FedAR: Attribute-Guided Representation Learning for Heterogeneous Federated Learning

Supplementary Material

6. Experimental Setup Details

6.1. Experimental Environment

All experiments were conducted on a server with 64 Intel Xeon Gold 6326 CPUs @ 2.90GHz, 125GB RAM, six NVIDIA GeForce RTX 3090 GPUs, and Ubuntu 22.04.4 LTS. Our implementation is based on PyTorch 2.7.0 and leverages the HuggingFace transformers library for accessing and utilizing the pre-trained CLIP text encoder.

6.2. Notation Table

For clarity, we provide a comprehensive summary of the mathematical notations used throughout this paper in Table 7.

Table 7. Summary of Key Notations

Symbol	Description
Federated Learning Setup (Section 3.1)	
M	Total number of clients
m	Client index, $m \in \{1, \ldots, M\}$
$\mathcal{D}_m$	Private dataset of client m
C	Total number of global classes
Attribute Space (Section 3.2)	
P	Number of attribute prototypes
$\mathbf{A}^*$	Attribute prototype matrix
Ω	Class–attribute association matrix (binary)
T_c	Textual prototype for class c
Feature Modulation (Section 3.3.1)	
$F_m(x_i)$	Spatial features from client m's extractor
S_{mod}	Spatial modulation map ($H \times W$)
$F'_m(x_i)$	Modulated features after modulation
General Notation	
D_f, D_t	Feature and text embedding dimensions
τ	Temperature in contrastive loss
λ	Loss balancing weight

6.3. Dataset and Heterogeneity Specifications

Dataset Specifications. We evaluate FedAR on three benchmark datasets widely adopted in heterogeneous federated learning. These datasets span increasing visual complexity and class granularity, ensuring a comprehensive evaluation across small-scale (Cifar10), medium-scale (Cifar100), and large-scale (Tiny-ImageNet) settings. The detailed statistics are reported in Table 8.

Table 8. Statistics of datasets used in our experiments.

Dataset	Classes	Train Size	Test Size	Image Size
Cifar10	10	50,000	10,000	$32 \times 32 \times 3$
Cifar100	100	50,000	10,000	$32 \times 32 \times 3$
Tiny-ImageNet	200	100,000	10,000	$64 \times 64 \times 3$

Model Heterogeneity Specifications. We consider four heterogeneous feature extractor (HtFE) groups, each reflecting a different degree of architectural diversity. Specifically, these settings are defined as follows:

- **HtFE$_2$**: 4-layer CNN and ResNet18.
- **HtFE$_3$**: ResNet10 [44], ResNet18, and ResNet34.
- **HtFE$_4$**: 4-layer CNN, GoogleNet [28], MobileNet_v2 [25], and ResNet18.
- **HtFE$_9$**: ResNet4, ResNet6, ResNet8 [44], ResNet10, ResNet18, ResNet34, ResNet50, ResNet101, and ResNet152 [6].

6.4. Hyperparameter Settings

In addition to the hyperparameter settings provided in the main body, we adhere to each baseline method's original paper for their respective hyperparameter settings.

- **LG-FedAvg [18]:** No additional hyperparameters.
- **FedGen [48]:** Noise dimension is 32, generator learning rate is 0.1, hidden dimension equals the feature dimension 512, and server learning epochs are 100.
- **FML [26]:** Knowledge distillation hyperparameters are $\alpha = 0.5$ and $\beta = 0.5$. The auxiliary model is chosen as the smallest model in the group to minimize communication overhead.
- **FedKD [31]:** Auxiliary model learning rate is 0.01, $T_{start} = 0.95$, and $T_{end} = 0.95$. The smallest model is chosen as the auxiliary model.
- **FedDistill [7]:** Knowledge distillation factor $\gamma = 1$.
- **FedProto [29]:** Prototype loss weight $\lambda = 0.1$.
- **FedTGP [41]:** $\lambda = 0.1$, margin threshold $\tau = 100$, and server learning epochs $S = 100$.
- **FedSA [46]:** $\lambda_1 = 0.1$, $\lambda_2 = 1$, $\lambda_3 = 1$, and $\alpha = 0.9999$.
- **FedTSP [32]:** Contrastive Temperature $\tau = 0.07$; FedTSP produces k descriptions, k = 3.

7. Pseudo-code of FedAR

For completeness, the pseudo-code of FedAR is presented in Algorithm 1. Each client receives the shared attribute prototypes $\mathbf{A}^*$, adapts them via a lightweight adapter, and applies attribute-guided modulation to its local features.

The local loss combines the contrastive classification loss $\mathcal{L}_{\text{con}}$ and attribute supervision loss $\mathcal{L}_{\text{attr}}$.

Algorithm 1: FedAR

Input: Clients number M; Class number C; Server-side frozen CLIP text encoder Φ_{text}; Server-side LLM (*e.g.*, GPT-4o); Number of attribute clusters P; Training rounds T; Client epochs E; Loss balance λ.

Output: Personalized models $\{\theta_m, w_m\}_{m=1}^{M}$.

Server-side: Semantic Space Construction

- Encode class prompts to obtain $\{T_c\}$ by Eq. (2).
- Generate 4–6 visual attribute phrases per class using LLM.
- Encode all phrases to obtain $\{A_{c,j}\}$ by Eq. (3).
- Cluster embeddings via K-means: $\mathbf{A}^* = [\mu_1, \ldots, \mu_P]$ by Eq. (4).
- Build class–attribute association matrix $\Omega \in \{0,1\}^{C \times P}$.
- Send $\mathbf{A}^*$, Ω, and $\{T_c\}$ to all clients.

Client-side: Attribute-Guided Representation Learning

for *round* $t = 0$ **to** $T-1$ **do**
  for *client* $m = 1$ **to** M ***in parallel*** **do**
    for *epoch* $e = 1$ **to** E **do**
      (1) Feature Extraction:
      Extract spatial feature map $F_m(x_i)$ from local backbone f_m.
      (2) Attribute-Conditioned Modulation:
      Adapt prototypes: $\mathbf{A}_{\text{adapt}}$ by Eq. (5).
      Compute correlation scores S by Eq. (9).
      Compute normalized scores $\tilde{S}$ by Eq. (10).
      Compute modulation map S_{mod} by Eq. (11) and reshaping.
      Obtain modulated features $F'_m(x_i)$ by Eq. (12).
      Apply global average pooling to get $F'_m(x_i)$.
      (3) Optimization:
      Compute $\mathcal{L}_{\text{con}}$ by Eq. (13).
      Compute $\mathcal{L}_{\text{attr}}$ by Eq. (16).
      Update parameters using $\mathcal{L}$ by Eq. (17).

Inference: Predict by cosine similarity: $\hat{y} = \arg\max_c \hat{F}'_m(x_i)^\top \hat{T}_c$.

8. Additional Experimental Results

8.1. Sensitivity to Local Training Epochs

Increasing the number of local training epochs (E) is a common strategy to reduce communication costs in federated learning. However, this often exacerbates client drift due to extended local optimization on heterogeneous data. As shown in Table 9, methods like FML and FedKD show degraded performance with more local epochs due to increased heterogeneity in auxiliary models. In contrast, FedAR maintains stable accuracy above 51% across all settings, demonstrating robustness to extended local training. This stability stems from our language-driven semantic space, which provides consistent supervision unaffected by local optimization trajectories.

Table 9. The test accuracy(%) on Cifar100 under different Local training epochs in the practical setting using the HtFE_8 model group.

Method	$E = 5$	$E = 10$	$E = 20$
LG-FedAvg [18]	40.33±0.15	40.46±0.08	40.93±0.23
FedGen [48]	40.00±0.41	39.66±0.31	40.07±0.12
FML [26]	39.08±0.27	37.97±0.19	36.02±0.22
FedKD [31]	41.06±0.13	40.36±0.20	39.08±0.33
FedDistill [7]	42.02±0.30	41.29±0.23	41.13±0.41
FedProto [29]	38.04±0.52	38.13±0.42	38.74±0.51
FedTGP [41]	46.44±0.26	46.59±0.31	46.65±0.29
FedSA [46]	45.73±0.22	46.40±0.19	46.10±0.23
FedTSP [32]	47.04±0.25	46.23±0.21	46.08±0.20
FedAR	**51.09±0.21**	**51.62±0.23**	**51.23±0.18**

8.2. Hyperparameter Sensitivity Analysis

We conduct a sensitivity analysis on the key hyperparameters specific to the FedAR framework, which were fixed at default values in the main body of the paper. This analysis justifies the choice of default settings and demonstrates the robustness of our method. All analyses are performed on the Cifar100 dataset using the HtFE_8 setting.

8.2.1. Influence of Contrastive Temperature τ

The contrastive temperature τ controls the concentration of the similarity distribution in the contrastive classification loss $\mathcal{L}_{\text{con}}$. A smaller τ leads to a sharper distribution, enforcing a tighter alignment between the visual features and the correct class prototype, thus encouraging stronger semantic clustering. Conversely, a larger τ smooths the distribution. We set the default value $\tau = 0.07$, following the common practice in CLIP-style contrastive learning. As shown in Table 10, the performance is robust around the default setting. When τ is slightly decreased to 0.03, the accuracy experiences a negligible drop (from 51.84% to 51.72%), indicating that the model remains stable and is

not overly sensitive to the increased penalty from hard negatives. However, increasing τ beyond the default value leads to a consistent degradation in performance. This confirms that over-smoothing the similarity distribution hinders the quality of the semantic alignment between the client-side features and the global language prototypes. The chosen default value $\tau = 0.07$ achieves the optimal balance, demonstrating the effectiveness of the standard setting in our heterogeneous federated context.

Table 10. The test accuracy on Cifar100 under different contrastive temperatures in the practical setting using the HtFE_8 model group.

τ	0.03	0.07 (default)	0.10	0.20	0.50
Acc (%)	51.72	**51.84**	51.23	50.25	49.57

8.2.2. Influence of Spatial Selection Ratio α

Table 11 shows a clear trend for the spatial selection ratio α, which determines the proportion of high-activation pixels used for attribute supervision. A low ratio such as $\alpha = 0.1$ selects only a very small portion of attribute-relevant regions and therefore omits useful spatial cues, resulting in lower accuracy. The performance peaks at $\alpha = 0.2$, where the mask captures sufficient discriminative structure without introducing excessive noise. Higher ratios progressively include more background and irrelevant regions, and at $\alpha = 1.0$ the mask degenerates into uniform averaging, yielding the weakest results. These observations confirm that attribute grounding benefits from a moderately sparse, discriminative spatial mask.

Table 11. The test accuracy on Cifar100 under different attribute activation ratios α in the practical setting using the HtFE_8 model group.

α	0.1	0.2 (default)	0.4	0.8	1.0
Acc (%)	50.62	**51.84**	51.25	49.02	48.34

8.3. Analysis of Text Encoders

Table 12 evaluates how different text encoders influence the global semantic space. BERT produces the weakest results because it lacks visual grounding and therefore generates prototypes that do not align well with image features. CLIP-based encoders consistently outperform BERT due to their exposure to large-scale image–text pairs during pretraining. Among them, CLIP/ResNet-50 provides a reasonable improvement, while the ViT-based encoders yield the strongest alignment. Interestingly, CLIP/ViT-B/16 and ViT-B/32 perform nearly identically across all datasets, suggesting that FedAR is largely insensitive to the specific ViT configuration as long as the encoder supplies sufficiently rich cross-modal semantics. This stability further indicates that the proposed attribute-guided framework does not depend on model-specific quirks of the textual backbone.

Table 12. The test accuracy(%) on different datasets under various text encoders in the practical setting using the HtFE_8 model group.

Text Encoder	Cifar10	Cifar100	Tiny-ImageNet
BERT	89.03±0.11	46.82±0.15	23.48±0.17
CLIP/ResNet-50	88.40±0.12	49.21±0.16	27.87±0.14
CLIP/ViT-B-16	89.24±0.13	51.83±0.14	35.13±0.11
CLIP/ViT-B-32 (default)	**89.33±0.12**	**51.84±0.15**	**35.25±0.13**

9. Training Cost Analysis

Client-side Cost. On the client side, FedAR introduces a lightweight attribute-conditioned feature modulation module, which involves an adapter (LayerNorm and linear projection) and cross-attention. Given the typically small spatial size of feature maps (*e.g.*, 7×7 for high-resolution images), the computational overhead of this modulation is minimal. The memory cost for clients remains low, comparable to other prototype-based methods, as they only need to store their local model weights f_m, the shared class textual prototypes $\{T_c\}$, the attribute textual prototypes $\mathbf{A}^*$, and the class–attribute association matrix Ω. Importantly, clients do not need to store or exchange local image prototypes, which further saves memory and communication bandwidth compared to traditional prototype aggregation methods.

Server-side Cost. The language model components are primarily involved on the server. The LLM (*e.g.*, GPT-4o) is used only once at the very start of the process to generate concise, visually grounded attribute phrases per class, incurring a negligible total computational cost over the entire training duration. This generation can also be done via a commercial API, eliminating the need for direct LLM deployment and its associated significant memory footprint. The frozen CLIP text encoder Φ_{text} is also used only once during the initialization phase to encode class and attribute texts and perform K-means clustering to construct the fixed semantic prototypes ($\mathbf{A}^*$ and $\{T_c\}$). Since all these operations are a one-time setup and the model CLIP remains frozen throughout the federated training rounds, the server-side overhead is dominated by these initial steps and is not a system bottleneck during the iterative training process.