Communication-Efficient Heterogeneous Federated Learning with Sparse Prototypes in Resource-Constrained Environments

Gyuejeong Lee[1]*, **Daeyoung Choi**[2,3]*
[1]SAKAK Inc., Seoul, South Korea
[2]Division of AI and Data Science, Korea Cyber University, Seoul, South Korea
[3]Intellectus, Seoul, South Korea
regulation.lee@sakak.co.kr, choidy@koreacu.ac.kr

Abstract

Communication efficiency in federated learning (FL) remains a critical challenge in resource-constrained environments. While prototype-based FL reduces communication overhead by sharing class prototypes—mean activations in the penultimate layer—instead of model parameters, its efficiency degrades with larger feature dimensions and class counts. We propose TinyProto, which addresses these limitations through Class-wise Prototype Sparsification (CPS) and Adaptive Prototype Scaling (APS). CPS enables structured sparsity by allocating specific dimensions to class prototypes and transmitting only non-zero elements, thereby achieving higher communication efficiency, while APS scales prototypes based on class distributions to improve performance. Our experiments demonstrate that TinyProto reduces communication costs by up to $10\times$ compared to existing methods while improving performance. Beyond communication efficiency, TinyProto offers crucial advantages: it achieves compression without client-side computational overhead and supports heterogeneous architectures, making it particularly suitable for resource-constrained heterogeneous FL scenarios.

1 Introduction

Federated learning (FL) has emerged as a transformative framework for collaborative model training without the need for centralized data aggregation (McMahan et al. 2017). It offers substantial advantages, including enhanced data privacy, optimized computational resource utilization, and improved communication efficiency (Zhang et al. 2021b). Among these benefits, reducing communication overhead is particularly critical for the scalability of FL (Sattler et al. 2019). Traditional FL frameworks transmit model weights or gradients, rather than raw data, between the server and clients, thereby conserving communication resources. However, as deep networks continue to grow in size, the communication overhead associated with transmitting model parameters increases significantly, undermining these advantages. This presents substantial challenges for large-scale deployments in resource-constrained environments, such as Internet of Things networks (Nguyen et al. 2021).

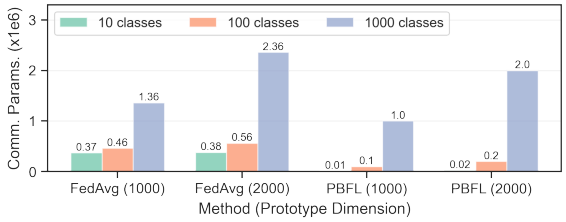


Figure 1: Comparison of communication costs between FedAvg and PBFL approaches for ShuffleNet v2.

To address this challenge, various strategies have been proposed to optimize either communication frequency (Wang and Joshi 2019; Wang et al. 2019; Karimireddy et al. 2020) or communication volume, with this work focusing specifically on the latter—reducing communication volume. Some approaches achieve this by transmitting only a subset of deep network parameters (Chen and Chao 2021; Wang et al. 2024) or gradients (Sattler et al. 2019; Han, Wang, and Leung 2020), while others utilize low-rank representations of parameter matrices (Wang et al. 2023; Wu et al. 2024, 2022). Additionally, techniques such as pruning model weights have proven effective in mitigating communication overhead (Jiang et al. 2022; Zhang et al. 2022; Wu, Song, and Zeng 2023). In contrast to transmitting model parameters, alternative methods exchange knowledge between the server and clients by sharing logits (Jeong et al. 2018; Li and Wang 2019), intermediate features (Tan et al. 2022; Zhang et al. 2024), auxiliary networks (Shen et al. 2020), or data generators (Zhu, Hong, and Zhou 2021). Although these methods were initially developed to address challenges such as data and model heterogeneity, they also contribute to improving communication efficiency.

Among these approaches, prototype-based FL (PBFL) stands out due to its communication efficiency. Rather than sharing model parameters, PBFL exchanges prototypes—mean activations in the feature layer (penultimate layer) for each class (Tan et al. 2022). This communication efficiency, combined with its effectiveness in handling model and data heterogeneity, has made PBFL approaches like FedProto widely researched and influential in heterogeneous FL (HtFL). However, our analysis revealed significant scalability limitations despite these communication advantages. Consider a classification task using ShuffleNet v2 (approximately 1.36M parameters for 1000 classes, with 1.0M pa-

*These authors contributed equally.

rameters between the penultimate and output layers) as illustrated in Figure 1. PBFL's communication costs, determined by the product of class count and prototype (feature) dimensions, are significantly lower than FedAvg (McMahan et al. 2017) when these factors are small. However, this advantage diminishes rapidly as either class count or prototype dimensions increase, limiting PBFL's effectiveness in resource-constrained networks. This scalability challenge becomes more pronounced in deep networks with higher ratios of classifier-to-feature extractor parameters.

In this work, we address the challenge of mitigating the communication overhead caused by scalability issues in PBFL, aiming to make PBFL more practical for resource-constrained environments. We propose TinyProto, a novel framework that combines Class-wise Prototype Sparsification (CPS) and Adaptive Prototype Scaling (APS) methods. The CPS method reduces communication overhead by introducing structured sparsity, where each class prototype utilizes only specific dimensions while setting the others to zero. While PBFL methods show promise for HtFL, they often underperform compared to other HtFL approaches, making the direct application of CPS impractical. To address this limitation, we introduce the APS method, which adjusts prototype scales based on class-wise sample sizes to reflect the importance of each client. By integrating CPS with APS, TinyProto achieves substantial communication efficiency without compromising performance. Experimental results demonstrate that TinyProto reduces communication costs by up to $10\times$ compared to the original PBFL approaches and $4\times$ compared to the most efficient baselines.

TinyProto differs from existing sparsification-based FL approaches by sparsifying prototypes rather than model parameters, which provides three key advantages:

- **Model Heterogeneity Support**: Enables heterogeneous architectures across clients, providing model flexibility.
- **Adaptive Compression**: Adjusts compression rates to match bandwidth constraints, optimizing resource use.
- **Computational Efficiency**: Eliminates the need for fine-tuning, reducing client-side computational overhead.

2 Related Work

Heterogeneous Federated Learning HtFL addresses challenges arising from heterogeneity in data distributions and model architectures. Existing HtFL approaches can be broadly categorized based on the type of information exchanged: model parameters, auxiliary model or data generator parameters, and logits or intermediate features. To accommodate varying model architectures, LG-FedAvg (Liang et al. 2020) allows clients to share upper layers while retaining architecture-specific lower layers. Auxiliary model-based methods, such as FML (Shen et al. 2020), use mutual distillation (Zhang et al. 2018) to train and share compact models. Similarly, sharing a data generator improves generalization (Zhu, Hong, and Zhou 2021). Other approaches avoid sharing model parameters by using logits or intermediate features. FedMD (Li and Wang 2019), for instance, uses a public dataset to enable knowledge transfer among heterogeneous clients. FedDistill (Jeong et al. 2018) transmits averaged class-wise logits from clients to the server; however, this method risks exposing the number of classes and the logit distribution, which can lead to potential privacy concerns. To address these risks, FedProto (Tan et al. 2022) takes a privacy-preserving approach by exchanging prototypes of the penultimate layer. Recent works have further refined PBFL techniques. FedTGP (Zhang et al. 2024) enhances performance through contrastive learning to improve prototype separability. Our work also builds upon PBFL, focusing on communication efficiency by introducing class-wise prototype sparsity.

Sparsity in Federated Learning Connection pruning (Han, Mao, and Dally 2015; Han et al. 2015) and structured weight sparsification (Wen et al. 2016) have been widely studied in deep learning to reduce memory usage and improve computational efficiency. These techniques have been adapted to FL to address communication constraints. Gradient sparsification enhances communication efficiency by transmitting only the most significant updates. Stich, Cordonnier, and Jaggi (2018) demonstrate that k-sparsification in stochastic gradient descent can significantly reduce communication costs. Sparse Ternary Compression (Sattler et al. 2019) extends this top-k gradient sparsification for federated learning, while Han, Wang, and Leung (2020) propose a fairness-aware method to ensure equitable updates across clients. Model weight pruning provides an alternative approach to reducing parameter transmission. PruneFL (Jiang et al. 2022) employs adaptive pruning to reduce model size, while FedDUAP (Zhang et al. 2022) prunes based on layer dimensions and importance. Several methods use masks to avoid transmitting unnecessary weights. Wu, Song, and Zeng (2023) employ masks to prune units or kernels dynamically, and FedMask (Li et al. 2021) transmits sparse binary masks instead of parameters. However, these approaches often require additional client-side computation, such as fine-tuning after pruning, or support only homogeneous models. Our framework also employs sparsification with masks, but applies it to prototypes, eliminating extra client-side computation and supporting heterogeneous models.

3 Problem Setting and Motivation

This section presents the problem setting and motivation.

3.1 Problem Setting

We consider an FL system comprising M clients and a server for a K-class classification task. The clients interact with the server to jointly develop personalized models without sharing their private data directly. Each client i has its data distribution P_i with K_i classes. These distributions can differ between clients, reflecting the typical scenario in FL. We define a loss function ℓ that evaluates the performance of each client's local model $\boldsymbol{w}_i$ on data points from their respective distributions. The system's aim can be described as minimizing the mean expected loss across all clients:

$$\min_{\mathbf{W}} \left\{ F(\mathbf{W}) := \frac{1}{M} \sum_{i=1}^{M} \mathbb{E}_{(x,y)\sim P_i} \left[\ell(\boldsymbol{w}_i; x, y) \right] \right\}, \quad (1)$$

where $\mathbf{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_M]$ represents a matrix containing all individual client models. Given that we only have a limited number of data points from each client (n_i), we estimate this expected loss using the empirical risk calculated on each client's local training dataset $\mathcal{D}_i = \{(x_i^{(l)}, y_i^{(l)})\}_{l=1}^{n_i}$ with empirical distribution $\hat{P}_i$. Thus, the training objective becomes finding the optimal set of local models that minimizes the average empirical risk across all clients:

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_i(\boldsymbol{w}_i). \tag{2}$$

Here, $\mathcal{L}_i(\boldsymbol{w}_i) = \frac{1}{n_i} \sum_{l=1}^{n_i} \ell(\boldsymbol{w}_i; x_i^{(l)}, y_i^{(l)})$ represents the average loss for each client, calculated over their private data.

In this work, we split each client's deep network $\boldsymbol{w}_i$ into two components: feature extractor and classifier. The feature extractor f_i of client i, parameterized by $\boldsymbol{\theta}_i$, transforms input data from $\mathbb{R}^D$ into a feature space $\mathbb{R}^d$. For the sample x, it produces a feature vector $\boldsymbol{c} = f_i(\boldsymbol{\theta}_i; x)$. The classifier g_i, parameterized by $\boldsymbol{\phi}_i$, maps these features to the final output space $\mathbb{R}^K$.

Prototype-Based Federated Learning In PBFL, clients exchange prototypes—the means of feature layer activations per class—rather than model parameters. Thus, the key process for handling prototypes consists of three steps. At the initial round, each client trains its model without any regularization before following these steps:

Step 1. Local Prototype Generation: Each client generates local prototypes from its private dataset and transmits them to the server. For each class j on client i, the local prototype $\bar{\boldsymbol{c}}_{i,j}^L$ is computed as:

$$\bar{\boldsymbol{c}}_{i,j}^L = \frac{1}{n_{i,j}} \sum_{(x,y)\in\mathcal{D}_{i,j}} f_i(\boldsymbol{\theta}_i; x), \tag{3}$$

where $\mathcal{D}_{i,j} \subseteq \mathcal{D}_i$ is client i's subset of class-j samples, and $n_{i,j} = |\mathcal{D}_{i,j}|$ is the number of such samples.

Step 2. Global Prototype Generation: The server generates global prototypes by aggregating local prototypes and transmits them to the clients. The global prototype $\bar{\boldsymbol{c}}_j^G$ for class j is obtained through weighted averaging (Tan et al. 2022):

$$\bar{\boldsymbol{c}}_j^G = \frac{1}{|\mathcal{N}_j|} \sum_{i\in\mathcal{N}_j} \frac{n_{i,j}}{\sum_{i=1}^{M} n_{i,j}} \bar{\boldsymbol{c}}_{i,j}^L, \tag{4}$$

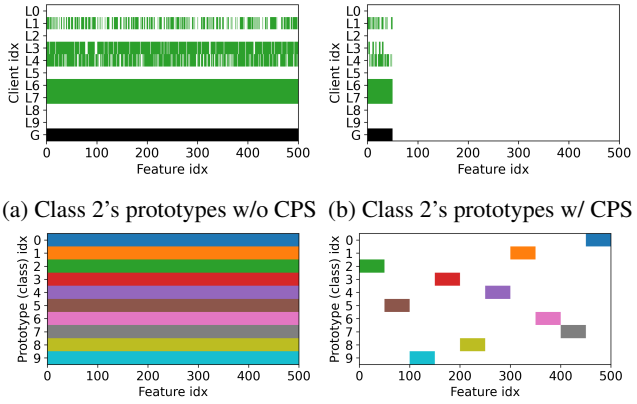
where $\mathcal{N}_j$ denotes the set of clients that have samples from class j, and $\sum_{i=1}^{M} n_{i,j}$ represents the total count of class-j samples in the system.

Step 3. Local Model Training: The local model at each client is trained using a combined loss function:

$$\tilde{\mathcal{L}}_i(\boldsymbol{w}_i) = \mathcal{L}_i(\boldsymbol{w}_i) + \lambda \mathcal{R}_i, \tag{5}$$

where $\mathcal{R}_i$ is the regularization term for knowledge distillation, and λ is a hyperparameter that regulates the regularization intensity. The term $\mathcal{R}_i$ is formulated as:

$$\mathcal{R}_i = \sum_j \rho(\bar{\boldsymbol{c}}_{i,j}^L, \bar{\boldsymbol{c}}_j^G), \tag{6}$$



(a) Class 2's prototypes w/o CPS (b) Class 2's prototypes w/ CPS

(c) Global prototypes w/o CPS (d) Global prototypes w/ CPS

Figure 2: Prototype comparison of FedProto with and without CPS for the CIFAR-10 dataset. The CPS dimension s set to 50 for (b) and (d).

where the function $\rho(\cdot, \cdot)$ computes the Euclidean distance between the local and global prototypes.

This process iterates until convergence, with clients continually updating their local models while maintaining communication efficiency through the exchange of prototypes.

3.2 Motivation

In a deep network, the feature layer (the penultimate layer) often exhibits sparse activation patterns under ReLU, leading to the emergence of dead units. Dead units are hidden units that consistently output zero across all training samples (Lu et al. 2019; Choi et al. 2021), thereby eliminating their contribution to learning and inference. This sparsity extends to class-level activation patterns, where certain units consistently remain inactive across all samples from specific classes, yielding sparse class prototypes. Figure 2a illustrates this sparse activation phenomenon through a heatmap of 500-dimensional local prototypes across 10 clients and the global prototype for class #2 in CIFAR-10. Colored areas indicate non-zero values, while blank areas represent zeros. Several clients (L1, L3, L4) exhibit partial prototype utilization, while L6 and L7 display full utilization. Notably, the others display zero prototypes due to the absence of class #2 in their local datasets.

Despite this sparse activation phenomenon, deep networks maintain high performance due to their substantial capacity and robust generalization capabilities (Arpit et al. 2017; Zhang et al. 2021a). This suggests that sparse prototypes could reduce server-client communication costs without performance degradation if sparse locations were consistent across clients. However, model and data heterogeneity cause sparse locations to vary among clients, as demonstrated in Figure 2a. While individual local prototypes are sparse, their union spans all dimensions, as shown in row 'G' of Figure 2a. Consequently, the aggregated global prototype utilizes the full dimensionality (Figure 2c), resulting in communication inefficiency. To realize the communication benefits of sparse prototypes, we propose a method that coordinates sparse locations across clients.

4 Methods

This section presents two components of TinyProto and concludes with its integration algorithm for PBFL approaches.

4.1 Class-Wise Prototype Sparsification

The CPS uses binary masks to impose class-specific sparsity patterns on prototypes. It employs random allocation of sparse locations for each mask while optimizing for maximum inter-class Hamming distance separation between masks. Let $\boldsymbol{m}_j = (m_{j,1}, m_{j,2}, \ldots, m_{j,d}) \in \{0,1\}^d$ be a mask vector and $\bar{\boldsymbol{c}}_j = (\bar{c}_{j,1}, \bar{c}_{j,2}, \ldots, \bar{c}_{j,d}) \in \mathbb{R}^d$ be a local or global prototype of class j. For notational simplicity, we omit class-specific subscripts j from $\boldsymbol{m}_j$ and $\bar{\boldsymbol{c}}_j$ when the context is clear. Through these masks, only a subset of elements of prototypes with mask values of 1 are selected and communicated. Mask vectors are distributed once at the beginning of FL rounds and remain fixed.

We now formally define prototype forms that are used in TinyProto with mask vectors: the structured sparse prototype and the compressed prototype.

Definition 1 (Structured Sparse Prototype)**.** *A structured sparse prototype $\tilde{\boldsymbol{c}} \in \mathbb{R}^d$ is obtained by applying the sparsification operator $S : \mathbb{R}^d \to \mathbb{R}^d$ to an original prototype $\bar{\boldsymbol{c}} \in \mathbb{R}^d$ with a binary mask $\boldsymbol{m} \in \{0,1\}^d$ as:*

$$\tilde{\boldsymbol{c}} = S(\bar{\boldsymbol{c}}; \boldsymbol{m}) = \bar{\boldsymbol{c}} \odot \boldsymbol{m}, \tag{7}$$

where $\odot$ denotes the Hadamard product.

Definition 2 (Compressed Prototype)**.** *The compressed prototype $\hat{\boldsymbol{c}} \in \mathbb{R}^s$ is obtained by applying the compression operator $C : \mathbb{R}^d \to \mathbb{R}^s$ to an original prototype $\bar{\boldsymbol{c}} \in \mathbb{R}^d$ with a binary mask $\boldsymbol{m} \in \{0,1\}^d$ as:*

$$\hat{\boldsymbol{c}} = C(\bar{\boldsymbol{c}}; \boldsymbol{m}) = (\bar{c}_i : m_i = 1), \tag{8}$$

where $s = \sum_{i=1}^{d} m_i$ is the number of non-zero elements.

In Definition 2, the CPS dimension s is a predetermined parameter that controls the degree of prototype compression. Rather than transmitting complete prototypes ($\bar{\boldsymbol{c}}$), compressed prototypes ($\hat{\boldsymbol{c}}$) are exchanged between the server and clients, corresponding to the colored dimensions where mask values are 1s, as illustrated in Figures 2b and 2d.

Figure 3 demonstrates the complete TinyProto process flow, showing how CPS reduces communication overhead using these modified prototypes from Definitions 1 and 2. Consider a 5-class classification example with 5-dimensional prototypes across M clients, where mask vectors are $\boldsymbol{m}_1 = (1,0,0,0,0)$, $\boldsymbol{m}_2 = (0,1,0,0,0)$, …, $\boldsymbol{m}_5 = (0,0,0,0,1)$. Then, ① Each client trains its local model with regularization and generates local prototypes. ②–③ Local prototypes ($\bar{\boldsymbol{c}}^L$) are compressed according to their respective masks and transmitted to the server. ④–⑤ The server aggregates the compressed prototypes ($\hat{\boldsymbol{c}}^L$) and distributes the compressed global prototypes ($\hat{\boldsymbol{c}}^G$) back to clients. ⑥ Clients reconstruct the compressed global prototypes into structured sparse prototypes ($\tilde{\boldsymbol{c}}^G$), which then guide local prototype sparsification during subsequent training rounds, as formalized in Eq. (6).

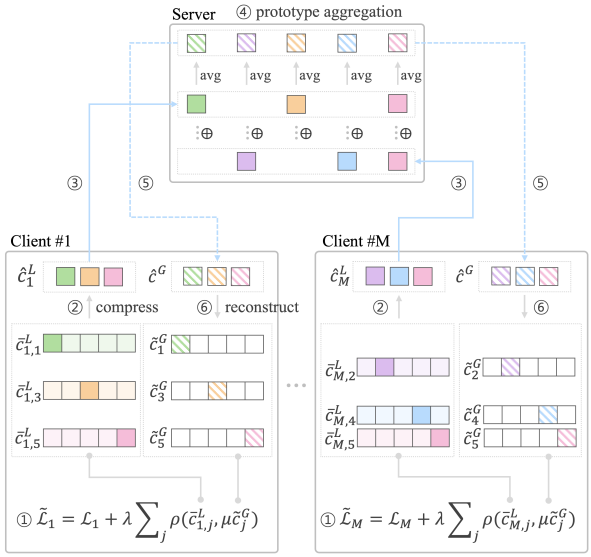


Figure 3: Process flow of TinyProto illustrated using a 5-class classification example with 5-dimensional prototypes and M clients.

Selection of CPS Dimension s In resource-constrained networks, s can be determined by available bandwidth constraints. This constraint-driven approach proves practically viable, as our empirical evaluation demonstrates that TinyProto outperforms the best baselines across various values of s, with detailed results in Section 5. This robustness enables flexible deployment across varying resource constraints without compromising performance.

Convergence Analysis of CPS Our convergence analysis builds upon FedProto (Tan et al. 2022). Theorem 2 in FedProto establishes its non-convex convergence rate under specific hyperparameter selections. By extending Lemma 2 of FedProto, we show that CPS achieves the same convergence rate. FedProto's Lemma 2 proves that the loss function for any client can be bounded after server-side prototype aggregation. To adapt this lemma to our framework, we introduce an additional assumption: the sparsification operator S (Definition 1) must maintain a consistent sparsity pattern per class across iterations and satisfy non-expansiveness. This extension preserves the lemma's bounded conditions for CPS, maintaining the convergence guarantees from FedProto's Theorem 2 within our framework. The complete details of this additional assumption and modified lemma are provided in the appendix.

4.2 Adaptive Prototype Scaling

In this subsection, we address two critical challenges when CPS is applied to PBFL. First, while global prototypes can be computed using Eq. (4) to incorporate each client's contribution in PBFL, this approach requires transmitting local class sample size $n_{i,j}$ to compute $\sum_{i=1}^{M} n_{i,j}$, which raises privacy concerns (Zhang et al. 2024). Therefore, simple averaging is commonly adopted in practice to avoid exposing

this sensitive information. For instance, the implementations in (Tan et al. 2022) and (Zhang et al. 2024) adopt:

$$\bar{\boldsymbol{c}}_j^G = \frac{1}{|\mathcal{N}_j|} \sum_{i \in \mathcal{N}_j} \bar{\boldsymbol{c}}_{i,j}^L, \tag{9}$$

where $\mathcal{N}_j$ represents the set of clients containing class j. Second, when implemented using the simple averaging approach (Eq. (9)), PBFL methods often underperform compared to existing HtFL approaches, as they fail to reflect the class-specific importance of each client, denoted by $n_{i,j}$. This parallels FedAvg's performance degradation when simple averaging omits n_i during aggregation, failing to account for the importance of each client.

To address these issues, we propose the APS method, a straightforward yet effective prototype scaling method that improves performance without directly transmitting $n_{i,j}$ to the server. The method scales local prototypes with $n_{i,j}$ and rescales global prototypes with an appropriate scaling factor.

Local Prototype Scaling APS scales local prototypes by $n_{i,j}$ to preserve each client's importance in federated learning as in Eq. (4). Instead of transmitting both $n_{i,j}$ and $\hat{\boldsymbol{c}}_{i,j}^L$ separately, the method sends the scaled compressed prototype $n_{i,j}\hat{\boldsymbol{c}}_{i,j}^L$, computed on the client side. This approach prevents privacy leakage by obfuscating $n_{i,j}$ during communication and at the server level. At the server, the global prototype for each class is simply computed as the mean of the received scaled prototypes, like Eq. (9):

$$\hat{\boldsymbol{c}}_j^G = \frac{1}{|\mathcal{N}_j|} \sum_{i \in \mathcal{N}_j} n_{i,j}\hat{\boldsymbol{c}}_{i,j}^L. \tag{10}$$

Global Prototype Scaling APS rescales the global prototypes using a hyperparameter-defined scaling constant to ensure stable training of local models. This adjustment mitigates potential stability issues caused by the varying scales of prototypes across clients and classes. The adjusted prototype regularization term for TinyProto is then defined as:

$$\mathcal{R}_i = \sum_j \rho(\bar{\boldsymbol{c}}_{i,j}^L, \mu\tilde{\boldsymbol{c}}_j^G), \tag{11}$$

where μ represents the scaling constant, and $\tilde{\boldsymbol{c}}_j^G$ denotes the structured sparse prototype for class j reconstructed from compressed global prototype $\hat{\boldsymbol{c}}_j^G$.

Despite its simplicity, APS achieves significant performance improvements, as demonstrated by our empirical results in Section 5. While Zhang et al. (2024) highlights that FedTGP avoids using $n_{i,j}$ during aggregation to address privacy concerns, our experimental results reveal that APS securely delivers significant performance improvements for FedTGP as well. Details on its application to FedTGP are provided in the appendix.

4.3 TinyProto Algorithm

The strength of TinyProto lies in its compatibility with existing PBFL algorithms. CPS can be integrated into FedProto by creating mask vectors for prototype sparsification and reconstruction, while APS can be applied by weighting local prototypes with $n_{i,j}$. Both techniques can be seamlessly extended to other PBFL methods, such as FedTGP. The integration process is detailed in Algorithm 1.

Algorithm 1: TinyProto-FP

Input: $\mathcal{D}_i$, $\boldsymbol{w}_i$, $i = 1, ..., M$
Server executes:

Initialize masking vector set $\{\boldsymbol{m}_j\}$ and transmit it to clients.
for iteration $t = 1, \ldots, T$ **do**
  Sample a client subset $\mathcal{C}^t$
  for client $i \in \mathcal{C}^t$ in parallel **do**
    $n_{i,j}\hat{\boldsymbol{c}}_{i,j}^L \leftarrow \text{LocalUpdate}\big(i, \hat{\boldsymbol{c}}_j^G\big)$
  Update $\hat{\boldsymbol{c}}_j^G$ with Eq. (10)

LocalUpdate$\big(i, \hat{\boldsymbol{c}}_j^G\big)$:

Reconstruct $\tilde{\boldsymbol{c}}_j^G$ from $\hat{\boldsymbol{c}}_j^G$
for each local epoch **do**
  for batch $(x_i^{(l)}, y_i^{(l)}) \in \mathcal{D}_i$ **do**
    Update model using the loss in Eq. (5) and (11)
Compute $\bar{\boldsymbol{c}}_{i,j}^L$ by Eq. (3) and convert it to $\hat{\boldsymbol{c}}_{i,j}^L$
return $n_{i,j}\hat{\boldsymbol{c}}_{i,j}^L$

5 Experiments

This section details the experimental setup, evaluations, and analysis. Our code is publicly available at: https://github.com/regulationLee/TinyProto

5.1 Experimental Setup

Datasets and Models We conducted our experiments on five datasets widely used for federated learning: CIFAR-10/100 (Krizhevsky 2009), GTSRB-43 (Houben et al. 2013), Flowers-102 (Nilsback and Zisserman 2008), and Tiny ImageNet-200 (Le and Yang 2015). Each dataset was divided into training (75%) and test (25%) subsets. To simulate non-IID data distributions across clients, we partitioned the data using a Dirichlet distribution (Dir(α)) with $\alpha = 0.1$ (Lin et al. 2020). We employed four lightweight models designed for resource-constrained devices, including ResNet-8 (Zhong et al. 2017), EfficientNet (Tan 2019), ShuffleNet v2 (Ma et al. 2018), and MobileNet v2 (Sandler et al. 2018). Each model integrates a global average pooling layer, with the default feature dimension fixed at $d = 500$.

Federated Learning Configuration Our federated learning setup involved 20 clients, all of whom actively participated in each of the 300 communication rounds. The client-side training configuration consisted of a learning rate of 0.01, a batch size of 32, and one local training epoch per round. We evaluated TinyProto integrated into FedProto and FedTGP (referred to as TinyProto-FP and TinyProto-FT), against six data-free federated learning algorithms: LG-FedAvg (Liang et al. 2020), FML (Shen et al. 2020), FedKD (Wu et al. 2022), FedDistill (Jeong et al. 2018), FedProto (Tan et al. 2022), and FedTGP (Zhang et al. 2024). For the prototype regularization term, we set $\lambda = 1$ following FedProto. The default CPS dimension was set to 50, corresponding to a 90% compression rate.

Algorithm	CIFAR-10		GTSRB-43		CIFAR-100		Flowers-102		Tiny ImageNet-200	
	Accuracy	Cost	Accuracy	Cost	Accuracy	Cost	Accuracy	Cost	Accuracy	Cost
LG-FedAvg	87.81 ± 0.23	0.20	95.82 ± 0.18	0.86	45.00 ± 0.32	2.00	50.56 ± 0.18	2.04	26.12 ± 0.27	4.00
FML	87.60 ± 0.01	34.32	95.15 ± 0.15	35.03	43.73 ± 0.11	36.12	46.47 ± 0.21	36.21	24.80 ± 0.21	38.12
FedKD	88.11 ± 0.01	30.66	95.68 ± 0.59	31.29	44.66 ± 0.30	32.26	49.00 ± 0.20	32.34	26.44 ± 0.17	34.05
FedDistill	87.91 ± 0.11	<0.01	96.45 ± 0.43	0.05	44.38 ± 0.45	0.29	51.58 ± 0.11	0.28	25.65 ± 0.16	1.17
FedProto	86.02 ± 0.38	0.15	88.00 ± 0.49	0.60	40.72 ± 0.84	1.46	39.85 ± 0.13	1.37	17.82 ± 0.07	2.93
FedTGP	87.62 ± 0.13	0.15	96.99 ± 0.60	0.60	45.30 ± 0.53	1.46	51.97 ± 0.15	1.37	24.70 ± 0.13	2.93
TinyProto-FP	86.20 ± 0.08	0.02	85.84 ± 0.59	0.06	40.39 ± 0.34	0.15	36.25 ± 0.10	0.14	20.06 ± 0.20	0.29
TinyProto-FT	$\mathbf{89.06 \pm 0.11}$	**0.02**	$\mathbf{97.68 \pm 0.10}$	**0.06**	$\mathbf{47.85 \pm 0.30}$	**0.15**	$\mathbf{55.23 \pm 0.13}$	**0.14**	$\mathbf{29.59 \pm 0.24}$	**0.29**

Table 1: Classification test accuracy (%) and communication cost across datasets with data heterogeneity $\alpha = 0.1$. Communication cost is measured by the number of parameters (in millions) shared per FL round. TinyProto-FP and TinyProto-FT denote TinyProto integrated into FedProto and FedTGP, respectively. The CPS dimension is set to 50.

Algorithm	Comm. cost formulation	Feature dimension		Scalability	Data heterogeneity	
		$d = 100$	$d = 1000$	$M = 50$	$\alpha = 0.01$	$\alpha = 0.5$
LG-FedAvg	$\sum_{i=1}^{M} \lvert\boldsymbol{\phi}_i\rvert \times 2$	43.74 ± 0.21	45.03 ± 0.15	45.00 ± 0.32	69.98 ± 0.22	25.59 ± 0.15
FML	$M \times (\lvert\boldsymbol{\theta}_{aux}\rvert + \lvert\boldsymbol{\phi}_{aux}\rvert) \times 2$	42.35 ± 0.18	43.66 ± 0.22	43.73 ± 0.11	69.11 ± 0.13	24.39 ± 0.39
FedKD	$M \times (\lvert\boldsymbol{\theta}_{aux}\rvert + \lvert\boldsymbol{\phi}_{aux}\rvert) \times 2 \times r$	43.95 ± 0.19	45.03 ± 0.17	44.66 ± 0.30	70.12 ± 0.20	25.62 ± 0.33
FedDistill	$\sum_{i=1}^{M}(K_i + K) \times K$	43.40 ± 0.25	43.91 ± 0.19	44.38 ± 0.45	70.16 ± 0.11	24.13 ± 0.05
FedProto	$\sum_{i=1}^{M}(K_i + K) \times d$	38.91 ± 0.41	40.37 ± 0.38	40.72 ± 0.84	64.85 ± 0.20	22.31 ± 0.30
FedTGP	$\sum_{i=1}^{M}(K_i + K) \times d$	44.30 ± 0.20	42.92 ± 0.24	45.30 ± 0.53	70.99 ± 0.31	24.11 ± 0.25
TinyProto-FP	$\sum_{i=1}^{M}(K_i + K) \times s$	39.60 ± 0.33	41.02 ± 0.29	40.39 ± 0.31	68.24 ± 0.34	19.46 ± 0.50
TinyProto-FT	$\sum_{i=1}^{M}(K_i + K) \times s$	$\mathbf{47.60 \pm 0.27}$	$\mathbf{47.32 \pm 0.31}$	$\mathbf{47.85 \pm 0.40}$	$\mathbf{74.35 \pm 0.52}$	$\mathbf{26.68 \pm 0.21}$

Table 2: Communication cost formulation and classification test accuracy (%) on CIFAR-100 under varying feature dimension, client scalability, and data heterogeneity. For FML and FedKD, $|\boldsymbol{\theta}_{aux}|$ and $|\boldsymbol{\phi}_{aux}|$ represent auxiliary feature extractor and classifier parameter sizes, with FedKD using dimensionality reduction factor r.

Selection of Hyperparameter μ We determined μ through systematic parameter selection based on the number of classes and data points: 1.5×10^{-4} for CIFAR-10, 5.0×10^{-4} for GTSRB-43, 5.0×10^{-3} for Flowers-102, and 1.5×10^{-3} for CIFAR-100 and Tiny ImageNet-200. The selection method is detailed in the appendix.

Evaluation Protocol For PBFL methods (FedProto, FedTGP, and TinyProto), predictions $\hat{y}$ were determined by finding the class with minimum ℓ_2-distance between the feature vector $f_i(\boldsymbol{\theta}_i; x)$ and local prototypes $\bar{\boldsymbol{c}}_{i,j}^{L}$ (Tan et al. 2022):

$$\hat{y} = \arg\min_{j} \| f_i(\boldsymbol{\theta}_i; x) - \bar{\boldsymbol{c}}_{i,j}^{L} \|_2. \tag{12}$$

The evaluation metric is the average test accuracy across all clients per round. We reported results from the best-performing global round, following standard practice in federated learning research (Zhang et al. 2024). Each experiment was repeated three times with different random seeds, and the average results were reported to ensure statistical robustness. No hyperparameter schedulers were applied during training to maintain fairness. Detailed configurations, including hyperparameter settings and additional results, are provided in the appendix.

5.2 Experimental Results

Communication Cost Reduction Communication costs across different approaches are compared in Table 1, with corresponding cost formulations—defined as the number of parameters transmitted per FL round—detailed in Table 2. TinyProto-FT ($s = 50$) achieved substantial communication reductions of up to $10\times$ and $4\times$ compared to existing PBFL methods and FedDistill, the most communication-efficient baseline, respectively (underlined). FedDistill achieved the lowest communication costs on datasets with few classes, such as CIFAR-10, although the advantage over TinyProto is marginal. FedDistill incurs trade-offs: it introduces higher privacy risks due to logit transmission and is incompatible with CPS-style compression owing to its dense logit requirements. TinyProto-FT demonstrated consistently superior performance across various configurations, as indicated by bold entries in Tables 1 and 2.

Communication Cost Scaling with Class Count and Feature Dimensions We analyzed how communication costs scale with class count and feature dimensions in Tables 1 and 2. As expected, all algorithms generally exhibit increased communication costs as class count grows due to the expansion of output layer neurons (Table 1). However, TinyProto with CPS dimension $s = 50$ demonstrates supe-

rior scaling behavior, showing minimal cost increases compared to FedProto and FedTGP. For feature dimension scaling from 500 to 100 and 1000, FedDistill shows no cost variation, as it depends solely on the class count. While LG-FedAvg and PBFL methods exhibit cost sensitivity to increasing feature dimensions, TinyProto maintains consistently high performance and low communication overhead by applying CPS.

Client Scalability and Data Heterogeneity The robustness of our approach was further validated in Table 2. When increasing client count (M) from 20 to 50, our method maintained superior performance with significantly lower communication costs. Our method consistently outperformed baselines across various data distributions, ranging from more heterogeneous ($\alpha = 0.01$) to more homogeneous ($\alpha = 0.5$), compared to the default setting ($\alpha = 0.1$).

Text Dataset Evaluation To evaluate TinyProto's generalizability across different modalities, we conducted experiments on the AG News dataset using FastText and TextCNN architectures. Table 3 presents results for the top-4 performing algorithms, showing both classification accuracy and communication cost. TinyProto-FT achieved the highest accuracy while maintaining the lowest communication cost, demonstrating its effectiveness across different data modalities beyond image classification.

Algo.	LG-FedAvg	FedKD	FedTGP	TinyProto-FT
Acc.	97.26 ± 0.05	97.20 ± 0.02	97.23 ± 0.07	$\mathbf{97.39 \pm 0.02}$
Cost	0.080	2.100	0.063	**0.006**

Table 3: Classification accuracy (Acc., %) and communication cost (Cost) comparison on AG News. Communication cost is quantified as the number of parameters (in millions) transmitted per FL round. The CPS dimension is set to 50.

5.3 Analysis

Ablation Study To evaluate the distinct contributions of CPS and APS components, we carried out ablation experiments. The results in Tables 1 and 2 show that TinyProto consistently outperforms the baselines FedProto and FedTGP, which incorporate neither CPS nor APS. TinyProto-FT's ablation study in Table 4 further confirms that APS is a critical component for improving performance, while CPS does not harm performance when combined with APS, despite a $10\times$ reduction in communication cost. Moreover, their combined use was essential for peak performance.

CPS	APS	CIFAR-10	CIFAR-100	Tiny ImageNet-200
✗	✗	87.62	45.30	24.70
✓	✗	87.68	43.51	22.73
✗	✓	88.69	47.27	28.40
✓	✓	**89.06**	**47.85**	**29.59**

Table 4: Ablation results for TinyProto-FT ($s = 50$).

Performance vs. Compression Trade-off Figure 4 demonstrates TinyProto-FT's trade-off between compression rate and performance across varying CPS dimensions. TinyProto-FT (circular markers) outperformed the best-performing baselines (red dashed lines) from Table 1 at all compression rates and remains robust across different CPS dimensions.

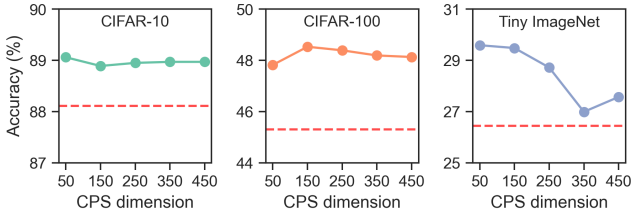
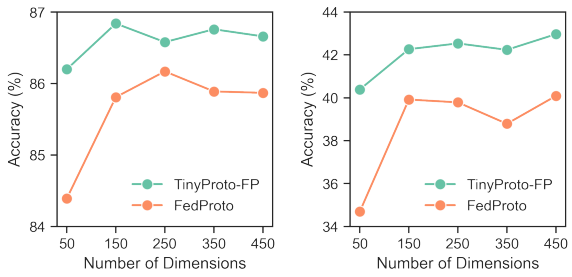


Figure 4: Compression-accuracy trade-off for TinyProto-FT.

CPS Dimension vs. Feature Dimension Figure 5 presents a comparative analysis of CPS-based sparsification in TinyProto-FP versus neuron count adjustment in FedProto under equivalent communication constraints. TinyProto-FP consistently demonstrated superior performance, attributed to its preservation of full representational capacity throughout the training process, while restricting sparsification exclusively to the communication phase. In contrast, neuron reduction imposes permanent constraints on model expressiveness, resulting in degraded performance.



(a) CIFAR-10 (b) CIFAR-100

Figure 5: Classification accuracy comparison between TinyProto-FP with varying CPS dimension s and FedProto with varying feature dimensions d.

6 Conclusion

The TinyProto framework leverages structured sparsity and adaptive prototype scaling to address the inherent scalability challenges of PBFL. Our extensive experiments demonstrate that TinyProto achieves a reduction of up to $10\times$ in communication costs compared to baselines, while maintaining or improving accuracy across diverse datasets. Beyond these efficiency gains, TinyProto offers distinct advantages over existing communication-efficient FL methods: it eliminates the need for additional computation, such as fine-tuning after model pruning, and provides support for heterogeneous models. These characteristics make TinyProto particularly well-suited for real-world deployments involving resource-constrained and heterogeneous clients.

7 Acknowledgments

This research was supported by Seoul R&BD Program (VC240022) through the Seoul Business Agency (SBA) funded by Seoul Metropolitan Government and supported by the Technology Incubator Program for Startup (TIPS) under Grant No. RS-2023-00271451, funded by the Ministry of SMEs and Startups (MSS), Republic of Korea.

References

Arpit, D.; Jastrzebski, S.; Ballas, N.; Krueger, D.; Bengio, E.; Kanwal, M. S.; Maharaj, T.; Fischer, A.; Courville, A.; Bengio, Y.; et al. 2017. A closer look at memorization in deep networks. In *International conference on machine learning*, 233–242. PMLR.

Chen, H.-Y.; and Chao, W.-L. 2021. On bridging generic and personalized federated learning for image classification. *arXiv preprint arXiv:2107.00778*.

Choi, D.; Lee, K.; Hwang, D.; and Rhee, W. 2021. Statistical characteristics of deep representations: An empirical investigation. In *International conference on artificial neural networks*, 43–55. Springer.

Han, P.; Wang, S.; and Leung, K. K. 2020. Adaptive gradient sparsification for efficient federated learning: An online learning approach. In *2020 IEEE 40th international conference on distributed computing systems (ICDCS)*, 300–310. IEEE.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

Houben, S.; Stallkamp, J.; Salmen, J.; Schlipsing, M.; and Igel, C. 2013. Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, 1288.

Jeong, E.; Oh, S.; Kim, H.; Park, J.; Bennis, M.; and Kim, S.-L. 2018. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *arXiv preprint arXiv:1811.11479*.

Jiang, Y.; Wang, S.; Valls, V.; Ko, B. J.; Lee, W.-H.; Leung, K. K.; and Tassiulas, L. 2022. Model pruning enables efficient federated learning on edge devices. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12): 10374–10386.

Karimireddy, S. P.; Kale, S.; Mohri, M.; Reddi, S.; Stich, S.; and Suresh, A. T. 2020. Scaffold: Stochastic controlled averaging for federated learning. In *International conference on machine learning*, 5132–5143. PMLR.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Le, Y.; and Yang, X. 2015. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7): 3.

Li, A.; Sun, J.; Zeng, X.; Zhang, M.; Li, H.; and Chen, Y. 2021. Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 42–55.

Li, D.; and Wang, J. 2019. FedMD: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*.

Liang, P. P.; Liu, T.; Ziyin, L.; Allen, N. B.; Auerbach, R. P.; Brent, D.; Salakhutdinov, R.; and Morency, L.-P. 2020. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523*.

Lin, T.; Kong, L.; Stich, S. U.; and Jaggi, M. 2020. Ensemble distillation for robust model fusion in federated learning. *Advances in neural information processing systems*, 33: 2351–2363.

Lu, L.; Shin, Y.; Su, Y.; and Karniadakis, G. E. 2019. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*.

Ma, N.; Zhang, X.; Zheng, H.-T.; and Sun, J. 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, 116–131.

McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; and y Arcas, B. A. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, 1273–1282. PMLR.

Nguyen, D. C.; Ding, M.; Pathirana, P. N.; Seneviratne, A.; Li, J.; and Poor, H. V. 2021. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3): 1622–1658.

Nilsback, M.-E.; and Zisserman, A. 2008. Automated Flower Classification over a Large Number of Classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520.

Sattler, F.; Wiedemann, S.; Müller, K.-R.; and Samek, W. 2019. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, 31(9): 3400–3413.

Shen, T.; Zhang, J.; Jia, X.; Zhang, F.; Huang, G.; Zhou, P.; Kuang, K.; Wu, F.; and Wu, C. 2020. Federated mutual learning. *arXiv preprint arXiv:2006.16765*.

Stich, S. U.; Cordonnier, J.-B.; and Jaggi, M. 2018. Sparsified SGD with memory. *Advances in neural information processing systems*, 31.

Tan, M. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.

Tan, Y.; Long, G.; Liu, L.; Zhou, T.; Lu, Q.; Jiang, J.; and Zhang, C. 2022. Fedproto: Federated prototype learning across heterogeneous clients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 8432–8440.

Wang, H.; Liu, X.; Niu, J.; Guo, W.; and Tang, S. 2024. Why Go Full? Elevating Federated Learning Through Partial Network Updates. arXiv:2410.11559.

Wang, H.; Liu, X.; Niu, J.; and Tang, S. 2023. SVDFed: Enabling Communication-Efficient Federated Learning via Singular-Value-Decomposition. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, 1–10. IEEE.

Wang, J.; and Joshi, G. 2019. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD. *Proceedings of Machine Learning and Systems*, 1: 212–229.

Wang, S.; Tuor, T.; Salonidis, T.; Leung, K. K.; Makaya, C.; He, T.; and Chan, K. 2019. Adaptive federated learning in resource constrained edge computing systems. *IEEE journal on selected areas in communications*, 37(6): 1205–1221.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, 2074–2082.

Wu, C.; Wu, F.; Lyu, L.; Huang, Y.; and Xie, X. 2022. Communication-efficient federated learning via knowledge distillation. *Nature communications*, 13(1): 2032.

Wu, T.; Song, C.; and Zeng, P. 2023. Efficient federated learning on resource-constrained edge devices based on model pruning. *Complex & Intelligent Systems*, 9(6): 6999–7013.

Wu, X.; Liu, X.; Niu, J.; Wang, H.; Tang, S.; Zhu, G.; and Su, H. 2024. Decoupling general and personalized knowledge in federated learning via additive and low-rank decomposition. In *Proceedings of the 32nd ACM International Conference on Multimedia*, 7172–7181.

Zhang, C.; Bengio, S.; Hardt, M.; Recht, B.; and Vinyals, O. 2021a. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3): 107–115.

Zhang, C.; Xie, Y.; Bai, H.; Yu, B.; Li, W.; and Gao, Y. 2021b. A survey on federated learning. *Knowledge-Based Systems*, 216: 106775.

Zhang, H.; Liu, J.; Jia, J.; Zhou, Y.; Dai, H.; and Dou, D. 2022. Fedduap: Federated learning with dynamic update and adaptive pruning using shared data on the server. *arXiv preprint arXiv:2204.11536*.

Zhang, J.; Liu, Y.; Hua, Y.; and Cao, J. 2024. Fedtgp: Trainable global prototypes with adaptive-margin-enhanced contrastive learning for data and model heterogeneity in federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 16768–16776.

Zhang, Y.; Xiang, T.; Hospedales, T. M.; and Lu, H. 2018. Deep mutual learning. In *CVPR*.

Zhong, Z.; Li, J.; Ma, L.; Jiang, H.; and Zhao, H. 2017. Deep residual networks for hyperspectral image classification. In *2017 IEEE international geoscience and remote sensing symposium (IGARSS)*, 1824–1827. IEEE.

Zhu, Z.; Hong, J.; and Zhou, J. 2021. Data-Free Knowledge Distillation for Heterogeneous Federated Learning. In *ICML*.