FedMBridge: Bridgeable Multimodal Federated Learning

Jiayi Chen [1] **Aidong Zhang** [1]

Abstract

Multimodal Federated Learning (MFL) addresses the setup of multiple clients with diversified modality types (e.g. image, text, video, and audio) working together to improve their local personal models in a data-privacy manner. Prior MFL works rely on restrictive compositional neural architecture designs to ensure inter-client information sharing via blockwise model aggregation, limiting their applicability in the real-world **Architecture-personalized MFL** (**AMFL**) scenarios, where clients may have distinguished multimodal interaction strategies and there is no restriction on local architecture design. The key challenge in AMFL is how to automatically and efficiently tackle the two heterogeneity patterns–statistical and architecture heterogeneity–while maximizing the beneficial information sharing among clients. To solve this challenge, we propose **FedMBridge**, which leverages a topology-aware hypernetwork to act as a bridge that can automatically balance and digest the two heterogeneity patterns in a communication-efficient manner. Our experiments on four AMFL simulations demonstrate the efficiency and effectiveness of our proposed approach.

1. Introduction

Personalized Federated Learning (PFL) (Arivazhagan et al., 2019; T Dinh et al., 2020; Deng et al., 2020) provides a collaborative training scheme, allowing multiple clients to train their personal models simultaneously while taking benefits from each other's learning experiences with a guarantee of data privacy. With recent advancements of multi-sensory devices, **Multimodal Federated Learning** (**MFL**) (Chen & Zhang, 2022; Zhao et al., 2021; Xiong et al., 2022) incorporating diverse modality types (e.g. image, video, text, and audio) into the PFL systems has become a significant research area. For example, in vision-centered MFL, clients could leverage various auxiliary modalities, such as text and audio, to boost the unimodal vision-only models; some clients learn only the image modality, some learn to fuse image and audio modalities, and some others may focus on modeling image-text interactions.

A key challenge in MFL is how to encourage knowledge sharing among clients with *diversified neural architectures* due to their inconsistent input modality types. To address this challenge, prior MFL approaches (Zhao et al., 2021; Chen & Zhang, 2022; Yu et al., 2023) have leveraged a compositional neural architecture design (i.e., neural architectures are made of common smaller blocks) for a blockwise knowledge sharing scheme. However, these methods require that all clients' models should be *splittable* into blocks as well as use the *same multimodal fusion flow*, which have ignored three real-world characteristics of MFL. **(1) Task complexity difference**. Due to personalization, clients may vary greatly in their *complexity* of modeling the inter-modal interactions, with some local tasks being more complex than others. In this situation, it is essential for different clients to employ diverse neural network sizes or even utilize varied network families that align with their personal data distributions. **(2) Multimodal pattern difference**. Studies in Multimodal Fusion (Atrey et al., 2010; Gao et al., 2020) have demonstrated that the *patterns* of inter-modal interactions can also shift across clients, necessitating distinct mechanisms for effective learning (Atrey et al., 2010; Gao et al., 2020). For example, some *image-text* clients may benefit from just a straightforward concatenation of different modalities' features; some image-audio clients may need an element-wise alignment; and some image-text-audio clients may need a complex intra- and inter-modal attention mechanism. **(3) Resource budget difference**. In real systems, client devices can be mobile phones, tablets, and personal computers, thus vary greatly in *computation resource budgets* (e.g., computation capacity, memory, storage, and network bandwidth). A client with poor resource cannot afford large-size models, such as pre-trained large foundation models, while a resource-rich client can benefit from it.

Motivated by the above practical MFL properties underexplored in prior works, it is natural to consider the coexistence of diverse multimodal fusion strategies, as well as

[1]Department of Computer Science, University of Virginia, Charlottesville, VA 22903, USA. Correspondence to: Jiayi Chen <jc4td@virginia.edu>, Aidong Zhang <aidong@virginia.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

different widths, depths, and architecture topologies across the clients in an MFL system. We name such MFL scenarios as **Architecture-personalized MFL** (**AMFL**), where different clients' neural networks might adopt *structurally distinct computational flows* and thus are *non-splittable* into global common blocks. As an analogy, AMFL can be likened to a social scenario where individuals who think in different ways to engage in efficient communication with each other. Different from traditional MFL that can be solved by manual block split, in AMFL we have no prior knowledge on how heterogeneous architectures are correlated with each other. Therefore, the unique **challenge** of AMFL, which is different from traditional MFL, is how to employ an *automatic way* to tackle the architecture gap during knowledge sharing among statistically-heterogeneous and architecture-heterogeneous clients.

Real-world AMFL systems have observed limitations of existing straightforward approaches in tackling this novel challenge. First, one might consider that using a globally-shared large language model to unify all different local tasks could be enough to overcome AMFL. However, the communication would be forbiddingly expansive and many local devices may not afford such an expansive model. Moreover, although some unimodal PFL approaches have explored the settings with diversified neural architectures, including feature-sharing methods (Ahmad & Aral, 2022; Zhu et al., 2021; Yang et al., 2021; Yao et al., 2023) and explicit parameter-sharing methods (Diao et al., 2021; Hong et al., 2022; Kim et al., 2023; Dai et al., 2022), these methods have shown to be *inefficient* in AMFL scenarios, since the diverse model topologies in AMFL either impose a costly workload on the server or require long time for uploading and downloading a large super-network at each communication round. Also, these methods struggle to maximize the beneficial knowledge sharing among clients, especially with significant topology differences among models.

Different from existing MFL/PFL works, we propose a novel *implicit parameter-sharing* PFL framework to solve the AMFL challenge, namely **FedMBridge**. The main idea is that, instead of sharing original weights across diverse weight spaces, we introduce a global *bridge function* that learns to perform knowledge sharing on a globally-shareable *latent* space. Intuitively, the bridge function is held on the server to act as a "bridge", which *balances and digests* the two disentangled heterogeneity patterns (i.e., statistical and architecture heterogeneity) and then *generates* local weights in the raw statistical-architectural entangled heterogeneity pattern. To achieve such a bridge function, we introduce a **Topology-Aware HyperNetwork** (**TAHN**), which is formulated as a two-stage process: the first stage encodes the implicit roles of each layer using graph neural networks, and the second stage aims to combine the layer-role information with task information to reconstruct local weights.

Our contributions are threefold: **(1)** We study an under-explored AMFL problem. To the best of our knowledge, this is one of the first works that tackle the collaboration of diversified multimodal fusion strategies for general-purpose federated AI systems. **(2)** We propose the FedMBridge framework to solve AMFL, where we introduce a brand-new Topology-aware HyperNetwork to automatically balance and digest architecture gap and statistical heterogeneity in an efficient manner. **(3)** We evaluate our approach on four AMFL simulations, which demonstrates the effectiveness of our approach to addressing AMFL.

2. Related Works

Personalized Federated Learning (PFL). PFL (Arivazhagan et al., 2019; T Dinh et al., 2020; Deng et al., 2020) is a learning paradigm that enables multiple clients to train their own *personal* models in parallel while *collaborating* with each other such that each local model can draw upon the learning experiences of others. PFL typically deals with the challenge of how to share valuable knowledge among client models who are trained from distinct local distributions, i.e., *statistical heterogeneity* or *Non-IIDness*. Considering what types of knowledge to be shared among clients, PFL methods can be categorized into two families: **(1)** *Feature-sharing PFL* employs the local models' *predictions or intermediate features* as the forms of knowledge shared among clients. For example, FedDistill (Jiang et al., 2020; Ahmad & Aral, 2022; Zhu et al., 2021), FML (Shen et al., 2020), KT-pFL (Zhang et al., 2021), and FedGKD (Yao et al., 2023) adopt collaborative knowledge distillation to align predictions among diverse client models on the server. **(2)** *Parameter-sharing PFL* employs the *weights or gradients* of local models as the forms of the knowledge exchanged between clients, which can be further divided into fine-tuning methods (Arivazhagan et al., 2019; T Dinh et al., 2020), meta-learning methods (Finn et al., 2017), factorization-based methods (Deng et al., 2020; Guo et al., 2021; Jeong & Hwang, 2022), multi-task learning methods (Smith et al., 2017; Chen & Zhang, 2022), and HyperNet-based methods (Shamsian et al., 2021). The two lines of research, Feature-sharing PFL and Parameter-sharing PFL, are orthogonal to each other and have their own distinct assumptions: while Feature-sharing PFL relies on an extra public dataset (Zhang et al., 2021) or additional data generators (Zhu et al., 2021; Zhang et al., 2022) to compute pseudo-supervision signals on the server, Parameter-sharing PFL typically assumes the symmetry of weight spaces across different clients to aggregate them at the server side.

Architecture-heterogeneous PFL. Recently, there has been significant attention towards adapting PFL to scenarios where local models have diversified structures and sizes, particularly in budget-limited or personalized-use contexts. In pursuit of addressing architecture heterogeneity,

Feature-sharing PFL frameworks have been widely adopted as they naturally bypass this challenge through knowledge distillation (see above for details). The drawback of these methods lies in their reliance on unrealistic and auxiliary data pipelines. Meanwhile, the research community within *Parameter-sharing PFL* also explores ways to address heterogeneity of client architectures. A stream of research employs super-networks, such as HeteroFL (Diao et al., 2021), Split-Mix (Hong et al., 2022), DisPFL (Dai et al., 2022), DepthFL (Kim et al., 2023), and FlexiFed (Wang et al., 2023), where each client is aligned to a smaller subset of a global super-network, using a local super-mask that are manually designed or computed through pruning techniques. Yet in our AMFL scenarios, these methods show inefficient and suboptimal communication–due to the significant variations in computational flows among local models, the masked weight aggregation referring to the supernet becomes costly and limits the knowledge-sharing opportunities among local weights. Few works (Litany et al., 2022) explored using hyper-networks to generalize different architectures in the federated context; they do not yet effectively address the joint statistical-architectural client heterogeneity and study only single-modality cases. *In contrast* to existing works, we introduce a more efficient and effective parameter-sharing scheme to handle joint statistical-architectural heterogeneity among multimodal clients.

Multimodal Federated Learning (MFL). MFL (Che et al., 2023; Lin et al., 2023; Barry et al., 2024) is a special case of Architecture-heterogeneous PFL that incorporates multiple modality types into the systems, where clients are equipped with different sensory devices requiring different architectures to process them. Existing MFL works have mainly employed global super-networks to serve both unimodal clients and clients employing various combinations of multiple modalities, including (Zhao et al., 2021; Chen & Zhang, 2022) that employ cross-modal alignment and (Xiong et al., 2022; Yu et al., 2023; Feng et al., 2023; Chen & Zhang, 2024) that utilize advanced mechanisms to fuse different modalities. However, the robustness of these methods relies on an assumption that all clients employ the same multimodal fusion strategy (e.g., feature alignment) in their model designs. *In contrast*, our work relaxes the use cases to diversified multimodal fusion strategies, which we believe reflect real-world situations (as argued in the Introduction), and we seek to automatically maximize the transfer of multimodal-interaction experiences among clients.

3. Problem Definition

Multimodal Federated Learning (MFL) (Zhao et al., 2021) addresses the setup of N **clients** with M **modality types** (e.g. image, video, text, and audio) working together to improve their **local personal models** $\theta_1, \theta_2, ..., \theta_N$. Each client $i \in \{1, 2, ..., N\}$ focuses on learning a subset of modality types $\mathcal{I}_i \subseteq \{1, 2...M\}$ and has a combinatorial *input space* $\mathcal{X}_{\mathcal{I}_i} := (\mathcal{U}^{(m)} | \forall m \in \mathcal{I}_i)$, where $\mathcal{U}^{(m)}$ is the subspace associated with the modality type m. For example, as illustrated in Figure 1(a), "client 2" focuses on an image-text bimodal task; "client 3" focuses on an audio-visual bimodal task; and "client 1" learns a text-only unimodal task. Each client i also has a personalized label space $\mathcal{Y}_i$. Each client i aims to obtain a **local mapping function** $f_{\mathcal{A}_i}(\cdot; \theta_i) : \mathcal{X}_{\mathcal{I}_i} \to \mathcal{Y}_i$ characterized by a *client-specific model architecture* $\mathcal{A}_i$ and parameterized by *trainable weights* $\theta_i \in \mathbb{R}^{d_i}$, where d_i indicates the structure of the weight space associated with $\mathcal{A}_i$.

Each client i has access *only* to its **local dataset** $\mathcal{D}_i = \{(\boldsymbol{x}_{ij}, y_{ij})\}_{j=1}^{n_i}$, sampled from $\boldsymbol{x}_{ij} \sim \mathcal{P}_i(\boldsymbol{x})$ and $y_{ij} \sim \mathcal{Q}_i(y|\boldsymbol{x}_{ij})$, where $\mathcal{P}_i$ is the client-specific input distribution over the combinatorial input space $\mathcal{X}_{\mathcal{I}_i}$, and $\mathcal{Q}_i$ is the conditional output distribution over the space $\mathcal{Y}_i$. Each sample's input consists of the modalities $\boldsymbol{x}_{ij} = (\boldsymbol{u}_{ij}^{(m)})_{m \in \mathcal{I}_i}$ present as in $\mathcal{I}_i$, where $\boldsymbol{u}_{ij}^{(m)}$ denotes the modality m in $\boldsymbol{x}_{ij}$. The **global objective** of MFL is formulated as

$$\min_{\theta_1, \theta_2, ..., \theta_N} \left[\frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i(\theta_i) \right] + \mathcal{R}(\theta_1, \theta_2, ..., \theta_N), \quad (1)$$

which aims to **(1)** jointly optimize the local objectives of all clients $\min_{\theta_i} \mathcal{L}_i(\theta_i) := \mathbb{E}_{(\boldsymbol{x}, y) \sim \mathcal{D}_i} l\left(y, f_{\mathcal{A}_i}(\boldsymbol{x}; \theta_i)\right)$, where $l(\cdot, \cdot)$ is the loss function, and meanwhile, **(2)** leverage a central server to encourage a privacy-preserving *knowledge sharing* scheme among clients $\mathcal{R}(\cdot)$ in order to boost each client's local model performance.

MFL is a problem that naturally suffers from network architecture heterogeneity among clients. Therefore, one of the main **challenges** of solving Eq.(1) is how to design and maximize the benefits of the inter-client knowledge sharing scheme $\mathcal{R}(\cdot)$, wherein there are simultaneous *architecture heterogeneity* ($\mathcal{A}_1 \neq \mathcal{A}_2 \neq ... \neq \mathcal{A}_N$) as well as *statistical heterogeneity* (Non-IIDness) among clients.

Definition 1 (Architecture-compositional MFL): When addressing the knowledge sharing among heterogeneous multimodal model architectures, traditional MFL systems typically leverage a restrictive design of *compositional* neural architectures: $\mathcal{A}_i := \{\mathcal{B}_{\text{enc}}^{(m)} | \forall m \in \mathcal{I}_i\} \cup \{\mathcal{B}_{\text{share}}\} \cup \{\mathcal{B}_{\text{dec},i}\}$, such that heterogeneous model architectures are *manually* split into smaller homogeneous *blocks*, allowing any pair of clients $\forall i, i' \in [N]$ share some common blocks:

$$\mathcal{A}_i \cap \mathcal{A}_{i'} = \{\mathcal{B}_{\text{enc}}^{(m)} | \forall m \in \mathcal{I}_i \cap \mathcal{I}_{i'}\} \cup \{\mathcal{B}_{\text{share}}\}, \quad (2)$$

as illustrated in Figure 1(b). Such design allows $\mathcal{R}(\theta_{1:N})$ to be achieved through *blockwise* weight sharing schemes, such as (Chen & Zhang, 2022; Zhao et al., 2021).

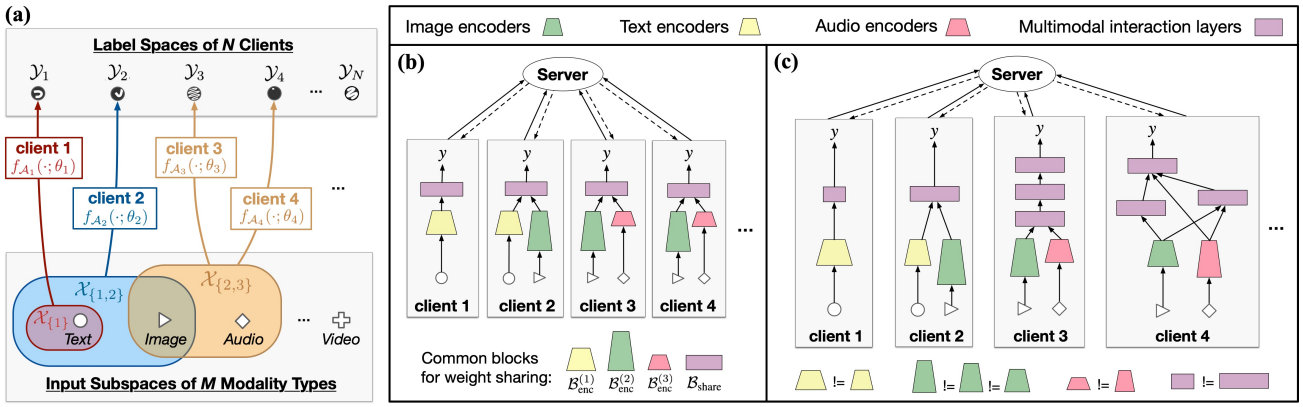


Figure 1. (a) Local mapping functions per client in Multimodal Federated Learning (MFL). (b) Problem setting of traditional MFL that requires restrictive compositional neural architectures. (c) Problem setting of **Architecture-personalized MFL (AMFL)**, without a restriction on local model architectures. In this AMFL example, "client 1" and "client 2" show a layer-*width* difference, "client 2" and "client 3" show a *depth* difference, and "client 3" and "client 4" show a *topology* difference, at multimodal interaction modules.

Definition 2 (Architecture-personalized MFL (AMFL)): We relax the traditional constraint Eq.(2), focusing on *more general* MFL scenarios without setting any restriction on the architecture design. $\mathcal{A}_i$ can be any neural architectures specified by local users, which is *non-compositional* so that the server has no prior knowledge about the inter-client weight-space sharing scheme. That is, for $\forall i, i' \in [N]$,

$$\mathcal{A}_i \cap \mathcal{A}_{i'} = \emptyset. \tag{3}$$

Given this relaxed design, there will be three particular **cases of architecture heterogeneity** that are not permitted in traditional MFL: **(1)** *Topology Difference* is a most common situation in multimodal FL systems. Two clients might use different model types (e.g., one client is based on Transformer while the other is based on ResNet) or use different multimodal fusion strategies for different input modality types (e.g., one client uses alignment while the other uses concatenation). **(2)** *Depth Difference* refers that two clients having the same topology (e.g., both are based on ResNet) but their numbers of layers/modules are different. **(3)** *Width Difference* describes a situation where two clients having the same topology and same depth, but their numbers of neurons at each layer are different. Examples of the three cases are illustrated in Figure 1(c). While traditional MFL does not address these cases of architecture heterogeneity, the **goal of AMFL** particularly deals with how to *automatically bridge the architecture gap*, for efficient and effective knowledge sharing among clients with heterogeneous neural architectures and distributions.

4. Methodology

In order to solve Eq.(1) under AMFL settings (Eq.(3)), we propose a new multimodal FL framework, **FedMBridge**, which *automatically bridges the architecture gap* among statistically heterogeneous clients. We will first introduce the main idea of FedMBridge and then present its three components: (1) the topological graph representation of local multimodal architectures; (2) the hypernetwork that generates personal weights conditioned on the topological graphs of architectures; and (3) the federated training workflow.

4.1. Main Idea of FedMBridge

Rethinking Implicit and Explicit Weight Sharing: Explicit weight sharing, or the simple weight aggregation within a globally-shared weight space, is seen in standard FL with homogeneous architectures (Li et al., 2019; Shamsian et al., 2021) or some Pruning-based FL methods with only width or depth differences (Jiang et al., 2022; Vahidian et al., 2021; Jiang et al., 2023). However, in AMFL, an *explicit* weight sharing is *not available* since a globally shared weight space does not even exist, especially if client models vary significantly in their topologies or depths. Alternatively, we explore an *implicit* weight sharing mechanism for AMFL: instead of sharing *original* weights across diverse weight spaces, we aim to perform knowledge sharing among clients within globally-shared *latent* space(s).

Definition 3 (Bridge Function): We propose an implicit weight sharing mechanism for AMFL by introducing a global "***bridge***" function $h(\cdot,\cdot;\phi)$, where $\phi \in \mathbb{R}^D$ is the trainable weights of the bridge function. The original locally-trained weights of N clients from diverse weight spaces $\theta_i \in \mathbb{R}^{d_i}, d_1 \neq d_2 \neq ... \neq d_N$, are re-parameterized as the output of the bridge function conditioned on two client-specific generative factors

$$\theta_i := h(\mathcal{A}_i, \boldsymbol{c}_i; \phi), \quad \forall i \in [N], \tag{4}$$

where the first generative factor $\mathcal{A}_i \in \mathcal{G}$ is the local neural architecture from a globally-shared *latent topology space* $\mathcal{G}$ and the second generative factor $\boldsymbol{c}_i \in \mathcal{T}$ represents the lo-

cal task from a globally-shared *latent task space* $\mathcal{T}$. While $\mathcal{G}$ manages only architectural heterogeneity, $\mathcal{T}$ manages only statistical heterogeneity. Intuitively, the bridge function h can be treated as a generative meta-learner that can digest two *disentangled* heterogeneity patterns to solve the raw *statistical-architectural entangled* heterogeneity pattern. We will show design details in the following sections.

4.2. Multimodal Neural Architectures as Graphs

The multimodal neural architecture $\mathcal{A}_i$ at each client i is represented as a **directed acyclic graph** structure:

$$\mathcal{A}_i := (\mathcal{V}_i, \mathcal{E}_i, \boldsymbol{Z}_i^{(0)}). \tag{5}$$

Each node $v \in \mathcal{V}_i$ stands for a *computational operator* f_v in the neural architecture. f_v can be either non-parametric (e.g., a concatenation operator) or parametric (e.g., a linear layer with weights of size 16×64). Edges $\mathcal{E}_i$ represents the *computational flow* of the neural architecture, where each edge $e_{v'\to v} \in \mathcal{E}_i$ indicates that the output of the operator $f_{v'}$ is the input of the operator f_v. Every node v is equipped with K types of *configuration or prior information* for the operator f_v, including layer types, layer levels, layer shapes, modality types or fusion stage, etc. The node feature matrix $\boldsymbol{Z}_i^{(0)} \in \mathbb{R}^{|V_i|\times K}$ holds such K configuration/prior information types for all operators in the graph.

Particularly, following standard multimodal learning (Zadeh et al., 2017; 2018b), the neural architectures of local models employ a *three-step* procedure. **(1) Unimodal Encoders**: At the first step, we employ unimodal encoders to extract modality-specific features. We consider multiple different architecture families (including ResNets, CNNs, MLPs, RNNs, and Small Transformers) simultaneously appearing across AMFL as unimodal encoders for different/same modality types. **(2) Multi-modality Interaction Module**: At the second step, we model the complex intra- and inter-modality interactions to effectively fuse the complementary information from multiple modalities. Since such interaction modes can be diverse across client tasks, herein, we allow AMFL to cover many existing multimodal fusion strategies, including concatenation, element-wise alignment, tensor fusion (Zadeh et al., 2017), low-rank fusion (Zadeh et al., 2018a), and so on. Figure 2 (top) shows three example clients in AMFL–client-1 fuses image and text using feature summation; client-2 combines audio and image using outer-product interactions; and client-3 uses cross-attention strategies to fuse the input audio and image. **(3) Personal Final Layer**: Each client's final layer handles client-specific decision making. In broader contexts, it can be a classifier, a value prediction head, or a data generator.

The graph defined in Eq.(5) is *constructed* by traversing the computational flow of the user-designed model architecture as follows. **(1) Collection of Nodes and Edges:** First, we borrow the ideas from DARTS (Differentiable Architecture Search) (Liu et al., 2018; Funoki & Ono, 2021) to gather nodes $\mathcal{V}_i$ and edges $\mathcal{E}_i$, by tracing the chain of backward gradients of variables. Specifically, a dummy multimodal input sample is fed to the model and undergoes forward function execution. After this, starting from the output variable's gradient function, we iteratively traverse the chain of gradient functions in reverse order. During the traversal, the gradient functions associated with parametric modules are gathered as *parametric nodes*, while other gradient functions, such as ConcatBackward and BmmBackward, are collected as *non-parametric nodes*. $|\mathcal{I}_i|$ *input nodes* are additionally attached to those operators with zero in-degrees, respectively. Directed *edges* are gathered along with the traversal. **(2) Construction of Node Features:** Second, we construct node features $\boldsymbol{Z}_i^{(0)}$ such that they provide sufficient contexts for learning the functionality role of each layer. In particular, we utilize $K = 7$ information types and each of them is categorical, including *branch* types ($k = 1$), *operator* types ($k = 2$), *layer level*s ($k = 3$), and parameter *shapes* ($k = 4, 5, 6, 7$). Branch types refer to which branch the operator f_v is located within the computational flow, which should be one of the unimodal branches or be the fusion branch. Operator types refer to how f_v transforms the input message it receives from previous nodes. For example, parametric node have operator types including "Linear/Conv weights", "bias", "layer normalization", and so on; non-parametric nodes have operator types such as "sum", "concatenation", "element-wise dot product", and "outer product". In addition, we enable each node to be aware of which knowledge granularity it learns, by computing its relative layer level within its parent branch. Furthermore, different parameter sizes for the same layer necessitate varying densities of message during weight generation, and therefore, the weight-tensor shapes of each parametric layer are considered as parts of nodes features. Specifically, the raw shape sizes, since they leads to learning sparsity, are ranked into several shape scales using a lookup table. We use a held-out shape scale "<HO>" for non-parametric nodes and input nodes.

4.3. Topology-aware HyperNetwork

As in Eq.(4), we propose to learn a bridge function that can jointly digest the two heterogeneity patterns (i.e., statistics and architecture heterogeneity). A challenge underlying this goal is that *how to balance and combine the two separate heterogeneity patterns*, such as which pattern is more crucial and whether there is any inter-pattern interactions.

We propose a **T**opology-**A**ware **H**yper**N**etwork (**TAHN**) to build such a bridge function. The key idea of TAHN is to encourage $h(\cdot, \cdot; \phi)$ to capture the **implicit roles of each layer** within the neural architecture, which are then combined with layer-invariant client-specific task information. This is inspired by an intuition that for a pair of layers from

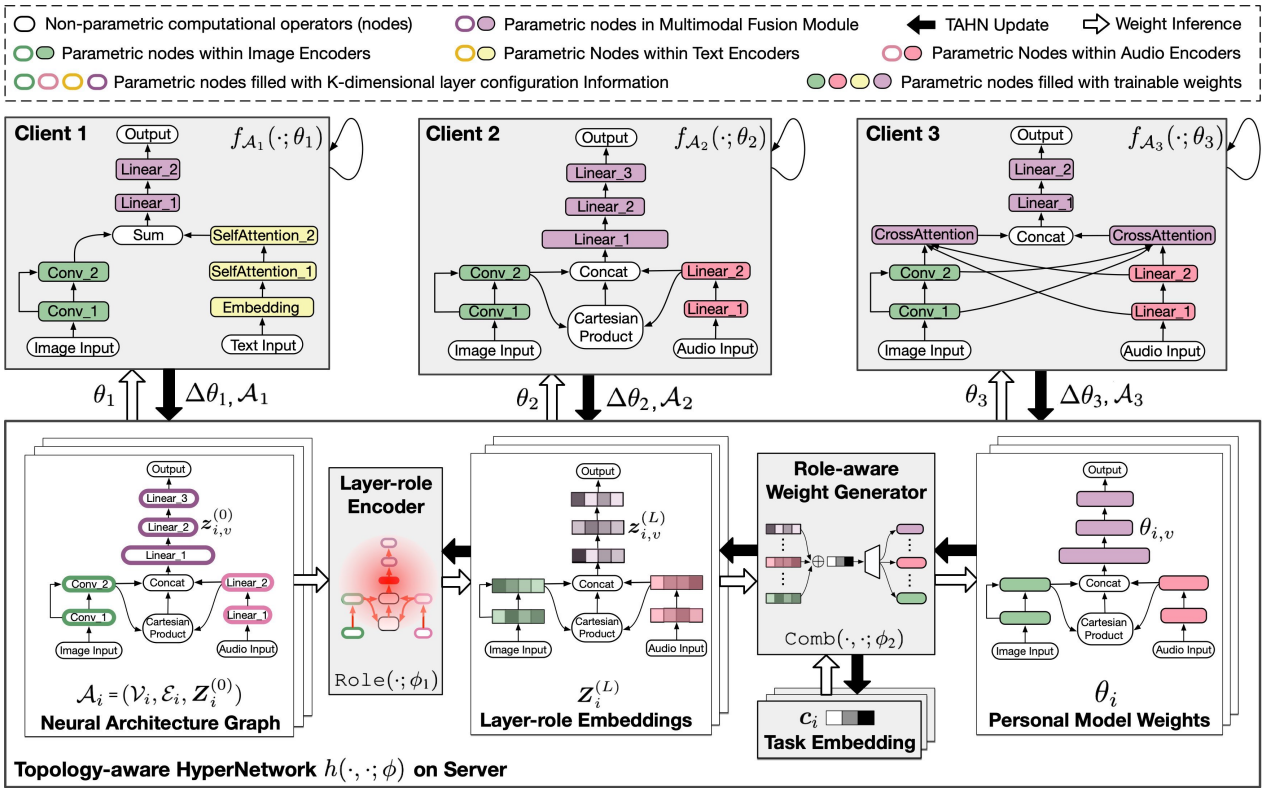


Figure 2. The proposed FedMBridge framework. The three example clients use different multimodal fusion strategies. The server holds the Topology-aware HyperNetwork (TAHN), a trainable bridge function for implicit weight sharing, which simultaneously overcomes the statistical heterogeneity and architecture heterogeneity among clients.

two different clients, if they act as similar roles within their models, would tend to have similar operations and weights.

Specifically, our TAHN consists of a **two-stage** process

$$h(\mathcal{A}_i, \boldsymbol{c}_i; \phi) = \texttt{Comb}\left(\boldsymbol{c}_i, \texttt{Role}(\mathcal{A}_i; \phi_1); \phi_2\right) \tag{6}$$

where the first stage $\texttt{Role}(\cdot;\phi_1)$ parameterized by ϕ_1 learns the implicit roles of layers such that layers across clients share a unified *layer-role embedding* space, and the second stage $\texttt{Comb}(\cdot,\cdot;\phi_2)$ parameterized by ϕ_2 aims to combine the two heterogeneity patterns and directly generates the weights. We represent $\phi = \{\phi_1, \phi_2\}$.

4.3.1. Stage One: Layer-role Encoder

In order to encode the implicit roles of layers, we consider two types of information. *First*, each layer's configuration information is important to determine the layer role. For example, if two layers from different architecture both are the convolutional layer and both are in the early level in the entire network, they tend to have similar filter and role during the computational flow. Such information is specified in $\boldsymbol{Z}_i^{(0)}$. *Second*, the position and contexts of each layer within the graphical computational flow is also important. For example, if two layers from different architectures are located in the same position in the same computational flow, they tend to have the same role. Such information is specified as the graphical structure $\mathcal{V}_i, \mathcal{E}_i$ of the computational flow.

The two types of information can be incorporated by applying a Graph Neural Network (GNN) on the neural architecture graph $\mathcal{A}_i = (\mathcal{V}_i, \mathcal{E}_i, \boldsymbol{Z}_i^{(0)})$, inspired from (Zhang et al., 2018; Knyazev et al., 2021; Lim et al., 2023). We formulate the layer-role encoder as an L-layer GNN

$$\begin{aligned} \boldsymbol{Z}_i^{(L)} &= \texttt{Role}(\mathcal{A}_i; \phi_1) \\ &:= g_L \circ g_{L-1} \circ \ldots \circ g_1(\boldsymbol{Z}_i^{(0)}; \mathcal{V}_i, \mathcal{E}_i), \end{aligned} \tag{7}$$

where $\boldsymbol{Z}_i^{(l)} = g_l(\boldsymbol{Z}_i^{(l-1)}; \mathcal{V}_i, \mathcal{E}_i, \psi_l)$ is the l-th GNN layer with trainable weights ψ_l. Every computational operator $\boldsymbol{z}_{i,v}^{(l)} \in \boldsymbol{Z}_i^{(l)}$ is encoded through message passing as

$$\begin{aligned} \boldsymbol{z}_{i,v}^{(l)} = \sigma(&\boldsymbol{W}_{\text{self}}^{(l)} \boldsymbol{z}_{i,v}^{(l-1)} + \boldsymbol{W}_{\text{in}}^{(l)} \sum_{(v',v)\in\mathcal{E}_i} \boldsymbol{z}_{i,v'}^{(l-1)} \\ &+ \boldsymbol{W}_{\text{out}}^{(l)} \sum_{(v,v')\in\mathcal{E}_i} \boldsymbol{z}_{i,v'}^{(l-1)} + \boldsymbol{b}^{(l)}), \end{aligned} \tag{8}$$

where $\psi_l = \{\boldsymbol{W}_{\text{self}}^{(l)}, \boldsymbol{W}_{\text{in}}^{(l)}, \boldsymbol{W}_{\text{out}}^{(l)}, \boldsymbol{b}^{(l)}\}$ are trainable parameters. $\phi_1 = \{\psi_1, \psi_2, ..., \psi_L\}$. The output of the final GNN layer $\boldsymbol{Z}_i^{(L)} = \{\boldsymbol{z}_{i,v}^{(L)} \in \mathbb{R}^S\}_{v\in\mathcal{V}_i}$ is a collection of **layer-role embeddings** for all parametric computational operators in $\mathcal{A}_i$, where S is the size of the layer-role embedding space.

4.3.2. Stage Two: Role-aware Weight Generator

The layer-role information obtained from the first stage $\boldsymbol{Z}_i^{(L)}$ is combined with client-specific task information $\boldsymbol{c}_i$ and then is used to generate the client weights in a node-wise manner: $\theta_i = \texttt{Comb}\left(\boldsymbol{c}_i, \boldsymbol{Z}_i^{(L)}; \phi_2\right)$. We represent the client model weights as a collection of weights for all computational operators $\theta_i = \{\theta_{i,v} | v \in \mathcal{V}_i\}$. Specifically, θ_i is obtained using a HyperNetwork-based **node decoder** g_{nodec} applied to each node in the neural architecture graph. Let $\theta_{i,v}$ denote the weights associated with the parametric computational operator v of client i. Every $\theta_{i,v}$ is computed

$$\theta_{i,v} := g_{\text{nodec}}\left(\boldsymbol{c}_i \oplus \boldsymbol{z}_{i,v}^{(L)}; \phi_2\right), \ \forall v \in \mathcal{V}_i \tag{9}$$

where $\oplus$ denotes an operation (e.g., concatenation or summation) combining two embedding vectors: **layer-specific role** embedding $\boldsymbol{z}_{i,v}^{(L)}$ and a trainable client-specific **layer-invariant task** embedding $\boldsymbol{c}_i \in \mathbb{R}^F$, where F is the size of task embedding space. g_{nodec} is an MLP-based neural network in all experiments.

4.4. FedMBridge Training

We let clients hold only their local personal models but the server holds the TAHN model that acts as a bridge for knowledge sharing. During training, clients perform their local model updates, and meanwhile, they communicate frequently with the server to help to optimize the TAHN.

The training workflow of FedMBridge is as follows. Each communication round r contains the following steps: **(1) Download.** The server predicts the weights $\{\theta_i\}_{i\in\mathcal{N}_r} = \{h(\mathcal{A}_i, \boldsymbol{c}_i; \phi)\}_{i\in\mathcal{N}_r}$ or a subset of clients $\mathcal{N}_r \subset [N]$, using the current TAHN parameters and the current task embedding $\boldsymbol{c}_i$ and conditioned on client architecture graphs $\mathcal{A}_i$. Note that the graphs $\mathcal{A}_i$ can be auto-recognized and constructed on the server based on the uploaded client models before the first round starts, and therefore, they do not raise significant privacy issue. **(2) Local Updates:** Each selected client $i \in \mathcal{N}_r$ begins from the downloaded θ_i, performs several local optimization steps based on its local data $\mathcal{D}_i$, and finally obtain new weights $\widetilde{\theta}_i$. **(3) Upload.** Each client send its update direction $\Delta\theta_i = \widetilde{\theta}_i - \theta_i$ to the server. **(4) Global Update and Knowledge Sharing.** The server computes the updates for TAHN inspired by the chain rule:

$$\begin{aligned}
\Delta\boldsymbol{c}_i &= \nabla_{\boldsymbol{c}_i}\mathcal{L}_i(\theta_i) = \Delta\theta_i \cdot \nabla_{\boldsymbol{c}_i}\theta_i \\
\Delta\phi_2 &= \frac{1}{|\mathcal{N}_r|}\sum_{i\in\mathcal{N}_r}\left(\Delta\theta_i \cdot \nabla_{\phi_2}\theta_i\right) \\
\Delta\phi_1 &= \frac{1}{|\mathcal{N}_r|}\sum_{i\in\mathcal{N}_r}\left(\Delta\theta_i \cdot \nabla_{\boldsymbol{z}_i^{(L)}}\theta_i \cdot \nabla_{\phi_1}\boldsymbol{Z}_i^{(L)}\right),
\end{aligned} \tag{10}$$

where the multi-step local update direction $\Delta\theta_i$ has replaced the original single-step local gradients $\nabla_{\theta_i}\mathcal{L}_i(\theta_i)$ that are not efficient in FL. We perform an average of TAHN updates over clients $\mathcal{N}_r$ for implicit knowledge sharing. Figure 2 shows an illustration of the workflow. Algorithm 1 in the appendix summarizes the training workflow of FedMBridge.

5. Experiments

5.1. Setups

AMFL Simulations: We evaluated our approach in *four* AMFL simulation scenarios, whose statistics are summarized in Table 1. (1) **SceneAMF** is constructed from the bimodal NYU-v2 dataset (Nathan Silberman & Fergus, 2012) that recognizes scenes from pairs of aligned RGB and depth images for these scenes. We create 80 clients covering 2 modality types, 3 types of input signals (RGB-only, depth-only, and RGB-depth bimodal inputs), and 40 types of neural architectures for local models. Each bimodal client adopts one of the two traditional multimodal fusion strategies: *concatenation* and *element-wise product*. Each client has its personal label space of size 50 sampled from a pool of 464 scenes. (2) **ObjectAMF** is constructed from the bimodal ModelNet40 dataset (Wu et al., 2015) whose task is 3D object recognition from two views of 3D models. We create 112 clients in this simulation covering 56 types of neural architectures. For each bimodal client, we employ one of three multimodal fusion strategies: *concatenation*, *average alignment*, and *tensor fusion* (Zadeh et al., 2017). (3) **EmotionAMF** is created from the CMU-MOSEI dataset (Liang et al., 2021) that focuses on emotion recognition task from real-world online videos consisting of 3 modalities (video, language script, and audio). Each video is annotated for the presence of 9 discrete emotions (angry, excited, fear, sad, surprised, frustrated, happy, disappointed, and neutral). The local tasks can be unimodal, bimodal, or trimodal. We employ three multimodal fusion strategies across clients: *average alignment*, *tensor fusion* (Zadeh et al., 2017), and *MultiEMO* (Shi & Huang, 2023). (4) **MnistAMF** is made from AVMnist (Liang et al., 2021) and MultiMnist (Sabour et al., 2017) datasets, covering three modalities (image of style one, image of style two, and the audio for digit). MnistAMF uses 4 multimodal fusion strategies: *average alignment, tensor fusion, MultiEMO*, and *cross-attention fusion* (Praveen et al., 2022). More details of local datasets and the local neural architecture configurations in these simulations are provided in Table 4 in Appendix A.

Baseline Methods: We compare FedMBridge with three families of baselines. (1) *No-knowledge-sharing* method, namely **Local**, which separately trains local models that have different neural architectures, without any knowledge sharing among clients, i.e., $\mathcal{R}(\cdot)$=0. (2) *Feature-sharing* approaches, such as **FedDistill** (McMahan et al., 2018) and **FedGKD** (Yao et al., 2023), which employ a public dataset at server that facilitates mutual knowledge distillation across

Table 1. Summary of the 4 simulations of AMFL. **FS**: number of different multimodal fusion strategies. **W**: number of different widths for each unimodal encoder. **D**: number of different depths for each unimodal encoder. **T**: number of topology types for visual modality's encoder. Acronyms for some modality types: **V** (Video), **L** (Language), **A** (Audio), **I1** (Style-one image), **I2** (Style-two image).

Simulation	#Clients	Input modalities per client	#Target classes per client	#Architectures (W, D, T, FS)
SceneAMF	80	{RGB}, {Depth}, or {RGB, Depth}	random 50 in 464 scenes	40 (1, 2, 2, 2)
ObjectAMF	112	{3D View1}, {3D View2}, or {3D View1, 3D View2}	random 5 in 40 objects	56 (2, 2, 1, 3)
EmotionAMF	90	{V}, {L}, {A}, {V, L}, {V, A}, {L, A}, or {V, L, A}	9 emotions	66 (1, 2, 1, 3)
MnistAMF	86	{I1}, {I2}, {A}, {I1,I2}, {I1,A}, {I2,A}, or {I1,I2,A}	random 4 digits in 0∼9	86 (2, 1, 1, 4)

heterogeneous architectures. For fair comparison, the public dataset should not result in much privacy risk and thus we allow only 5% clients submit only 5% of their samples to the server during the setup phase before training begins. We skip data-free federated distillation methods as they require extra decoders to generate multimodal data. (3) *Parameter-sharing* approaches, including those using fixed local subnet allocations, such as **HeteroFL** (Diao et al., 2021), as well as proper extensions of architecture-homogeneous PFL (such as **HyperPFL** (Shamsian et al., 2021) and **APFL** (Jeong & Hwang, 2022)) using pruning techniques. Our FedMBridge belongs to parameter-sharing approaches, while we also show that FedMBridge can be combined with the feature-sharing methods and achieves better performance.

Reproductibility: Implementation details and hyperparameters are provided in Appendix B.

5.2. Main Results

Table 2 reports the results on all simulations, comparing FedMBridge with baseline approaches. We use *two* evaluation metrics: **ACC** (%) refers to the final average accuracy on the testing datasets over all clients; **COT** (seconds) refers to the average time spent by each round of server-client communication, including downloading, uploading, and global knowledge sharing stages, whose weights in the metric are 0.46, 0.46, and 0.08, respectively. Each experiment was executed by 5 trials using different random seeds.

Performance Comparison: From Table 2, in general, the FL methods that utilize either features or weights for knowledge sharing outperform the non-knowledge-sharing Local, showing that the knowledge learned at local clients were successfully exchanged among clients and improved local performance. *However*, we observe that the feature-sharing baselines (rows 4-5) were sensitive to the modality gap and statistical heterogeneity in AMFL since feature distillation is not sufficiently robust to distribution shift. In addition to robustness issue, these methods relied on a public dataset with complete modalities, which raises privacy risk. *Also*, we observe that the Parameter-sharing baselines (rows 6-8) significantly suffered from the architecture heterogeneity in AMFL: the more heterogeneous the local neural architectures, the less shareable weights between clients, and therefore, trained weights or gradients might be not sufficiently transferred among local models. *In contrast*, FedMBridge outperformed both feature- and Parameter-sharing baselines. This is because FedMBridge does not rely on public data; does not rely on knowledge transfer losses that is difficult to balance task shifts; and leverages TAHN to implicitly maximize sufficient weight sharing instead of explicit aggregation of unaligned weight spaces.

Communication Efficiency: As in Table 2, Local has the best efficiency as it requires no inter-client knowledge sharing and no communication. Figure 3 breakdowns the time costs (row 2) and memory costs (row 1) on ObjectAMF during each communication round into multiple stages, where its column-1 reports the average costs of local training over all selected clients with different sizes of models and data, its column-2 shows the average costs during the information exchange between the server and selected clients (including reading, copy, transmission, and writing of models), and its column-3 shows the costs associated with cross-client knowledge sharing facilitated by the server. Among knowledge-sharing FL methods, FedMBridge and feature-sharing baselines required the same time for uploading local models. However, FedMBridge was faster in knowledge aggregation compared to on-server distillation methods and more efficient in downloading compared to on-device distillation. This is because feature-sharing methods require computing pseudo-labels for distillation using all data on local or global sites, while FedMBridge avoids this by not needing data loaders on the server or additional client-side processing. Moreover, FedMBridge showed higher efficiency in uploading and downloading than pruning-based parameter-sharing baselines. This is because FedMBridge transmits the original local models, whereas these baselines transmit a large supermask indicating parameter shareability. Although parameter-sharing baselines using fixed masks can mitigate this issue, they suffer performance drops due to connection collapse.

5.3. Ablation Study

Table 3 reports the ablation study for our FedMBridge framework. We investigated the impacts of *four* components or factors in FedMBridge. **(1) Impact of TAHN Stage One.**

Table 2. Average performance comparison between different methods on all AMFL simulations. "s": seconds. "*": privacy leakage risk.

Method	SceneAMF		ObjectAMF		EmotionAMF		MnistAMF	
	ACC ↑	COT ↓	ACC ↑	COT ↓	ACC ↑	COT ↓	ACC ↑	COT ↓
Local	76.56 ± 0.99	0	91.41 ± 0.90	0	67.83 ± 0.93	0	91.83 ± 1.32	0
FedDistill*	78.20 ± 0.84	41.4	91.49 ± 0.33	21.4	71.47 ± 0.56	33.2	92.33 ± 1.21	37.8
FedGKD*	81.32 ± 0.83	35.2	93.82 ± 0.98	17.6	73.97 ± 0.86	28.4	93.91 ± 0.63	29.7
HeteroFL w/ Mask	76.32 ± 1.03	25.8	91.63 ± 0.79	9.69	68.47 ± 1.35	20.8	91.94 ± 0.77	18.7
HyperPFL w/ Prune	78.92 ± 0.91	117.9	92.39 ± 0.33	63.6	72.85 ± 0.91	152.9	92.67 ± 0.43	140.2
APFL w/ Prune	77.36 ± 0.83	36.6	92.65 ± 0.92	16.2	71.61 ± 0.32	25.0	91.95 ± 0.80	31.6
FedMBridge (Ours)	**83.92** ± 0.95	37.4	**94.64** ± 0.94	18.6	**75.96** ± 0.83	26.6	**95.78** ± 0.61	29.5

Figure 3. Comparison of time costs and memory costs per communication round between different approaches on ObjectAMF dataset.

Table 3. Ablation Study for FedMBridge using MnistAMF dataset.

FedMBridge Hyperparameters				ACC ↑	COT ↓
L	$\oplus$	$\lvert\mathcal{N}_r\rvert/N$	w/wo KD		
4	concat	0.25	$\beta = 0$	95.78	29.5
0	concat	0.25	$\beta = 0$	87.75	**26.1**
4	**sum**	0.25	$\beta = 0$	95.51	28.8
4	concat	**0.40**	$\beta = 0$	**96.15**	36.2
4	concat	0.25	$\beta = \mathbf{0.01}$	95.92	41.9
4	concat	0.25	$\beta = \mathbf{0.05}$	**96.53**	41.9

The first stage of TAHN leverages a GNN-based network to learn layerwise role embeddings. Table 3 (row 4) removes the stage one by setting $L = 0$. The performance drops after this removal demonstrate the importance of this module. **(2) Impact of Role-Task Fusion Operator ($\oplus$) in TAHN Stage Two.** The second stage of TAHN combines the layer-role embeddings with the task embedding. Table 3 (row 5) replaces the default concatenation with sum operation. The performance remains almost unchanged. **(3) Impact of Client Selection ($|\mathcal{N}_r|/N$).** Table 3 (row 6) slightly improves the performance by selecting more clients at each communication round. Yet the efficiency drops along with the performance increase. The main reason of such communication time increase is that the number of input instances fed to the TAHN network increases. **(4) Combining with Feature-sharing FL.** Table 3 (rows 7-8) show that FedMBridge can be combined with the feature-sharing methods, i.e. the vanilla federated knowledge distillation, and can achieve slightly better performance, where β denotes the importance of the distillation losses during the feature sharing on the server. Although the current improvements are not obvious, in future work we may explore advanced methods to better balance the impacts of parameter sharing and feature sharing in AMFL scenarios.

6. Conclusion

In this paper, we focused on the novel Architecture-personalized MFL (AMFL) problem, which allows for free local multimodal neural architecture design with diversified multimodal fusion strategies. To attain a communication-efficient solution and improve beneficial parameter sharing in AMFL, we proposed FedMBridge, which leverages a topology-aware hypernetwork as a bridge function to balance and digest the architecture heterogeneity and statistical heterogeneity. We conduct comprehensive experiments on several AMFL simulations and the result demonstrates the efficiency and effectiveness of FedMBridge over baselines.

Acknowledgements

We would like to express sincere appreciation to all the reviewers for their constructive feedbacks, which greatly improved the quality of this paper. This work is supported in part by the US National Science Foundation under grants 2217071, 2213700, 2106913, 2008208, 1955151.

Impact Statement

Our work focuses on a general setting of Multimodal Federated Learning (MFL), which can be widely used in the real-world scenarios, including future applications in privacy-preserving Artificial General Intelligence (AGI), distributed and efficient edge-driven training of Multimodal Large Foundation Models, and so on. This work considers task complexity difference and allows for *multimodal pattern divergence* across clients, which will support a wide range of user-personalization tasks, where clients may vary greatly in their complexity of modeling the inter-modal interactions.

As for the technical impacts, while hypernetwork-based approach has been proposed in previous works for consistent neural architectures, we think our topology-aware hypernetwork approach is one of the first attempts that formally emphasize the *freedom of architecture gap* in MFL by modeling the computational flows of multimodal interactions as *graphs*. Our work may drive the future research on parameter-level knowledge transfer or collaborative AI between different models, potentially promoting the tradeoff between large-scale large model training on the server and efficient and private small model training on local devices.

References

Ahmad, S. and Aral, A. Fedcd: Personalized federated learning via collaborative distillation. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, pp. 189–194. IEEE, 2022.

Arivazhagan, M. G., Aggarwal, V., Singh, A. K., and Choudhary, S. Federated learning with personalization layers, 2019.

Atrey, P. K., Hossain, M. A., El Saddik, A., and Kankanhalli, M. S. Multimodal fusion for multimedia analysis: a survey. *Multimedia systems*, 16:345–379, 2010.

Barry, G., Konyar, E., Harvill, B., and Johnstone, C. A survey of advances in multimodal federated learning with applications. In *Multimodal and Tensor Data Analytics for Industrial Systems Improvement*, pp. 315–344. Springer, 2024.

Che, L., Wang, J., Zhou, Y., and Ma, F. Multimodal federated learning: A survey. *Sensors*, 23(15):6986, 2023.

Chen, J. and Zhang, A. Fedmsplit: Correlation-adaptive federated multi-task learning across multimodal split networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 87–96, 2022.

Chen, J. and Zhang, A. On disentanglement of asymmetrical knowledge transfer for modality-task agnostic federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 11311–11319, 2024.

Dai, R., Shen, L., He, F., Tian, X., and Tao, D. Dispfl: Towards communication-efficient personalized federated learning via decentralized sparse training. In *International Conference on Machine Learning*, pp. 4587–4604. PMLR, 2022.

Deng, Y., Kamani, M. M., and Mahdavi, M. Adaptive personalized federated learning. *arXiv preprint arXiv:2003.13461*, 2020.

Diao, E., Ding, J., and Tarokh, V. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. In *International Conference on Learning Representations*, 2021.

Feng, T., Bose, D., Zhang, T., Hebbar, R., Ramakrishna, A., Gupta, R., Zhang, M., Avestimehr, S., and Narayanan, S. Fedmultimodal: A benchmark for multimodal federated learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4035–4045, 2023.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Funoki, Y. and Ono, S. Dmnas: Differentiable multi-modal neural architecture search. In *International Workshop on Advanced Imaging Technology (IWAIT) 2021*, volume 11766, pp. 465–470. SPIE, 2021.

Gao, J., Li, P., Chen, Z., and Zhang, J. A survey on deep learning for multimodal data fusion. *Neural Computation*, 32(5):829–864, 2020.

Guo, B., Mei, Y., Xiao, D., and Wu, W. Pfl-moe: Personalized federated learning based on mixture of experts. In *Web and Big Data: 5th International Joint Conference, APWeb-WAIM 2021, Guangzhou, China, August 23–25, 2021, Proceedings, Part I*, pp. 480–486, 2021.

Hong, J., Wang, H., Wang, Z., and Zhou, J. Efficient split-mix federated learning for on-demand and in-situ customization. In *International Conference on Learning Representations (ICLR 2022)*, 2022.

Jeong, W. and Hwang, S. J. Factorized-fl: Agnostic personalized federated learning with kernel factorization similarity matching, 2022.

Jiang, D., Shan, C., and Zhang, Z. Federated learning algorithm based on knowledge distillation. In *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*, pp. 163–167. IEEE, 2020.

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

Jiang, Z., Xu, Y., Xu, H., Wang, Z., Liu, J., Chen, Q., and Qiao, C. Computation and communication efficient federated learning with adaptive model pruning. *IEEE Transactions on Mobile Computing*, 2023.

Kim, M., Yu, S., Kim, S., and Moon, S.-M. Depthfl: Depthwise federated learning for heterogeneous clients. In *The Eleventh International Conference on Learning Representations*, 2023.

Knyazev, B., Drozdzal, M., Taylor, G. W., and Romero-Soriano, A. Parameter prediction for unseen deep architectures. In *Advances in Neural Information Processing Systems*, 2021.

Kumar, G. and Toshniwal, D. Neuron specific pruning for communication efficient federated learning. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pp. 4148–4152, 2022.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Li, X., Huang, K., Yang, W., Wang, S., and Zhang, Z. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019.

Liang, P. P., Lyu, Y., Fan, X., Wu, Z., Cheng, Y., Wu, J., Chen, L., Wu, P., Lee, M. A., Zhu, Y., et al. Multibench: Multiscale benchmarks for multimodal representation learning. *arXiv preprint arXiv:2107.07502*, 2021.

Lim, D., Maron, H., Law, M. T., Lorraine, J., and Lucas, J. Graph metanetworks for processing diverse neural architectures. In *The Twelfth International Conference on Learning Representations*, 2023.

Lin, Y.-M., Gao, Y., Gong, M.-G., Zhang, S.-J., Zhang, Y.-Q., and Li, Z.-Y. Federated learning on multimodal data: A comprehensive survey. *Machine Intelligence Research*, pp. 1–15, 2023.

Litany, O., Maron, H., Acuna, D., Kautz, J., Chechik, G., and Fidler, S. Federated learning with heterogeneous architectures using graph hypernetworks. *arXiv preprint arXiv:2201.08459*, 2022.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2018.

McMahan, H. B., Ramage, D., Talwar, K., and Zhang, L. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

Nathan Silberman, Derek Hoiem, P. K. and Fergus, R. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

Praveen, R. G., de Melo, W. C., Ullah, N., Aslam, H., Zeeshan, O., Denorme, T., Pedersoli, M., Koerich, A. L., Bacon, S., Cardinal, P., et al. A joint cross-attention model for audio-visual fusion in dimensional emotion recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2486–2495, 2022.

Sabour, S., Frosst, N., and Hinton, G. E. Dynamic routing between capsules. *Advances in neural information processing systems*, 30, 2017.

Shamsian, A., Navon, A., Fetaya, E., and Chechik, G. Personalized federated learning using hypernetworks. In *International Conference on Machine Learning*, pp. 9489–9502. PMLR, 2021.

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

Shi, T. and Huang, S.-L. Multiemo: An attention-based correlation-aware multimodal fusion framework for emotion recognition in conversations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14752–14766, 2023.

Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A. S. Federated multi-task learning. *Advances in neural information processing systems*, 30, 2017.

T Dinh, C., Tran, N., and Nguyen, J. Personalized federated learning with moreau envelopes. *Advances in Neural Information Processing Systems*, 33:21394–21405, 2020.

Vahidian, S., Morafah, M., and Lin, B. Personalized federated learning by structured and unstructured pruning under data heterogeneity. In *2021 IEEE 41st international conference on distributed computing systems workshops (ICDCSW)*, pp. 27–34. IEEE, 2021.

Wang, K., He, Q., Chen, F., Chen, C., Huang, F., Jin, H., and Yang, Y. Flexifed: Personalized federated learning for edge clients with heterogeneous model architectures. In *Proceedings of the ACM Web Conference 2023*, pp. 2979–2990, 2023.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

Xiong, B., Yang, X., Qi, F., and Xu, C. A unified framework for multi-modal federated learning. *Neurocomputing*, 2022.

Yang, R., Tian, J., and Zhang, Y. Regularized mutual learning for personalized federated learning. In *Asian Conference on Machine Learning*, pp. 1521–1536. PMLR, 2021.

Yao, D., Pan, W., Dai, Y., Wan, Y., Ding, X., Yu, C., Jin, H., Xu, Z., and Sun, L. Fed gkd: Towards heterogeneous federated learning via global knowledge distillation. *IEEE Transactions on Computers*, 2023.

Yu, Q., Liu, Y., Wang, Y., Xu, K., and Liu, J. Multimodal federated learning via contrastive representation ensemble. *arXiv preprint arXiv:2302.08888*, 2023.

Zadeh, A., Chen, M., Poria, S., Cambria, E., and Morency, L.-P. Tensor fusion network for multimodal sentiment analysis. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1103–1114, 2017.

Zadeh, A., Liang, P. P., Mazumder, N., Poria, S., Cambria, E., and Morency, L.-P. Memory fusion network for multi-view sequential learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018a.

Zadeh, A., Liang, P. P., Poria, S., Vij, P., Cambria, E., and Morency, L.-P. Multi-attention recurrent network for human communication comprehension. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018b. URL https://github.com/A2Zadeh/CMU-MultimodalSDK.

Zhang, C., Ren, M., and Urtasun, R. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018.

Zhang, J., Guo, S., Ma, X., Wang, H., Xu, W., and Wu, F. Parameterized knowledge transfer for personalized federated learning. *Advances in Neural Information Processing Systems*, 34:10092–10104, 2021.

Zhang, L., Shen, L., Ding, L., Tao, D., and Duan, L.-Y. Fine-tuning global model via data-free knowledge distillation for non-iid federated learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10174–10183, 2022.

Zhao, Y., Barnaghi, P., and Haddadi, H. Multimodal federated learning. *arXiv preprint arXiv:2109.04833*, 2021.

Zhu, Z., Hong, J., and Zhou, J. Data-free knowledge distillation for heterogeneous federated learning. In *International Conference on Machine Learning*, pp. 12878–12889. PMLR, 2021.

A. AMFL Simulations

This paper focuses on the AMFL (short for Architecture-agnostic Multimodal Federated Learning) problem setting. Since it is an overlooked area we found there is no existing dataset that can be directly used to evaluate our method. Therefore, we create **four simulation scenarios of AMFL**: SceneAMF, ObjectAMF, MnistAMF, and EmotionAMF.

Each simulation involves at least one visual modality type, plus other auxiliary modality types such as audio and text. Table 4 illustrates overviews of the four simulations. We will release our AMFL simulations in the future.

A.1. Simulation One: "SceneAMF"

We use the NYU-Depth-v2 dataset (Nathan Silberman & Fergus, 2012) as the source to create SceneAMF. The NYU-Depth-v2 contains 1,449 RGBD images recorded by both the RGB and Depth cameras from the Microsoft Kinect, spanning 464 different scenes taken from 3 cities with 26 scene types. Each instance consists of two image modalities: an **RGB image** (the first modality) and a **Depth map** (the second modality) of the scene.

Derived from NYU-Depth-v2, SceneAMF consists of $N = 80$ clients with $M = 2$ modality types and 40 different local neural architectures. The statistical and architecture heterogeneity patterns are simulated as follows.

- **Simulation of Statistical Heterogeneity:**
  - *Tasks & Label Spaces*: The label space sizes on local clients are set to $|\mathcal{Y}_i|= 50, \forall i \in [N]$. In order to create a diversity of local tasks with distribution shifts, we let each client randomly sample 50 scenes from the pool of 464 scenes of NYU-Depth-v2, and use the 50 scenes as its target classes. Clients annotate their target classes using relative labels 0/1/2/.../49 and have no knowledge on which classes match the others'. Note that a global label space does not exist.
  - *Size of Local Datasets*: Each client i has n_i training samples and 60 testing samples from the selected 50 scenes. Here, n_i is sampled from a normal distribution $n_i \sim \text{Normal}(70, 10)$, which simulates a diverse numbers of samples across clients.
  - *Input Modalities & Input Spaces*: We let a total of 3 types of input spaces across the 80 local datasets. Specifically, we let 16 clients take only RGB images as the inputs, 16 clients take only Depth maps as the inputs, and 48 clients take pairs of the RGB image and Depth map as the inputs.
- **Simulation of Architecture Heterogeneity:**
  - *Feature Extractors for RGB Images*: There are 64 clients having RGB inputs. We use two architecture families (***CNN*** and ***ResNets***) and two depths per family (***3*** and ***5***) across 64 clients. Specifically, 32 clients use the CNN-based feature extractors for RGB images, where 16 of them contains 3 CNN layers and the other 18 clients contain 5 CNN layers. The other 32 clients use the ResNet-based feature extractors for RGB images, among which 16 clients contains 3 ResNet blocks and the others use 5-block ResNets. For those using the same architecture family and having the same depth, their CNN filter sizes and channel numbers for each layer are fixed to {3x3, 3x3, 5x5, 3x3, 3x3} and {16, 32, 32, 64, 64}, respectively. We use MaxPooling in all the networks.
  - *Feature Extractors for Depth Maps*: There are 64 clients taking Depth maps as inputs. We let each client use either ***ViT***-based or ***CNN***-based model, with either 2 or 4 layers. Specifically, 32 clients use the ViT-based feature extractors for Depth maps, with 768-dimensional embedding size, 9 patches, and 3 heads per attention layer for all clients. Among them, 16 ViT clients contains 2 self-attention layers and the others contain 4 self-attention layers. The other 32 clients use the CNN-based feature extractors for RGB images, among which 16 clients contains 2 CNN layers and the others use 4-layer CNNs. The CNN filter sizes and channel numbers for each layer are fixed to {3x3, 3x3, 5x5, 3x3} and {8, 16, 16, 32}, respectively. Again, we use MaxPooling in all the networks.
  - *Multimodal Fusion Modules*: For each of the 32 unimodal clients, we use a 2-layer or 3-layer MLP right after the feature extractor. The two types of MLP have the same width (counted beginning from the deeper to shallower layers) but different depths. Then, for each of the 48 bimodal clients, we use either Concatenation or Element-wise Product to combine the two modality-specific extracted features, which is then followed by a 2-layer or 3-layer MLP. In bimodal clients, there could exist all the 4 types of feature extractors per modality (as presented above) as well as 4 types of fusion modules, as illustrated in Table 4 (row 1).
  - *Client-personal Final Layers*: The final layer on each client is a dense layer with the output size of 50.

A.2. Simulation Two: "ObjectAMF"

We use the ModelNet40 dataset (Wu et al., 2015) as the source to create ObjectAMF. ModelNet40 contains 12,311 3D shapes covering 40 common object categories, including airplane, bathtub, bed, bookshelf, chair, cone, cup, and so on. Each 3D object has two image modalities, the **two viewpoints** of its shape. Derived from ModelNet40, ObjectAMF consists of $N = 112$ clients with $M = 2$ modality types and 56 different local neural architectures. The statistical and architecture heterogeneity patterns are simulated as follows.

- **Simulation of Statistical Heterogeneity:**
  - *Tasks & Label Spaces*: The label space sizes on local clients are set to $|\mathcal{Y}_i| = 5, \forall i \in [N]$. We let each client randomly sample 5 object types from the pool of 40 objects of ModelNet40, and use the 5 object categories as its target classes. Clients annotate their target classes using relative labels 0/1/2/3/4 and have no knowledge on which classes match the others'.
  - *Size of Local Datasets*: Each client i have $n_i \sim Normal(500, 100)$ training samples and 1,000 testing samples from the selected 5 classes.
  - *Input Modalities & Input Spaces*: We let a total of 3 types of input spaces across the 112 local datasets. Specifically, we let 24 clients take only View-1 images as the inputs, 24 clients take only View-2 images as the inputs, and 64 clients take pairs of the View-1 and View-2 images as the inputs.
- **Simulation of Architecture Heterogeneity:**
  - *Unimodal Feature Extractors*: We use MLP models extracting features for view1 or view2 images. For each view point, there are 4 different MLP configurations across clients, including two different network widths and two different network depths: 2-layer MLPs with a width of [64 ,32] or [128, 64], and 3-layer MLPs with a width of [128, 64, 32] or [256, 128, 64].
  - *Multimodal Fusion Modules*: For the 48 unimodal clients, there feature extractors are followed by the same 2-layer MLP with a width of [64, 32], where there is no need for multimodal fusion. Then, for each of the 64 bimodal clients, we use one of the three multimodal fusion strategies: Concatenation, Tensor Fusion (Zadeh et al., 2017), and Average Alignment. As illustrated in Table 4 (row 2), those using alignment leverage a 2-layer MLP with [64, 32] to process the aligned features; those using concatenation fusion leverage a 3-layer MLP with [64, 64, 32] to process the concatenated features; and, those using the tensor fusion leverage a 3-layer MLP with a wider configuration of [128, 64, 32].
  - *Client-personal Final Layers*: The final layer on each client is a dense layer with the output size of 5.

A.3. Simulation Three: "MnistAMF"

We create AMFL scenarios with *more than two modalities* and a *broader range of fusion strategies*. We begin with a lightweight setup, MnistAMF, using the widely-adopted MNIST dataset for simple digit classification tasks.

We first collect our source modalities from several different versions of MNIST. In the last decades, there have been many modified versions of MNIST datasets extending the original data distribution. For example, in AVMNIST (Liang et al., 2021), each digit image is accompanied by the corresponding spoken pronunciation of that digit, as articulated by a speaker. MultiMNIST (Sabour et al., 2017) generates colored digit images, where each image contains both the target digit (in the red color) as well as a random digit overlayed on the target digit (in the green color). There are also NoisyMNIST, RotationMNIST, GridMnist, and so on, proposed by previous works. We consider each of these modifications as an "image style" for the same target digit. Therefore, it is reasonable to create a multimodal MNIST using the combinations of these modified datasets. In this paper, we collect **three source modalities** by combining AVMNIST and MultiMNIST, as well as adding some statistical noise to the original MNIST. Each instance consists of three modalities: the original digit image after adding noise to its background (**Style 1**); the colored digit image overlayed with a random beyond-target digit, generated by from MultiMNIST (**Style 2**); and, the audio signals of the spoken digit from AVMNIST (**Sound**).

Derived from the above collections, we construct the MnistAFL, which consists of $N = 86$ clients with 86 different local neural architectures across the $M = 3$ modality types. The statistical and architecture heterogeneity patterns are simulated as follows and illustrated in Table 4 (row 3).

- **Simulation of Statistical Heterogeneity:**

- *Tasks & Label Spaces*: The label space sizes are set to $|\mathcal{Y}_i|= 4, \forall i \in [N]$. We let each client randomly sample 4 digits from the total of 10 digits, and use them as its target classes. Clients maps their target digits using relative label ids 0/1/2/3.
- *Local Data Distributions*: When creating Style-1 images, different clients use different random seeds to generate the noise added to the target images. Also, when creating Style-2 images, different clients use different random seeds to select the beyond-target digits that overlay on each target image.
- *Size of Local Datasets*: Each client i have 1,000 testing samples and $n_i \sim Normal(1800, 600)$ training samples from the selected 4 digits.
- *Input Modalities & Input Spaces*: We set up a total of 6 input spaces across the 86 local datasets. There are 3 clients taking only Style-1 images as the inputs and 3 clients taking only Style-2 images as the inputs. Also, there are 3 types of bimodal clients, including 12 clients taking both styles as inputs, 12 clients taking the sound in accompany with Style-1 images, and 12 clients taking the sound in accompany with Style-2 images. Finally, there are 48 clients taking both styles as well as the sound as inputs.

- **Simulation of Architecture Heterogeneity:**
  - *Unimodal Feature Extractors*: We employ CNN-based feature extractors for Style-1 and Style-2 images. For the sound modality, we convert the audio signals into visual representations, i.e., spectrograms, as illustrated in Table 4 (row 3). Thus we can also use CNN-based feature extractors for the spoken digits. For every modality type, its corresponding feature extractors on different clients maintain a consistent depth (i.e., fixed two CNN layers) but vary in their widths (i.e., the number of channels can be either 32 or 64). The kernel sizes of all feature extractors are set to 3×3.
  - *Multimodal Fusion Modules*: As mentioned above, we set up 36 bimodal clients and 48 trimodal clients in this simulation scenario. Before fusion, we use MLP adaptors for each incoming modality's encoded features, each of which is an MLP that maps the encoded unimodal features into a 256-dimensional vector. Then, we consider 4 multimodal fusion strategies: Tensor Fusion, MultiEMO (Shi & Huang, 2023), Cross-attention Fusion (Praveen et al., 2022), and Average Aalignment. As in Figure 4 (row 3), there are 2^2 clients using MultiEMO to fuse the Style-1 image and the sound spectrogram, where the widths of their feature extractors are different. Likewise, 2^2 clients using MultiEMO to fuse the Style-2 image and the sound spectrogram; 2^3 clients using MultiEMO to fuse Style-1, Style-2, and the sound spectrogram; etc. We use a 2-layer MLP with the width [64, 32] after obtaining the fused feature. Furthermore, as edge cases, the remaining 6 unimodal clients do not have the fusion operators so that we use the similar 2-layer MLP right after the modality-specific feature extractors.
  - *Client-personal Final Layers*: The final layer on each client is a dense layer with the output size of 4.

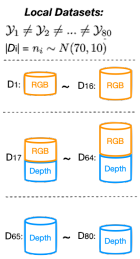
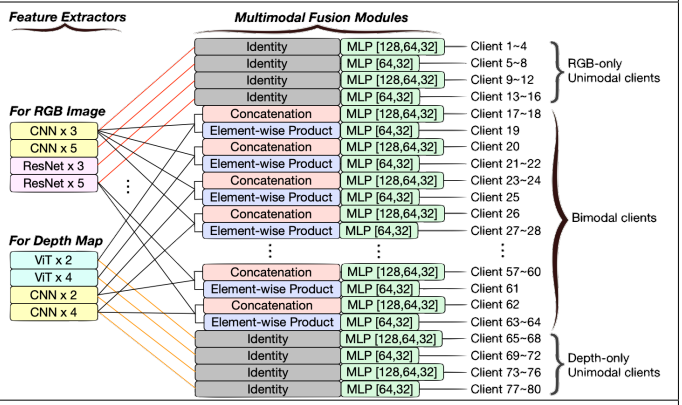
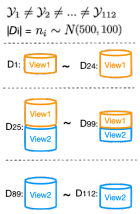
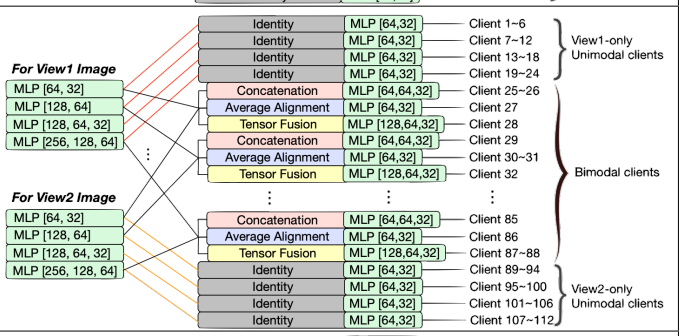
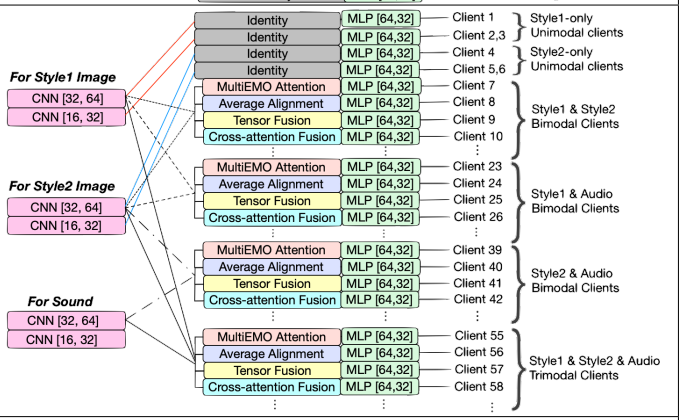
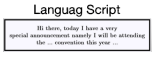
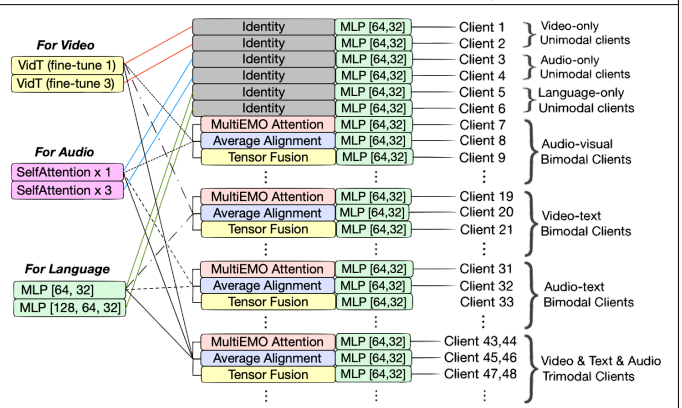
A.4. Simulation Four: "EmotionAMF"

We use the Multimodal Opinion Sentiment and Emotion Intensity (CMU-MOSEI) dataset (Liang et al., 2021) as the source to create EmotionAMF. CMU-MOSEI contains more than 23,500 sentence utterance videos from more than 1000 online YouTube speakers. Each video sample consists of **three multimedia modalities**, including the *visual* sequence, the *audio* channels, plus the annotated *language* script. Each video is annotated for the presence of 9 discrete emotions, including angry, excited, fear, sad, surprised, frustrated, happy, disappointed, and neutral.

Derived from CMU-MOSEI, EmotionAMF consists of $N = 90$ clients with $M = 3$ modality types and 66 different local neural architectures. The statistical and architecture heterogeneity patterns are simulated as follows.

- **Simulation of Statistical Heterogeneity:**
  - *Tasks & Label Spaces*: Since there is only 9 emotion types in CMU-MOSEI, we set the label space sizes of all clients as $|\mathcal{Y}_i|= 9, \forall i \in [N]$. This is reasonable as the sense for distinguishing different emotions is often a commonsense shared by human annotators. Yet in EmotionAMF the same emotion type might be annotated as different label ids on different clients, which simulates the task heterogeneity. We perturb the label indices for each client.
  - *Size of Local Datasets*: Each client i have 400 testing samples and $n_i \sim Normal(215, 40)$ training samples. To guarantee data privacy, the videos recorded from each individual speaker appears on only one client.
  - *Input Modalities & Input Spaces*: We consider a situation that many clients do not have enough computation power to model the complex relationships between video, language, and audio. Instead, some clients may only

Table 4. Statistics of the 4 Simulations of Architecture-agnostic Multimodal Federated Learning (AMFL).

Simulations	All Modality Types	Client Heterogeneity Illustration	
		Data Heterogeneity	Heterogeneity of Neural Architectures
SceneAMF	RGB image  Depth map 		
ObjectAMF	1st View Point of 3D Shape  2nd View Point of 3D Shape 		
MnistAMF	Style1 of Digit  Style2 of Digit  Sound of Digits (Spectrogram) 		
EmotionAMF	Video  Audio  Languag Script 		

learn to model its video features but receive such relationship knowledge from other more powerfule clients. Therefore, we let a total of 7 types of input spaces across the 90 local datasets. Clients can be unimodal, bimodal, as well as trimodal. There are 3 types of unimodal clients, including 2 clients taking only videos as the inputs, 2 clients taking only texts, and 2 clients taking only audios as the inputs. Also, there are 3 types of bimodal clients, including 12 audio-visual clients, 12 visual-textual clients, and 12 audio-textual clients. Finally, there are 48 clients taking all the three modalities as inputs.

- **Simulation of Architecture Heterogeneity:**
  - *Feature Extractors for the Video Modality*: We adopt a pre-trained Video Transformer (VidT) (Shi & Huang, 2023) to extract *spatial* and *temporal* features in videos. For simplicity, during the federated training, we do not fine tune the entire Video Transformer. Instead, we employ parameter-efficient fine-tuning (PEFT), such as LoRA, which operates at only a couple of multi-head attention layers. Among the 74 clients that contain video modalities, we let 37 of them fine-tune the final attention layer while the remaining ones fine-tune the last three attention layers.
  - *Feature Extractors for the Language Modality*: First, following (Liang et al., 2021), we utilize a pre-trained BERT to extract features for language scripts. Then, each local client aims to train a "self-attentional feature adaptor", whose responsibility is to adapt these pre-extracted features to each local task distribution, that is, feature adaptation. The adapted features will be used for multimodal fusion. As for the neural architecture design for this adaptor, we let a half of clients that take the language inputs train a single 4-head self-attention layer, while the other half use a 3-layer 4-head self-attention mechanism as this adaptor. Both use 256 as the size of key/query/value vectors.
  - *Feature Extractors for the Audio Modality*: We use MLP-based models for the audio modalities. Again, we let a half of clients that take the audio inputs train a 2-layer MLP with the widths [64, 32] while the others train a 3-layer MLP with the widths [128, 64, 32]. We use ReLU activation functions for all clients.
  - *Multimodal Fusion Modules*: As for the multimodal fusion strategie, we consider both the straightforward way Average Aalignment as well as those good at modeling complex inter- and intra-modality interactions, such as Tensor Fusion (Zadeh et al., 2017) and MultiEMO. The detailed configurations can be found in Figure 4 (row 4). There are 2^2 audio-visual (or audio-text, video-text) clients simply align the two modalities' feature onto a common latent space, where the depths of their feature extractors are different. Likewise, 2^2 audio-visual (or audio-text, video-text) clients leverage the Tensor Fusion with different backbone depths; and, 2^2 audio-visual (or audio-text, video-text) clients leverage the MultiEMO fusion with different backbone depths. For the trimodal case, there are 2^3 audio-visual-text clients using each fusion strategy, where the depths of their feature extractors are different. After the fusion, we use a common 2-layer MLP with the width [64, 32] to process the fused information. Furthermore, as edge cases, the remaining 6 unimodal clients do not have the fusion operators so that we use the similar 2-layer MLP right after the modality-specific feature extractors.
  - *Client-personal Final Layers*: The final layer on each client is a dense layer with the output size of 9.

B. Reproducibility

All the approaches are implemented using PyTorch 3.7 and we ran all experiments on a single A800.

B.1. Implementation of Baselines

We compared our approach against seven baselines across three families, whose implementation details are as follows.

B.1.1. *Separate Training*

Local. Clients separately train their own models. There is no communication and knowledge transfer between clients. Hence a server is not needed. As clients may converge at varying speeds, to ensure a fair comparison, we adopt the practice of allowing each client to showcase its optimal performance post-convergence. Subsequently, we calculate the average accuracy over the best performances of all clients.

B.1.2. *Federated Training via Feature Sharing*

FedDistill (McMahan et al., 2018). The server utilizes an unlabeled public dataset to enable feature sharing among uploaded client models through Knowledge Distillation (KD). When applying FedDistill to the AMFL scenarios, we make the following three adjustments. *First*, the original FedDistill relies on a unified global weight space–the student model in the

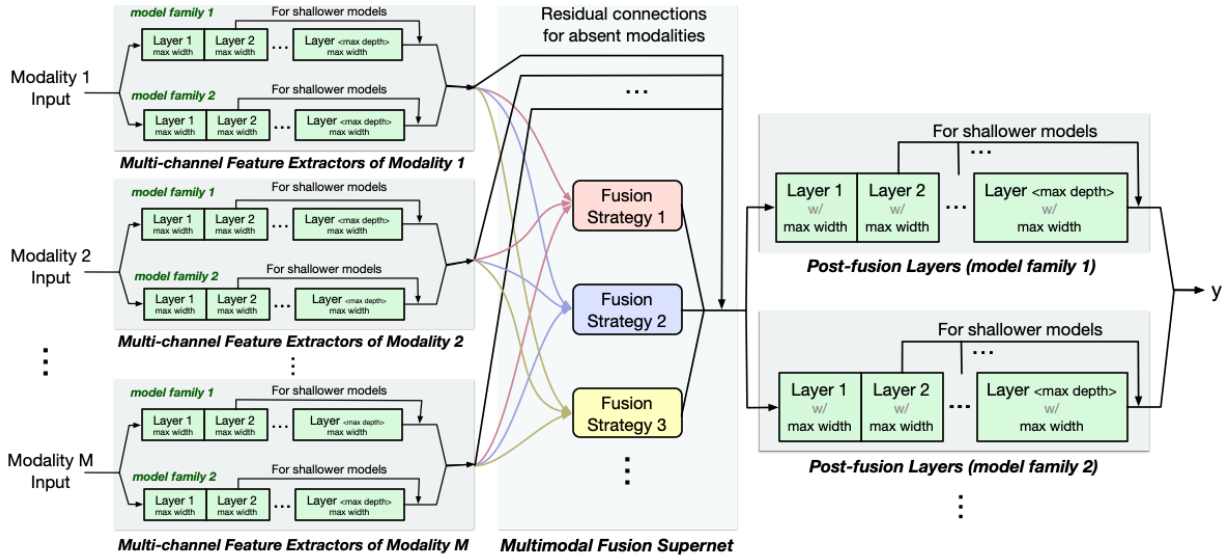


Figure 4. Baselines' supernet architecture on the server.

original FedDistill is the average of all uploaded models. This condition is not met in AMFL. Thus we conduct client-wise KD, where the uploaded model of each client is considered as a student. *Second*, FedDistill leverages several models $\widehat{\theta}_1, \widehat{\theta}_2, ..., \widehat{\theta}_s$ as teachers to compute the pseudo-labels, which then guide the student training. Upon receiving the weights $\{\theta_i | \forall i \in \mathcal{N}_r\}$ from selects clients, instead of constructing a Gaussian weight distribution $p(\theta|\mathcal{D}) = q(\{\theta_i | \forall i \in \mathcal{N}_r\})$ and computing the pseudo-labels from re-sampled models $\widehat{\theta}_1, \widehat{\theta}_2, ..., \widehat{\theta}_s \sim p(\theta|\mathcal{D})$, we directly select a subset $\{\widehat{\theta}_1, \widehat{\theta}_2, ..., \widehat{\theta}_s\} \subseteq \{\theta_i | \forall i \in \mathcal{N}_r\}$ and use these models to compute the pseudo-labels. In all experiments, we set $s = 10$ and use the step size $\beta = 0.1$ for the gradients of KD losses. *Third*, whereas FedDistill gathers public data from sources outside the client pool, this is not feasible for Personalized and Multimodal Federated Learning, where each client is engaged in distinct tasks and employs diverse modalities. In our scenario, before the start of federated training, the public dataset is collected from the clients such that every pattern of modality combinations has public samples available for client-wise KD. Specifically, we randomly sample $\rho\%$ clients from each group of clients having the same input structure (the same combination of modality types). For example, for SceneAMF, we sample $16 \times \rho\%$ RGB-only client, $16 \times \rho\%$ Depth-only client, and $48 \times \rho\%$ RGB-Depth bimodal clients. Then, we request each of them to submit $\rho\%$ of their local training samples. Considering the potential privacy leakage of such data collection way, we mitigate this risk by deliberately keeping the hyperparameter ρ at a small value, specifically $\rho = 5$.

FedGKD (Yao et al., 2023). The server employs a memory buffer to store multiple historical global models from recent rounds, which are used as teachers to guide the local model training via Knowledge Distillation (KD). Students are the local models trained on the data owners. Different from FedDistill, a public dataset is not needed in FedGKD since KD is performed locally using the local dataset. Yet a tradeoff is that FedGKD requires the server to (1) capture the accurate global knowledge as the teacher, as well as (2) each client has sufficient computation resource to download the historical teachers to perform local KD. Both cannot be satisfied in AMFL given the weight-space heterogeneity and limited storage at clients. Thus, when we apply FedGKD to the AMFL scenarios, we make the following adjustments. *First*, about what are the teachers stored in the memory buffer at server, since the aggregated knowledge in the original FedGKD is unavailable, we opt to directly store the historical uploaded client models. To mitigate excessive memory usage in storing all clients' historical models, the server selects only s uploaded models to be added to the memory buffer at each round r. The models before round $(r-3)$ are removed. These models are randomly selected from $\{\widehat{\theta}_1^r, \widehat{\theta}_2^r, ..., \widehat{\theta}_s^r\} \subseteq \{\theta_i | \forall i \in \mathcal{N}_r\}$ and are associated with their round ids and client ids. In all experiments, we set $s = 10$ and use the step size $\beta = 0.1$ for the gradients of KD losses. *Second*, considering the limited storage capacity at local clients, the number of downloaded teachers is determined according to local memory budgets. In our implementations, each client sends requests to the server to obtain $s' \in [4, 8]$ teachers, and then, the server randomly sample s' teachers from its the buffer to give it back.

We did not consider data-free federated distillation baselines as they require extra computational efforts on the server for generating multimodal data via Generative Models (such as GAN and VAE). This is particularly an issue when the local multimodal interactions are too complex to reconstruct. It's worth to mention that the proposed framework, FedMBridge,

belongs to another line of approaches; thus, we can combine it with any feature-sharing FL methods for further enhancements.

B.1.3. *Federated Training via Parameter Sharing*

HeteroFL (Diao et al., 2021) deals with the setup that local models do not share the same architecture, by utilizing a global supernet Θ^{super} on the server such that every local model is a sub-network of this supernet $\theta_i \subseteq \Theta^{\text{super}}, \forall i \in [N]$. Each local model θ_i is associated with a supermask m_i indicating which parameters in the supernet are activated in the local model. Although HeteroFL is designed for single modality and assumes the same model family and the same network depths, we can adapt it to our AMFL scenarios as follows. We need to manually construct a Θ^{super} supernet to cover all local networks. To obtain the supernet architecture, we merge all local neural networks following three rules: (1) If there are different network widths using the same model class (i.e. network topology) and the same depth, we can merge them into a single network having the same model family and depth, but using the maximum width per layer for the supernet. (2) If there are not only different widths but also different network depths using the same model class, we can also merge them into a single network, with the maximum depths for the supernet and add residual connections beginning from the layers corresponding to each local model's end layer to the final layer. The merge of widths is similar to the first rule. (3) If two clients use different network topologies (i.e. model families) at the same module–for example, client-A uses a CNN-based network for the video feature extractor and client-B uses a ViT-based network for the video extractor, we cannot merge this part of their networks. Instead, we construct a two-channel network, where each channel refers to a model family, and the outputs of the two channels are added together. An overview of the implemented supernet is shown in Figure 4.

HyperPFL + Pruning (Shamsian et al., 2021; Kumar & Toshniwal, 2022). HyperPFL solves the similar setup as APFL, except using the HyperNetwork on the server to generate the local weights. Yet the original HyperPFL can only handle the same architecture across clients. We extend HyperPFL like APFL as follows. We let the HyperNetwork generate the supernet instead of the original local models. We employ a 16-dimensional task embedding and a 5-layer MLP-based HyperNetwork with intermediate widths [128, 128, 16, 30, *], where * denotes the number of parameters in the supernet. Since the supernet is very large, the HyperNetwork contains a large number of parameters. In practice, the forward and backward process of the HyperNetwork is conducted in a parallel manner.

APFL + Pruning (Jeong & Hwang, 2022; Kumar & Toshniwal, 2022). The original paper introduced the Agnostic Personalized Federated Learning (APFL) problem and addressed it using Factorized-FL framework, which factorizes the model parameters into a pair of rank-1 vectors, where one captures the common knowledge across different labels and tasks and the other captures knowledge specific to the task for each local model. Yet APFL assumes all local models share the same architecture. Thus, when applying APFL to our AMFL scenarios, we make the following two adjustments. *First*, we extend APFL to the architecture-heterogeneous setting by straightforwardly combining AFPL with the supernet idea of HeteroFL. We let both global and local models of APFL use the supernet architecture instead. The supernet architecture is constructed using the same way as in HeteroFL. Since local datasets may not contain all modality types to feed to the supernet, those missing input modalities are imputed by zeros. *Second*, model aggregation on the server is guided by the supermasks of each client. In contrast to HeteroFL, supermasks are dynamically updated at each round using structured magnitude Pruning, with the target compression ratio determined by the specific local network size.

All these parameter-sharing FL baselines depend on a manually constructed supernet, which essentially provides prior knowledge about the relationships between local parameters. However, we found such heuristic prior knowledge provided by a supernet does not contain sufficient parameter-level correlation information, thereby limiting their potential of knowledge transfer. For example, in Figure 4, there is no knowledge sharing across clients between *different model families*, between *different fusion strategies*, between *different network layers*, and between each *pair of parameters* in the same layer. In addition, operating on the supernet is not efficient. In contrast, our proposed framework does not construct but automatically learn the knowledge about the relationships between local parameters.

B.2. FedMBridge Algorithm

The FedMBridge training process is summarized in Algorithm 1.

B.3. Hyperparameters

The hyperparameters are listed in Table 5.

Algorithm 1 FedMBridge

Input: Local dataset $\mathcal{D}_i, i = 1, 2, ..., N$; number of communication rounds T; local learning rate α; global learning rate η; local neural architecture configurations.
Initialization: On each $i \in [N]$, construct local neural architecture graph $\mathcal{A}_i$ and initializes the local model θ_i.
Initialization: Each $i \in [N]$ upload $\mathcal{A}_i$ to server.
for round $r = 1, 2, ..., T$ **do**
  Select $\mathcal{N}_r \subset [N]$ clients as participants.
  // Inference and Download
  for client $i \in \mathcal{N}_r$ in parallel **do**
    Compute $\theta_i = h(\mathcal{A}_i, \boldsymbol{c}_i; \phi)$ and send to client i.
  end for
  // Local updates & Upload
  for client $i \in \mathcal{N}_r$ in parallel **do**
    Copy $\widetilde{\theta}_i \leftarrow \theta_i$
    for each local update step **do**
      Sample mini-batch $B \in \mathcal{D}_i$ and local update $\widetilde{\theta}_i \leftarrow \widetilde{\theta}_i - \alpha \nabla_{\widetilde{\theta}_i} \mathcal{L}_i(\widetilde{\theta}_i; B)$
    end for
    Compute update direction $\Delta\theta_i = \widetilde{\theta}_i - \theta_i$
    Upload $\Delta\theta_i$ to server.
  end for
  // Global update and knowledge sharing on the server
  for client $i \in \mathcal{N}_r$ in parallel **do**
    Update task embeddings $\boldsymbol{c}_i \leftarrow \boldsymbol{c}_i - \eta \Delta\theta_i \cdot \nabla_{\boldsymbol{c}_i} \theta_i$
  end for
  Update layer-role encoder $\phi_1 \leftarrow \phi_1 - \eta\Delta\phi_1$, where $\Delta\phi_1 = \frac{1}{|\mathcal{N}_r|} \sum_{i \in \mathcal{N}_r} \left(\Delta\theta_i \cdot \nabla_{\boldsymbol{Z}_i^{(L)}} \theta_i \cdot \nabla_{\phi_1} \boldsymbol{Z}_i^{(L)} \right)$.
  Update weight generator $\phi_2 \leftarrow \phi_2 - \eta\Delta\phi_2$, where $\Delta\phi_2 = \frac{1}{|\mathcal{N}_r|} \sum_{i \in \mathcal{N}_r} \left(\Delta\theta_i \cdot \nabla_{\phi_2} \theta_i \right)$.
  Combine $\phi = \{\phi_1, \phi_2\}$
end for
Return (Server): Global topology-aware hypernetwork ϕ and client embeddings $\boldsymbol{c}_i, \forall i \in [N]$.
Return (Clients): Personal models $\theta_i, \forall i \in [N]$ saved or obtained from the server via weight generation as in Eq.(4).

Table 5. List of Hyperparameters.

Where	Notations/Descriptions	Values
Clients	local optimizers	Adam
	local learning rate (α)	0.1 $\sim$ 0.02
	# training epoch per round	15
	training batch size	128
Server	client selection ratio ($\lvert\mathcal{N}_r\rvert/N$)	0.25
	# GNN layers (L)	4
	node feature size per GNN layer	[32, 64, 64, 32]
	layer-role embedding size (S)	32
	task embedding size (F)	16
	learning rate of TAHN (η)	0.07$\sim$0.005
	optimizer of TAHN	SGD
	weight decay	0.06