preprint arXiv:1808.07217*, 2018.
[27] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 212–229, 2019.
[28] F. Zhou and G. Cong, "On the convergence properties of a k-step averaging stochastic gradient descent algorithm for nonconvex optimization," *arXiv preprint arXiv:1708.01012*, 2017.
[29] A. Al-Saedi, E. Casalicchio, and V. Boeva, "An energy-aware multi-criteria federated learning model for edge computing," in *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2021, pp. 134–143.
[30] L. Li, D. Shi, R. Hou, H. Li, M. Pan, and Z. Han, "To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.
[31] Z. Jiang, Y. Xu, H. Xu, Z. Wang, J. Liu, Q. Chen, and C. Qiao, "Computation and communication efficient federated learning with adaptive model pruning," *IEEE Transactions on Mobile Computing*, vol. 23, pp. 2003–2021, 2024.
[32] B. Isik, F. Pase, D. Gündüz, S. Koyejo, T. Weissman, and M. Zorzi, "Adaptive compression in federated learning via side information," in *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024*, vol. 238. PMLR, 2024, pp. 487–495. [Online]. Available: https://proceedings.mlr.press/v238/isik24a.html
[33] X. Huang, P. Li, and X. Li, "Stochastic controlled averaging for federated learning with communication compression," in *Proceedings of the International Conference on Learning Representations (ICLR) 2024*, 2024. [Online]. Available: https://openreview.net/forum?id=jj5ZjZsWJe
[34] Z. Oz, C. S. Oz, A. Malekjafarian, N. Afraz, and F. Golpayegani, "submfl: Compatible submodel generation for federated learning in device heterogenous environment," *arXiv preprint arXiv:2405.20014*, 2024. [Online]. Available: https://arxiv.org/abs/2405.20014
[35] J. Zheng, K. Li, E. Tovar, and M. Guizani, "Federated learning for energy-balanced client selection in mobile edge computing," in *2021 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 2021, pp. 1942–1947.
[36] M. K. Nori, S. Yun, and I. Kim, "Fast federated learning by balancing communication trade-offs," *IEEE Transactions on Communications*, vol. 69, pp. 5168–5182, 2021.
[37] P. Li, G. Cheng, X. Huang, J. Kang, R. Yu, Y. Wu, M. Pan, and D. Niyato, "Snowball: Energy efficient and accurate federated learning with coarse-to-fine compression over heterogeneous wireless edge devices," *IEEE Transactions on Wireless Communications*, vol. 22, no. 10, pp. 6778–6792, 2023.
[38] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[39] S. Wang, T. Tuor, T. Salonidis, K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, pp. 1205–1221, 2019.
[40] N. C. Kakwani, "Applications of lorenz curves in economic analysis," *Econometrica*, vol. 45, no. 3, pp. 719–727, 1977. [Online]. Available: http://www.jstor.org/stable/1911684
[41] J. Gorski, F. Pfeuffer, and K. Klamroth, "Biconvex sets and optimization with biconvex functions: a survey and extensions," *Mathematical methods of operations research*, vol. 66, no. 3, pp. 373–407, 2007.
[42] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, "Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications: Centralized and federated learning," 2022. [Online]. Available: https://dx.doi.org/10.21227/mbc1-1h68
[43] T.-M. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution on federated learning in non-convex settings," *arXiv preprint arXiv:1909.06335*, 2019.

Javad Dogani is currently a Postdoctoral Researcher with IMDEA Networks Institute, Madrid, Spain. His research interests include decentralized learning, LLMs, and edge AI.

Reza Namvar is a PhD Fellow at IMDEA Networks Institute and UC3M. His research interests include decentralized learning and AI-based solutions for solving mobile network problems

Masoumeh Khodarahmi is a Ph.D. student at IMDEA Networks Institute and UC3M. Her research interests include network constraints in AI applications and multimedia streaming.

Farshad Khunjush is currently a Full Professor at Shiraz University, Shiraz, Iran. His research interests include computer architectures, cloud computing, and big data.

Nikolaos Laoutaris is a Research Professor at IMDEA Networks Institute in Madrid and Director of its Data Transparency Group (DTG). His research interests include privacy, transparency, data protection, network economics, distributed systems, protocols, and network measurement.