

Article

FedAnchor: Anchored and Adaptive Federated Learning for Fault Diagnosis in Resource-Constrained Industrial IoT

Yanxin Hu ¹ , Xiaoman Liu ², Zhenzhen Xie ³, Junjie Pang ⁴ and Chao Cheng ^{2,*}¹ School of Mechanical and Electrical Engineering, Changchun University of Technology, Changchun 130012, China; 1202401018@stu.ccut.edu.cn² School of Computer Science and Engineering, Changchun University of Technology, Changchun 130012, China; 2202504116@stu.ccut.edu.cn³ School of Computer Science and Technology, Shandong University, Qingdao 266237, China⁴ School of Computer Science and Engineering, Qingdao University, Qingdao 266000, China

* Correspondence: chengchao@ccut.edu.cn

Abstract

Federated learning (FL) enables privacy-preserving fault diagnosis across distributed industrial devices, but most existing methods assume homogeneous model architectures and comparable client resources. This assumption is unrealistic in resource-constrained Industrial Internet of Things (IIoT) scenarios, where clients may have substantially different memory and computation capacities. To address this challenge, we propose FedAnchor, an anchored and adaptive FL framework for resource-heterogeneous fault diagnosis. FedAnchor decomposes each client submodel into a shared anchored core and a client-specific adaptive extension. The anchored core provides a common parameter subspace for consistent masked aggregation, while the adaptive extension is selected by a server-side reinforcement-guided policy under client memory budgets. This design couples resource-aware submodel allocation with structurally aligned aggregation. Experiments on four benchmark datasets and six heterogeneous memory configurations show that FedAnchor achieves competitive or superior accuracy compared with representative homogeneous and model-heterogeneous FL baselines. Under the evaluated non-IID settings, FedAnchor improves accuracy by up to **9.8 percentage points** over the strongest baseline, while maintaining favorable communication–accuracy trade-offs and empirical stability.

Keywords: federated learning; industrial internet of things; fault diagnosis; resource-aware machine learning; heterogeneous clients; deep reinforcement learning



Academic Editor: Zhuming Bi

Received: 16 June 2026

Revised: 8 July 2026

Accepted: 14 July 2026

Published: 16 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Rotating machinery, such as bearings, motors, and gearboxes, plays a critical role in modern industrial systems. Failures can cause costly downtime and safety risks, making accurate fault diagnosis essential. Deep learning (DL) has achieved strong diagnostic performance by learning discriminative patterns from vibration and acoustic signals [1–4]. However, accurate models usually require large and diverse labeled datasets, which are rarely available at a single site. In addition, privacy regulations (e.g., GDPR [5]) and network limitations often prevent raw industrial data from being shared across factories or organizations. Federated learning (FL) addresses this issue by allowing clients to train a shared model collaboratively while keeping raw data local [6–8].

A major challenge in federated fault diagnosis is client-side resource heterogeneity. Edge devices may differ substantially in memory capacity and computational power. Excluding weak clients can introduce participation bias, whereas uniformly shrinking the model underutilizes high-resource devices [9,10]. A more flexible solution is to let clients train resource-feasible submodels. However, this introduces structural mismatches: Different clients update different parameter subsets, and naive aggregation may average parameters with inconsistent update histories. Therefore, a resource-adaptive FL method must not only assign feasible submodels but also preserve a shared aggregation structure.

Knowledge-distillation-based methods [11–13] can align heterogeneous models, but they often assume a shared public dataset or require exchanging intermediate representations. Existing submodel-based methods usually rely on random dropout, fixed width scaling, or cyclic activation of model components [14–16]. These strategies reduce local resource usage, but they do not explicitly separate a shared aggregation subspace from a client-specific adaptive subspace. As a result, resource adaptation and aggregation consistency are often handled separately or implicitly.

To address these challenges, we propose **FedAnchor**, an anchored and adaptive federated learning framework for resource-heterogeneous fault diagnosis. FedAnchor decomposes each client submodel into a small anchored core shared by all clients and a client-specific adaptive extension. The anchored core provides a common parameter subspace for consistent masked aggregation, while the adaptive extension is selected by a server-side *deep reinforcement learning* policy according to client memory budgets and training feedback. This design couples resource-aware submodel allocation with structurally aligned aggregation in a unified framework.

As illustrated in Figure 1, conventional homogeneous FL relies on one full model for all clients, which is often unsuitable for resource-heterogeneous IIoT devices. FedAnchor addresses this mismatch by assigning different submodels according to client memory budgets while retaining a shared anchored core for aggregation alignment. The server-side selector determines the submodel structure, whereas clients only train their assigned submodels. This design highlights the central idea of FedAnchor: data-driven, resource-aware submodel coordination with a shared anchored subspace for more consistent aggregation under heterogeneous client resources. Different from random or fixed submodel pruning, the parameters used for cross-client aggregation are explicitly protected by the anchored core, whereas the remaining extension is adapted to client-specific budgets.

Our main contributions are summarized as follows:

- We formalize federated fault diagnosis under **multi-tiered memory constraints**, where clients may train structurally different submodels while raw industrial data remain local. This formulation bridges the gap between homogeneous FL assumptions and resource-heterogeneous IIoT fault diagnosis scenarios.
- We propose **FedAnchor**, an anchored and adaptive FL framework for structurally heterogeneous clients. Unlike homogeneous FedAvg-style training, dropout-based submodel training, or fixed nested submodels, FedAnchor decomposes each client submodel into a shared anchored core and a client-specific adaptive extension. The anchored core supports consistent masked aggregation, while the adaptive extension is selected by a server-side reinforcement-guided policy under client memory budgets. Section 4 further decomposes the roles of reinforcement-guided allocation, anchored alignment, and masked aggregation.
- We conduct experiments on four public fault diagnosis datasets, namely CWRU, JNU, XJTU, and MFPT, under six heterogeneous memory configurations and both IID and non-IID data partitions. The results show that FedAnchor achieves competitive or superior diagnostic accuracy compared with representative homogeneous and

model-heterogeneous FL baselines. Additional analyses of signal-level responses, feature-shift partitions, client selection policies, convergence behavior, and raw communication volume further examine the behavior and practical trade-offs of the proposed framework.

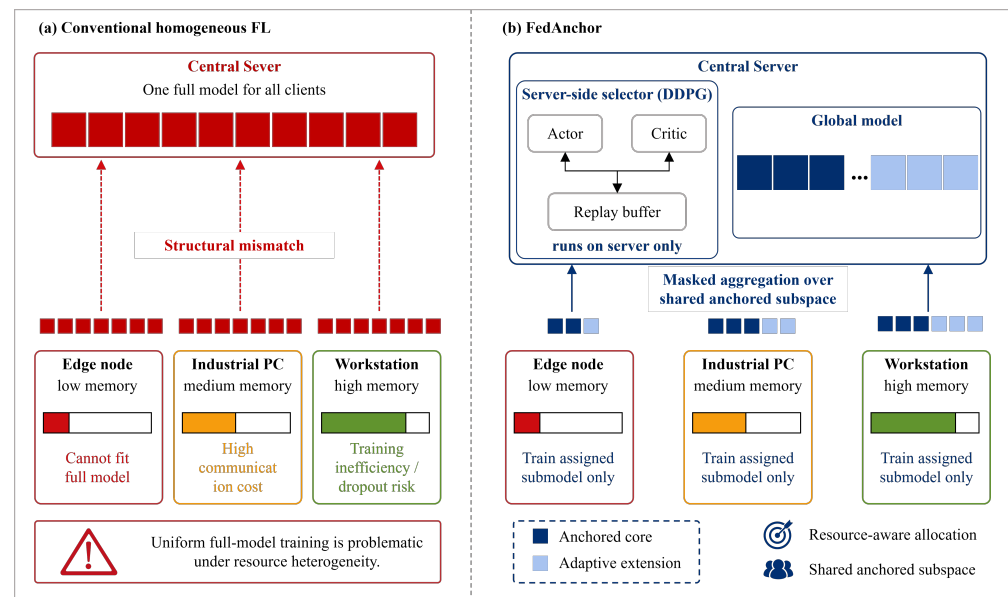


Figure 1. Motivation and overview of FedAnchor under resource heterogeneity. **(a)** Conventional homogeneous FL requires every client to train the same full global model, which may cause memory overflow, high communication cost, or participant dropout when client resources differ. **(b)** FedAnchor uses a server-side selector to assign resource-aware submodels while preserving a shared anchored core for masked aggregation over a common parameter subspace. Dark blue blocks denote the anchored core shared across clients, and light blue blocks denote client-specific adaptive extensions.

2. Related Work

In recent years, FL has emerged as a machine learning paradigm for collaborative fault diagnosis in industrial settings, particularly for rotating machinery, where privacy-preserving and efficient model training are critical. Although FL has demonstrated substantial potential, several challenges remain in representative applications, especially in handling client heterogeneity, diverse resource constraints, and the adaptability of learning frameworks to representative industrial conditions. This section reviews recent advances in federated learning for fault diagnosis, with a particular focus on model heterogeneity and resource-adaptive strategies.

2.1. Federated Learning for Fault Diagnosis

Federated learning has attracted increasing attention as a viable framework for intelligent fault diagnosis in rotating machinery, offering a privacy-preserving solution for collaborative model training across distributed industrial devices. By enabling sensors and edge nodes to jointly train a global model without transferring raw data, FL mitigates data isolation while complying with regulatory constraints such as GDPR [6]. Recent studies have extended FL-based fault diagnosis to dynamic operational environments through techniques such as adaptive validation [7] and cross-device domain generalization [17], which improve performance under changing working conditions.

In parallel, personalized federated learning (PFL) approaches have been proposed to accommodate client-specific characteristics. For example, Kolmogorov–Arnold-enhanced convolutional neural networks have been applied to personalized rotating machinery fault diagnosis [18]. To address limited and imbalanced data distributions across clients,

meta-learning [19] and transfer learning [20] have been integrated with FL to enhance generalization and client-level adaptation. Transfer-based FL frameworks, such as FTSDC [21], introduce dual-correction mechanisms for cross-machine fault scenarios, while knowledge distillation techniques facilitate heterogeneous learning across IoT systems [22]. Despite these advances, many existing FL-based fault diagnosis methods implicitly assume homogeneous model architectures and overlook practical hardware diversity, particularly in terms of memory and computational constraints. These limitations motivate the development of resource-adaptive federated learning frameworks, such as *FedAnchor*, which support personalized submodel construction and consistent aggregation under multi-tiered device constraints.

2.2. Federated Learning with Model Heterogeneity

Model heterogeneity across clients poses a fundamental challenge to federated learning, especially in industrial environments where devices exhibit substantial variation in memory capacity, computational power, and communication bandwidth. Traditional FL algorithms, which assume uniform model architectures, often struggle to scale effectively under such heterogeneous conditions. Federated dropout (FD) [14] addresses this issue by randomly pruning the global model to generate lightweight submodels for individual clients. However, the stochastic nature of FD can lead to less consistent training and insufficient training of shared parameters, particularly for resource-constrained devices.

To improve structural efficiency, several approaches explicitly design heterogeneous model architectures. Methods such as FjORD [23], HeteroFL [24], and Fluid [15] allocate fixed substructures according to predefined client resource profiles. While these approaches improve parameter utilization, they lack adaptability to dynamic changes in client capabilities and cannot adjust model structures as training progresses. More flexible strategies, including FedRolex [16], employ rolling submodel activation across communication rounds to enhance allocation diversity. Adaptive federated dropout [25] further introduces structure–performance tables to guide submodel reuse, albeit at the cost of increased convergence latency. Successive layer training (SLT) [26] incrementally expands a model’s depth, enabling progressive learning from shallow to deeper networks for constrained devices, but it remains limited in lateral scalability and sensitivity to task heterogeneity.

Recent hypernetwork-based approaches provide another direction for personalized and model-heterogeneous federated learning. Shamsian et al. proposed personalized Federated HyperNetworks (pFedHN), where a server-side hypernetwork generates personalized model parameters for different clients, enabling parameter sharing while maintaining client-specific models [27]. Litany et al. further studied federated learning with heterogeneous architectures using graph hypernetworks, where client architectures are represented and processed through a graph-based hypernetwork to support parameter generation across different model structures [28]. These methods are highly relevant to model heterogeneity because they allow client models to differ in architecture or parameterization. However, their system’s assumptions differ from the submodel-coordination setting considered in this work. Hypernetwork-based methods usually introduce an additional model generator on the server and aim to produce personalized client models, whereas *FedAnchor* maintains a single global diagnostic model and extracts resource-feasible submodels from a shared parameter space.

In addition, personalized federated learning has also been investigated in natural language processing scenarios. For example, Wang et al. studied personalized federated text classification with gradient-free prompt tuning, where personalization is achieved through prompt-level adaptation for language models [29]. Such NLP-oriented personalized FL methods broaden the application scope of federated learning, but they are not directly comparable to the present vibration-signal fault diagnosis setting because their data

modality, backbone model, task objective, and personalization mechanism are designed for text classification rather than 1D vibration-based industrial diagnosis. Therefore, these methods are discussed as relevant personalized or heterogeneous FL studies, while the experimental comparison in this work focuses on submodel-based baselines that can be directly adapted to resource-constrained fault diagnosis.

Recent Resource-Adaptive and Model-Heterogeneous FL Methods: Recent studies have further investigated resource-adaptive and model-heterogeneous FL under system-heterogeneous clients. NeFL introduces nested model scaling for clients with heterogeneous system resources and analyzes the inconsistency caused by training multiple submodel architectures [30]. FIARSE proposes Federated Importance-Aware Submodel Extraction, which dynamically adjusts client submodels according to parameter importance for model-heterogeneous FL with clients of different computational capacities [31]. These studies are closely related to the present work because they also address resource-aware model adaptation. However, their mechanisms differ from FedAnchor: NeFL follows the nested submodel-scaling family, while FIARSE focuses on importance-aware extraction. In contrast, FedAnchor explicitly couples a shared anchored aggregation subspace with a client-specific adaptive extension. This design aims to maintain aggregation consistency while still exploiting heterogeneous client memory budgets.

Although these recent methods are highly relevant, not all of them are directly comparable under the experimental protocol of this paper. Our direct baselines are selected to use the same 1D-CNN backbone, the same synchronous partial-participation FL protocol, the same train/test splits, and the same memory-tier settings. Under this criterion, the comparison already includes SLT, a recent resource-constrained submodel-training method, together with HeteroFL and FedRolex as representative nested and rolling submodel baselines. Recent methods such as NeFL and FIARSE are therefore discussed to clarify the latest research landscape and the relationship between FedAnchor and recent resource-adaptive FL, while the direct experiments are kept under a unified implementation protocol for fair comparison. The key characteristics of these methods are summarized in Table 1.

Table 1. Qualitative positioning of FedAnchor against representative heterogeneous and personalized federated learning methods.

Method	Main Mechanism	Supports Resource Budgets	Additional Generator or Task-Specific Module	Role in This Work
FedAvg-Small	Smallest homogeneous feasible model	Yes	No	Direct baseline
FederatedDropout	Random structured dropout	Yes	No	Direct baseline
HeteroFL	Nested width-scaled submodels	Yes	No	Direct baseline
FedRolex	Rolling submodel activation	Yes	No	Direct baseline
SLT	Progressive layer-wise training	Yes	No	Direct baseline
pFedHN [27]	Hypernetwork-generated personalized models	Partially	Server-side hypernetwork	Discussed
Graph hypernetwork FL [28]	Architecture-aware parameter generation	Partially	Graph hypernetwork	Discussed
NeFL [30]	Nested model scaling for system-heterogeneous clients	Yes	No generator	Discussed
FIARSE [31]	Importance-aware submodel extraction	Yes	No generator	Discussed
Gradient-free prompt-tuning PFL [29]	Prompt-level personalization for text classification	Task-dependent	NLP prompt module	Discussed
FedAnchor	Anchored shared core + adaptive extension	Yes	No public data or generator	Proposed

Other approaches, such as selective knowledge distillation [22] and cross-domain collaborative updates [17], provide partial flexibility in heterogeneous settings but often rely on additional data assumptions or implicit architectural compatibility. Overall, existing methods either lack fine-grained adaptivity, incur additional communication or coordination overhead, or fail to explicitly promote aggregation consistency under structural diversity. In contrast, *FedAnchor* addresses these limitations by combining server-side adaptive submodel allocation with an anchored aggregation mechanism, enabling resource-aware federated learning under multi-tiered resource heterogeneity.

3. Problem Statement

Notation: To facilitate understanding of the proposed framework, the main symbols used throughout this paper are summarized in Table 2. These symbols are formally introduced in subsequent sections but are listed here for clarity and quick reference.

Table 2. List of main symbols used in this work.

Symbol	Definition
K	Number of clients in the federated network
$\mathcal{D}_k$	Private dataset on client k
ρ_k	Memory constraint of client k
$\mathcal{G}$	Global diagnostic model
$\mathcal{H}_k$	Customized submodel for client k
α_k	Anchoring ratio shared across clients, based on $\min_k \rho_k$
N_ℓ	Number of neurons in layer ℓ
$\text{Acc}(\cdot)$	Aggregated client-reported feedback accuracy used for policy learning
a_t	Action vector generated by the policy agent
s_t	System state at communication round t
r_t	Reward signal at round t
$\mu(\cdot)$	Actor policy network
$Q(\cdot, \cdot)$	Critic network
$\mathcal{N}_t$	Exploration noise
γ	Discount factor in DRL reward computation
θ_t	Parameter vector of the global model at round t
$\mathbf{m}_{k,t}$	Binary mask defining the executable submodel of client k
$\mathbf{m}^A$	Shared anchored mask retained by all clients
$\mathbf{m}_{k,t}^D$	Client-specific adaptive mask outside the anchored subset
$\Delta\theta_{k,t}$	Masked local parameter update uploaded by client k
$Z_{j,t}$	Active weighted denominator for parameter-wise masked aggregation
$\mathcal{B}$	Replay buffer for experience storage
B	Mini-batch size for policy update
θ^μ, θ^Q	Parameters of actor and critic networks
τ	Target network update coefficient
d_t	Done flag in experience tuple, typically false
T	Total number of training rounds

System Setup: Modern industrial systems, such as wind farms and smart factories, deploy distributed sensors and edge devices to monitor rotating machinery in resource-constrained Industrial IoT environments. Such systems typically consist of a heterogeneous mix of low-power edge nodes and high-performance servers, forming a federated and resource-diverse network. Each device collects local vibration or acoustic signals, while raw data sharing across sites is restricted due to privacy regulations and communication limitations.

We consider a federated learning system comprising K clients indexed by $k \in \{1, \dots, K\}$. Each client k owns a local dataset $\mathcal{D}_k = \{(x_i^{(k)}, y_i^{(k)})\}_{i=1}^{n_k}$, where $x_i^{(k)}$ and $y_i^{(k)}$ denote the input signal and corresponding fault label, respectively. Client k is subject to

a normalized memory budget $\rho_k \in [\rho_{\min}, \rho_{\max}]$, which specifies the maximum scalable training–memory proportion of the global model that can be allocated to the client-side submodel. We use ρ_k as a resource-capacity constraint rather than a raw parameter-count constraint. During local training, the peak memory consumption of a submodel includes not only model weights but also gradients, optimizer states, intermediate activations, and the local minibatch buffer. Therefore, the memory budget is interpreted through an estimated training–memory model rather than by parameter count alone. To reflect representative IIoT resource conditions, we consider multi-tiered resource heterogeneity, e.g., $\rho_k \in \{0.125, 0.25, 0.5, 0.75, 1.0\}$, which captures a spectrum of devices ranging from embedded edge nodes to GPU-accelerated servers.

For a submodel $\mathcal{H}_k$ trained with minibatch size b , its estimated peak training memory can be written as

$$\hat{M}_{\text{peak}}(\mathcal{H}_k; b) = M_w(\mathcal{H}_k) + M_g(\mathcal{H}_k) + M_{\text{opt}}(\mathcal{H}_k) + M_{\text{act}}(\mathcal{H}_k; b) + M_{\text{batch}}(b) + M_{\text{misc}}, \quad (1)$$

where M_w , M_g , and M_{opt} denote the memory occupied by model weights, gradients, and optimizer states, respectively. M_{act} denotes the memory required to store intermediate activations for back-propagation, M_{batch} is the input minibatch buffer, and M_{misc} denotes framework-dependent temporary buffers. The memory components considered in this work are summarized in Table 3.

Table 3. Training–memory components considered in the memory-budget model.

Component	Symbol	Description
Model weights	M_w	Trainable parameters stored on the client
Gradients	M_g	Gradient tensors required for back-propagation
Optimizer states	M_{opt}	Momentum buffer for SGD with momentum
Activations	M_{act}	Intermediate feature maps retained for back-propagation
Minibatch buffer	M_{batch}	Input windows and labels in the local minibatch
Temporary buffers	M_{misc}	Framework-dependent workspace and temporary tensors

Let $P(\mathcal{H}_k)$ be the number of trainable parameters in $\mathcal{H}_k$, and let q be the number of bytes per floating-point value. In our implementation of FP32 with training and SGD with momentum, the parameter-related memory is approximated as

$$M_w(\mathcal{H}_k) + M_g(\mathcal{H}_k) + M_{\text{opt}}(\mathcal{H}_k) \approx q(1 + 1 + \eta_{\text{opt}})P(\mathcal{H}_k), \quad (2)$$

where $\eta_{\text{opt}} = 1$ for SGD with momentum because one additional momentum buffer is maintained. The activation memory is estimated from the feature-map sizes of the selected submodel:

$$M_{\text{act}}(\mathcal{H}_k; b) \approx qb \sum_{\ell \in \mathcal{L}} A_{\ell}(\mathcal{H}_k), \quad (3)$$

where $A_{\ell}(\mathcal{H}_k)$ denotes the number of activation elements retained at layer ℓ of submodel $\mathcal{H}_k$. For the 1D-CNN used in this work, $A_{\ell}(\mathcal{H}_k)$ is determined by the temporal length and channel width of the corresponding layer. The local minibatch buffer $M_{\text{batch}}(b)$ is fixed across methods because all experiments use the same input window length and minibatch size. Accordingly, the client memory constraint is enforced on the scalable part of the training memory:

$$\frac{\hat{M}_{\text{sc}}(\mathcal{H}_k; b)}{\hat{M}_{\text{sc}}(\mathcal{G}; b)} \leq \rho_k, \quad \hat{M}_{\text{sc}}(\mathcal{H}_k; b) = M_w(\mathcal{H}_k) + M_g(\mathcal{H}_k) + M_{\text{opt}}(\mathcal{H}_k) + M_{\text{act}}(\mathcal{H}_k; b). \quad (4)$$

The fixed minibatch buffer and framework-dependent buffers are included when discussing actual device memory, but they are not used to differentiate methods because they are kept identical across all compared approaches. Thus, ρ_k should be understood as a normalized scalable training-memory budget that controls the size of the client-side trainable submodel.

The global objective is to collaboratively train a fault diagnosis model $\mathcal{G}$ under strict privacy constraints such that raw data never leaves local clients. Due to heterogeneous resource budgets, each client trains a personalized submodel $\mathcal{H}_k \subset \mathcal{G}$ that satisfies its memory constraint ρ_k . The federated system must therefore allow updates from structurally heterogeneous submodels to be aggregated in a consistent manner while maintaining diagnostic performance across clients.

Practical Challenges: This setup introduces three key challenges.

First, enforcing a uniform global model across clients with diverse memory and computational capacities is impractical. Naively truncating layers or filters to fit resource-limited devices may discard diagnostically critical components, such as fault-induced impulses or high-frequency modulations, which are relevant for fault localization. Such indiscriminate pruning can reduce diagnostic performance in practical industrial scenarios.

Second, heterogeneous clients train structurally different submodels, which leads to *structural misalignment* during model aggregation. In this case, parameters with the same index across different submodels may correspond to different update histories or functional roles. Directly averaging such heterogeneous updates can result in invalid aggregation, less consistent training, and degraded global performance, particularly under non-IID data distributions.

Third, many existing approaches rely on static pruning strategies or heuristic dropout mechanisms that fail to adapt to evolving training dynamics. These methods cannot fully exploit device-specific capabilities or respond effectively to changing optimization states, limiting their applicability in dynamic and resource-constrained industrial environments.

Therefore, an effective federated learning framework must jointly support adaptive submodel construction under client-specific resource constraints and promote aggregation consistency and reduce empirical training variability across structurally heterogeneous models.

Learning Objective: The federated optimization objective is formulated as

$$\min_{\mathcal{G}} \sum_{k=1}^K \frac{|\mathcal{D}_k|}{|\mathcal{D}|} \mathbb{E}_{(x,y) \sim \mathcal{D}_k} [\mathcal{L}(\mathcal{G}(x), y)], \quad (5)$$

where $\mathcal{L}(\cdot)$ denotes a supervised loss function (e.g., cross-entropy), and $|\mathcal{D}| = \sum_{k=1}^K |\mathcal{D}_k|$ is the total data size. This objective captures the weighted empirical risk across clients, subject to two constraints: (i) Each client k trains a submodel $\mathcal{H}_k$ that satisfies its memory budget ρ_k , and (ii) data privacy is preserved by exchanging only model parameters rather than raw data. The global model $\mathcal{G}$ must therefore learn effectively from heterogeneous submodels under these constraints.

Constrained Resource-Heterogeneous FL formulation: To make the resource-aware learning problem explicit, let $\theta_t \in \mathbb{R}^P$ denote the parameter vector of the global model at communication round t . For client k , the executable submodel is represented by a binary mask $\mathbf{m}_{k,t} \in \{0, 1\}^P$, and the corresponding local trainable parameter vector is

$$\theta_{k,t} = \mathbf{m}_{k,t} \odot \theta_t. \quad (6)$$

The resource-heterogeneous federated diagnosis problem can then be written as the following mask-constrained empirical-risk minimization problem:

$$\min_{\theta, \{\mathbf{m}_{k,t}\}} \sum_{k=1}^K \frac{|\mathcal{D}_k|}{|\mathcal{D}|} \mathbb{E}_{(x,y) \sim \mathcal{D}_k} [\mathcal{L}(f(x; \mathbf{m}_{k,t} \odot \theta), y)], \quad (7)$$

subject to

$$\frac{M_{sc}(\mathbf{m}_{k,t} \odot \theta_t; b)}{M_{sc}(\theta_t; b)} \leq \rho_k, \quad \forall k, t, \quad (8)$$

$$\mathbf{m}^A \preceq \mathbf{m}_{k,t}, \quad \forall k, t, \quad (9)$$

where $\mathbf{m}^A$ denotes the anchored mask shared by all clients, and $\preceq$ indicates element-wise inclusion. Equation (8) enforces client-specific scalable-memory feasibility, while Equation (9) ensures that every client submodel contains the same anchored parameter subset for aggregation alignment. This formulation makes explicit that FedAnchor jointly addresses resource feasibility, submodel heterogeneity, and anchored aggregation consistency.

4. Methodology

FedAnchor connects the problem formulation in Section 3 with a practical learning pipeline for resource-heterogeneous federated fault diagnosis. It contains three interacting components: a server-side reinforcement-guided submodel selector, an anchored alignment mechanism, and a masked aggregation scheme. The selector allocates adaptive capacity within each client's feasible memory budget. The anchoring mechanism defines a fixed shared parameter subset that every client submodel must retain. Masked aggregation then aggregates valid parameters from heterogeneous submodels while applying FedAvg-like updates on the anchored subspace, as illustrated in Figure 2.

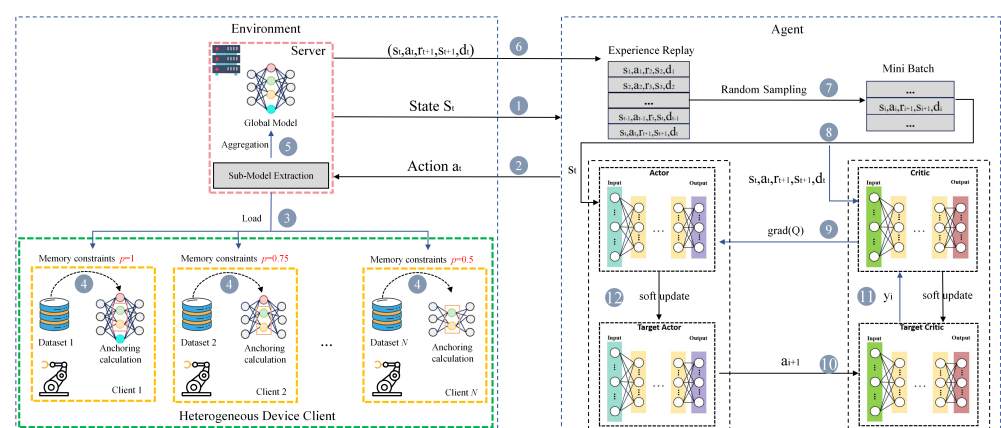


Figure 2. Overall architecture of FedAnchor. The pipeline includes federated state observation, action generation by the server-side actor network, resource-feasible submodel allocation, local training, masked aggregation, replay-buffer storage, and actor–critic policy updates.

Summary of the Proposed Method: The proposed method is FedAnchor, an anchored and adaptive federated learning framework for resource-heterogeneous fault diagnosis. It consists of three core designs. First, a resource-feasible submodel construction mechanism maps each client's memory budget to an executable submodel. Second, a server-side reinforcement-guided selector learns where to place the client-specific adaptive extension within the feasible submodel space. Third, an anchored core is deterministically retained by all clients, and masked aggregation updates this shared subspace consistently across heterogeneous submodels. Therefore, FedAnchor is not a new optimizer or a replacement

for the FL communication protocol. Instead, it is a resource-aware submodel coordination strategy that extends FL to clients with heterogeneous memory budgets.

Unified Mathematical Formulation of FedAnchor: FedAnchor constructs each client mask as a union of two disjoint components:

$$\mathbf{m}_{k,t} = \mathbf{m}^A \vee \mathbf{m}_{k,t}^D, \quad (10)$$

where $\mathbf{m}^A$ is the deterministic anchored mask shared by all clients, $\mathbf{m}_{k,t}^D$ is the client-specific adaptive mask selected under memory budget ρ_k , and $\vee$ denotes element-wise logical OR. The two masks satisfy

$$\mathbf{m}^A \odot \mathbf{m}_{k,t}^D = \mathbf{0}, \quad (11)$$

which prevents overlap between anchored and adaptive parameters. The adaptive mask is generated by a server-side policy and a feasibility projection:

$$\mathbf{m}_{k,t}^D = \Pi_{\rho_k}(\Omega(\theta_t, \mu_\phi(s_t))), \quad (12)$$

where μ_ϕ is the actor policy, s_t is the server-side state, $\Omega(\cdot)$ maps the policy action to layer-wise adaptive candidate indices, and $\Pi_{\rho_k}(\cdot)$ enforces the client-specific memory constraint. The final submodel sent to client k is therefore

$$\mathcal{H}_{k,t} = f(\cdot; \mathbf{m}_{k,t} \odot \theta_t). \quad (13)$$

This unified formulation links the resource-feasible projection, reinforcement-guided adaptive selection, anchored core preservation, and masked aggregation in a single mask-based mathematical model. The main differences from representative resource-heterogeneous federated learning approaches are summarized in Table 4.

Table 4. Comparison between FedAnchor and typical federated learning approaches under resource-heterogeneous settings.

Approach	Main Assumption or Mechanism	Limitation Under Resource Heterogeneity	FedAnchor Improvement
FedAvg/FedAvg-Small	All clients train the same model; FedAvg-Small uses the smallest homogeneous feasible submodel.	Standard FedAvg cannot directly handle different memory budgets; FedAvg-Small satisfies weak clients but underutilizes high-resource clients. Dropped structures may vary across clients and rounds, making shared-parameter training less consistent.	FedAnchor allows different clients to train resource-feasible submodels while preserving a shared anchored aggregation subspace. FedAnchor explicitly protects a deterministic anchored core and uses adaptive extensions only outside the anchor boundary.
FederatedDropout	Generates lightweight submodels by structured or random dropout.	The submodel structure is mainly predefined and may not adapt to round-wise training feedback.	FedAnchor learns adaptive submodel placement through server-side policy learning while enforcing resource feasibility.
HeteroFL	Uses nested width-scaled submodels according to client capacity.	Rolling activation improves coverage but does not explicitly define a fixed shared diagnostic subspace for all clients. Layer-wise progression may limit lateral channel adaptivity and can be sensitive to task heterogeneity.	FedAnchor combines adaptive allocation with an always-retained anchored core for consistent masked aggregation. FedAnchor performs layer-wise adaptive channel selection while retaining shared anchors across layers.
FedRolex	Rolls active submodel components across communication rounds to improve parameter coverage.		Provides a measured trade-off among diagnostic accuracy, empirical stability, and raw communication volume under the evaluated settings.
SLT	Progressively trains model layers or depth under resource constraints.		
FedAnchor	Anchored core plus client-specific adaptive extension under memory budgets.	Introduces server-side policy-learning overhead.	

Module-Level Contribution Decomposition: To clarify the boundary between reinforcement-guided allocation and anchoring, Table 5 summarizes the role and dependency of each module. DRL is responsible for adaptive capacity allocation, while anchoring

defines the shared aggregation subspace. Their main interaction occurs in boundary-aware submodel construction, where the DRL-selected adaptive extension is placed outside the fixed anchored subset under the client-specific memory constraint. Masked aggregation then uses the resulting masks to aggregate heterogeneous submodel updates consistently.

Table 5. Module-level contribution decomposition and dependency analysis of FedAnchor.

Module	Dominant Mechanism	Main Role	Dependency and Interaction
Resource-budget modeling	Resource-constraint-dominated	Defines the feasible scalable memory budget and prevents infeasible submodel assignment.	Provides the hard constraint for DRL-guided allocation and anchored submodel construction.
State and reward design	DRL-dominated	Converts feedback accuracy, previous allocation, and training progress into a policy-learning signal.	Supplies the input and reward for actor–critic learning; it does not enforce structural alignment.
Adaptive submodel allocation	DRL-dominated	Learns layer-wise positions of client-specific adaptive neuron blocks.	Operates within each memory budget and outside the anchor boundary; it does not select the anchor subset.
Anchored subset construction	Anchoring-dominated	Preserves a fixed shared parameter subset across all clients.	Provides the common update subspace for consistent aggregation and constrains adaptive block placement.
Boundary-aware submodel construction	Coupled DRL–anchor mechanism	Combines the fixed anchored subset with the DRL-selected adaptive extension.	Enforces non-overlap between anchored and adaptive neurons and checks memory feasibility.
Masked aggregation	Anchoring-dominated with mask-based execution	Aggregates valid parameters from heterogeneous submodels and applies FedAvg-like updates on anchors.	Uses the constructed masks; anchors are updated by all selected clients, while adaptive parameters are updated only when active.
Policy update	DRL-dominated	Updates the server-side actor and critic using replayed round-wise experience.	Uses post-training feedback to improve future allocation without changing the anchor definition.

4.1. Reinforcement-Guided Submodel Extraction

Client devices in federated networks exhibit substantial heterogeneity in memory budgets ρ_k , where $k \in \{1, \dots, K\}$. To respect these constraints, each client trains a partial submodel $\mathcal{H}_k \subset \mathcal{G}$ of the global model $\mathcal{G}$ such that $\text{size}(\mathcal{H}_k) \leq \rho_k \cdot \text{size}(\mathcal{G})$. Because submodel structure selection affects not only immediate performance but also future aggregation behavior, we formulate submodel allocation as a sequential decision-making problem. Accordingly, we model this process as a Markov decision process (MDP) and adopt the Deep Deterministic Policy Gradient (DDPG) algorithm [32] to learn a continuous, resource-aware allocation policy. The DDPG agent is maintained and updated on the server side, where sufficient computational and memory resources are assumed to be available. Resource-constrained clients do not execute the actor–critic networks, maintain replay buffers, or perform policy updates; they only receive the selected submodel structure or mask for local training. Therefore, using DRL for allocation does not contradict the resource-constrained client setting, because the additional decision-making overhead is centralized at the server.

State Representation: The system state at communication round t is defined as

$$s_t = [\bar{A}_{t-1}^{\text{fb}}, a_{t-1}^{(1)}, a_{t-1}^{(2)}, a_{t-1}^{(3)}, t/T] \in \mathbb{R}^5, \quad (14)$$

where $\bar{A}_{t-1}^{\text{fb}}$ denotes the aggregated client-reported feedback accuracy from the previous communication round; $a_{t-1}^{(1)}$, $a_{t-1}^{(2)}$, and $a_{t-1}^{(3)}$ are the three components of the previous layer-wise action vector corresponding to Conv1, Conv2, and the fully connected layer, respectively; t/T represents normalized training progress. Therefore, the state dimension is five, consisting of one scalar feedback-performance indicator, three historical allocation indicators, and one training-progress indicator.

Feedback accuracy is collected without introducing an additional public validation dataset or exchanging raw samples, logits, or intermediate features. At the end of each communication round, each selected client reports only a scalar local feedback accuracy value computed during its local training/evaluation procedure. The server aggregates these scalar reports as

$$\bar{A}_{t-1}^{\text{fb}} = \sum_{k \in \mathcal{S}_{t-1}} \frac{|\mathcal{D}_k|}{\sum_{j \in \mathcal{S}_{t-1}} |\mathcal{D}_j|} A_{k,t-1}^{\text{fb}}, \quad (15)$$

where $\mathcal{S}_{t-1}$ is the set of clients selected in round $t-1$, $|\mathcal{D}_k|$ is the number of local training samples on client k , and $A_{k,t-1}^{\text{fb}}$ is the scalar feedback accuracy reported by client k . This feedback signal is used only for server-side policy learning and is not used as the final test metric. Final diagnostic performance is evaluated separately on the held-out test set described in Section 5. This state formulation captures current feedback performance, historical allocation context, and training progress, enabling temporally informed allocation decisions.

Action Space: The actor network generates a continuous action vector

$$a_t = \mu(s_t | \theta^\mu) + \mathcal{N}_t, \quad (16)$$

where $\mu(\cdot)$ denotes the deterministic policy parameterized by θ^μ , and $\mathcal{N}_t$ is Gaussian noise added for exploration during training.

Resource-Feasible Action Mapping: The actor output does not directly determine an unconstrained model size. Instead, it determines the layer-wise positions of adaptive neuron blocks within a client-specific feasible submodel space. For each selected client k , the action a_t is mapped to a submodel through a resource-feasible projection:

$$\mathcal{H}_{k,t} = \Pi_{\rho_k}(\Phi(\mathcal{G}_t, a_t)), \quad (17)$$

subject to

$$\frac{\hat{M}_{\text{sc}}(\mathcal{H}_{k,t}; b)}{\hat{M}_{\text{sc}}(\mathcal{G}_t; b)} \leq \rho_k, \quad (18)$$

where $\Phi(\cdot)$ denotes the action-guided submodel construction operator, and $\Pi_{\rho_k}(\cdot)$ denotes the feasibility projection under the memory budget of client k . Therefore, the resource constraint is enforced before local training begins. The policy learns which parts of the global model should be activated within the allowed budget rather than learning to arbitrarily increase or decrease the client-side model's size. It is important to note that the DRL policy controls only the adaptive extension of each submodel. The shared anchored subset is not selected by the policy; it is deterministically preserved to maintain a common aggregation subspace across clients.

Reward Function: The reward at round t is defined as

$$r_t = e^{\text{Acc}_t - 1} + (\text{Acc}_t - \text{Acc}_{t-1}), \quad (19)$$

where Acc_t and Acc_{t-1} denote the normalized aggregated client-reported feedback accuracies of the current and previous communication rounds, respectively. The first term encourages policies that maintain a high feedback-performance level, while the second term explicitly captures incremental improvement across rounds.

We do not include an additional memory or communication penalty in the reward because resource feasibility is enforced as a hard constraint during submodel construction. For each client k , the selected submodel must satisfy the scalable memory constraint defined in Equation (4) before local training is performed. Under a fixed memory-tier configuration,

the role of the policy is therefore to determine where the adaptive capacity is allocated within the feasible submodel rather than how much memory the client is allowed to use. Similarly, the communication volume of each client update is upper-bounded by the size of its feasible submodel and is evaluated separately in the communication-cost analysis. Adding a soft resource penalty to the reward would provide limited additional information under fixed memory budgets and could bias the policy toward overly small or under-trained submodels.

Therefore, FedAnchor implements resource awareness through constrained submodel feasibility and client-specific memory budgets, while the reward focuses on learning diagnostically effective allocations within those budgets. Since this feedback is computed locally and reported only as scalar values, it does not require a separate public validation dataset or any exchange of raw data, logits, or intermediate representations. The feedback signal is used only to guide the server-side allocation policy, whereas the final diagnostic accuracy is reported on the held-out test set. This composite reward design uses a learning signal that reflects both long-term performance levels and short-term optimization dynamics for temporally coupled submodel allocation decisions in federated training.

This reinforcement-guided module establishes the foundation for adaptive and resource-aware submodel selection. However, to promote consistent aggregation across heterogeneous submodels, additional structural constraints are required. We address this challenge through an anchor-based submodel construction, which is described next.

4.2. Anchored Submodel Construction

Although the DRL agent dynamically selects personalized neuron blocks for each client, unconstrained submodel extraction may remove parameters that are repeatedly updated across clients, which can lead to aggregation variability under structural heterogeneity. To address this issue, we introduce an *anchored submodel construction* strategy that explicitly preserves a small, shared parameter subset across all clients, thereby enforcing a consistent update subspace during federated aggregation.

Specifically, for each layer ℓ , we reserve a proportion $\alpha_k = \alpha \cdot \min(\rho_1, \dots, \rho_K)$ of neurons as *anchor parameters*, where ρ_k denotes the memory constraint of client k . Anchors are implemented as a fixed index set consisting of the first $\lfloor \alpha_k N_\ell \rfloor$ neurons in layer ℓ , where N_ℓ is the total number of neurons. This design serves as a deterministic coordination mechanism that preserves identical parameter subsets across heterogeneous clients, without introducing additional communication, data exchange, or feature-level matching. The anchoring mechanism is therefore not responsible for learning the client-specific allocation policy. Its role is to define a stable structural constraint that every policy-generated submodel must satisfy.

The anchoring ratio α is treated as a structural design parameter that controls the size of the shared update subspace. Rather than assuming semantic correspondence at the feature level, the anchors are intended to maintain a statistically consistent and repeatedly updated parameter subset across clients. In practice, we set $\alpha = 0.15$ as a default value, which provides a measured balance between aggregation consistency and personalized model capacity under diverse resource configurations. Importantly, this choice does not affect the generality of the framework and can be adjusted to accommodate different resource constraints or application requirements.

Signal-level interpretation of anchored parameters. It should be noted that the anchored neurons are not pre-assigned to manually defined physical frequency bands. In a 1D-CNN-based diagnostic model, convolutional channels act as learnable temporal filters, and their frequency responses emerge from data-driven training rather than from fixed signal-processing rules. Therefore, FedAnchor does not assume a one-to-one correspon-

dence between an anchored neuron and a specific bearing characteristic frequency, such as BPFO, BPFI, or BSF, or a manually defined spectral band.

Instead, the purpose of anchoring is to ensure that a small subset of learnable filters is retained and repeatedly updated by all participating clients during federated aggregation. This shared update exposure encourages the anchored channels to capture diagnostic patterns that are more stable across clients, such as fault-induced impulses, periodic modulation, and broadband or high-frequency vibration components that commonly appear in bearing fault signals. Meanwhile, the non-anchored adaptive channels can still model client-specific operating conditions, load variations, and local data distributions. The anchored subspace is therefore not intended to encode fixed physical features but to preserve a commonly updated diagnostic representation that stabilizes aggregation under structural heterogeneity without requiring explicit feature-level alignment.

Beyond the anchored parameters, the remaining neurons are selected adaptively by the DRL agent under explicit boundary constraints. For client k and layer ℓ , we first define the layer-wise neuron budget as

$$m_{k,\ell} = \lfloor \rho_k N_\ell \rfloor, \quad (20)$$

where N_ℓ is the number of neurons or channels in layer ℓ . The number of anchored neurons is defined as

$$c_\ell = \min(\lfloor \alpha_k N_\ell \rfloor, m_{k,\ell}), \quad (21)$$

and the number of dynamically selected neurons is

$$d_{k,\ell} = m_{k,\ell} - c_\ell. \quad (22)$$

The anchor index set is fixed as

$$\mathcal{A}_\ell = \{0, 1, \dots, c_\ell - 1\}. \quad (23)$$

If $d_{k,\ell} = 0$, no dynamic neuron is selected for this layer. Otherwise, the start index of the dynamic block is computed as

$$s_{k,\ell} = c_\ell + \left\lfloor \frac{a_t[\ell] + 1}{2} \cdot \max(0, N_\ell - c_\ell - d_{k,\ell}) \right\rfloor, \quad (24)$$

where $a_t[\ell] \in [-1, 1]$ is the layer-wise action generated by the policy network. The dynamic index set is then

$$\mathcal{B}_{k,\ell} = \{s_{k,\ell}, s_{k,\ell} + 1, \dots, s_{k,\ell} + d_{k,\ell} - 1\}. \quad (25)$$

This construction guarantees that $\mathcal{A}_\ell \cap \mathcal{B}_{k,\ell} = \emptyset$ because $s_{k,\ell} \geq c_\ell$, and it prevents out-of-bound selection because $s_{k,\ell} + d_{k,\ell} \leq N_\ell$. The final valid neuron set is

$$\mathcal{S}_{k,\ell} = \mathcal{A}_\ell \cup \mathcal{B}_{k,\ell}, \quad (26)$$

which satisfies

$$|\mathcal{S}_{k,\ell}| = c_\ell + d_{k,\ell} = m_{k,\ell} \leq \lfloor \rho_k N_\ell \rfloor. \quad (27)$$

Therefore, anchor preservation and adaptive selection are performed within the same client-specific layer-wise budget rather than adding dynamic neurons on top of the budget. After the layer-wise selection is completed, the resulting submodel is further checked by the scalable-memory feasibility condition in Equation (18).

Algorithm 1 summarizes the anchored submodel extraction procedure. By construction, every client submodel retains an identical anchored parameter subset while allowing flexible, resource-aware personalization through non-overlapping DRL-controlled dynamic

components. This structure provides the basis for masked aggregation under heterogeneous model architectures, as detailed next.

Algorithm 1 Anchored Submodel Extraction Guided by DRL Action with Boundary Handling

Require: Global model $\mathcal{G}$; memory constraint ρ_k ; action vector a_t ; anchoring ratio α

Ensure: Customized submodel $\mathcal{H}_k$

```

1: Compute anchoring ratio:  $\alpha_k \leftarrow \alpha \times \min(\rho_1, \dots, \rho_K)$ 
2: Clone model:  $\mathcal{H}_k \leftarrow \text{clone}(\mathcal{G})$ 
3: for each layer  $\ell \in \{\text{Conv1}, \text{Conv2}, \text{FC}\}$  do
4:    $N_\ell \leftarrow \text{get\_neurons}(\ell)$ 
5:    $m_{k,\ell} \leftarrow \lfloor \rho_k N_\ell \rfloor$ 
6:    $c_\ell \leftarrow \min(\lfloor \alpha_k N_\ell \rfloor, m_{k,\ell})$ 
7:    $d_{k,\ell} \leftarrow m_{k,\ell} - c_\ell$ 
8:    $\text{anchor\_idx} \leftarrow [0, c_\ell)$ 
9:   if  $d_{k,\ell} > 0$  then
10:     $s_{k,\ell} \leftarrow c_\ell + \left\lfloor \frac{a_t[\ell]+1}{2} \cdot \max(0, N_\ell - c_\ell - d_{k,\ell}) \right\rfloor$ 
11:     $\text{dynamic\_idx} \leftarrow [s_{k,\ell}, s_{k,\ell} + d_{k,\ell})$ 
12:   else
13:     $\text{dynamic\_idx} \leftarrow \emptyset$ 
14:   end if
15:    $\text{valid\_idx} \leftarrow \text{anchor\_idx} \cup \text{dynamic\_idx}$ 
16:   Prune layer  $\ell$  of  $\mathcal{H}_k$  according to  $\text{valid\_idx}$ 
17: end for
18: Check the scalable-memory feasibility condition in Equation (18)
19: return  $\mathcal{H}_k$ 

```

4.3. Masked Aggregation with Alignment

After local training, client k uploads a masked update

$$\Delta\theta_{k,t} = \mathbf{m}_{k,t} \odot (\theta_{k,t}^{\text{loc}} - \theta_t), \quad (28)$$

where $\theta_{k,t}^{\text{loc}}$ denotes the locally trained parameter vector, and inactive parameters are set to zero by $\mathbf{m}_{k,t}$. Because heterogeneous clients may activate different non-anchored parameters, aggregation should be normalized over the clients that actually update each parameter. For each parameter index j , the server computes the active weighted denominator

$$Z_{j,t} = \sum_{k \in \mathcal{S}_t} |\mathcal{D}_k| m_{k,t}^{(j)}. \quad (29)$$

The global update is then performed by parameter-wise normalized masked aggregation:

$$\theta_{t+1}^{(j)} = \begin{cases} \theta_t^{(j)} + \frac{\sum_{k \in \mathcal{S}_t} |\mathcal{D}_k| m_{k,t}^{(j)} \Delta\theta_{k,t}^{(j)}}{Z_{j,t}}, & Z_{j,t} > 0, \\ \theta_t^{(j)}, & Z_{j,t} = 0. \end{cases} \quad (30)$$

This normalization ensures that a non-anchored parameter is averaged only over clients that actually train and upload that parameter, while parameters inactive for all selected clients remain unchanged.

To analyze the aggregation behavior under structural heterogeneity, we interpret Equation (30) from the perspective of a shared anchored parameter subspace. Let $\mathcal{A}$ denote

the anchored parameter subset, and let $\neg\mathcal{A}$ denote the non-anchored subset. Each local update can be decomposed as

$$\Delta\theta_{k,t} = \Delta\theta_{k,t}^{(\mathcal{A})} \oplus \Delta\theta_{k,t}^{(\neg\mathcal{A})}. \quad (31)$$

The anchored subset is retained by every selected client, whereas the non-anchored adaptive subset may vary across clients and rounds.

Proposition 1. For any anchored parameter index j satisfying $m_A^{(j)} = 1$, the FedAnchor update is equivalent to FedAvg restricted to the anchored subspace.

Proof. Since the anchored mask is retained by every selected client, we have $m_{k,t}^{(j)} = 1$ for all $k \in \mathcal{S}_t$. Therefore,

$$Z_{j,t} = \sum_{k \in \mathcal{S}_t} |\mathcal{D}_k|. \quad (32)$$

Substituting Equation (32) into the parameter-wise masked aggregation rule gives

$$\theta_{t+1}^{(j)} = \theta_t^{(j)} + \sum_{k \in \mathcal{S}_t} \frac{|\mathcal{D}_k|}{\sum_{r \in \mathcal{S}_t} |\mathcal{D}_r|} \Delta\theta_{k,t}^{(j)}, \quad (33)$$

which is the standard FedAvg update applied to the anchored parameter j . $\square$

Consequently, the overall FedAnchor update can be interpreted as the combination of a FedAvg-like update on the shared anchored subspace and a mask-normalized update on the non-anchored adaptive subspace:

$$\theta_{t+1} = \theta_t + \underbrace{\sum_{k \in \mathcal{S}_t} \frac{|\mathcal{D}_k|}{\sum_{r \in \mathcal{S}_t} |\mathcal{D}_r|} \Delta\theta_{k,t}^{(\mathcal{A})}}_{\text{FedAvg-like anchor update}} + \underbrace{\mathcal{U}_{\text{mask}}\left(\{\Delta\theta_{k,t}^{(\neg\mathcal{A})}, \mathbf{m}_{k,t}^{(\neg\mathcal{A})}\}_{k \in \mathcal{S}_t}\right)}_{\text{parameter-wise normalized adaptive update}}, \quad (34)$$

where $\mathcal{U}_{\text{mask}}(\cdot)$ denotes the parameter-wise normalized update in Equation (30) restricted to the non-anchored parameters. This decomposition indicates that FedAnchor supports aggregation consistency and structural adaptability without requiring full model homogeneity or feature-level alignment. It also clarifies the interaction between modules: DRL affects which non-anchored adaptive parameters are activated, whereas anchoring guarantees that a FedAvg-like update is always performed on the shared anchored subspace.

4.4. Policy Optimization Procedure

The DDPG policy is optimized on the server side using replay-based off-policy learning. The server-side policy aims to maximize the expected discounted return

$$J(\phi) = \mathbb{E}_{\mu_\phi} \left[\sum_{t=1}^T \gamma^{t-1} r_t \right], \quad (35)$$

where ϕ denotes the actor parameters, and r_t is the feedback-based reward defined in Equation (19). At each communication round t , the server observes the state s_t and generates one global layer-wise action

$$a_t = \mu(s_t | \theta^\mu) + \mathcal{N}_t, \quad (36)$$

where $a_t \in \mathbb{R}^3$ corresponds to the adaptive selection decisions for Conv1, Conv2, and the fully connected layer, respectively. This action does not directly specify an independent model size for each client. Instead, the same action is combined with each selected

client's memory budget ρ_k and mapped into a client-specific feasible submodel through the boundary-aware construction in Algorithm 1. Thus, one policy action is generated per communication round, while the resulting submodels are client-specific because their feasible budgets differ.

After local training and masked aggregation, the server obtains the next state s_{t+1} and reward r_t . The transition tuple

$$e_t = (s_t, a_t, r_t, s_{t+1}, d_t) \quad (37)$$

is stored in the replay buffer $\mathcal{B}$, where d_t is a done flag. In the finite-horizon federated training process, $d_t = 1$ only at the last communication round, and $d_t = 0$ otherwise. When the replay buffer contains at least B transitions, the server uniformly samples a mini-batch

$$\{(s_i, a_i, r_i, s_{i+1}, d_i)\}_{i=1}^B \sim \mathcal{B}. \quad (38)$$

For each sampled transition, the target value is computed using the target actor μ' and target critic Q' :

$$y_i = r_i + \gamma(1 - d_i)Q'(s_{i+1}, \mu'(s_{i+1} | \theta^{\mu'}) | \theta^{Q'}). \quad (39)$$

The critic network is updated by minimizing the mean-squared temporal-difference error:

$$\mathcal{L}_Q(\theta^Q) = \frac{1}{B} \sum_{i=1}^B \left(Q(s_i, a_i | \theta^Q) - y_i \right)^2. \quad (40)$$

The actor network is updated by maximizing the critic-estimated value of the action generated by the current policy. Equivalently, the actor loss is written as

$$\mathcal{L}_\mu(\theta^\mu) = -\frac{1}{B} \sum_{i=1}^B Q(s_i, \mu(s_i | \theta^\mu) | \theta^Q). \quad (41)$$

The deterministic policy gradient used for updating the actor is

$$\nabla_{\theta^\mu} J \approx \frac{1}{B} \sum_{i=1}^B \nabla_a Q(s_i, a | \theta^Q) \big|_{a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s_i | \theta^\mu). \quad (42)$$

After the actor and critic updates, the target networks are updated by soft replacement:

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}, \quad \theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}. \quad (43)$$

In our implementation, the policy is updated once per communication round after the replay buffer contains at least one mini-batch of transitions. This update is performed only on the server; clients neither store replay buffers nor execute actor–critic optimization.

Having defined the policy update routines, the server-side selector, anchored submodel construction, masked aggregation, and policy update form the complete FedAnchor training pipeline.

Mathematical Model Clarification: The formulation above separates three levels of FedAnchor. At the optimization level, FedAnchor solves a resource-constrained federated empirical-risk minimization problem. At the submodel-construction level, each client mask is decomposed into a shared anchored mask and a resource-feasible adaptive mask. At the aggregation level, parameter-wise normalized masked aggregation ensures that anchored parameters receive FedAvg-like updates from all selected clients, while non-anchored parameters are updated only by clients that activate them. This formulation clarifies how adaptive allocation and anchored aggregation are coupled mathematically.

5. Experimental Evaluation

In this section, we evaluate the proposed FedAnchor framework under representative industrial conditions for fault diagnosis in rotating machinery. The experiments are designed as reproducible benchmark-based evaluations rather than full real-factory deployment tests. We use public fault diagnosis datasets to ensure controlled comparison, while resource-heterogeneous IIoT conditions are simulated through memory-tier constraints, partial participation, non-IID partitions, feature-shift partitions, and client-selection policies. Our experiments assess the performance of FedAnchor in terms of diagnostic accuracy, training variability, and resource efficiency under these controlled resource-heterogeneous conditions. To directly reflect the three challenges in Section 3, we report diagnostic accuracy (capability), empirical training variability (aggregation behavior under structural misalignment), and communication/efficiency metrics (resource-aware learning under heterogeneous constraints). We compare FedAnchor with representative federated learning methods across several benchmark datasets and resource configurations to evaluate its effectiveness in resource-heterogeneous industrial settings.

5.1. Experimental Setup

To evaluate the proposed FedAnchor framework under representative resource-heterogeneous industrial diagnosis settings, we design a series of federated diagnostic experiments on rotating machinery. These experiments simulate heterogeneous resource environments and non-IID data partitions to assess diagnostic accuracy, variability, and efficiency under representative resource constraints.

Compared Methods: We compare FedAnchor with five representative federated learning baselines that cover homogeneous aggregation, dropout-based submodel training, nested model-heterogeneous FL, rolling submodel activation, and progressive layer-wise training. **FedAvg-Small** is a resource-feasible adaptation of FedAvg [33]. Since standard FedAvg assumes that all clients train the same model architecture, it cannot be directly executed when clients have different memory budgets. For each heterogeneous memory configuration, all clients are assigned the largest homogeneous submodel that can be loaded by the weakest client:

$$\rho_{\min}^{\text{cfg}} = \min_k \rho_k.$$

The global model used by FedAvg-Small is therefore a uniformly scaled submodel for which its scalable training-memory ratio does not exceed $\rho_{\min}^{\text{cfg}}$. During training, every selected client downloads, trains, and uploads this same smallest feasible model, and the server performs standard FedAvg aggregation over this homogeneous parameter space. Thus, FedAvg-Small is not a model-heterogeneous baseline; rather, it serves as a conservative homogeneous FL reference that satisfies all client resource budgets without excluding weak clients. Its limitation is that it underutilizes higher-resource clients, which motivates resource-adaptive heterogeneous methods such as FedAnchor. **FederatedDropout** [14] reduces memory usage via structured neuron dropout while retaining the global model format. **HeteroFL** [24] assigns nested submodels according to client capacity and is included as a directly relevant model-heterogeneous FL baseline. **FedRolex** [16] activates different submodel components cyclically to maximize parameter reuse and diversity. **SLT** [26] gradually increases model depth layer-by-layer, enabling execution on devices with dynamic or limited capacity. These baselines cover different types of references: FedAvg-Small represents a homogeneous resource-feasible lower-bound baseline, while FederatedDropout, HeteroFL, FedRolex, and SLT represent submodel-based or model-heterogeneous FL baselines that can operate under different client capacities. All methods are re-implemented in PyTorch 2.1.0 [34] and adapted to the same submodel training interface for fair comparison.

Rationale for Comparison: The selected baselines represent typical FL strategies under resource heterogeneity. FedAvg-Small represents a homogeneous resource-feasible reference, FederatedDropout represents dropout-based submodel training, HeteroFL represents nested width-scaled heterogeneous FL, FedRolex represents rolling submodel activation, and SLT represents progressive layer-wise training. Compared with these approaches, FedAnchor jointly considers resource-feasible adaptive allocation and aggregation consistency. The adaptive extension exploits client-specific memory budgets, while the anchored core preserves a shared update subspace for masked aggregation. Therefore, the comparison evaluates whether this coupled design improves diagnostic accuracy, empirical stability, and communication–accuracy trade-off over typical homogeneous and model-heterogeneous FL strategies.

Scope of Experimental Baselines: The direct experimental baselines are selected according to two criteria. First, the method should support submodel-level training under different client resource budgets. Second, it should be implementable using the same 1D-CNN backbone, synchronous FL protocol, client partitions, memory-tier settings, and train/test splits used in our fault-diagnosis experiments. Under these criteria, FedAvg-Small, FederatedDropout, HeteroFL, FedRolex, and SLT are used as direct baselines. Recent methods such as NeFL [30] and FIARSE [31] are discussed in the Related Work Section because they represent the latest progress in resource-adaptive and model-heterogeneous FL. They are not added as additional experimental baselines in this revision to keep the direct comparison under the same implemented protocol and to avoid changing the backbone architecture, submodel interface, or system setting.

Datasets and Diagnostic Task Construction: We adopt four publicly available rotating machinery fault diagnosis datasets: CWRU, JNU, XJTU, and MFPT. Before federated partitioning, each dataset is first converted into a supervised multi-class diagnostic task with a fixed label definition and a held-out test split. A diagnostic class is defined according to the fault category, operating condition, or their combination, depending on the annotation format of each dataset. All raw signals are segmented into fixed-length 1024-sample windows, and each window inherits the diagnostic label of its source signal record. For each dataset, 70% of the segmented windows are used for federated training, and 30% are held out for final testing. The train/test split is performed after window segmentation and before client partitioning. Therefore, the held-out test set is never used for local client training, server-side policy feedback, federated aggregation, or client partition construction. Only the training windows are distributed to clients under the IID, Dirichlet label-skew, or feature-shift settings. This protocol ensures that all compared methods use the same diagnostic task definition, the same held-out test set, and the same client-level training partitions.

Table 6 summarizes how the diagnostic task is constructed for each dataset. The CWRU dataset [35] provides bearing fault signals under controlled motor load and speed conditions and is constructed as a 10-class diagnosis task. The JNU dataset [36] contains bearing signals collected under varying speed and load conditions and is constructed as a 12-class diagnosis task based on working-condition and fault-type combinations. The XJTU dataset [37] provides run-to-failure bearing degradation signals under multiple operating conditions and is constructed as a 15-class diagnosis task. The MFPT dataset [38] contains bearing fault data from laboratory and field-related conditions and is also constructed as a 15-class diagnosis task. Although these datasets differ in acquisition platforms and operating regimes, all of them are converted into supervised multi-class fault diagnosis tasks using the same segmentation, train/test split, and federated partition protocol. Table 7 summarizes the dataset statistics and federated diagnostic task construction.

Table 6. Diagnostic task construction for each dataset before federated partitioning.

Dataset	Signal Source	Diagnostic Labels	Class Construction	Federated Task Setup
CWRU	Bearing vibration signals under controlled load/speed conditions	10 classes	Classes are constructed from the normal condition and selected bearing fault categories or severity levels. Each segmented window inherits the label of its source record.	1024-sample windows are generated first. A 70/30 train/test split is applied before distributing training windows to 20 clients.
JNU	Bearing vibration signals under variable speed/load conditions	12 classes	Classes are defined by working-condition and fault-type combinations according to the dataset annotations. Each window is labeled by the corresponding condition–fault combination.	The same 1024-sample segmentation and 70/30 split are used. Only training windows are partitioned among clients.
XJTU	Run-to-failure bearing degradation signals under multiple operating conditions	15 classes	Classes are constructed from the selected bearing degradation records or condition-specific fault/degradation labels used in the supervised diagnosis task.	Segmented windows are split into training and held-out test sets before federated IID/non-IID partitioning.
MFPT	Bearing fault signals from laboratory and field-related conditions	15 classes	Classes are formed from the selected normal and fault-condition records according to the dataset annotations.	The same preprocessing, train/test split, and client-partition protocol are applied for fair comparison.

Table 7. Dataset statistics and federated diagnostic task construction. The diagnostic labels are defined before federated partitioning. The numbers of windows are counted after 1024-sample segmentation, and the train/test split is applied before distributing training windows to clients.

Dataset	# Classes	# Total Windows	# Train Windows	# Test Windows	Operating Conditions	# Clients	Partition Settings
CWRU	10	4500	3150	1350	Controlled load/speed	20	IID / Dirichlet $\beta = 0.3$ /Feature-shift
JNU	12	3600	2520	1080	Variable speed/load	20	IID / Dirichlet $\beta = 0.3$ /Feature-shift
XJTU	15	4500	3150	1350	Run-to-failure conditions	20	IID / Dirichlet $\beta = 0.3$ /Feature-shift
MFPT	15	4500	3150	1350	Laboratory/field-related conditions	20	IID / Dirichlet $\beta = 0.3$ /Feature-shift

Benchmark Scope and Application Relevance: The four public datasets used in this work are representative rotating-machinery fault diagnosis benchmarks. They provide controlled vibration-signal labels and allow reproducible comparison across different federated learning methods. We acknowledge that these datasets are smaller than full industrial IIoT deployments and cannot fully capture all practical factors, such as continuous streaming data, intermittent connectivity, device failures, asynchronous availability, and factory-scale client populations. The purpose of using these benchmarks is therefore not to claim a complete real-factory deployment but to evaluate the diagnostic and aggregation behavior of different FL methods under controlled and repeatable fault-diagnosis tasks. To better reflect the computing-resource coordination scenario targeted by FedAnchor, we simulate heterogeneous client resources through six multi-tier memory configurations, deterministic client-to-tier assignments, partial client participation, IID and non-IID data partitions, feature-shift partitions, and client-selection policy variations. These settings allow us to examine whether FedAnchor can assign feasible submodels to clients with different resource budgets and maintain structurally consistent aggregation under representative resource-heterogeneous conditions.

Federated Data Partition Protocol: For each dataset, the segmented training windows are distributed to $K = 20$ clients. Under the IID setting, samples from each diagnostic class are uniformly and stratifiedly assigned to clients so that each client has approximately the same class distribution as the global training set. Under the non-IID setting, we use label-skew partitioning based on a Dirichlet distribution with concentration parameter $\beta = 0.3$. To make the non-IID split reproducible, the random seeds for the three independent runs are fixed as $\{2024, 2025, 2026\}$. For each seed, the non-IID split is generated by a class-wise Dirichlet allocation algorithm. For each diagnostic class c , all training windows belonging to class c are first shuffled using the specified seed. The client allocation proportions are then sampled as

$$\mathbf{p}_c \sim \text{Dirichlet}(\beta \mathbf{1}_K), \quad (44)$$

and samples of class c are assigned to clients according to $\mathbf{p}_c$. After all classes are allocated, the client sample counts are checked to avoid degenerate clients. Specifically, each client is required to contain at least $n_{\min} = 30$ training windows; otherwise, the partition is regenerated using the same seed sequence until the minimum-sample condition is satisfied. A smaller β indicates stronger label skew across clients. For each dataset, memory configuration, and seed, the generated client partitions are reused by all compared methods to ensure fair comparison. The detailed Dirichlet allocation algorithm, feature-shift partition protocol, and client-level distribution diagnostics are provided in Appendix A.1. This keeps the main experimental setup concise while preserving full reproducibility details.

Resource Constraint Settings: To emulate resource heterogeneity across clients, we assign each client k a normalized memory budget $\rho_k \in (0, 1]$, indicating the maximum scalable training-memory ratio of the client submodel relative to the full global model $\mathcal{G}$, as defined in Equation (4). Six heterogeneous memory configurations are explored. Four are three-tiered configurations: $[0.125, 0.25, 1]$, $[0.125, 0.5, 0.75]$, $[0.125, 0.5, 1]$, and $[0.125, 0.75, 1]$. Two are four-tiered configurations: $[0.125, 0.25, 0.5, 1]$ and $[0.125, 0.5, 0.75, 1]$. These cases reflect representative resource scenarios ranging from low-computing-power edge devices to industrial server nodes. Each submodel is constructed by selecting layer-wise channels or neurons under the client-specific budget ρ_k .

For reproducibility, the client-to-tier assignment is deterministic and fixed for each memory configuration, as summarized in Table 8. For three-tier configurations, the 20 clients are divided as evenly as possible into groups of seven, seven, and six clients. For four-tier configurations, clients are evenly divided into four groups of five clients. The same client identities, local data partitions, local data sizes, and random seeds are reused for all compared methods under the same dataset, memory configuration, and IID/non-IID setting. Thus, differences among methods are attributable to the learning and aggregation strategies rather than to different client assignments or data splits.

Table 8. Client assignment for each heterogeneous memory configuration.

Memory Configuration	Tier 1 Clients	Tier 2 Clients	Tier 3 Clients	Tier 4 Clients
$[0.125, 0.25, 1]$	$\rho = 0.125$: clients 0–6	$\rho = 0.25$: clients 7–13	$\rho = 1$: clients 14–19	–
$[0.125, 0.5, 0.75]$	$\rho = 0.125$: clients 0–6	$\rho = 0.5$: clients 7–13	$\rho = 0.75$: clients 14–19	–
$[0.125, 0.5, 1]$	$\rho = 0.125$: clients 0–6	$\rho = 0.5$: clients 7–13	$\rho = 1$: clients 14–19	–
$[0.125, 0.75, 1]$	$\rho = 0.125$: clients 0–6	$\rho = 0.75$: clients 7–13	$\rho = 1$: clients 14–19	–
$[0.125, 0.25, 0.5, 1]$	$\rho = 0.125$: clients 0–4	$\rho = 0.25$: clients 5–9	$\rho = 0.5$: clients 10–14	$\rho = 1$: clients 15–19
$[0.125, 0.5, 0.75, 1]$	$\rho = 0.125$: clients 0–4	$\rho = 0.5$: clients 5–9	$\rho = 0.75$: clients 10–14	$\rho = 1$: clients 15–19

Fairness and Seed Control: For each dataset and each IID/non-IID setting, the client data partition is generated once per random seed and then shared by all compared methods. The client IDs are also kept fixed when assigning memory tiers, so a client with a given

local dataset keeps the same memory budget within a memory configuration across all methods. The same random seeds are used for data partitioning, client sampling, model initialization, and local training schedules whenever applicable. The client sampling sequence is generated once per seed and reused by all compared methods. This protocol ensures that methods are evaluated under identical client identities, local sample sizes, class distributions, resource budgets, and participation sequences.

Memory-Budget Implementation: In the experiments, each memory tier ρ_k is implemented by constraining the scalable training memory of the client submodel rather than by parameter count alone. Since all methods use the same input window length, minibatch size, optimizer, and local training schedule, the input minibatch buffer and framework-level temporary buffers are shared across methods and do not affect relative comparisons. For each candidate submodel, the layer-wise channel or neuron selection determines both the number of trainable parameters and the number of activation elements, which are then used to estimate the scalable training–memory ratio. A submodel is considered feasible for client k only if its estimated scalable training–memory ratio does not exceed ρ_k . This design allows the memory tiers to approximate practical resource differences among edge devices while maintaining a reproducible and method-independent constraint protocol. It should be noted that the DDPG reward is not designed to minimize communication volume beyond the predefined client memory budgets. Instead, memory and communication efficiency are controlled by the client-specific feasible submodel constraints, while raw communication volume is reported as an independent evaluation metric.

Implementation Details: All experiments are run on an Intel Xeon E5-2678 v3 CPU and an NVIDIA RTX 3090 GPU. We simulate $K = 20$ clients, with $M = 5$ clients selected per communication round unless otherwise specified. **Client Selection Protocol:** In the main experiments, each communication round selects $M = 5$ out of $K = 20$ clients uniformly at random without replacement. This setting follows the common partial-participation protocol in federated learning and simulates practical IIoT scenarios where only a subset of edge devices may be available in each round due to communication, energy, or scheduling constraints. To ensure fairness, the random client-sampling sequence is generated once per seed and reused by all compared methods, so different methods are evaluated under the same participation sequence. In resource-heterogeneous settings, random selection may induce round-wise variation in the distribution of selected memory tiers. We therefore further evaluate alternative client selection policies in Appendix A.4. The main experiments retain random selection because it is method-independent, does not rely on privileged resource scheduling, and provides a neutral baseline for comparing learning and aggregation strategies under partial participation. The base diagnostic model is a compact 1D-CNN with two convolutional blocks and one fully connected head and equipped with ReLU activations and BatchNorm. Concretely, the feature extractor consists of $\text{Conv1d}(k = 64, \text{stride} = 16, \text{padding} = 24) \rightarrow \text{BN} \rightarrow \text{ReLU} \rightarrow \text{MaxPool1d}(k = 2, \text{stride} = 2)$, followed by $\text{Conv1d}(k = 3, \text{stride} = 1, \text{padding} = 1) \rightarrow \text{BN} \rightarrow \text{ReLU} \rightarrow \text{MaxPool1d}(k = 2, \text{stride} = 2)$. Channel widths are scaled by the client-specific subset factor sf : the first and second conv layers output $16 \times sf$ and $32 \times sf$ channels, respectively. Given an input window of length 1024, the temporal dimension evolves as $1024 \xrightarrow{\text{stride } 16} 64 \xrightarrow{\text{pool}} 32 \xrightarrow{\text{conv}} 32 \xrightarrow{\text{pool}} 16$, so the classifier receives $32 \times sf \times 16$ features after flattening. The classifier head uses $\text{FC}(100 \times sf) \rightarrow \text{ReLU} \rightarrow \text{FC}(\#classes)$. Input sequences are segmented into fixed-length 1024-sample windows; we apply per-client mean–std normalization on the training split (statistics frozen at test time) and light time-domain augmentations (Gaussian perturbation and random scaling) during training only. Non-IID settings are introduced via Dirichlet sampling with concentration parameter $\beta = 0.3$ to simulate skewed label distributions. A DDPG agent guides submodel structure by mapping global training context to pruning decisions, while anchor neurons

promote aggregation consistency across heterogeneous models. For server-side policy learning, the DDPG agent uses only scalar feedback accuracies reported by the selected clients after local training. No public validation dataset is introduced for policy feedback. These feedback values are used solely for policy optimization and are separated from the final test accuracy reported in the experimental results.

Evaluation Metrics: We report four evaluation metrics. **Accuracy** measures classification performance on unseen test data. **Training loss** reflects optimization behavior.

Raw communication volume records the total bidirectional traffic during training, including both server-to-client downloads and client-to-server uploads.

Communication and Runtime Accounting: To make the resource-efficiency analysis transparent, we report the raw communication volume, the number of transmitted parameters, and the wall-clock training time for each method. Raw communication volume is defined as the total bidirectional serialized traffic between the server and the selected clients, including both server-to-client downloads and client-to-server uploads. It is therefore not limited to the uploaded model update size. For method m at communication round t , let S_t denote the selected client set. Let $P_{m,k,t}^\uparrow$ and $P_{m,k,t}^\downarrow$ denote the numbers of trainable FP32 parameters actually uploaded by client k and downloaded from the server, respectively. For heterogeneous submodel methods, these quantities are computed from the actual assigned submodel of each selected client in each round, rather than from the full global model size. The cumulative transmitted parameter counts are

$$P_m^\uparrow = \sum_{t=1}^T \sum_{k \in S_t} P_{m,k,t}^\uparrow, \quad P_m^\downarrow = \sum_{t=1}^T \sum_{k \in S_t} P_{m,k,t}^\downarrow. \quad (45)$$

The corresponding raw communication volume is computed as

$$C_m = q(P_m^\uparrow + P_m^\downarrow) + C_m^{\text{mask/index}} + C_m^{\text{meta}}, \quad (46)$$

where $q = 4$ bytes for FP32 parameters, $C_m^{\text{mask/index}}$ denotes the serialized mask or active-index overhead, and C_m^{meta} denotes small metadata such as client IDs, layer descriptors, and submodel descriptors. For FedAnchor, the mask or active-index overhead required for masked aggregation is explicitly included in $C_m^{\text{mask/index}}$ and is not treated as free. For methods for which their submodel structure is deterministic and known to both the server and clients, the per-round mask/index overhead is set to zero except for negligible metadata. Unless otherwise stated, C_m is reported as the total bidirectional communication volume over all T communication rounds.

Training time is measured as the end-to-end wall-clock time from the start of federated training to the completion of the final aggregation round, including local submodel training, server aggregation, submodel construction, and server-side policy updates. All methods are evaluated under the same hardware, client participation sequence, local epochs, optimizer, and random seeds to ensure fair runtime comparison. Because the experiments are conducted in a single-machine simulation environment with sequential client execution, the reported training time is used for relative comparison under identical implementation conditions rather than as direct deployment latency on physically distributed devices.

Efficiency, as defined in Equation (47), is used as an auxiliary compound indicator that summarizes the trade-off among diagnostic accuracy, empirical variability, and raw communication volume. The primary evaluation still relies on the separately reported accuracy, standard deviation, training loss, raw communication volume, transmitted parameters, and training time. The logarithmic communication penalty is adopted as a practical normalization choice to reduce the dominance of large communication values; therefore,

this score is interpreted as a supplementary ranking indicator rather than a standalone theoretical metric:

$$\text{Efficiency}_m = \frac{\text{Accuracy}_m}{\sqrt{\text{Var}_m} \cdot \log\left(1 + \frac{C_m}{C_0}\right)}, \quad (47)$$

where C_m is the total bidirectional raw communication volume defined above, and $C_0 = 1$ MB is used only to make the logarithmic communication term dimensionless. This efficiency score is used as a supplementary compound indicator; the primary evaluation relies on the separately reported accuracy, empirical standard deviation, raw communication volume, transmitted parameters, and training time. Each experiment is repeated three times using the fixed random seeds $\{2024, 2025, 2026\}$. We report the mean and standard deviation to describe empirical variability across controlled runs. Because three repetitions are insufficient for making strong statistical claims about variance reduction, the reported standard deviation is interpreted as an empirical stability indicator rather than as evidence of statistically significant variance superiority.

Hyperparameter Configuration: Table 9 presents the detailed configuration used across all components, including model training, federated optimization, and policy learning. These settings remain fixed across datasets unless otherwise noted.

Table 9. Hyperparameter settings used in all experiments.

Model, Federated, and Client Settings		DDPG Policy Settings	
Parameter	Value	Parameter	Value
Optimizer	SGD with momentum	State dimension	5
Momentum coefficient	0.9	Action dimension	3
Weight decay	0.001	Action range	$[-1, 1]$
Initial/minimum learning rate	0.1/0.01	Exploration noise	0.1 Gaussian
Learning rate schedule	Cosine annealing	Actor hidden layers	$[256, 256]$
Loss function	Cross-entropy	Critic hidden layers	$[256, 256]$
Number of clients	20	Discount factor γ	0.9
Active clients per round	5	Target update coefficient τ	0.99
Communication rounds T	100	Replay buffer size	10^5
Local epochs per client	5	Batch size	32
Memory constraints ρ_k	$\{0.125, 0.25, 0.5, 0.75, 1.0\}$	Policy update frequency	1 update/round after $ \mathcal{B} \geq B$
IID partition	Uniform label distribution	Policy execution location	Server side only
Non-IID partition	Dirichlet $\beta = 0.3$	Client-side policy overhead	None
Random seeds	$\{2024, 2025, 2026\}$	–	–
Minimum samples/client	$n_{\min} = 30$	–	–
Anchoring ratio α_k	$0.15 \cdot \min_k(\rho_k)$	–	–

5.2. Comparative Evaluation Under Resource Heterogeneity

To evaluate the performance of FedAnchor under representative resource conditions, we compare its diagnostic accuracy with five federated learning baselines across six heterogeneous memory settings and four representative fault diagnosis datasets. Tables 10 and 11 report the results under IID and non-IID settings, respectively.

Table 10. IID setting: diagnostic accuracy of different federated learning methods under six multi-tiered memory constraint settings across four bearing fault diagnosis datasets. FedAvg-Small denotes the resource-feasible FedAvg variant in which all clients train the smallest homogeneous submodel allowed by the weakest memory tier.

Dataset	Method	[0.125, 0.5, 0.75]	[0.125, 0.5, 1]	[0.125, 0.75, 1]	[0.125, 0.25, 1]	[0.125, 0.25, 0.5, 1]	[0.125, 0.5, 0.75, 1]
CWRU	FedAvg-Small	0.8126 ± 0.0210	0.7558 ± 0.0268	0.7847 ± 0.0334	0.7891 ± 0.0336	0.7956 ± 0.0158	0.7822 ± 0.0346
	FedRolex	0.7842 ± 0.0698	0.7620 ± 0.1049	0.9134 ± 0.0131	0.7671 ± 0.1194	0.7361 ± 0.2085	0.8931 ± 0.0583
	HeteroFL	0.8091 ± 0.0121	<u>0.8447 ± 0.0071</u>	0.9047 ± 0.0071	<u>0.8634 ± 0.0575</u>	0.7980 ± 0.0318	<u>0.9064 ± 0.0126</u>
	FederatedDropout	0.6138 ± 0.1049	0.6642 ± 0.1823	0.7854 ± 0.1794	0.5923 ± 0.2308	0.4664 ± 0.2231	0.8375 ± 0.1122
	SLT	0.7775 ± 0.0463	0.8027 ± 0.0427	0.7973 ± 0.0410	0.8017 ± 0.0434	<u>0.8020 ± 0.0340</u>	0.7793 ± 0.0454
	FedAnchor	0.9002 ± 0.0213	0.9232 ± 0.0104	0.9375 ± 0.0055	0.9030 ± 0.0116	0.9141 ± 0.0034	0.9286 ± 0.0037
XJTU	FedAvg-Small	0.8076 ± 0.0283	0.7773 ± 0.0156	0.8136 ± 0.0296	0.8161 ± 0.0407	0.8198 ± 0.0295	0.7822 ± 0.0346
	FedRolex	0.8775 ± 0.0537	0.9292 ± 0.0201	<u>0.9543 ± 0.0258</u>	<u>0.8879 ± 0.0141</u>	0.8269 ± 0.1391	0.8931 ± 0.0583
	HeteroFL	<u>0.9040 ± 0.0552</u>	0.9032 ± 0.0256	0.9262 ± 0.0309	0.8543 ± 0.0730	<u>0.8995 ± 0.0219</u>	<u>0.9527 ± 0.0089</u>
	FederatedDropout	0.6398 ± 0.2403	<u>0.9526 ± 0.0204</u>	0.9190 ± 0.0847	0.8279 ± 0.1507	0.6281 ± 0.2700	0.8375 ± 0.1122
	SLT	0.8417 ± 0.0365	0.8141 ± 0.0131	0.8109 ± 0.0114	0.7970 ± 0.0240	0.8287 ± 0.0259	0.8025 ± 0.0401
	FedAnchor	0.9301 ± 0.0198	0.9649 ± 0.0070	0.9761 ± 0.0070	0.9442 ± 0.0065	0.9521 ± 0.0021	0.9775 ± 0.0077
JNU	FedAvg-Small	0.5127 ± 0.0416	0.5345 ± 0.0262	0.5472 ± 0.0383	0.5151 ± 0.0397	0.5329 ± 0.0273	0.5280 ± 0.0436
	FedRolex	0.5389 ± 0.0708	0.6273 ± 0.0878	0.6827 ± 0.0723	<u>0.5857 ± 0.0235</u>	0.5118 ± 0.0967	<u>0.6560 ± 0.0711</u>
	HeteroFL	<u>0.5535 ± 0.0143</u>	0.4028 ± 0.0765	0.6565 ± 0.0108	0.5618 ± 0.0054	<u>0.5873 ± 0.0362</u>	0.6516 ± 0.0129
	FederatedDropout	0.4176 ± 0.0837	<u>0.6324 ± 0.0363</u>	0.6780 ± 0.0377	0.5650 ± 0.0495	0.4894 ± 0.1422	0.6232 ± 0.0800
	SLT	0.5509 ± 0.0299	0.5458 ± 0.0317	0.5523 ± 0.0408	0.5484 ± 0.0272	0.5616 ± 0.0422	0.6205 ± 0.0517
	FedAnchor	0.6169 ± 0.0726	0.6741 ± 0.0164	<u>0.6801 ± 0.0234</u>	0.6042 ± 0.0140	0.6532 ± 0.0038	0.6984 ± 0.0034
MFPT	FedAvg-Small	0.6279 ± 0.0125	0.6583 ± 0.0130	0.6341 ± 0.0115	0.6039 ± 0.0337	0.6309 ± 0.0080	0.6331 ± 0.0154
	FedRolex	<u>0.6585 ± 0.0719</u>	<u>0.6872 ± 0.0632</u>	0.7262 ± 0.0335	0.7165 ± 0.0144	0.6301 ± 0.0829	<u>0.7102 ± 0.0564</u>
	HeteroFL	0.6429 ± 0.0398	0.6501 ± 0.0441	0.7587 ± 0.0096	0.6395 ± 0.0270	<u>0.6773 ± 0.0252</u>	0.6773 ± 0.0252
	FederatedDropout	0.5832 ± 0.0309	0.6857 ± 0.0351	<u>0.7857 ± 0.0474</u>	0.6647 ± 0.0375	0.5797 ± 0.1415	0.6827 ± 0.1342
	SLT	0.5978 ± 0.0347	0.5988 ± 0.0586	0.5955 ± 0.0530	0.6079 ± 0.0442	0.6146 ± 0.0478	0.6205 ± 0.0517
	FedAnchor	0.7644 ± 0.0136	0.7437 ± 0.0059	0.7968 ± 0.0174	<u>0.6839 ± 0.0065</u>	0.7284 ± 0.0211	0.7918 ± 0.0054

Bold values indicate the highest accuracy, and underlined values denote the second-best result in each setting.

Table 11. Non-IID setting: diagnostic accuracy of different federated learning methods under six multi-tiered memory constraint settings across four bearing fault diagnosis datasets. FedAvg-Small denotes the resource-feasible FedAvg variant in which all clients train the smallest homogeneous submodel allowed by the weakest memory tier.

Dataset	Method	[0.125, 0.5, 0.75]	[0.125, 0.5, 1]	[0.125, 0.75, 1]	[0.125, 0.25, 1]	[0.125, 0.25, 0.5, 1]	[0.125, 0.5, 0.75, 1]
CWRU	FedAvg-Small	0.5807 ± 0.1020	0.5185 ± 0.0842	0.6321 ± 0.0430	0.5452 ± 0.0924	0.5400 ± 0.1226	0.6324 ± 0.0829
	FedRolex	0.5013 ± 0.1091	0.6439 ± 0.1005	0.5610 ± 0.0421	0.5477 ± 0.2644	0.5230 ± 0.0950	0.6134 ± 0.0583
	HeteroFL	<u>0.7080 ± 0.0318</u>	<u>0.7234 ± 0.0575</u>	<u>0.7827 ± 0.0491</u>	<u>0.6444 ± 0.0704</u>	<u>0.6467 ± 0.0646</u>	<u>0.7343 ± 0.0753</u>
	FederatedDropout	0.3649 ± 0.0949	0.5170 ± 0.0696	0.6629 ± 0.0609	0.4482 ± 0.1350	0.3741 ± 0.1990	0.5664 ± 0.0929
	SLT	0.5800 ± 0.0491	0.5113 ± 0.1223	0.5032 ± 0.0528	0.5934 ± 0.0226	0.6156 ± 0.1402	0.6193 ± 0.1260
	FedAnchor	0.7511 ± 0.0365	0.7282 ± 0.1224	0.8022 ± 0.0516	0.7420 ± 0.0624	0.6879 ± 0.0924	0.7760 ± 0.0059
XJTU	FedAvg-Small	0.5543 ± 0.1006	0.5677 ± 0.0430	0.5378 ± 0.1761	0.5528 ± 0.1284	0.6035 ± 0.0335	0.6388 ± 0.0795
	FedRolex	0.4965 ± 0.1133	0.6968 ± 0.0983	0.7553 ± 0.0309	0.6148 ± 0.1800	0.3649 ± 0.0393	0.6992 ± 0.1334
	HeteroFL	<u>0.6694 ± 0.0537</u>	<u>0.8197 ± 0.0436</u>	<u>0.8818 ± 0.0100</u>	0.6294 ± 0.0792	<u>0.8060 ± 0.0420</u>	<u>0.8190 ± 0.0734</u>
	FederatedDropout	0.3445 ± 0.0513	0.4659 ± 0.1063	0.7454 ± 0.1000	0.4563 ± 0.0877	0.3726 ± 0.0977	0.6477 ± 0.1841
	SLT	0.6351 ± 0.0699	0.7035 ± 0.0528	0.5773 ± 0.1195	<u>0.6797 ± 0.0758</u>	0.6970 ± 0.0568	0.6010 ± 0.0767
	FedAnchor	0.7625 ± 0.0374	0.8351 ± 0.0549	0.9158 ± 0.0161	0.7750 ± 0.0591	0.8104 ± 0.0535	0.8827 ± 0.0364
JNU	FedAvg-Small	0.3688 ± 0.0497	0.3873 ± 0.0659	0.3882 ± 0.0425	<u>0.3572 ± 0.0496</u>	0.4222 ± 0.0730	0.4026 ± 0.0466
	FedRolex	0.3051 ± 0.0805	0.2623 ± 0.1817	0.3852 ± 0.1596	0.2958 ± 0.1478	0.2829 ± 0.1612	<u>0.4389 ± 0.0887</u>
	HeteroFL	<u>0.4028 ± 0.0765</u>	<u>0.4289 ± 0.0553</u>	0.4028 ± 0.0765	0.3391 ± 0.0403	<u>0.4382 ± 0.0388</u>	0.4099 ± 0.1001
	FederatedDropout	0.1896 ± 0.0692	0.2447 ± 0.1787	0.3785 ± 0.0828	0.2405 ± 0.1641	0.2812 ± 0.1581	0.4231 ± 0.0913
	SLT	0.2952 ± 0.0725	0.4042 ± 0.0692	<u>0.4637 ± 0.0799</u>	0.3218 ± 0.0705	0.4100 ± 0.0951	0.3417 ± 0.0790
	FedAnchor	0.4253 ± 0.0623	0.4727 ± 0.0509	<u>0.5053 ± 0.0369</u>	0.3753 ± 0.0969	0.4394 ± 0.0385	0.4720 ± 0.0552
MFPT	FedAvg-Small	0.4163 ± 0.0648	0.4230 ± 0.0738	0.4422 ± 0.0541	0.4524 ± 0.0298	0.4008 ± 0.0268	0.4575 ± 0.0575
	FedRolex	0.4775 ± 0.1039	0.5550 ± 0.0683	0.5896 ± 0.0820	0.5229 ± 0.0108	0.4481 ± 0.1294	0.6084 ± 0.0199
	HeteroFL	<u>0.5863 ± 0.0409</u>	<u>0.6089 ± 0.0048</u>	<u>0.6736 ± 0.0202</u>	<u>0.5614 ± 0.0166</u>	<u>0.5197 ± 0.0901</u>	<u>0.6420 ± 0.0216</u>
	FederatedDropout	0.2773 ± 0.0475	0.5469 ± 0.0625	0.5778 ± 0.1378	0.4316 ± 0.0768	0.3274 ± 0.1453	0.5726 ± 0.0182
	SLT	0.4568 ± 0.0204	0.4398 ± 0.0248	0.4439 ± 0.0419	0.4281 ± 0.0375	0.4378 ± 0.0115	0.4062 ± 0.0322
	FedAnchor	0.6013 ± 0.0236	0.6336 ± 0.0104	0.7005 ± 0.0112	0.5733 ± 0.0176	0.5822 ± 0.0117	0.6689 ± 0.0085

Bold values indicate the highest accuracy, and underlined values denote the second-best result in each setting.

5.2.1. Performance Under IID Settings

Under IID settings, FedAnchor obtains the highest accuracy in 22 out of 24 dataset-configuration pairs. For instance, it reaches **93.75%** on CWRU under the [0.125, 0.75, 1] setting and **97.75%** on XJTU under the [0.125, 0.5, 0.75, 1] setting. These results indicate that

the anchored submodel coordination strategy can support heterogeneous memory budgets when client label distributions are relatively balanced.

HeteroFL results show that nested model-heterogeneous training remains a relevant reference method, although its accuracy is generally lower than FedAnchor under the evaluated settings. HeteroFL is frequently the second-highest method, particularly on CWRU and XJTU, where it reaches **90.64%** and **95.27%** under the [0.125, 0.5, 0.75, 1] setting, respectively. On JNU, FedAnchor obtains the highest result in five out of six IID configurations, while FedRolex achieves the highest value under [0.125, 0.75, 1]. On MFPT, FedAnchor also obtains the highest accuracy in five configurations, while FederatedDropout achieves the highest value under [0.125, 0.75, 1].

These findings indicate that several baselines can be competitive in specific resource configurations, whereas FedAnchor reports a consistent accuracy–variability trade-offs across diverse datasets and structural regimes.

5.2.2. Performance Under Non-IID Data

Under Dirichlet-distributed label skew ($\beta = 0.3$), the performance gap between methods becomes more visible in several datasets. FedAnchor obtains the highest accuracy in all 24 non-IID configurations, achieving **91.58%** on XJTU and **77.60%** on CWRU in representative settings. Compared with the highest-performing baseline in each configuration, the maximum non-IID improvement is **9.8 percentage points** on CWRU under [0.125, 0.25, 1].

HeteroFL is the second-highest method in many non-IID configurations, especially on CWRU, XJTU, and MFPT, indicating that nested heterogeneous submodels remain effective under label skew. For example, HeteroFL reaches **78.27%** on CWRU and **88.18%** on XJTU under the [0.125, 0.75, 1] setting. However, FedAnchor remains higher across all evaluated non-IID settings. On JNU, all methods show accuracy reduction under Dirichlet label skew, while FedAnchor still obtains the highest results in all six configurations. These observations indicate that server-side adaptive allocation and structural anchoring can help reduce the combined effects of statistical and structural heterogeneity.

5.2.3. Performance Across Memory-Tier Configurations

As the resource tiers become more diverse, baseline methods may exhibit larger fluctuations due to structural mismatch or fixed allocation rules. FedAnchor remains comparatively consistent as memory tiers vary. On MFPT under [0.125, 0.25, 0.5, 1], it achieves **72.84%** (IID) and **58.22%** (non-IID), exceeding the highest-performing baseline by 5.1 and 6.3 percentage points, respectively.

In the four-tier setting [0.125, 0.5, 0.75, 1], FedAnchor obtains the highest results on all IID and non-IID datasets, while HeteroFL is the closest competitor in several CWRU, XJTU, and MFPT cases. This pattern indicates that FedAnchor reports a measured empirical trade-off across different resource partitions and data distributions.

5.3. Sensitivity Analysis of the Anchoring Ratio

The anchoring ratio controls the size of the shared parameter subset retained by all clients. To examine its influence, we vary α from 0 to 0.30 and evaluate FedAnchor on four datasets under both IID and non-IID partitions. The sensitivity analysis is conducted under two representative heterogeneous memory configurations: the four-tier setting [0.125, 0.5, 0.75, 1] and the three-tier setting [0.125, 0.5, 1]. Figure 3 reports the resulting diagnostic accuracy and standard deviation.

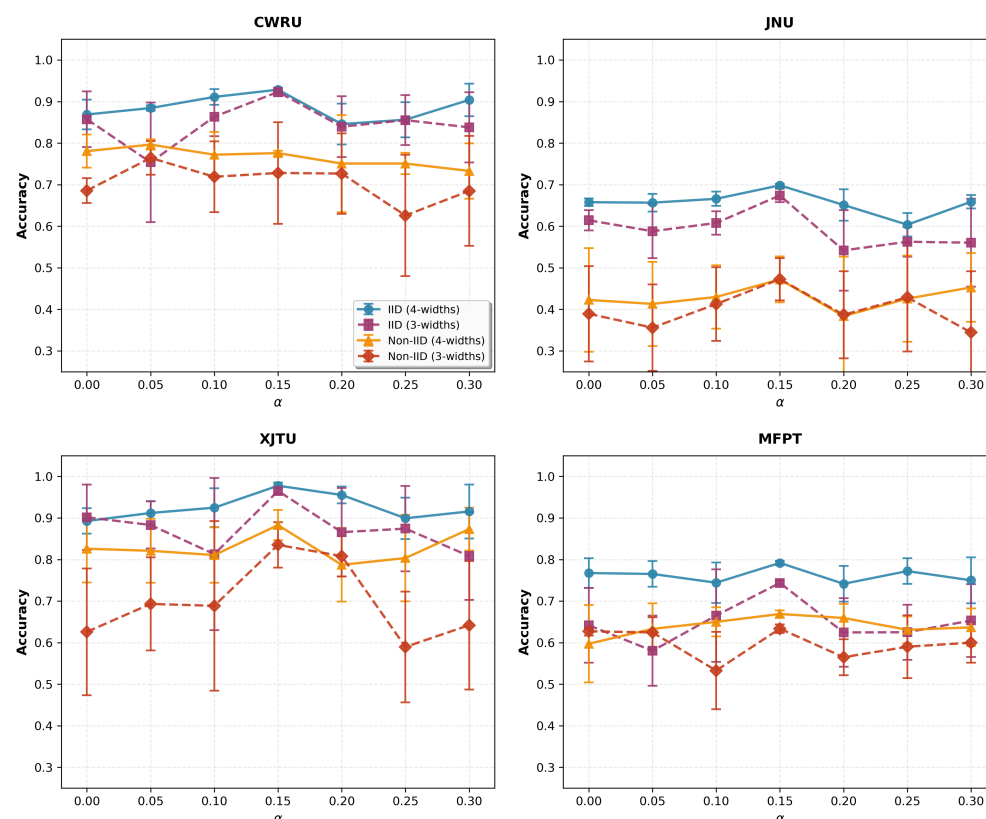


Figure 3. Sensitivity analysis of the anchoring ratio α across four datasets. The results are reported under IID and non-IID partitions using two representative heterogeneous memory configurations: $[0.125, 0.5, 0.75, 1]$ and $[0.125, 0.5, 1]$. Error bars denote empirical standard deviations over the three fixed-seed runs. A moderate anchoring ratio generally provides a better balance between shared aggregation consistency and adaptive submodel flexibility.

Aggregation protocol for sensitivity analysis. To clarify how the table-level sensitivity results are computed, we define the evaluated setting set as

$$\mathcal{C} = \mathcal{D} \times \mathcal{P} \times \mathcal{M}, \quad (48)$$

where $\mathcal{D} = \{\text{CWRU}, \text{JNU}, \text{XJTU}, \text{MFPT}\}$ denotes the four datasets, $\mathcal{P} = \{\text{IID}, \text{Non-IID}\}$ denotes the two data partitions, and $\mathcal{M} = \{[0.125, 0.5, 0.75, 1], [0.125, 0.5, 1]\}$ denotes the two representative memory configurations. Thus, $|\mathcal{C}| = 4 \times 2 \times 2 = 16$.

For each anchoring ratio α and each setting $c \in \mathcal{C}$, we first compute the mean accuracy over the three fixed random seeds $\mathcal{R} = \{2024, 2025, 2026\}$:

$$\bar{A}_{\alpha,c} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} A_{\alpha,c,r}, \quad (49)$$

where $A_{\alpha,c,r}$ denotes the test accuracy obtained with anchoring ratio α under setting c and seed r . The run-to-run empirical standard deviation for the same setting is computed as

$$S_{\alpha,c} = \sqrt{\frac{1}{|\mathcal{R}| - 1} \sum_{r \in \mathcal{R}} (A_{\alpha,c,r} - \bar{A}_{\alpha,c})^2}. \quad (50)$$

The reported average accuracy is then computed as an unweighted macro-average over the 16 evaluated settings:

$$\text{AvgAcc}_\alpha = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \bar{A}_{\alpha,c}. \quad (51)$$

Similarly, the reported average empirical standard deviation is computed as

$$\text{AvgStd}_\alpha = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} S_{\alpha,c}. \quad (52)$$

The IID and non-IID averages are computed by applying the same macro-average only to the corresponding subsets of $\mathcal{C}$. The best-performing count is computed using the seed-averaged accuracy $\bar{A}_{\alpha,c}$ for each setting rather than using individual runs:

$$N_{\text{best}}(\alpha) = \sum_{c \in \mathcal{C}} \mathbb{I} \left[\alpha = \arg \max_{\alpha'} \bar{A}_{\alpha',c} \right], \quad (53)$$

where $\mathbb{I}[\cdot]$ is the indicator function. We use an unweighted macro-average so that each dataset–partition–memory–configuration setting contributes equally to the default anchoring-ratio selection, preventing larger datasets or easier partitions from dominating the sensitivity summary.

Using this aggregation protocol, the table-level sensitivity results provide a quantitative basis for the default choice. Averaged over the 16 evaluated dataset–partition–memory–configuration settings, $\alpha = 0.15$ obtains an average accuracy of **76.07%**, compared with 69.73% for $\alpha = 0$ and 70.07% for the second-highest average setting ($\alpha = 0.10$). The same value also gives the highest averaged IID accuracy (83.78%) and non-IID accuracy (68.37%). In addition, based on the seed-averaged accuracy within each setting, $\alpha = 0.15$ obtains the highest accuracy in 14 out of the 16 evaluated settings, indicating that this setting is not determined by a single dataset, partition, or memory configuration.

Using the same macro-average protocol, the variability results show a similar empirical trend. The average empirical standard deviation at $\alpha = 0.15$ is **2.53%**, whereas the other tested values range from 6.13% to 8.04% in the evaluated three-seed runs. Compared with the no-anchor case ($\alpha = 0$), the observed average empirical standard deviation is lower by about 60.2%. This result is interpreted as an empirical stability indicator rather than a formal statistical variance claim. When $\alpha = 0$, the shared anchored subspace is removed, which weakens aggregation consistency under structurally heterogeneous submodels. Small values such as $\alpha = 0.05$ and $\alpha = 0.10$ preserve more adaptive capacity but provide less common structure for aggregation. In contrast, larger values such as $\alpha \geq 0.20$ may allocate more parameters to the shared anchor and reduce the flexibility of DRL-guided dynamic allocation.

Although the highest value may vary slightly across dataset–configuration pairs, these quantitative results indicate that $\alpha = 0.15$ provides an empirical trade-off across the evaluated settings. Therefore, we use $\alpha = 0.15$ as the default anchoring ratio in the main experiments while noting that application-specific tuning may further improve performance in particular scenarios.

5.4. Evaluation of Resource-Efficient Diagnostic Performance

To further analyze the performance–resource trade-off, we evaluate FedAnchor on the CWRU dataset using the multi-tiered memory setup $\rho_k \in \{0.125, 0.5, 0.75, 1.0\}$. This analysis considers diagnostic accuracy, empirical training variability, and communication

efficiency, which jointly characterize the behavior of resource-heterogeneous federated fault diagnosis.

The quantitative results show that FedAnchor obtains the highest diagnostic accuracy in this representative setting, reaching **92.86%**. It also reports the lowest empirical standard deviation across the three controlled training runs, with a standard deviation of 0.37%, compared with 5.83% for FedRolex and 11.22% for FederatedDropout. These results suggest improved empirical stability in this setting, although we do not claim statistical significance from three repetitions alone. The observed stability is associated with the structural anchoring mechanism, which preserves a shared update subspace across structurally diverse clients.

To provide a transparent resource-efficiency analysis, we report raw communication and runtime statistics in Table 12, and then, we use the resource-efficiency score defined in Equation (47) as a supplementary compound indicator. The table explicitly separates uploaded parameters, downloaded parameters, FP32 parameter traffic, and mask/index overhead. Therefore, the reported raw communication volume is not only the uploaded model size but the total bidirectional traffic accumulated over all communication rounds. Because the same accounting rule is applied to all datasets and memory configurations, Table 12 reports the representative CWRU setting used in the multi-dimensional trade-off analysis. The communication values for other configurations follow the same calculation procedure and scale with the actually assigned submodel sizes.

Table 12. Breakdown of bidirectional communication volume and runtime on CWRU under $\rho_k \in \{0.125, 0.5, 0.75, 1.0\}$. All values are averaged over the three fixed seeds {2024, 2025, 2026}. Communication is counted over all $T = 100$ rounds with 5 selected clients per round. Uploaded and downloaded parameter counts are computed from the actually transmitted submodels in each round. Total communication includes FP32 parameter traffic in both directions, mask/index overhead, and metadata. Training time is measured on a single NVIDIA RTX 3090 GPU with sequential client simulation.

Method	Uploaded Params (M)	Downloaded Params (M)	FP32 Param Traffic (MB)	Mask/Index Overhead (MB)	Total Comm. (MB)	Time/Round (s)	Total Time (min)
FedAvg-Small	250.0	250.0	1908.3	0	1908.3	15.0	25.0
FedRolex	31.2	31.2	239.4	0	239.4	15.8	26.2
HeteroFL	148.4	148.4	1133.5	0	1133.5	16.5	27.5
FederatedDropout	250.0	250.0	1908.3	59.60	1967.9	17.2	28.8
SLT	31.2	31.2	239.4	59.60	299.0	19.5	32.5
FedAnchor	31.2	31.2	239.4	59.60	299.0	19.5	32.5

The raw statistics show that FedAnchor transmits 31.2 M uploaded parameters and 31.2 M downloaded parameters in this representative CWRU setting. Its mask/index overhead is 59.60 MB and is explicitly included in the total communication volume. As a result, FedAnchor obtains a total communication volume of 299.0 MB over 100 rounds. Compared with FedAvg-Small and HeteroFL, FedAnchor reduces total communication by 84.3% and 73.6%, respectively. This saving is achieved by transmitting client-specific feasible submodels instead of full global models or larger heterogeneous submodels.

It should also be noted that FedAnchor does not report the lowest raw communication volume in this setting. FedRolex has a lower total communication volume because it does not require the same mask/index overhead in our accounting. However, FedAnchor provides a more favorable empirical trade-off when diagnostic accuracy and empirical training stability are considered together. Therefore, the communication analysis is interpreted jointly with accuracy and empirical variability, rather than as a standalone claim that FedAnchor minimizes raw traffic in all cases.

SLT and FedAnchor have the same transmitted-parameter count and mask/index overhead in this representative setting because both operate under the same feasible

submodel-size budget determined by the memory constraint. Their difference lies in the submodel allocation strategy and aggregation mechanism rather than in the nominal transmitted model size.

The reported training time includes local submodel training, aggregation, submodel construction, and server-side DDPG policy updates. FedAnchor requires 32.5 min for 100 rounds compared with 25.0 min for FedAvg-Small. This 30% increase reflects the additional adaptive submodel management and policy optimization overhead. However, FedAnchor does not claim the shortest raw training time. Its practical advantage is instead a communication-efficient and accuracy-stable trade-off: the additional server-side adaptive-allocation overhead remains manageable, while the communication volume remains substantially lower than full-model or larger-submodel transmission baselines. The absolute training time remains manageable in the simulated training environment and indicates that the additional adaptive-allocation overhead is not prohibitive under the evaluated setting.

Using the supplementary efficiency score in Equation (47), FedAnchor attains the highest score (39.04) among the compared methods in this representative setting. This indicates a measured empirical trade-off between diagnostic performance, empirical variability, and raw communication volume. Because the same quantitative information is already reported through the tables and text discussion, the analysis focuses on numerical comparisons rather than an additional redundant summary plot.

Summary of Improvements over Typical FL Approaches: The empirical results suggest three main improvements of FedAnchor over typical FL baselines in the evaluated resource-heterogeneous settings. First, compared with homogeneous resource-feasible training such as FedAvg-Small, FedAnchor better utilizes high-resource clients through adaptive extensions. Second, compared with dropout-based or rolling submodel methods, the anchored core provides a common update subspace and reduces structural inconsistency during masked aggregation. Third, compared with static heterogeneous submodel assignment, the server-side policy uses round-wise feedback to adapt the non-anchored extension within feasible memory budgets. These improvements lead to a measured trade-off among diagnostic accuracy, empirical variability, and raw communication volume. We note that FedAnchor does not claim the lowest raw runtime or the lowest communication volume in every case; rather, its advantage lies in the combined accuracy–stability–resource trade-off. Compared with recent resource-adaptive FL methods such as NeFL [30] and FIARSE [31], FedAnchor focuses on a complementary design point. Instead of relying only on nested scaling or importance-aware extraction, FedAnchor maintains a single global diagnostic model and coordinates resource-feasible submodels through an anchored shared subspace. This makes the proposed framework directly compatible with 1D-CNN-based vibration diagnosis and synchronous partial-participation FL while explicitly promoting consistent aggregation under structural heterogeneity.

Overall, this multi-perspective analysis indicates that FedAnchor reports a measured empirical trade-off among accuracy, empirical variability, and resource-efficiency indicators under the evaluated federated fault diagnosis settings.

5.5. Supplementary Robustness and Interpretability Analyses

To keep the main experimental section focused, several additional analyses are reported in the appendices. Appendix A.2 provides representative convergence curves and discusses early-stage behavior under server-side online policy learning. Appendix A.3 reports the additional Feature-shift non-IID robustness experiment. Appendix A.4 evaluates random, tier-balanced, and round-robin client selection policies. Appendix A.5 provides signal-level visualizations of anchored and non-anchored channels. Together, these supple-

mentary results support the robustness, stability, and interpretability discussions without interrupting the main comparison, resource-efficiency analysis, and ablation study.

5.6. Ablation Study

To assess the individual contributions of the core components in FedAnchor, we conduct ablation studies by selectively disabling or modifying key modules. The following variants are evaluated: (i) **FedAnchor-NoAnchor**, which removes the anchor-based aggregation mechanism; (ii) **FedAnchor-Random**, which replaces the DRL-guided submodel selector with uniform random allocation; (iii) **FedAnchor-AccOnly**, where the DRL reward depends solely on global accuracy; (iv) **FedAnchor-DeltaOnly**, where the reward is based only on round-to-round accuracy improvement. All experiments are conducted under a heterogeneous memory configuration $\rho_k \in \{0.125, 0.25, 0.5, 1.0\}$, with the results summarized in Figure 4.

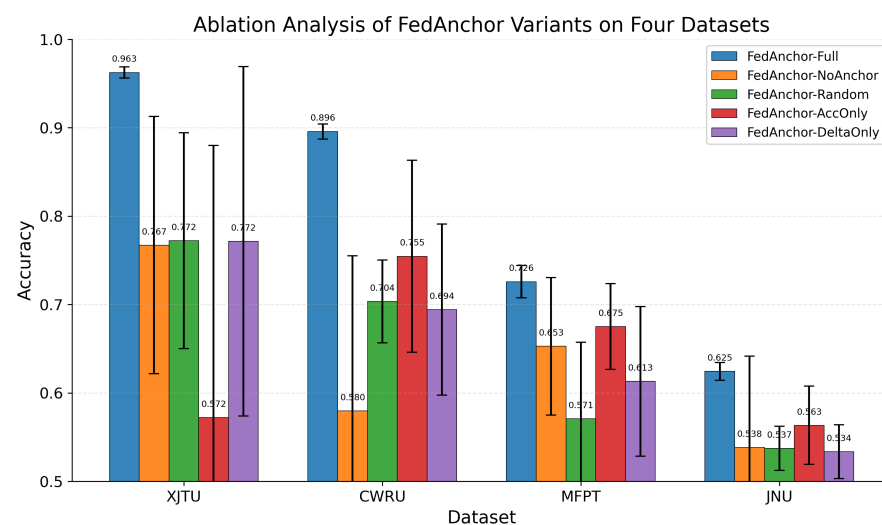


Figure 4. Ablation analysis of FedAnchor on four datasets under memory constraints $\rho_k \in \{0.125, 0.25, 0.5, 1.0\}$. **FedAnchor-Full** integrates both DRL-guided submodel selection and anchor-based aggregation. Removing either module (**NoAnchor**, **Random**) weakens the observed accuracy–variability profile. Using partial reward signals (**AccOnly**, **DeltaOnly**) weakens policy performance. The full method reports the highest accuracy with comparatively low empirical variability across the evaluated datasets.

The full version of FedAnchor obtains the highest diagnostic accuracy across all datasets. For example, on the XJTU and CWRU datasets, FedAnchor attains **96.3%** and **89.6%** accuracy, respectively, with standard deviations below 1%. Removing the anchor-based aggregation mechanism leads to a clear decrease in accuracy. **FedAnchor-NoAnchor** decreases to **76.7%** on XJTU and **58.0%** on CWRU, corresponding to decreases of **19.6** and **31.6 percentage points**, respectively. Meanwhile, the standard deviation increases to 14.6% and 17.5%, respectively. These results indicate that anchor-based aggregation contributes to reducing update inconsistency under structural heterogeneity.

The contribution of the DRL-based submodel selector is also observed. Replacing it with random submodel allocation (**FedAnchor-Random**) results in accuracy decreases of **19.2** and **15.5 percentage points** on CWRU and MFPT, respectively. Moreover, the corresponding standard deviation increases by more than 200% in several cases, indicating that uncontrolled variation in submodel width increases the variability of local updates.

The reward design also affects training dynamics. When global accuracy alone is used as the reward signal (**FedAnchor-AccOnly**), performance on XJTU decreases from 96.3% to **57.2%**, corresponding to a decrease of **39.1 percentage points**. Similar accuracy decreases

are observed on MFPT and JNU, with decreases of **24.9** and **14.2 percentage points**, respectively. Using only the incremental accuracy improvement signal (**FedAnchor-DeltaOnly**) gives higher accuracy than **FedAnchor-AccOnly** in several cases, but it still underperforms the full version by approximately **10–25 percentage points** across the evaluated datasets. These observations indicate that a composite reward, which jointly accounts for absolute performance and improvement dynamics, provides a more informative learning signal for long-horizon policy optimization.

In summary, the ablation results show that both anchor-based aggregation and the adaptive DRL policy contribute to the performance of FedAnchor. Anchored aggregation maintains a shared update subspace across heterogeneous submodels, thereby reducing update inconsistency, while the server-side DRL-guided submodel selector supports capacity-aware utilization of client resources. Removing either component reduces accuracy and increases variability, indicating their complementary roles in resource-heterogeneous federated optimization.

6. Limitations and Future Work

This study has several limitations. First, the experiments are conducted on public rotating-machinery fault diagnosis benchmarks. These datasets provide reproducible labeled vibration signals and enable controlled comparison across FL methods, but they are smaller than full industrial IIoT deployments. They cannot fully capture continuous streaming data, long-term equipment evolution, intermittent connectivity, device failures, asynchronous client availability, factory-scale client populations, or deployment-specific network latency. Therefore, the results should be interpreted as benchmark-based evidence for the proposed resource-aware submodel coordination strategy rather than as a complete validation of real factory-scale deployment. Second, the current implementation uses a compact 1D-CNN backbone and synchronized communication rounds. The behavior of the proposed anchored submodel coordination strategy with other architectures, asynchronous updates, or larger-scale client populations requires further evaluation. Third, although we additionally evaluate random, tier-balanced, and round-robin client selection policies, this study still assumes synchronous partial participation with a fixed number of selected clients per round. More complex availability-driven, network-aware, or latency-aware client scheduling policies are left for future work. Fourth, the DDPG-based selector is executed on the server side and does not introduce additional client-side policy-learning cost, and the added convergence analysis shows manageable overhead in the evaluated single-machine sequential simulation. However, real distributed deployments may exhibit different latency patterns due to network bandwidth, device parallelism, straggler effects, and asynchronous availability. Finally, although each experiment is repeated with three fixed random seeds to report mean and standard deviation, this number of repetitions is limited for strong statistical claims about variance reduction. Therefore, the reported standard deviations are interpreted as empirical stability indicators. Future work will evaluate FedAnchor on larger private industrial datasets and real heterogeneous edge platforms, examine asynchronous and large-client settings, include more independent runs with formal statistical testing, and investigate lighter allocation policies or theoretical bounds for anchored submodel aggregation.

7. Conclusions

This paper proposes **FedAnchor**, a federated learning framework for fault diagnosis in resource-heterogeneous industrial environments. By integrating a server-side DRL-guided submodel extraction policy with an anchor-based aggregation mechanism, FedAnchor en-

ables adaptive, capacity-aware training while promoting structurally consistent aggregation across heterogeneous clients.

Experiments on four benchmark datasets show that FedAnchor obtains higher diagnostic accuracy than the evaluated baselines under many IID and non-IID settings, with up to **9.8 percentage points** improvement over the highest-performing baseline in the evaluated non-IID scenarios. The results also indicate favorable empirical stability in the evaluated three-seed runs and measured auxiliary resource-efficiency scores, suggesting an accuracy–variability–communication trade-off for representative resource-heterogeneous IIoT fault diagnosis tasks. Although the experiments are conducted on public benchmark datasets, the resource-heterogeneous settings, partial participation, feature-shift partition, and client-selection analyses provide controlled evidence for the resource-coordination behavior of FedAnchor. Ablation and sensitivity studies further show that anchored aggregation and server-side adaptive submodel allocation are complementary components and that a moderate anchoring ratio provides an empirical default across the evaluated datasets. The additional feature-shift and signal-level analyses, reported in the Appendix, further show that FedAnchor remains effective beyond a single Dirichlet label-skew partition and that the anchored subset develops more concentrated fault-related spectral and activation responses. These results support its role as a commonly updated diagnostic representation rather than a manually predefined physical-frequency component.

Author Contributions: Conceptualization, Y.H. and C.C.; methodology, Y.H. and X.L.; software, Y.H. and X.L.; validation, Y.H., X.L. and Z.X.; formal analysis, Y.H.; investigation, Y.H. and X.L.; resources, C.C.; data curation, Y.H. and X.L.; writing—original draft preparation, Y.H.; writing—review and editing, X.L., Z.X., J.P. and C.C.; visualization, Y.H.; supervision, C.C.; project administration, C.C.; funding acquisition, C.C. and Z.X. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China under Grant No. 62372063 and the Young Scientists Fund of the National Natural Science Foundation of China under Grant No. 62502284.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Supplementary Experimental Details and Analyses

This appendix provides additional experimental protocols, robustness checks, convergence curves, and interpretability analyses that support the main findings while keeping the main text concise.

Appendix A.1. Data Partition Details

Additional Non-IID Mode: Feature Distribution Shift: The main non-IID setting uses Dirichlet label skew with $\beta = 0.3$, which primarily controls the class distribution across clients. To further evaluate robustness under another type of industrial statistical heterogeneity, we add an additional non-IID mode with feature distribution shifts, denoted as *feature-shift*. This setting reflects practical industrial scenarios where different clients operate under different working conditions, such as load levels, rotation speeds, environmental conditions, or acquisition periods, leading to distribution shifts in vibration-signal features even when the observed fault categories are the same.

Algorithm A1 Dirichlet Label-Skew of Non-IID Partition

Require: Training set D^{train} ; number of clients K ; concentration parameter β ; minimum sample threshold $n_{\min}$; random seed s

Ensure: Client datasets $\{D_1, \dots, D_K\}$

- 1: Set random seed s
- 2: **repeat**
- 3: Initialize $D_k \leftarrow \emptyset$ for all $k \in \{1, \dots, K\}$
- 4: **for** each diagnostic class c **do**
- 5: Collect and shuffle all training samples of class c
- 6: Sample client proportions $\mathbf{p}_c \sim \text{Dirichlet}(\beta \mathbf{1}_K)$
- 7: Split samples of class c according to $\mathbf{p}_c$
- 8: Add the split subsets to the corresponding client datasets $\{D_k\}_{k=1}^K$
- 9: **end for**
- 10: **until** $\min_k |D_k| \geq n_{\min}$
- 11: **return** $\{D_1, \dots, D_K\}$

Different from Dirichlet label skew, the feature-shift setting keeps the class coverage approximately balanced across clients. For each diagnostic class c , we first divide its training samples into K disjoint feature blocks:

$$\mathcal{P}_{c,1}, \mathcal{P}_{c,2}, \dots, \mathcal{P}_{c,K}, \quad (\text{A1})$$

where K is the number of clients. When operating-condition or acquisition-segment information is available, the samples are grouped according to such condition information before block construction. When explicit condition labels are unavailable, samples are ordered by their acquisition sequence before partitioning, because adjacent vibration windows are more likely to share similar signal characteristics. Client k then receives a rotated block assignment from class c :

$$\mathcal{D}_k^{(c)} = \mathcal{P}_{c,((k+c-2) \bmod K)+1}. \quad (\text{A2})$$

This assignment ensures that each client observes all fault classes, while different clients receive different feature blocks within each class. Therefore, the resulting heterogeneity mainly comes from feature-distribution differences rather than label-distribution differences. The same feature-shift partition is reused across the evaluated methods under the same dataset, memory configuration, and random seed.

Algorithm A2 Feature Distribution Shift of Non-IID Partition

Require: Training set $\mathcal{D}_{\text{train}}$; number of clients K ; random seed s

Ensure: Client datasets $\{\mathcal{D}_1, \dots, \mathcal{D}_K\}$

- 1: Set random seed s
- 2: Initialize $\mathcal{D}_k \leftarrow \emptyset$ for all $k \in \{1, \dots, K\}$
- 3: **for** each diagnostic class c **do**
- 4: Collect all training samples of class c as $\mathcal{S}_c$
- 5: Order or group $\mathcal{S}_c$ according to operating condition, acquisition segment, or acquisition sequence
- 6: Divide $\mathcal{S}_c$ into K disjoint feature blocks $\{\mathcal{P}_{c,1}, \dots, \mathcal{P}_{c,K}\}$
- 7: **for** each client $k \in \{1, \dots, K\}$ **do**
- 8: Compute block index $p \leftarrow ((k+c-2) \bmod K) + 1$
- 9: Assign $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \mathcal{P}_{c,p}$
- 10: **end for**
- 11: **end for**
- 12: **return** $\{\mathcal{D}_1, \dots, \mathcal{D}_K\}$

Table A1. Data partition settings used in the experiments.

Partition Mode	Main Heterogeneity Type	Key Characteristics
IID	None	Stratified allocation with balanced class distributions across clients.
Dirichlet label skew	Label-distribution heterogeneity	Class-wise Dirichlet allocation with concentration parameter $\beta = 0.3$, producing different class proportions across clients.
Feature-shift	Feature-distribution heterogeneity	Condition- or acquisition-based feature blocks; all clients have all classes, but clients receive different feature blocks within each class.

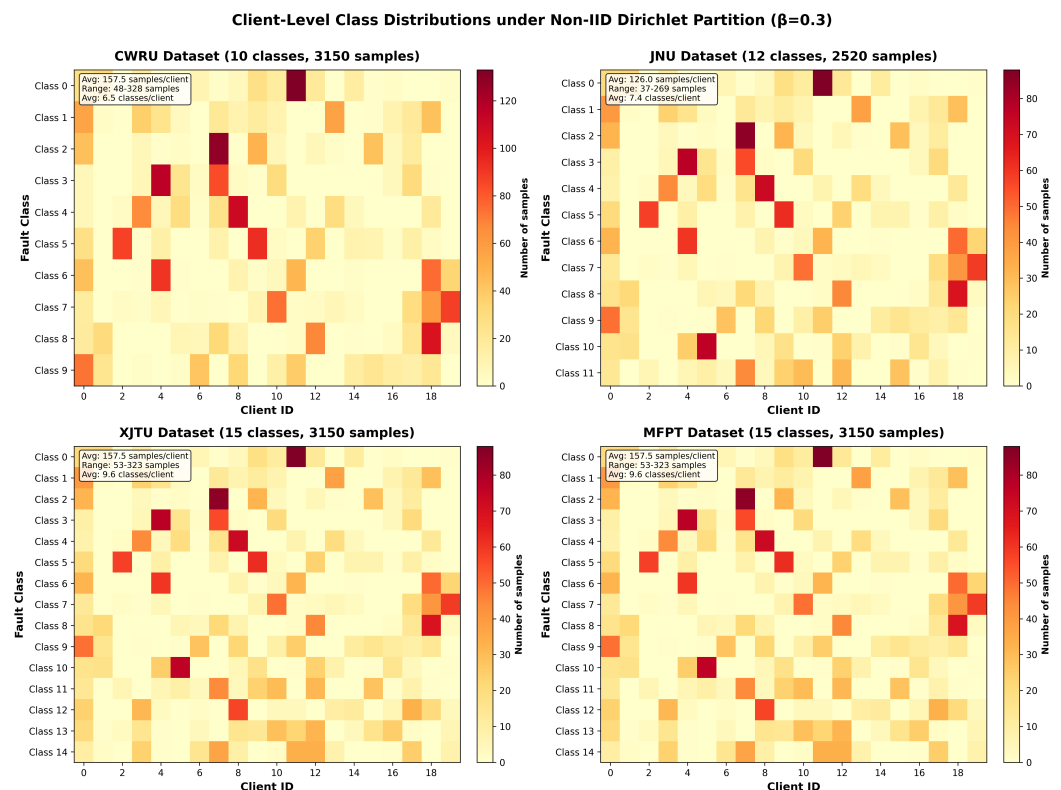


Figure A1. Client-level class distributions under the non-IID Dirichlet partition with $\beta = 0.3$ for all four datasets. Rows denote diagnostic classes, columns denote clients, and color intensity indicates the number of training windows assigned to each client for the corresponding class. The sample numbers shown in the subplot titles refer to training windows after the 70/30 train/test split. The partition creates clear label skew across clients: CWRU has 3–10 classes per client, JNU has 4–12 classes per client, and XJTU and MFPT have 5–15 classes per client. The same partition generated under each random seed is reused across all compared methods for fair evaluation.

Table A2. Client-level sample and class statistics under non-IID partitioning with Dirichlet $\beta = 0.3$ and minimum-sample threshold $n_{\min} = 30$. The minimum sample counts correspond to the smallest number of training windows assigned to any client after applying Algorithm A1.

Dataset	Avg. Samples/Client	Min–Max Samples	Avg. Classes/Client	Min–Max Classes
CWRU	157.5	48–328	6.5	3–10
JNU	126.0	37–269	7.4	4–12
XJTU	157.5	53–323	9.6	5–15
MFPT	157.5	53–323	9.6	5–15

Appendix A.2. Convergence Behavior Analysis

Although the previous sections report final diagnostic accuracy and communication–accuracy trade-offs, convergence behavior is also important for practical IIoT deployment. In particular, FedAnchor introduces server-side online policy learning for adaptive sub-model allocation, which may raise concerns about early-stage convergence delay compared with static allocation methods. To examine this issue, we provide a representative convergence comparison of CWRU under the heterogeneous memory configuration $\rho_k \in \{0.125, 0.25, 0.5, 1.0\}$ and the Dirichlet label-skew non-IID partition with $\beta = 0.3$.

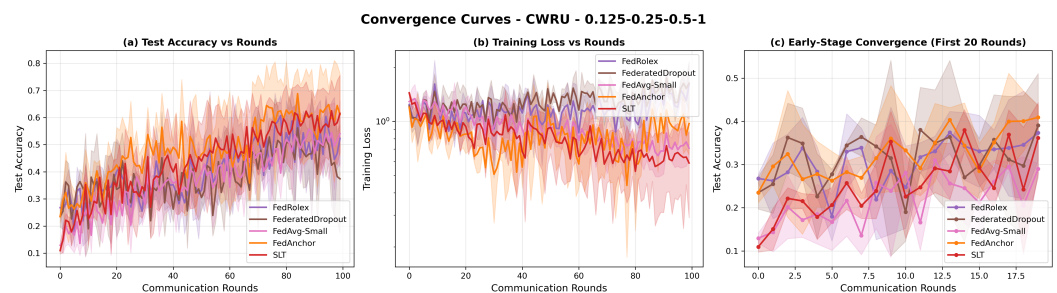


Figure A2. Representative convergence behavior on CWRU under heterogeneous memory configuration $\rho_k \in \{0.125, 0.25, 0.5, 1.0\}$ and Dirichlet label-skew non-IID partition with $\beta = 0.3$. Curves are averaged over three fixed seeds {2024, 2025, 2026}, and shaded regions denote empirical standard deviation. (a) Test accuracy versus communication rounds. (b) Training loss versus communication rounds. (c) Early-stage test accuracy during the first 20 communication rounds.

Figure A2a shows that FedAnchor does not suffer from an obvious early-stage convergence delay despite the use of server-side online policy learning. In the first 20 communication rounds, as enlarged in Figure A2c, FedAnchor improves test accuracy at a rate comparable to or faster than the representative baselines. This indicates that the fixed anchored subset provides a stable aggregation path from the beginning of training, while the DRL policy gradually adapts the non-anchored extension within the feasible memory budget.

Figure A2b further shows that FedAnchor converges smoothly without divergence or severe oscillation. Although FedAvg-Small may exhibit lower training loss in later rounds, this does not necessarily translate into better test accuracy under non-IID and resource-constrained settings. FedAvg-Small uses the smallest homogeneous feasible model to satisfy all client budgets, whereas FedAnchor allows higher-resource clients to exploit additional adaptive capacity while preserving a shared anchored aggregation subspace. Therefore, the convergence curves support the interpretation that FedAnchor balances adaptive allocation and aggregation stability rather than relying on server-side policy learning alone.

Regarding practical deployment time, FedAnchor should not be interpreted as minimizing raw wall-clock training time. As reported in Table 12, FedAnchor requires 19.5 s per round and 32.5 min for 100 rounds in the single-machine sequential simulation compared with 15.0 s per round and 25.0 min for FedAvg-Small. The additional overhead comes from adaptive submodel management and server-side policy optimization. Therefore, the practical advantage of FedAnchor lies in achieving a favorable communication–accuracy trade-off with manageable server-side overhead rather than being the fastest method in raw training time. In communication-limited IIoT deployments, reducing transmitted model traffic while improving diagnostic accuracy can be more valuable than minimizing server-side computation time alone.

Appendix A.3. Robustness Under Feature Distribution Shift

To evaluate whether FedAnchor remains effective beyond Dirichlet label-skew non-IID, we further test representative baselines under the feature-shift partition defined in Algorithm A2. This setting simulates industrial scenarios where different clients operate under different working conditions or acquisition periods, leading to shifts in vibration-signal feature distributions even when the same fault categories are observed. Unlike Dirichlet label skew, where clients have different class distributions, the feature-shift setting ensures that each client has access to all fault classes but receives different feature blocks within each class.

Unless otherwise specified, we use the representative four-tier memory configuration $\rho_k \in \{0.125, 0.5, 0.75, 1.0\}$, $K = 20$ clients, five selected clients per round, and three fixed random seeds $\{2024, 2025, 2026\}$. Table A3 reports the diagnostic accuracy under feature-shift non-IID.

Table A3. Diagnostic accuracy under the feature distribution shift non-IID setting. Results are reported under the representative four-tier memory configuration $\rho_k \in \{0.125, 0.5, 0.75, 1.0\}$. All values are mean accuracy $\pm$ standard deviation over three fixed seeds $\{2024, 2025, 2026\}$. Bold values indicate the highest accuracy for each dataset.

Algorithm	CWRU	JNU	MFPT	XJTU
FedAvg-Small	81.04 $\pm$ 1.82	49.91 $\pm$ 2.08	65.33 $\pm$ 0.34	79.63 $\pm$ 2.99
FedRolex	64.57 $\pm$ 3.28	48.50 $\pm$ 6.58	52.47 $\pm$ 8.44	85.93 $\pm$ 3.81
FederatedDropout	37.61 $\pm$ 4.45	38.54 $\pm$ 13.13	36.37 $\pm$ 3.70	46.62 $\pm$ 5.97
FedAnchor	91.46 $\pm$ 1.01	64.22 $\pm$ 0.14	74.72 $\pm$ 1.58	89.80 $\pm$ 7.35
SLT	79.58 $\pm$ 3.23	52.85 $\pm$ 7.03	64.82 $\pm$ 3.98	82.79 $\pm$ 1.80

The results show that feature-shift non-IID presents a different optimization challenge from Dirichlet label skew. Under feature shift, clients have access to all fault classes, but the feature statistics within each class differ across clients. This setting therefore tests whether the learned global representation remains stable when the heterogeneity comes from signal characteristics rather than class imbalance.

FedAnchor achieves the highest accuracy across all four datasets under this additional non-IID mode. Compared with FedAvg-Small, FedAnchor improves accuracy by 10.42, 14.31, 9.39, and 10.17 percentage points on CWRU, JNU, MFPT, and XJTU, respectively. The improvement indicates that the anchored shared subspace remains useful when clients have balanced class coverage but shifted feature distributions. The adaptive extension further allows clients with different memory budgets to exploit feasible model capacity without forcing all clients to train the weakest homogeneous model.

These results complement the Dirichlet label-skew experiments and show that FedAnchor is not only effective under a single non-IID partition type. They also suggest that industrial federated fault diagnosis should evaluate both label-distribution skew and feature-distribution shifts, because these two forms of heterogeneity impose different challenges on local training and global aggregation.

Appendix A.4. Robustness to Client Selection Policies

The main experiments use uniform random client selection with five clients per round. Although this protocol is widely used for partial-participation FL and provides a method-independent evaluation setting, it may introduce round-wise variation in the memory-tier composition of selected clients under resource heterogeneity. Such variation could potentially affect the update frequency of different submodel components and the stability of anchored-parameter aggregation.

To examine this effect, we conduct an additional sensitivity study on client selection policies. We compare the default random selection with two alternatives: tier-balanced selection and round-robin selection. *Random* selection samples $M = 5$ clients uniformly without replacement from all $K = 20$ clients. *Tier-balanced* selection explicitly controls the memory-tier composition of each selected client set; under the representative four-tier configuration $\rho_k \in \{0.125, 0.5, 0.75, 1.0\}$, one client is sampled from each tier, and the fifth client is assigned to a tier in a rotating manner across rounds. *Round-robin* selection follows a deterministic cyclic schedule to ensure that all clients participate approximately the same number of times during training.

All policies use the same number of selected clients per round, the same total number of communication rounds, and the same local training settings. The experiments are conducted under the representative four-tier memory configuration $\rho_k \in \{0.125, 0.5, 0.75, 1.0\}$ and Dirichlet label-skew non-IID partition with $\beta = 0.3$. All values are reported over the three fixed seeds $\{2024, 2025, 2026\}$.

Table A4. Effect of client selection policies on FedAnchor under the representative four-tier memory configuration $\rho_k \in \{0.125, 0.5, 0.75, 1.0\}$ and Dirichlet label-skew non-IID partition with $\beta = 0.3$. All values are mean accuracy $\pm$ standard deviation over the three fixed seeds $\{2024, 2025, 2026\}$.

Selection Policy	CWRU	JNU	MFPT	XJTU	Avg.
Random	94.22 $\pm$ 0.54	85.41 $\pm$ 1.26	92.85 $\pm$ 0.91	88.56 $\pm$ 1.69	90.26
Tier-balanced	94.41 $\pm$ 0.49	85.63 $\pm$ 1.21	93.08 $\pm$ 0.78	88.81 $\pm$ 1.54	90.48
Round-robin	94.18 $\pm$ 0.57	85.35 $\pm$ 1.33	92.79 $\pm$ 0.94	88.49 $\pm$ 1.73	90.20

Table A4 shows that FedAnchor remains stable across different client selection policies. Compared with random selection, tier-balanced selection slightly reduces the round-wise fluctuation of selected resource tiers, leading to a marginal average improvement of 0.22 percentage points across datasets. Round-robin selection ensures more uniform long-term participation across clients and achieves performance comparable to random selection, with an average difference of only -0.06 percentage points. Overall, the diagnostic accuracy varies by less than 0.3 percentage points across the three policies, indicating that FedAnchor is not overly dependent on a particular random client selection sequence.

This robustness can be attributed to the anchored submodel design. The anchored subset is retained by every selected client regardless of its memory tier. Therefore, even when the adaptive extension varies with client resources, the anchored parameters still receive updates from the selected participants in every communication round. This design helps maintain a shared diagnostic representation under partial participation, while the adaptive extension accommodates resource-specific capacity differences. In addition, the random sequence used in the main experiments is fixed per seed and shared across all compared methods, which prevents client-selection randomness from favoring a particular algorithm.

The small improvement observed under tier-balanced selection suggests that explicitly controlling the memory-tier composition can provide additional stability by reducing round-wise fluctuation in the distribution of selected resource tiers. However, FedAnchor under random selection already achieves competitive performance, indicating that the anchoring mechanism mitigates much of the update inconsistency caused by resource-heterogeneous participation without requiring controlled selection policies.

Appendix A.5. Activation and Signal-Level Analysis of Anchored Parameters

To further examine the signal-level behavior of anchored channels in industrial fault diagnosis, we analyze the learned first-layer filters and their activation responses. This analysis is intended to clarify whether the anchored subspace preserves common fault-

discriminative information rather than to claim a manually predefined correspondence between anchored neurons and specific physical frequency bands.

Filter Frequency Response: For the first convolutional layer, we compute the normalized frequency response of each learned kernel. For a convolutional kernel w_j , the normalized response is defined as

$$R_j(f) = \frac{|\mathcal{F}(w_j)(f)|}{\sum_{f'} |\mathcal{F}(w_j)(f')| + \epsilon}, \quad (\text{A3})$$

where $\mathcal{F}$ denotes the Fourier transform, and ϵ is a small constant for numerical stability. We average $R_j(f)$ over anchored and non-anchored channels, respectively, to compare the spectral regions emphasized by the two channel groups.

Activation Energy: For an input sample x , let $z_{\ell,j}(x)$ denote the activation map of channel j in layer ℓ . For a channel subset $\mathcal{G}_\ell$, where $\mathcal{G}_\ell$ can be the anchored subset or the non-anchored subset, the activation energy is computed as

$$E_\ell^{\mathcal{G}}(x) = \frac{1}{|\mathcal{G}_\ell|} \sum_{j \in \mathcal{G}_\ell} \frac{1}{T_\ell} \sum_{\tau=1}^{T_\ell} |z_{\ell,j}(x, \tau)|, \quad (\text{A4})$$

where T_ℓ denotes the temporal length of the activation map.

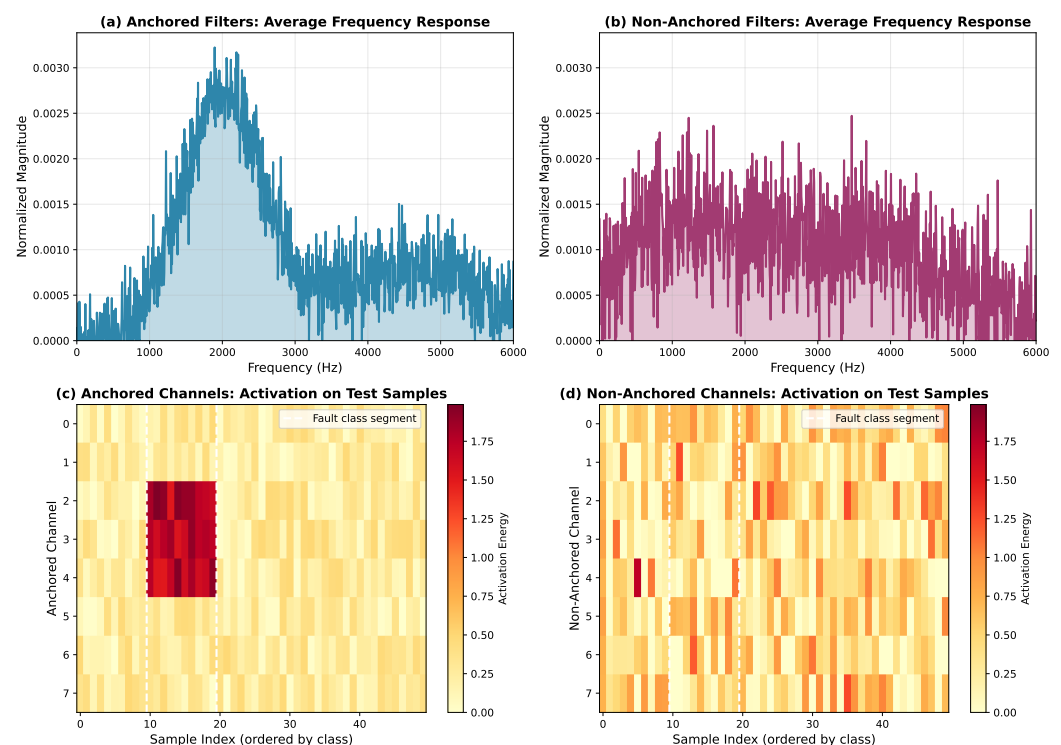


Figure A3. Signal-level interpretation of anchored and non-anchored channels. The visualization is obtained from the first convolutional layer of FedAnchor on CWRU under heterogeneous memory configurations $\rho_k \in \{0.125, 0.5, 0.75, 1.0\}$ and the non-IID partition with Dirichlet $\beta = 0.3$. (a,b) Average frequency responses of first-layer convolutional filters in the anchored and non-anchored subsets, with a unified y-axis for fair comparison. Anchored filters are not manually tied to predefined frequency bands; instead, they exhibit more concentrated and stable responses in fault-sensitive spectral regions through repeated cross-client updates. (c,d) Activation energy heatmaps on test samples ordered by class, with a unified colorbar and colormap for fair comparison. Dashed vertical lines indicate a selected fault-class segment. Anchored channels show more consistent fault-related activation patterns across the fault segment, whereas non-anchored channels retain more dispersed and client-specific adaptive responses.

Figure A3 provides a signal-level comparison between anchored and non-anchored channels. The frequency–response curves show that anchored filters do not correspond to a single manually specified frequency band. Instead, they form a group of learnable filters with more concentrated spectral responses after federated training. Compared with the anchored filters, the non-anchored filters exhibit more dispersed responses over the frequency range, which is consistent with their role as client-specific adaptive extensions.

The activation heatmaps further show that anchored channels produce more concentrated responses on the representative fault-class segment. In contrast, the non-anchored channels exhibit more scattered activation patterns. This observation suggests that the anchored subset tends to preserve common fault-discriminative information that is repeatedly updated across clients, whereas the non-anchored subset mainly provides additional adaptive capacity for client-specific variations. Therefore, the signal-level analysis supports the motivation of FedAnchor: anchoring does not impose predefined physical features but encourages a shared set of learnable diagnostic filters that helps stabilize aggregation under structural heterogeneity.

As a qualitative diagnostic visualization, this analysis should not be interpreted as a deterministic mapping between anchored channels and physical bearing-characteristic frequencies. Instead, it provides empirical evidence that the repeatedly updated anchored subset develops more stable fault-related spectral and activation responses under the evaluated federated setting.

References

1. Zhu, Z.; Lei, Y.; Qi, G.; Chai, Y.; Mazur, N.; An, Y.; Huang, X. A review of the application of deep learning in intelligent fault diagnosis of rotating machinery. *Measurement* **2023**, *206*, 112346. [\[CrossRef\]](#)
2. Hoang, D.T.; Kang, H.J. A survey on deep learning based bearing fault diagnosis. *Neurocomputing* **2019**, *335*, 327–335. [\[CrossRef\]](#)
3. Zhang, F.; Li, Z.; Cheng, Y.; Yang, Y.; Han, Y.; Yi, C.; Li, T.; Yan, J.; Dong, L.; Zhang, W. MHSNet: A Multi-Scale Hidden State Interaction Network for Fault Diagnosis of Rotating Machinery. *Tsinghua Sci. Technol.* **2025**, *31*, 2855–2876. [\[CrossRef\]](#)
4. Zou, L.; Ling, H.; Lei, M.; Fang, X.; Cai, M.; Yu, H. Domain-Independent Gear Pitting Fault Diagnosis Using Transformer Encoder and LinSoftmax. *Big Data Min. Anal.* **2025**, *8*, 1127–1147. [\[CrossRef\]](#)
5. The European Parliament and the Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council*; Official Journal of the European Union: Brussels, Belgium, 2016.
6. Qian, Q.; Zhang, B.; Li, C.; Mao, Y.; Qin, Y. Federated transfer learning for machinery fault diagnosis: A comprehensive review of technique and application. *Mech. Syst. Signal Process.* **2025**, *223*, 111837. [\[CrossRef\]](#)
7. Zhang, W.; Li, X.; Ma, H.; Luo, Z.; Li, X. Federated learning for machinery fault diagnosis with dynamic validation and self-supervision. *Knowl.-Based Syst.* **2021**, *213*, 106679. [\[CrossRef\]](#)
8. Jiang, H.; Chen, S.; Ouyang, T.; Lin, M.; Zhang, X.; Zhang, Q.; Chen, X. MFEL-H2B: Multimodal Federated Edge Learning with heterogeneity-aware balancing across modalities and models. *High-Confid. Comput.* **2026**, 100384. [\[CrossRef\]](#)
9. Yang, T.; Andrew, G.; Eichner, H.; Sun, H.; Li, W.; Kong, N.; Ramage, D.; Beaufays, F. Applied federated learning: Improving google keyboard query suggestions. *arXiv* **2018**, arXiv:1812.02903.
10. Maeng, K.; Lu, H.; Melis, L.; Nguyen, J.; Rabbat, M.; Wu, C.J. Towards fair federated recommendation learning: Characterizing the inter-dependence of system and data heterogeneity. In Proceedings of the 16th ACM Conference on Recommender Systems, Seattle, WA, USA, 18–23 September 2022; pp. 156–167.
11. Wu, M.; Li, L.; Chang, T.; Qiao, P.; Miao, C.; Zhou, J.; Wang, J.; Wang, X. FedEKT: Ensemble Knowledge Transfer for Model-Heterogeneous Federated Learning. In Proceedings of the 2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS), Guangzhou, China, 19–21 June 2024; pp. 1–10.
12. Qiao, Y.; Adhikary, A.; Kim, K.T.; Zhang, C.; Hong, C.S. Knowledge distillation assisted robust federated learning: Towards edge intelligence. In Proceedings of the ICC 2024-IEEE International Conference on Communications, Denver, CO, USA, 9–13 June 2024; pp. 843–848.
13. He, C.; Annamaram, M.; Avestimehr, S. Group knowledge transfer: Federated learning of large cnns at the edge. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 14068–14080.

14. Caldas, S.; Konečný, J.; McMahan, H.B.; Talwalkar, A. Expanding the reach of federated learning by reducing client resource requirements. *arXiv* **2018**, arXiv:1812.07210.
15. Wang, I.; Nair, P.; Mahajan, D. Fluid: Mitigating stragglers in federated learning using invariant dropout. *Adv. Neural Inf. Process. Syst.* **2024**, *36*, 73258–73273.
16. Alam, S.; Liu, L.; Yan, M.; Zhang, M. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 29677–29690. [[CrossRef](#)]
17. Ma, H.; Wei, J.; Zhang, G.; Wang, Q.; Kong, X.; Du, J. Heterogeneous Federated Learning: Client-side Collaborative Update Inter-Domain Generalization Method for Intelligent Fault Diagnosis. *IEEE Internet Things J.* **2024**, *12*, 5704–5718.
18. Kulevome, D.K.B.; Man, Q.; Mawuli, E.S.; Opoku-Mensah, E. Personalized Federated Learning with Kolmogorov-Arnold Enhanced CNN for Rotating Equipment Fault Diagnosis. In Proceedings of the 2024 21st International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), Chengdu, China, 14–16 December 2024; pp. 1–5.
19. Cui, J.; Li, J.; Mei, Z.; Wei, K.; Wei, S.; Ding, M.; Chen, W.; Guo, S. Federated meta-learning for few-shot fault diagnosis with representation encoding. *IEEE Trans. Instrum. Meas.* **2023**, *72*, 3536812. [[CrossRef](#)]
20. Li, X.; Zhang, C.; Li, X.; Zhang, W. Federated transfer learning in fault diagnosis under data privacy with target self-adaptation. *J. Manuf. Syst.* **2023**, *68*, 523–535. [[CrossRef](#)]
21. Yan, Z.; Xu, Z.; Zhang, Y.; Sun, J.; Liu, L.; Sun, Y. FTSDC: A novel federated transfer learning strategy for bearing cross-machine fault diagnosis based on dual-correction training. *Adv. Eng. Inform.* **2024**, *61*, 102499. [[CrossRef](#)]
22. Guo, S.; Chen, H.; Liu, Y.; Yang, C.; Li, Z.; Jin, C.H. Heterogeneous Federated Learning Framework for IIoT Based on Selective Knowledge Distillation. *IEEE Trans. Ind. Inform.* **2024**, *21*, 1078–1089. [[CrossRef](#)]
23. Horvath, S.; Laskaridis, S.; Almeida, M.; Leontiadis, I.; Venieris, S.; Lane, N. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 12876–12889.
24. Diao, E.; Ding, J.; Tarokh, V. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *arXiv* **2020**, arXiv:2010.01264.
25. Nader, B.; Jiahui, H.; Hui, Z.; Xin, L. Adaptive federated dropout: Improving communication efficiency and generalization for federated learning. In Proceedings of the Proceedings of the IEEE Conference on Computer Communications Workshops, Vancouver, BC, Canada, 10–13 May 2021; pp. 10–13.
26. Pfeiffer, K.; Khalili, R.; Henkel, J. Aggregating capacity in FL through successive layer training for computationally-constrained devices. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 35386–35402. [[CrossRef](#)]
27. Shamsian, A.; Navon, A.; Fetaya, E.; Chechik, G. Personalized Federated Learning using Hypernetworks. In *Proceedings of the 38th International Conference on Machine Learning*; Meila, M., Zhang, T., Eds.; PMLR: New York, NY, USA, 2021; Volume 139, pp. 9489–9502.
28. Litany, O.; Maron, H.; Acuna, D.; Kautz, J.; Chechik, G.; Fidler, S. Federated learning with heterogeneous architectures using graph hypernetworks. *arXiv* **2022**, arXiv:2201.08459.
29. Wang, R.; Yu, T.; Zhang, R.; Kim, S.; Rossi, R.; Zhao, H.; Wu, J.; Mitra, S.; Yao, L.; Henao, R. Personalized federated learning for text classification with gradient-free prompt tuning. In *Proceedings of the Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, 16–21 June 2024*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 4597–4612.
30. Kang, H.; Cha, S.; Shin, J.; Lee, J.; Kang, J. Nefl: Nested model scaling for federated learning with system heterogeneous clients. *IEEE Trans. Mob. Comput.* **2025**, *24*, 6734–6746. [[CrossRef](#)]
31. Wu, F.; Wang, X.; Wang, Y.; Liu, T.; Su, L.; Gao, J. Fiarse: Model-heterogeneous federated learning via importance-aware submodel extraction. *Adv. Neural Inf. Process. Syst.* **2024**, *37*, 115615–115651. [[CrossRef](#)]
32. Lillicrap, T. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
33. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the Artificial Intelligence and Statistics, Lauderdale, FL, USA, 20–22 April 2017*; PMLR: New York, NY, USA, 2017; pp. 1273–1282.
34. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 8026–8037.
35. Smith, W.A.; Randall, R.B. Rolling element bearing diagnostics using the Case Western Reserve University data: A benchmark study. *Mech. Syst. Signal Process.* **2015**, *64*, 100–131. [[CrossRef](#)]
36. Li, K.; Ping, X.; Wang, H.; Chen, P.; Cao, Y. Sequential fuzzy diagnosis method for motor roller bearing in variable operating conditions based on vibration analysis. *Sensors* **2013**, *13*, 8013–8041. [[CrossRef](#)] [[PubMed](#)]

37. Wang, B.; Lei, Y.; Li, N.; Li, N. A Hybrid Prognostics Approach for Estimating Remaining Useful Life of Rolling Element Bearings. *IEEE Trans. Reliab.* **2020**, *69*, 401–412. [[CrossRef](#)]
38. MFPT. Fault Data Sets. 2013. Available online: <http://www.mfpt.org/> (accessed on 10 April 2013).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.