Graph-Relational Federated Learning: Enhanced Personalization and Robustness

Wanyu Lin, *Member, IEEE*, Hao Lan, Hao He, and Baochun Li, *Fellow, IEEE*

***Abstract*—Hypernetwork has recently emerged as a promising technique to generate personalized models in federated learning (FL). However, existing works tend to treat each client equally and independently — each client contributes equally to learning the hypernetwork, and their representations are independent in the hypernetwork. Such an independent treatment ignores topological structures among different clients, which are usually reflected in the heterogeneity of client data distribution. In this work, we propose *Panacea*, a novel FL framework that can incorporate client relations as a graph to facilitate learning and personalization by using graph hypernetwork. Empirically, we show *Panacea* achieves state-of-the-art performance in terms of both accuracy and speed on multiple benchmarks. Further, *Panacea* improves the robustness by leveraging the client relation graph. Specifically, it (1) generalizes better to the novel clients outside of the training and (2) is more resilient to various adversarial attacks, including model poisoning and backdoor attacks, which is also proved by our theoretical analysis.**

***Index Terms*—Personalized federated learning, graph-relational enhanced federation, graph neural networks.**

I. Introduction

With the rise of the Internet of Things (IoT), a massive amount of data is being collected from geographically distributed sources, including mobile phones, wearable sensors, and other IoT devices. This data is highly informative and can be used for various AI-based applications, including predicting health events such as the risk of heart attacks [1]. To optimize these devices' storage and computational capabilities, storing data locally and pushing more computation to IoT devices is preferred. Additionally, privacy-preserving AI is imperative to comply with data privacy regulations like GDPR [2]. Federated Learning (FL) has emerged as a popular paradigm for training statistical models over distributed devices/clients while keeping the training data local [3].

In typical federated learning, each client/device holds a dataset for local training, and a server aggregates gradients from clients for global model updates [4]. A unique global model can then be applied to all clients [4], [5]. However, this paradigm might be sub-optimal in practice, as the clients' data distributions are usually heterogeneous. To address this issue, personalized federated learning (PFL) is proposed to train a personalized local model for each client while each client can still leverage the knowledge from other clients in the federation [6]. However, there is a key challenge in PFL: *How to enable beneficial collaborative training while preserving the uniqueness of the clients yet achieving communication efficiency.*

A promising approach to balance information sharing and uniqueness preservation among clients is Personalized Federated Hypernetworks (pFedHN) [7]. Hypernetworks [8] are neural networks that generate model parameters for another set of deep neural networks. In pFedHN, a multi-layer perceptron (MLP) hypernetwork takes the local client's representation as input and generates personalized model weights corresponding to each client. By providing a mapping from the clients' embedding space to the clients' model parameter space, pFedHN achieves a better generalization for clients that were unseen during training.

Despite providing a principled approach to personalized model learning, pFedHN has limitations. It treats every client equally and independently, ignoring the potential relations of data heterogeneity across the clients. Specifically, each client's representation and corresponding local gradients are regarded as independent data samples for learning the hypernetwork. However, the clients' data distributions are often interdependent, which often forms a graph where each client is a node, and the links reflect the data distribution dependency between clients. More importantly, such a graph is naturally accessible in real-world applications with IoT networks. For example, in road network traffic forecasting in the Bay Area, sensors measuring traffic speed can be modeled as nodes, with physical proximity between two sensors as a link [9]. By modeling the clients' relations as a graph, achieving more effective knowledge sharing among the clients in the federation is possible.

This paper presents *Panacea*, a novel framework for personalized federated learning based on hypernetworks. The framework comprises a graph neural network (GNN) module and a graph generator. The GNN module takes the client representation and their relationships as input to generate local model weights. To preserve the "uniqueness" of different clients, we introduce a

Received 28 May 2024; revised 28 March 2025; accepted 4 May 2025. Date of publication 3 June 2025; date of current version 4 September 2025. This work was supported in part by the Hong Kong (HK) Research Grant Council (RGC) General Research Fund under Grant 15208222, in part by the Hong Kong Polytechnic University, in part by NSFC Young Scientist Fund under Grant A0040473, and in part by the Hong Kong Polytechnic University. *(Wanyu Lin and Hao Lan are co-first authors.) (Corresponding author: Wanyu Lin.)*

Wanyu Lin is with the Department of Data Science and Artificial Intelligence, Department of Computing, The Hong Kong Polytechnic University, Hong Kong (e-mail: wanylin@comp.polyu.edu.hk).

Hao Lan is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100190, China.

Hao He is with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Baochun Li is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 1A1, Canada.

Digital Object Identifier 10.1109/TDSC.2025.3574329

graph generator that distinguishes discrepancies among clients in the federation. The graph generator uses the encoding of the local model weights, i.e., client embeddings, to reconstruct the client relation graph. With a GNN module and a graph generator, *Panacea* enables (1) effective knowledge sharing among clients, even when two clients have low similarities, i.e., distant neighbors in the relation graph, (2) avoids negative influence between clients with significant distribution discrepancies while ensuring personalization performance.

In summary, this paper makes the following contributions. First, we propose a novel personalized federated learning framework, *Panacea*, that leverages the inherent relationships among clients to encourage effective knowledge sharing while preserving each client's uniqueness. Second, unlike other methods that incur extra communication costs, *Panacea* requires no additional communication due to the introduced graph hypernetwork. Third, we conduct extensive empirical studies on many synthetic and real-world datasets across various learning tasks, demonstrating that *Panacea* outperforms prior state-of-the-art federated learning algorithms. Furthermore, we show that *Panacea* improves the robustness of the system in two ways: (1) better generalization to novel clients outside of the training and (2) more resilience to malicious clients, as demonstrated by both experiments and theoretical analysis.

II. Related Work

Federated learning: The federated learning (FL) setting assumes a federation of distributed devices/clients, each with its local dataset [3]. The goal of FL is to collaboratively train statistical models over distributed clients with data confidentiality and communication efficiency. Perhaps the standard and the most known algorithm is FedAvg [4] which learns a global model by aggregating local models trained with IID data [4]. In practice, the clients' data distributions typically exhibit various degrees of heterogeneity [5], [10]. Many variants have been proposed to tackle statistical heterogeneity issues, such as FedMA [10]. However, training a unique model in FL settings has been shown to be vulnerable to statistical heterogeneity over distributed clients in the literature [3], [6]. More specifically, FL shows poor convergence on highly heterogeneous data, and the statistical heterogeneity deteriorates the performance of the global model on individual clients [11], [12], [13], [14].

Model-agnostic approaches are proposed to learn meta-models for local personalization [15], [16], [17], [18], [19], [20]. Existing MAML-based approaches usually are computationally intensive as they require computing the Hessian matrix. Recently, multi-task learning (MTL) has emerged as an alternative solution for PFL, where each client was treated as a learning task [21], [22], [23]. Specifically, MOCHA [21] is proposed to generalize the distributed optimization methods [24], attempting to address the statistical challenges in federated settings. Marfoq et al. [22] proposed federated surrogate optimization algorithms by assuming each local data distribution is a mixture of unknown underlying distributions. Both pFedMe [20] and Ditto [23] maintain a personalized model for each client but leverage different optimization strategies. More specifically, pFedMe uses the Moreau envelope function and Ditto employs a simple L2-regularization term. However, these works do not explicitly model the inherent dependencies among clients, leading to sub-optimal solutions.

Some prior works implicitly model the statistical dependency among clients by enforcing the client relationship in regularizing model weights [21]. Nevertheless, these works bear the limitation of regularization-based methods due to the assumption that graphs only encode the similarity of clients, and they can not operate in settings where only a fraction of devices are observed during training. Instead, we propose to learn a parametric function approximation referred to as graph hypernetworks, which attempts to generate the personalized model weights directly. The most related to us is pFedHN [7], which used a multi-layer perceptron to generate personalized heterogeneous models. Considering the participants equally limits their ability to deal with the interdependency among the devices. To the best of our knowledge, the work proposed here is the first federated learning framework to leverage hypernetwork to generate personalized models considering the inherent dependency between devices.

Hypernetworks: Hypernetwork refers to a framework in which a neural network is trained to predict the weights of another neural network that performs the tasks of interest. The term "hypernetwork" was first coined in [8], and it has shown strong performance in large-scale language modeling and image classification tasks. In the literature, hypernetworks have been widely used in many learning tasks, such as neural network architecture search (NAS) [25], [26], molecule property prediction [27]. Specifically, in NAS, hypernetworks were proposed to generate target networks conditioned on neural network architectures [25]. In the context of personalized federated learning, hypernetworks could be a feasible solution for generating personalized local models conditioned on the clients' heterogeneous conditions, such as statistical heterogeneity.

The most relevant to ours is the use of Graph Hypernetworks (GHNs) — hypernetworks that take graphs as input. For example, Nachmani et al. used graph hypernetworks for molecule property prediction [27]; Zhang et al. leveraged graph hypernetworks for neural architecture search [25]. To the best of our knowledge, ours is the first work to use graph hypernetworks for generating personalized local models in federated learning settings. In this work, we show that our graph hypernetwork can explicitly model the clients' statistical relationship to ensure effective knowledge sharing among clients. By incorporating a graph generator, it can preserve clients' uniqueness to ensure personalized performance.

Federated learning with hypernetworks: The most related prior work is pFedHN [7] which uses hypernetworks to generate personalized model weights. The major difference between our work and pFedHN is that we further incorporate client interdependency. We will show that leveraging the client relation graph, *Panacea* can boost the performance of PFL from various aspects, including better generalizability to unseen clients and robustness to malicious clients using label-flipping attacks – a representative attacks in federated learning settings [28].

Federated learning with graphs: Several prior works do federated learning with graphs. Most of them focus on graph learning problems on graph-structured data [29], [30]. For example, FedGraphNN [30] provides an open-source FL system for GNNs, which enables federated training over various GNNs. Meng et al. [29] proposes a GNN-based federated learning architecture that attempts to capture complex spatial-temporal data dependencies among multiple participants. The closest to us is SFL [31], which also attempts to leverage graphs to boost the performance of personalized federated learning. SFL naively updates the client weights by averaging over its neighbors. In contrast, we generalize hypernetwork-based approach with a graph learning module equipped with a generator, which can incorporate the relational graph of clients for generating local model weights. Following the standard FL setting, we do not require all clients to participate in each round. More importantly, with a dedicated graph reconstruction component, our approach exhibits better performance, particularly robustness in adversarial settings.

III. Proposed Framework: Panacea

A. Problem Formulation and Notations

Our objective is to train personalized models collaboratively for a set of $\mathcal{T}$ clients, each with its unique local dataset. Each client $t \in \mathcal{T}$ has a distribution $\mathcal{P}_t$ on $\mathcal{X}_t \times \mathcal{Y}_t$. Since the local data distributions $\mathcal{P}_t \in \mathcal{T}$ differ, it's natural to fit a single model to each data distribution. We assume that each client has access to $|\mathcal{D}_t|$ IID data points drawn from $\mathcal{P}_t$, and client t's local dataset can be represented as $\mathcal{D}_t = \{(\mathbf{x}_j^{(t)}, y_j^{(t)})\}_{j=1}^{|\mathcal{D}_t|}$. The relationship among clients/devices is described by a graph with the adjacency matrix $\mathbf{A} = [\mathbf{A}_{uv}]$, where u and v index the clients in the graph. Let $\ell_t : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ describe the loss function corresponding to client t, and $\mathcal{L}_t(\theta_t) = \frac{1}{|\mathcal{D}_t|}\sum_j \ell_t(\mathbf{x}_j, y_j; \theta_t)$ is client t's average loss over its personal training data, where θ_t denotes the model weights of client t. The loss function varies for different tasks, e.g., *cross-entropy* is commonly used for classification while *mean square error* is preferred for forecasting. The goal is to solve the following optimization problem:

$$\Theta^* \in \arg\min_{\Theta} \frac{1}{|\mathcal{T}|}\sum_{t=1}^{|\mathcal{T}|} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{P}_t}\left[\ell_t\left(\mathbf{x}_j, y_j; \theta_t\right)\right]. \tag{1}$$

During training, the optimization is carried out on finite training samples as follows:

$$\arg\min_{\Theta} \frac{1}{|\mathcal{T}|}\sum_{t=1}^{|\mathcal{T}|}\mathcal{L}_t(\theta_t) := \frac{1}{|\mathcal{T}|}\sum_{t=1}^{|\mathcal{T}|}\frac{1}{|\mathcal{D}_t|}\sum_{j=1}^{|\mathcal{D}_t|}\ell_t\left(\mathbf{x}_j, y_j; \theta_t\right), \tag{2}$$

where $\Theta = \{\theta_t\}_{t=1}^{\mathcal{T}}$ are the set of personalized model parameters for all clients.

Remark: In this paper, we assume the existence and accessibility of a client relation graph that reflects the similarity of local models. we argue such an assumption is practical in many real-world scenarios. For example, in news recommender systems [32], clients are mobile phones containing user data, which can be connected as a graph via social networks. Intuitively, neighbors on the social network will have similar usage patterns of phones. For the cases where the graph naturally exists but is unobserved, we leave it to future work that might incorporate relation inference models [33] into our framework to estimate the graph during federated learning.

B. Our Framework

Pipeline overview: Our *Panacea* framework is composed of a GNN encoder, an MLP, and a graph generator, as shown in Fig. 1. The GNN encoder, denoted as $E(\cdot;\psi_e)$, takes the client's initial embedding and the client relationship (i.e., the adjacency matrix $\mathbf{A}$) as input. It generates informative client embeddings $\mathbf{Z}$ after multiple GNN propagation steps. The MLP, $M(\cdot;\psi_m)$, generates local model weights θ_t for each client t based on the client embeddings. Finally, a graph generator $D(\cdot)$ preserves the local preferences of each client by reconstructing the graph relation of the clients.

Training objective for the server: In *Panacea*, the GNN encoder and the MLP together is considered as the graph hypernetwork denoted as $\mathcal{GH}(\cdot;\psi) = M(\cdot;\psi_m) \circ E(\cdot;\psi_e)$, where $\psi = \{\psi_e, \psi_m\}$. To learn the hypernetwork, *Panacea* optimize the following objective:

$$\min_{\psi_e,\psi_m} \mathcal{L}_{\mathcal{GH}}(E, M) + \lambda_d \mathcal{L}_d(D, E), \tag{3}$$

where $\mathcal{L}_{\mathcal{GH}}(E, M)$ is the graph hypernetwork loss and $\mathcal{L}_d(D, E)$ is the graph reconstruction loss, and λ_d is the weight to balance these two terms. Note that the graph generator D is only involved during training to encourage the hypernetwork to preserve clients' uniqueness.

Graph hypernetwork: We use $h_t^{(L)}$ to denote the embedding of client t at the L's layer of the GNN encoder. Given client initial embedding $h_t^{(0)}$ and the graph relation of the clients $\mathbf{A}$, the graph hypernetwork outputs the local model weights for client t via $\mathcal{GH}(h_t^{(0)}, \mathbf{A}; \psi)$. Collaboratively, the graph hypernetwork learns a family of personalized local models for clients $\{\mathcal{GH}(h_t^{(0)}, \mathbf{A}; \psi) | t \in \mathcal{T}\}$. To train it, we use the mean squared error between the hypernetwork predicted model parameters and the local model parameters gathered from the clients' local updates in this communication round which is denoted as θ_t. Formally, the graph hypernetwork loss is the following,

$$\mathcal{L}_{\mathcal{GH}} = \frac{1}{2|\mathcal{T}|}\sum_{t=1}^{|\mathcal{T}|} ||\theta_t - \mathcal{GH}(h_t^{(0)}, \mathbf{A}; \psi_e, \psi_m)||_2^2. \tag{4}$$

Graph generator: We train the graph generator using a graph reconstruction loss. Specifically, we realize it with an inner-product operator, which admits the client embeddings as the input and reconstructs the graph relation of clients. The intuition is that good client embeddings should be able to preserve the "affinity" between clients. Therefore, they can inform us of the graph relation of clients. We independently sample client indices u and v from the marginal client index distribution $p(c)$ and compute the reconstruction loss as,

$$\mathcal{L}_d = \mathbb{E}_{u,v\sim p(c)}\left[-\mathbf{A}_{uv}\log\sigma\left(\hat{\mathbf{z}}_u^\intercal \hat{\mathbf{z}}_v\right)\right.$$

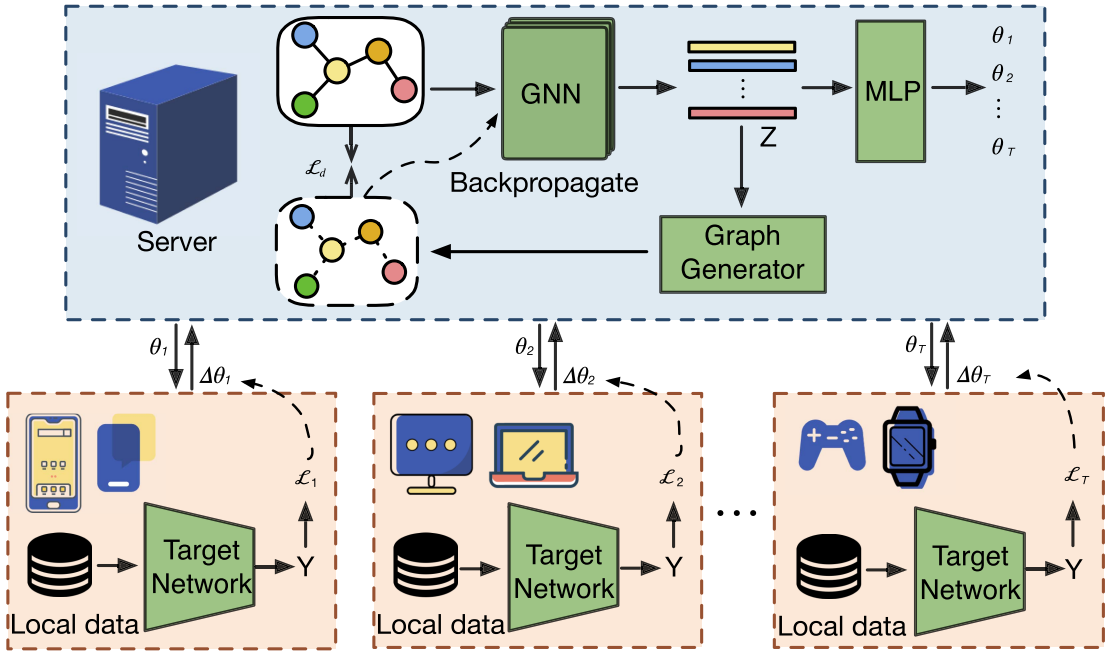


Fig. 1. Illustration of *Panacea*. The framework has two components 1) a graph hypernetwork consisting of a GNN encoder that embeds clients based on their relationships, and an MLP that generates local model weights based on the embeddings; 2) a graph generator that reconstructs the client relation graph to preserve the clients' uniqueness. Local models and hypernetwork parameters are jointly optimized using alternating optimization.

$$- (1 - \mathbf{A}_{uv}) \log \left(1 - \sigma \left(\hat{\mathbf{z}}_u^\intercal \hat{\mathbf{z}}_v\right)\right)], \tag{5}$$

where $\mathbf{z}_u = E(h_u; \psi_e)$ is the embedding of client u, $\hat{\mathbf{z}}_u$ is the transformation of $\mathbf{z}_u$ with the MLP and $\sigma(x) = \frac{1}{1+\exp^{-x}}$ is the sigmoid function.

Training objective for the clients: Each client i aims to reduce its own local task. Under the constraint that the local model's parameters are generated by the hypernetwork, we can formulate the learning objective of all clients as the following,

$$\begin{aligned} &\arg\min_{\Theta} \frac{1}{|\mathcal{T}|} \sum_{t=1}^{|\mathcal{T}|} \mathcal{L}_t(\theta_t) \\ &= \arg\min_{\Theta=\{\mathcal{GH}(h_t,\mathbf{A};\psi_e,\psi_m)\}_t} \frac{1}{|\mathcal{T}|} \sum_{t=1}^{|\mathcal{T}|} \frac{1}{|\mathcal{D}_t|} \sum_{j=1}^{|\mathcal{D}_t|} \ell_t(\mathbf{x}_j, y_j; \theta_t). \end{aligned} \tag{6}$$

Essentially, the goal of personalized federated learning is to enable each client to benefit from knowledge available from other clients to get a better approximation for their local model parameters θ_t — each client in the federation obtains a solitary model that can have a better generalization ability to unseen samples.

Joint optimization of Θ and ψ: By sharing the graph hypernetwork parameters, *Panacea* can enforce effective knowledge sharing among the clients in the federation. However, jointly optimizing formula (3) and formula (6) can be difficult due to the non-convexity of each term. Specifically, both graph hypernetwork and local models are deep neural networks. Thus optimizing them are non-convex problems. In addition, there is no explicit and stable supervision signal for learning the graph hypernetwork parameters ψ, as Θ is dynamically changing in each communication round before convergence. To facilitate the joint optimization of (3) and (6), we leverage following observations:

Observation 1: With a fixed hypernetwork ψ, the updates over Θ only depend on the local dataset $\{\mathcal{D}_t\}_{t\in\mathcal{T}}$ as illustrated in formula (6).

Observation 2: The gradient of the hypernetwork ψ only depends on Θ rather than on the client dataset $\{\mathcal{D}_t\}_{t=1}^{\mathcal{T}}$, as shown in formula (3), (4), and (5).

The above observations suggest that it is natural to use alternating optimization approach [21] to jointly solve formula (3) and formula (6) for learning the personalized models and graph hypernetwork parameters. Specifically, using the chain rule, we can have the gradient of ψ_e and ψ_m from formula (3), (4), and (5):

$$\begin{aligned} \nabla_{\psi_e} \{\mathcal{L}_{\mathcal{GH}} + \lambda_d \mathcal{L}_d\} &= \nabla_{\psi_e} \mathcal{L}_{\mathcal{GH}} + \lambda_d \nabla_{\psi_e} \mathcal{L}_d \\ &= (\nabla_{\psi_e} \theta_t)^\intercal \nabla_{\theta_t} \mathcal{L}_{\mathcal{GH}} + \lambda_d \nabla_{\psi_e} \mathcal{L}_d; \end{aligned} \tag{7}$$

$$\begin{aligned} \nabla_{\psi_m} \{\mathcal{L}_{\mathcal{GH}} + \lambda_d \mathcal{L}_d\} &= \nabla_{\psi_m} \mathcal{L}_{\mathcal{GH}} \\ &= (\nabla_{\psi_m} \theta_t)^\intercal \nabla_{\theta_t} \mathcal{L}_{\mathcal{GH}}. \end{aligned} \tag{8}$$

From formula (7), we can observe that only the first term of gradient ψ_e needs the local updates from the client. Therefore, we can simply use a general update rule $(\nabla_{\psi_e} \theta_t^\intercal) \triangle\theta_t$ to approximate the first term of ψ_e, i.e., $\triangle\psi_e = (\nabla_{\psi_e} \theta_t^\intercal) \triangle\theta_t$, where $\triangle\theta_t$ denotes the change of client t's model parameters after a round of local updates. For the second term $\mathcal{L}_{d_t}$, it does not need the gradients from the client; therefore, easily computable on the server side. Similarly, the gradient update of ψ_m can be approximated based on the update rule: $\triangle\psi_m = (\nabla_{\psi_m} \theta_t^\intercal) \triangle\theta_t$.

Algorithm 1: Panacea.

Require: α — learning rates, η — client learning rate, R — number of rounds, K_c, K_s — number of local rounds for clients and server.
Ensure: Personalized models $\{\theta_1, \theta_2, \ldots, \theta_{\mathcal{T}}\}$, and a graph hypernetwork model ψ.

```
for each communication round i ∈ [R] do
  sample a subset of clients S_t ⊂ [T]
  for each client t ∈ S_t do
    θ_t = GH(h_t^(0), A; ψ), and θ̃_t = θ_t
    for each local step k ∈ [K_c] do
      sample a mini-batch B ⊂ D_t
      θ̃_t = θ̃_t − η∇_{θ̃_t} L_t(B)
    end for
    △θ_t = θ̃_t − θ_t
  end for
  for each local step k ∈ [K_s] do
    ψ_e = ψ_e − α∇_{ψ_e} θ_t^⊺ △θ_t − αλ_d ∇_{ψ_e} L_d
    ψ_m = ψ_m − α∇_{ψ_m} θ_t^⊺ △θ_t
  end for
end for
Return the personalized models {θ̃_t}_{t∈T}, and the graph hypernetwork model ψ.
```

Extensive literature has shown the benefits of performing multiple local optimization steps per communication round in terms of both convergence rate and final accuracy [4], [7]. Thus, in *Panacea*, we perform multiple local updates. K_c and K_s steps for the client models and the hypernetowrk at the server.

Algorithm 1 demonstrates the detailed procedure of our proposed framework for learning the graph hypernetworks and the personalized local models for the clients. In each communication round, the clients download the latest personalized models from the server, then use local SGD to train K_c local steps to update the local model weights. After that, each client will upload their model updates to the server. Accordingly, the server will train K_s local steps to update the hypernetwork parameters.

Communication cost: Via the chain rule, the gradient of the hypernetwork w.r.t the local task loss can be composed as the following:

$$\nabla_{\psi}\mathcal{L}_t = (\nabla_{\psi}\theta_t)^{\intercal}\nabla_{\theta_t}\mathcal{L}_t. \tag{9}$$

Equation (9) shows that the client will only need to transmit $\nabla_{\theta_t}\mathcal{L}_t$ to the server for updating the graph hypernetwork parameters. Therefore, the communication cost is determined by the size of the local model parameter changes $\triangle\theta_t$ uploaded from the clients to the server and the size of θ_t sent from the server to the clients. It means that our Panacea causes no additional communication cost compared with traditional federated learning methods such as FedAvg [4].

C. Theoretical Analysis

In this section, we analyze the linear case of Panacea and compare it with the pFedHN to emphasize the benefit of introducing the client relation graph.

Notations: We have n clients. For each client i, data $\mathbf{x}_i \in \mathbb{R}^d$ with a dimension d follows a standard Gaussian distribution $x_i \sim \mathcal{N}(0, I_d)$ and labels y_i are generated by a ground truth linear model with a parameter θ_i^*, i.e., $y_i = \mathbf{x}_i^{\top}\theta_i^*$. We use $\Theta^* = [\theta_1^*, \ldots, \theta_n^*]$ to denote all ground truth parameters. For a model with parameter $\Theta = [\theta_1, \ldots, \theta_n]$, its expected risk at client i is $\mathcal{R}(\theta_i) := \mathbb{E}_{(x_i,y_i)}(y_i - x_i^{\top}\theta_i)^2 = \|\theta_i^* - \theta_i\|_F^2$. The averaged expected risk across clients is $\mathcal{R}(\Theta) := \frac{1}{n}\mathcal{R}(\theta_i) = \frac{1}{n}\|\Theta^* - \Theta\|_F^2$. pFedHN generate models via a linear hypernetwork with a latent dimension k and parameters $W \in \mathbb{R}^{d\times k}$ and $V := [v_1, \ldots, v_n] \in \mathbb{R}^{k\times n}$ which denote the weight of hypernetwork and the client embeddings respectively. The client parameters are generated by applying the hypernetwork to each client embedding, i.e., $\Theta = WV$. It is typically assumed that $d > n > k$. Our *Panacea* uses a GNN as the hypernetwork leading to the following decomposition of Θ as $\Theta = W_L(\cdots W_2(W_1(V\tilde{G}))\tilde{G}\cdots\tilde{G})$ where L is the number of GNN layers and in lth layer,W_l is per node transformation and $\tilde{G}$ is aggregation operation. For simplicity, we use mean aggregation, i.e., $[\tilde{G}]_{i,j} = \frac{1}{N(i)+1}1_{[j\in N(i)\cup\{i\}]}$ where $N(i) := \{j|\mathbf{A}_{ij} = 1\}$ is neighborhood of node i in the graph. We can express Θ in a compact form via $\Theta = WVG$ where $W = W_L\cdots W_1$ and $G = \tilde{G}^L$.

Optimum: As observed in the original paper of pFedHN, learning a linear hypernetwork by minimizing the expected risk $\mathcal{R}(\Theta) \propto \|\Theta^* - \Theta\|_F^2$ with $\Theta = WV$ is equivalently to solving the rank k approximation of the matrix Θ^*. Thus the optimum is achieved when it learns the top k PCA components, i.e., $\Theta_k^* = P\mathbf{diag}(\lambda_1, \ldots, \lambda_k, 0, \ldots, 0)Q^{\top}$ where $P\mathbf{diag}(\lambda_1, \ldots, \lambda_n)Q^{\top}$ is the singular value decomposing (SVD) of Θ^*. Theorem III.1 shows that *panacea*, although using a GNN, has the same optimum as long as the graph is not generated, i.e., $\mathbf{rank}(\tilde{G}) = n$.

Theorem III.1: Optimization $\min_{\Theta=WVG}\mathcal{R}(\Theta)$ has an unique minimum Θ_k^*, if $\mathbf{rank}(\tilde{G}) = n$.

Proof of Theorem III.1 We already know Θ_k^* is the unique minimizer of $\mathcal{R}(\Theta)$ under the constrain that $\mathbf{rank}(\Theta) = k$. We just need to show it is achievable. Given the fact that $\tilde{G}$ being invertible, we could easily have $WV = \Theta_k^* G^{-1}$ which gives $\Theta = \Theta_k^*$.

Robustness: In the following, we show leveraging extra knowledge of the client relations can bring us a gain of robustness. To facilitate the analysis, we will assume the ground truth model parameters can be decomposed via $\Theta^* = W^*V^*$ where $W^* \in \mathbb{R}^{d\times k}$ is the ground truth hypernetwork with elements i.i.d from $\mathcal{N}(0,1)$ and $V^* \in \mathbb{R}^{k\times n}$ is the ground truth client embeddings satisfies (1) *Consistency*: be consistent with the graph structure as $V^*G \simeq V^*$ and (2) *Gaussianity*: each entry has a Gaussian marginal $\mathcal{N}(0,1)$. For the algorithms, we assume the initialization of the model hypernetwork, W_0 and client embeddings, V_0 both have i.i.d Gaussian entries $\mathcal{N}(0,1)$. For the attack setting, label flip attack is originally defined in the context of classification and gives zero gradient in expectation since the label is randomly flipped. For simplicity, we assume

the attacked client provides no gradient during training. Further, say m clients (client 1 to m) are attacked. We call $\alpha := \frac{m}{n} < 1$, *the attack ratio*. Further, we restrict our analysis in a not too strong attack where $n - m \geq k$ implying the possibility of recovering the optimal hypernetwork with the remaining unattacked clients. Thus both pFedHN and Panacae will learns W^* (under an equivalence of orthogonal transform). So we will only analyze to optimization of clients' embeddings V. Optimization with the unattacked clients is equivalent to minimize the loss $\mathcal{R}_{\texttt{uatk}}(\Theta) = \frac{1}{n}\|(\Theta^* - \Theta)_{(m+1):n}\|_F^2$ where we use the subscripts $(m+1):n$ to denote $m+1$'th to n'th columns of the matrix. Let $\hat{\Theta}_{\texttt{mlp}}$ and $\hat{\Theta}_{\texttt{gnn}}$ be the learned model via pFedHN and Panacea, i.e., $\hat{\Theta}_{\texttt{mlp}} \in \operatorname{argmin}_{\Theta = W^* V} \mathcal{R}_{\texttt{uatk}}(\Theta)$ and $\hat{\Theta}_{\texttt{gnn}} \in \operatorname{argmin}_{\Theta = W^* V G} \mathcal{R}_{\texttt{uatk}}(\Theta)$.

Theorem III.2: For pFedHN, in expectation, its risk is $\mathbb{E}[\mathcal{R}(\hat{\Theta}_{\texttt{mlp}})] = 2dk\alpha$.

Proof of Theorem III.2: For pFedHN, it will be unable to recover the client representations for all the m lost clients. Thus it will suffered from an error $\mathcal{R}(\Theta) = \frac{1}{n}\sum_{i=1}^{m}\mathcal{R}(\theta_i) = \frac{1}{n}\|W^* v_i^* - W^* v_i\|_F^2$. In expectation, $\mathbb{E}[\|W^* v_i^* - W^* v_i\|_F^2] = \mathbb{E}[\text{tr}((v_i^* - v_i)^\top W^{*\top} W^* (v_i^* - v_i))] = 2dk$. Thus $\mathcal{R}(\Theta) = \frac{2dkm}{n} = 2dk\alpha$.

Theorem III.3: For Panacea, $\mathbb{E}[\mathcal{R}(\hat{\Theta}_{\texttt{gnn}})] = \frac{2dk}{n}\|(I - G_{(m+1):n} G^\dagger_{(m+1):n}) G_{1:m}\|_F^2$.

Proof of Theorem III.3: The full loss on all clients can be decomposed as the sum of the loss on attacked clients and the loss on unattacked clients, i.e., $\mathcal{R}(\Theta) = \mathcal{R}_{\texttt{atk}}(\Theta) + \mathcal{R}_{\texttt{uatk}}(\Theta)$ where $\mathcal{R}_{\texttt{atk}}(\Theta) := \frac{1}{n}\|(\Theta^* - \Theta)_{1:m}\|_F^2$ and $\mathcal{R}_{\texttt{uatk}}(\Theta) := \frac{1}{n}\|(\Theta^* - \Theta)_{(m+1):n}\|_F^2$. Recall that we use the subscripts to denote the column indices. For simplicity, we further denote $G_{(m+1:n)}$ and $G_{1:m}$ as P, Q. During training, the unattacked loss get minimized. Using the assumption of client embeddings's consistency with the graph, i.e., $V^* G = V^*$, we have $\mathcal{R}_{\texttt{uatk}}(\Theta) \propto \|W^*(V^* - V) G_{1:m}\|_F^2$. It is easy to see $\mathcal{R}_{\texttt{uatk}}(\Theta)$ is minimized to zero when $\|(V^* - V)P\|_F^2 = 0$. The solution might not unique depending on the rank of P. However, we use SGD to optimize V which has an implicit regularization of the deviation of the final solution $\hat{V}_{\texttt{gnn}}$ and the initialization V_0. Thus, we can have an unique solution $\hat{\Theta}_{\texttt{gnn}} = W^* \hat{V}_{\texttt{gnn}}$ with $\hat{V}_{\texttt{gnn}} := V_0 + (V^* - V_0) P P^\dagger$ where $\dagger$ indicates the Moore-Penrose inverse. The finish loss $\mathcal{R}(\hat{\Theta}_{\texttt{gnn}}) = \mathcal{R}_{\texttt{atk}}(\hat{\Theta}_{\texttt{gnn}}) = \frac{1}{n}\|W^*(V^* - \hat{V}_{\texttt{gnn}})Q\|_F^2$, which in expectation equals the following

$$
\begin{aligned}
&\mathbb{E}\left[\mathcal{R}\left(\hat{\Theta}_{\texttt{gnn}}\right)\right] \\
&= \frac{1}{n}\mathbb{E}\left[\text{tr}\left(Q^\top \left(V^* - \hat{V}_{\texttt{gnn}}\right)^\top \mathbb{E}\left[W^{*\top} W^*\right] (V^* - \hat{V}_{\texttt{gnn}}) Q\right)\right] \\
&= \frac{d}{n}\mathbb{E}\left[\| \left(V^* - \hat{V}_{\texttt{gnn}}\right) Q\|_F^2\right] \\
&= \frac{d}{n}\mathbb{E}\left[\| \left(V^* - V_0\right)\left(I - PP^\dagger\right) Q\|_F^2\right] \\
&= \frac{2dk}{n}\| \left(I - PP^\dagger\right) Q\|_F^2 \\
&= \frac{2dk}{n}\| \left(I - G_{(m+1):n} G^\dagger_{(m+1):n}\right) G_{1:m}\|_F^2 \qquad (10)
\end{aligned}
$$

Derivation from (10) to (11) is rooted from the Guassianality of V^* and V_0 and the independence between V^* and V_0.

Lemma III.4: $r(G, m) := \|(I - G_{(m+1):n} G^\dagger_{(m+1):n}) G_{1:m}\|_F^2$ denoting the distance of G's first m columns to the span of the rest columns of G. We have $0 \leq r(G, m) \leq m$. Further, the upperbound is achieved, iff $G_{1:m} = I_{1:m}$ and $G_{1:m}^\top G_{(m+1):n} = \mathbf{0}$. The lowerbound is achieved iff $\mathbf{col}(G_{1:m}) \subset \mathbf{col}(G_{(m+1):n})$.

Proof of Lemma III.4: We first recall the definition of the graph aggregation operation $\tilde{G}$ and the multi-layer aggregation $G = \tilde{G}^L$. We then point out a few properties holds for both $\tilde{G}$ and G. Finally, we prove the necessary and sufficient condition for the lowerbound and upperbound.

Definition III.5 (Graph aggregation operation). $\tilde{G}$ is a simple mean aggregation which averages embeddings according the graph adjacent matrix $\mathbf{A}$, i.e., $[\tilde{G}]_{i,j} = \frac{1}{N(i)+1} 1_{[j \in N(i) \cup \{i\}]}$ where $N(i) := \{j | \mathbf{A}_{ij} = 1\}$. $G := \tilde{G}^L$ is operation of aggregating L times.

Property III.1: (Value range) By definition, all elements in $\tilde{G}$ and G are non-negative and in the range of [0,1].

Property III.2: (Column summation) By definition, summation of any column of $\tilde{G}$ is one, i.e., $\mathbf{1}^\top \tilde{G} = \mathbf{1}^\top$. The property also holds for G since $\mathbf{1}^\top G = \mathbf{1}^\top \tilde{G} \cdots \tilde{G} = \mathbf{1}^\top$.

Property III.3: (Norm) We denote G's columns as g_i, i.e., $G = [g_1, \ldots, g_n]$. We have $\|g_i\|_2^2 \in [1/n, 1]$.

Proof of property 3: For the norm of g_i is lower bounded via Jensen's inequality as

$$
\|g_i\|^2 = \sum_{i,j} g_{i,j}^2 \geq n \left(\sum_j g_{i,j}/n\right)^2 = 1/n \qquad (11)
$$

For the norm of g_i is upper bounded due to the non-negativity of g_i's elements,

$$
\|g_i\|^2 = \sum_{i,j} g_{i,j}^2 \leq \left(\sum_j g_{i,j}\right)^2 = 1 \qquad (12)
$$

□

Proof of the lower-bound in Lemma First, it is trivial that $\|(I - G_{(m+1):n} G^\dagger_{(m+1):n}) G_{1:m}\|_F^2 \geq 0$. Second, $\|(I - G_{(m+1):n} G^\dagger_{(m+1):n}) G_{1:m}\|_F^2 = 0 \iff G_{1:m} = G_{(m+1):n} G^\dagger_{(m+1):n} G_{1:m} \iff \mathbf{col}(G_{1:m}) \subset \mathbf{col}(G_{(m+1):n})$. □

Proof of the upper-bound in Lemma Because, $G_{(m+1):n} G^\dagger_{(m+1):n} G_{1:m}$ is the projection of $G_{1:m}$ in the column space $G_{(m+1):n} G^\dagger_{(m+1):n}$, we have $\|(I - G_{(m+1):n} G^\dagger_{(m+1):n}) G_{1:m}\|_F^2 \leq \|G_{1:m}\|_F^2$. The equality holds when $G_{1:m}^\top G_{(m+1):n} = \mathbf{0}$. By property 3, we know $\|g_i\|_2^2 \leq 1$ for any

$i \in [m]$. Thus $\|G_{1:m}\|_F^2 \leq m$. The equality holds when each g_i is one-hot vector. Further since $g_{i,i} > 0$ by definition, g_i has to equal e_i which is the i'th standard basis vector. □

Based on the theorems, we make a few remarks: ❶ Panacea has a smaller expected risk than pFedHN, i.e., $\mathbb{E}[\mathcal{R}(\hat{\Theta}_{\texttt{gnn}})] \leq \mathbb{E}[\mathcal{R}(\hat{\Theta}_{\texttt{mlp}})]$, due to the fact that $r(G, m) \leq m$ from lemma III.4. ❷ pFedHN's loss is linearly scale with the attack ratio α while Panacea does not. ❸ Instead, Panacea's risk is related to the graph structure. We discuss two extreme cases: (1) *Empty graph* with no edges which is totally non-informative. Then G becomes the identity I which leads to Panacea degenerate to pFedHN with $\mathbb{E}[\mathcal{R}(\hat{\Theta}_{\texttt{gnn}})] = \mathbb{E}[\mathcal{R}(\hat{\Theta}_{\texttt{mlp}})]$; (2) *Clique graph* with all nodes are connected, which is highly informative, since knowing one client's embedding equals to knowing all of them. Thus we can always fully recover the client embeddings from the unattacked ones. Mathematically, $G = \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ which leads $r(G, m) = 0$ for any $m < n$ implying $\mathbb{E}[\mathcal{R}(\hat{\Theta}_{\texttt{gnn}})] = 0$ from Theorem III.3.

IV. Empirical Studies

We evaluate *Panacea* and all baseline methods across various tasks from different application domains.

1) Synthetic data for classification: FL is a synthetic binary classification for federated learning. We borrow the synthetic data distribution from a prior domain adaptation work [34]. For each client t, a 2-dimensional unit vector $[a_t, b_t]$ was randomly generated as the client embedding, and the angle of the unit vector was represented as $\omega_t = \arcsin(\frac{b_t}{a_t})$. Positive samples $(\mathbf{x}, 1, t)$ and negative samples $(\mathbf{x}, 0, t)$ are sampled from two different 2-dimensional Gaussian distributions, $\mathcal{N}(\mu_{t,1}, \mathbf{I})$ and $\mathcal{N}(\mu_{t,0}, \mathbf{I})$, respectively, where $\mu_{t,1} = [\frac{\omega_t}{\pi}a_t, \frac{\omega_t}{\pi}b_t]$ and $\mu_{t,0} = [-\frac{\omega_t}{\pi}a_t, -\frac{\omega_t}{\pi}b_t]$. Then we construct the client relation graph with a Bernoulli distribution, i.e., $\mathbf{A}_{uv} \sim \mathbf{Bern}(0.5a_u a_v + 0.5b_u b_v + 0.5)$. The generation process ensures that the datasets of all clients are non-IID and the adjacent clients in the graph have similar decision boundaries for classification.

2) Car image data for classification: Comprehensive Cars (CompCars) [35] contains 136,726 images of cars with labels including 4 car types (MPV, SUV, sedan, and hatchback), 5 viewpoints (front (F), rear (R), side (S), front-side (FS), and rear-side (RS)), and years of manufacture (YOMs, ranging from 2009 to 2014). We follow the data splitting from [34], and each client has car images only from one viewpoint and one YOM. The task is to predict the car type based on the image. Two clients are connected if either their viewpoints or YOMs are identical/nearby. For example, client A and B are connected if A's YOM is 2009, and B's is 2010.

3) State network temperature data for regression: TPT-48 is a real-world dataset for temperature prediction. It contains the monthly average temperature over 48 contiguous states in the US from 2008 to 2019. We use the data processed by Washington Post [36] [1]. The task is to forecast the next 6 months' temperature given the previous first six months' temperature.

[1]The raw data is from the National Oceanic and Atmospheric Administration's Climate Divisional Database (nClimDiv) and Gridded 5KM GHCN-Daily Temperature, and Precipitation Dataset (nClimGrid) [37].

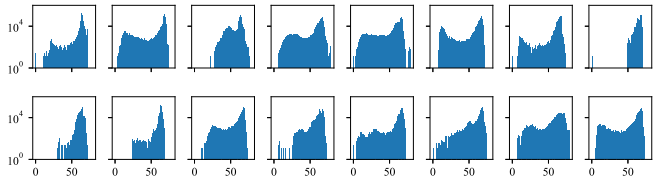


Fig. 2. Illustration of data heterogeneity on road network traffic dataset PEMS-BAY. We randomly selected 16 sensors and visualized the distributions of their data values.

Due to the diverse geographical environments, the collected temperature datasets from various states are inherent non-IID. The geographically adjacent states form links in the graph.

4) Road network traffic data for forecasting: PEMS-BAY and *METR-LA* [9], [34] are two datasets constructed based on the traffic data collected by sensors in the road network. Specifically, *PEMS-BAY* contains the traffic speed readings from 325 sensors in the Bay Area over six months, from January 1st, 2017, to May 31st, 2017. *METR-LA* contains the traffic speed readings from 207 loop detectors in Los Angeles County over 4 months from March 1st, 2012, to June 30th, 2012. To visualize the statistical heterogeneity, we show the histograms of traffic speed from different sensors of PEMS-BAY in Fig. 2. We also list the statistics of PEMS-BAY and METR-LA in Table I.

A. Experiment Settings

Baselines: We compare *Panacea* against various personalized federated learning (PFL) approaches. Our baselines include *Hypernetwork-based* methods: *pFedHN* [7], which treats each client equally and independently and realizes the hypernetwork with a MLP. *Graph-based* methods: *SFL* [31] that enhance personalization by using graph relation and doing information aggregation from nearby clients. *Standard* FL methods: 1) *FedAvg* [4], the classical federated learning algorithm based on federated averaging; 2) *Per-FedAvg* [15] that personalizes client models via meta-learning; 3) *pFedMe* [20] and *Ditto* [23] improve personalization by adding a regularization term in the objective function; *FedAvg-finetune* [38] that realizes personalization by incorporating fine-tuning over federated averaging.

For the implementation of baselines, we use the original implementations released by the authors if available; otherwise, we re-implement them by referring to the original paper.

FedAvg [4]: We re-implemented FedAvg by referring to the original paper. On the client side, the clients pull the global model from the server and train it using their own local data. The updated model weights are then uploaded back to the server. On the server side, a subset of clients is randomly selected to perform local training, and their model weight updates are aggregated to generate the new global model.

Per-FedAvg [15]: We utilized a third-party's PyTorch implementation at https://github.com/KarhouTam/Per-FedAvg , which claims to be implemented based on the source code shared by the original authors. On the client side, Per-FedAvg trains the local model with Hessian-Free MAML (Model-Agnostic Meta-Learning). On the server side, it performs the same aggregation process as FedAvg.

TABLE I
STATISTICS OF *PEMS-BAY* AND *METR-LA*

DATASET	# OF CLIENTS	# OF EDGES	# OF TRAIN SEQ. PER CLIENT	# OF VAL SEQ. PER CLIENT	# OF TEST SEQ. PER CLIENT
PEMS-BAY	325	2,369	36,465	5,209	10,419
METR-LA	207	1,515	23,974	3,425	6,850

pFedMe [20]: We use the original implementation at https://github.com/CharlieDinh/pFedMe. Different from other baselines, pFedMe maintains a personalized model on the client side and uses the Moreau envelope function to help decompose the personalized model optimization from global model learning. On the server side, pFedMe also adopts the same average aggregation as FedAvg.

Ditto [23]: We use the original implementation at https://github.com/s-huu/Ditto. Similar to pFedMe, Ditto maintains a personalized model on the client side but differs in its optimization approach. Instead of using the Moreau envelope to decouple personalized and global model learning, Ditto employs a simple L2-regularization term to align the personalized model with the global one.

SFL [31]: We use the original implementation at https://github.com/dawenzi098/SFL-Structural-Federated-Learning. To prevent the local model from deviating significantly from the global model, SFL introduces an additional regularization term during the client's local training process. For server aggregation, SFL aggregates the clients' weights from their neighbors and generates the global model by averaging the aggregated weights of all clients.

pFedHN [7]: We use the original implementation at https://github.com/AvivSham/pFedHN. In pFedHN, the client performs the local training in the same way as in FedAvg. However, the model weights that clients pull from the server are generated by a hypernetwork. On the server side, the hypernetwork is trained to share the knowledge between clients while also personalizing the model for each individual client. Unlike Panacea, pFedHN treats each client equally and independently and realizes the hypernetwork with an MLP.

Implementation of Panacea: Our framework consists of three modules: a GNN encoder, an MLP, and a graph generator (see Fig. 1 in Section III). For evaluation, we use a 3-layer GNN encoder with hidden dimension 100 and a 3-layer MLP for generating local model parameters. The graph generator is instantiated with an inner-product operator. The client embedding dimension was fixed to 100 for all datasets.

For the binary classification/regression tasks, we realize the target network model with a 3-layer MLP for each client, where the model's hidden dimension is 16. For the forecasting tasks with traffic datasets, we reuse the gated recurrent unit (GRU) model from [29], [39], which has 63 K parameters and it is a 1-layer GRU with hidden dimension 100. For the image classification task with CompCars dataset, we use the ResNet18 [40] model and the weights pre-trained with ImageNet dataset from torchvision [41] as the initial model.

Hyperparamters: In all methods, we use a batch size of 64. Unless otherwise stated, we applied a grid search for following hyperparameters: the learning rate of graph hypernetworks was tuned amongst $\{0.001, 0.003, 0.01, 0.03, 0.1\}$, the learning rate of target networks was tuned amongst $\{0.001, 0.003, 0.01, 0.03, 0.1\}$, and the coefficient λ_d was searched in $\{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1\}$. We used the SGD optimizer [42] for all client local updates. In each communication round, the number of client local steps K_c is set as 50, and the number of server local steps K_s is set as 10. For all algorithms, we limit the training process to 100 server-client communication rounds for CompCars dataset, and 800 rounds for other datasets. At each round, we randomly select 5 clients to participate in the training. For the baseline approaches, we set their hyperparameters as reported in the corresponding paper [4], [7], [15], [20], [31]. Without specification, we report the results under the hyperparameters with the best performance overall.

Hardware: We implemented our framework and all baseline approaches in PyTorch 1.12. All clients and servers are simulated on a workstation with 2 x AMD Milan 7413 @ 2.65 GHz CPU, 4 x NVIDIA A100 GPU with 40 GB memory, and 3.84 TB SSD.

Evaluation metrics: The evaluation for *Panacea* is geared towards personalized model performance, generalization capability to unseen clients, and robustness to malicious clients under label-flipping attacks. The evaluating metric of predictive performance varies in different learning tasks. We evaluate the performance with the classification accuracy of the held-out test set averaged over all clients for classification tasks and the Mean Square Error (MSE) for regression and forecasting tasks, respectively. To evaluate the generalization capability to unseen clients, we hold out 20% of the clients as novel clients and evaluate corresponding model performance, i.e., classification accuracy or MSE. We introduce malicious clients under various attack ratios during training to assess robustness. Specifically, the malicious clients randomly flip the labels of their local data samples for training. In each round, they report the jamming local-updates to the server, attempting to disrupt the entire federated training. The attack ratio ranges from 0% to 50%. All the reported results, e.g., mean and standard deviation, are calculated over five independent runs.

As for the data scale, FL-60 has 60 clients for the synthetic binary classification task where each client has 100 samples. TPT-48 has 48 clients and each client has 125 samples. In forecasting tasks, each sensor/detector is treated as a client; there are 325 clients in PEMS-BAY and 207 clients in METR-LA, respectively. For the image classification task, there are 30 clients (5 viewpoints $\times$ 6 YOMs) and 24,151 images in total. Moreover,

TABLE II
PREDICTIVE PERFORMANCE COMPARISONS OVER ALL DATASETS: WE REPORT CLASSIFICATION ACCURACY FOR THE CLASSIFICATION TASKS AND MSE FOR THE REGRESSION AND FORECASTING TASKS

	FL-60 (%)	CompCars (%)	TPT-48 (10^{-3})	METR-LA (10^{-3})	PEMS-BAY (10^{-3})
FedAvg [4]	97.9±9.7	88.2±1.3	2.6±0.9	112.4±43.2	33.5±38.5
FedAvg-finetune	**100.0±0.0**	**90.0±5.8**	2.6±1.0	20.0±7.8	13.2±12.9
Per-FedAvg [15]	87.7±24.2	66.0±1.8	2.6±0.9	289.5±89.9	63.7±70.4
pFedMe [20]	**100.0±0.0**	72.2±0.9	2.5±0.8	315.8±81.3	75.7±126.1
Ditto [23]	**100.0±0.0**	89.0±4.5	**2.1±0.5**	71.5±23.0	42.8±30.8
SFL [31]	73.3±33.7	77.0±7.9	2.9±1.0	73.2±23.0	36.4±27.3
pFedHN [7]	84.4±24.8	77.4±8.2	2.7±1.1	67.3±21.4	33.2±24.8
Panacea	**100.0±0.0**	88.2±1.2	2.6±0.7	**1.0±0.2**	**0.4±0.4**

Value in **bold** denotes the best result, and value with underline denotes the second-best result.

TABLE III
GENERALIZATION CAPABILITY COMPARISONS TO NOVEL CLIENTS: WE REPORT CLASSIFICATION ACCURACY FOR THE CLASSIFICATION TASKS AND MSE FOR THE REGRESSION AND FORECASTING TASKS

	FL-60 (%)	CompCars (%)	TPT-48 (10^{-3})	METR-LA (10^{-3})	PEMS-BAY (10^{-3})
FedAvg [4]	99.2±2.9	86.8±1.7	2.8±1.0	109.7±38.3	30.7±31.7
FedAvg-finetune	**100.0±0.0**	83.0±2.7	**2.8±0.8**	19.7±9.7	12.3±6.6
Per-FedAvg [15]	80.8±22.3	66.4±3.1	2.8±1.0	274.8±80.1	59.4±64.2
pFedMe [20]	42.5±37.5	72.0±2.2	3.5±1.5	305.3±83.2	98.5±97.5
Ditto [23]	95.0±10.0	81.5±7.0	2.9±0.9	68.0±24.3	38.1±22.8
SFL [31]	62.5±37.0	77.5±7.0	3.3±1.2	70.8±24.9	33.7±19.3
pFedHN [7]	80.0±25.6	74.6±7.4	3.0±1.0	64.5±23.0	31.1±17.5
Panacea	**100.0±0.0**	**87.8±1.8**	**2.8±0.9**	**1.0±0.3**	**0.4±0.5**

Value in **bold** denotes the best result, and value with underline denotes the second-best result.

all datasets in each client are randomly split into 80% / 20% for training / testing.

B. Results and Discussion

Averaged predictive performance across clients: Table II reports the average accuracy or MSE and variance across clients. *Panacea* outperforms other baselines on the more challenging and practical real-world datasets METR-LA and PEMS-BAY, achieving the best results with **1.0±0.2** and **0.4±0.4**, respectively. Additionally, *Panacea* achieves on par performance in CompCars and TPT-48, and matches the best performance in FL-60. These results demonstrate the robustness and effectiveness of *Panacea* across various datasets. FedAvg-finetune, Ditto, and pFedHN also achieved competitive performance across multiple datasets. FedAvg-finetune attained slightly better accuracy in CompCars, demonstrating its effectiveness in classification tasks. Though Ditto performs well in FL-60, CompCars, and TPT-48, it achieves worse performance in both MERT-LA and PEMS-BAY compared to ours. These results highlight the strengths of *Panacea*, especially in challenging real-world datasets.

Generalization to novel clients: We also conduct experiments to evaluate the generalization capability of different federated learning algorithms. Specifically, we study an important learning setup where new clients join. In general, if models are shared across clients, new clients joining in would require fine-tuning the shared model, such as pFedMe and Per-FedAvg. In contrast, it may not need fine-tuning or retraining in the federated learning framework based on hypernetworks, including pFedHN and *Panacea*. To verify the generalization capability of hypernetwork-based approaches, we take the client's initial embedding as the hypernetwork input to generate corresponding personalized models without fine-tuning and then evaluate the clients' averaged predictive performance with local test data. The results are shown in Table III.

We observe that *Panacea* exhibits the best generalization capability all over the datasets. This demonstrates its strong ability to generalize to novel clients. pFedHN also shows competitive performance, outperforming non-hypernetwork-based methods like pFedMe, Per-FedAvg, and SFL, particularly in PEMS-BAY. This can be explained by the fact that the federated hypernetwork framework essentially learns a meta-model over the distribution of clients with the proposed hypernetwork, thus achieving better generalization to novel clients. FedAvg-finetune achieves competitive generalization performance by finetuning the model on new clients with their local data, while other baselines all have significant performance drops on unseen clients.

The visualization of convergence speed and task performance: Fig. 3(a)–(e) shows the task performance during training with different personalized federated learning algorithms. We have the following observations: 1) Both SFL and Per-FedAvg show poor performance in the entire training process on FL-60; 2) Except for the FL-60 dataset, *Panacea* has the fastest convergence speed and exhibits comparable performance compared to other

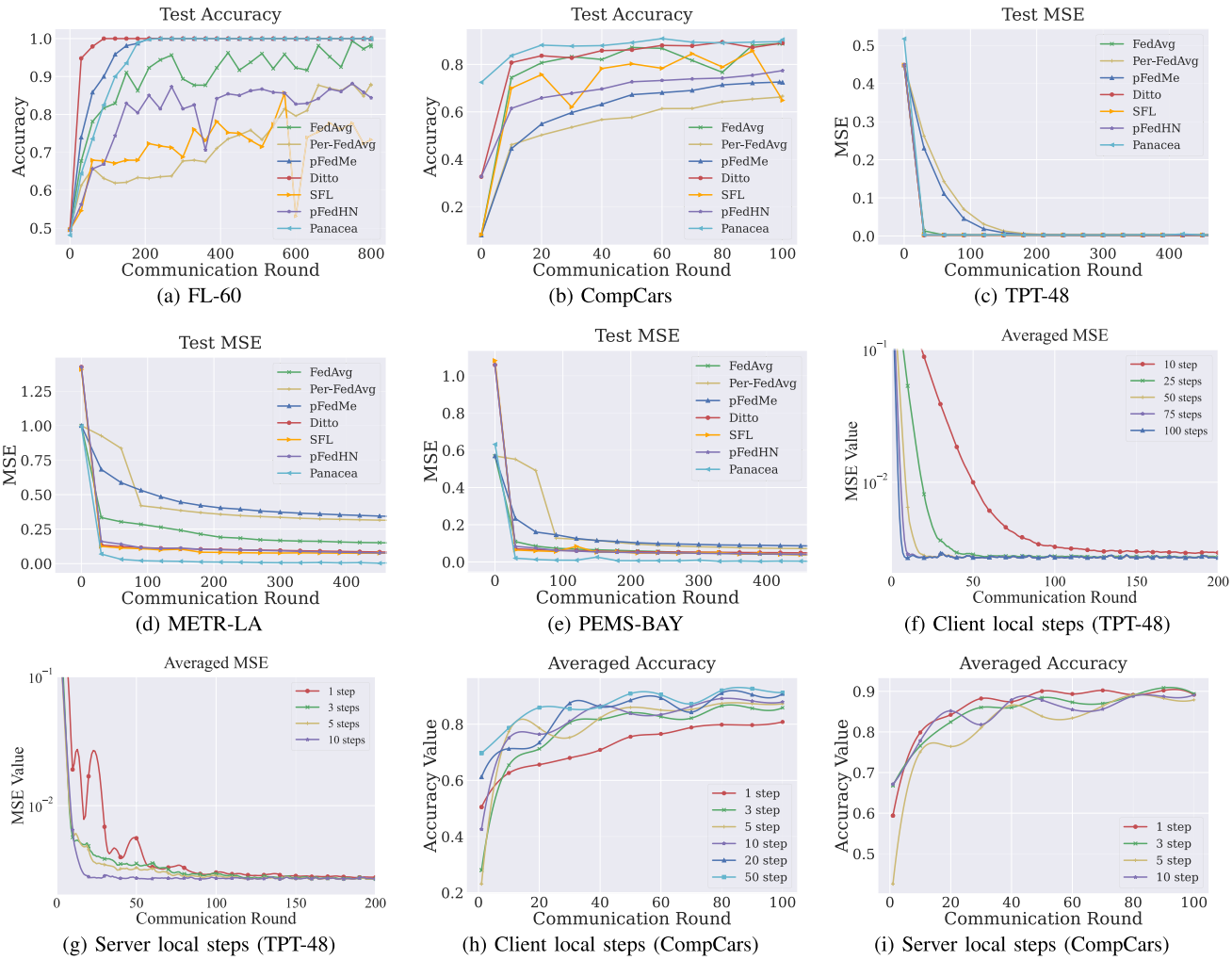


Fig. 3. (a)-(e) compare convergence speed and final performance, and (f)-(i) compare final performance of Panacea with various hyperparameter settings.

baselines; 3) both pFedMe and Per-FedAvg perform poorly on two real-world road network traffic datasets (PEMS-BAY shows a similar trend as METR-LA); 4) Although both *Panacea* and pFedHN are hypernetwork-based methods, *Panacea* achieves significantly faster and more stable convergence. This highlights the effectiveness of its graph encoder and graph generator in enhancing performance.

TABLE IV
PREDICTIVE PERFORMANCE

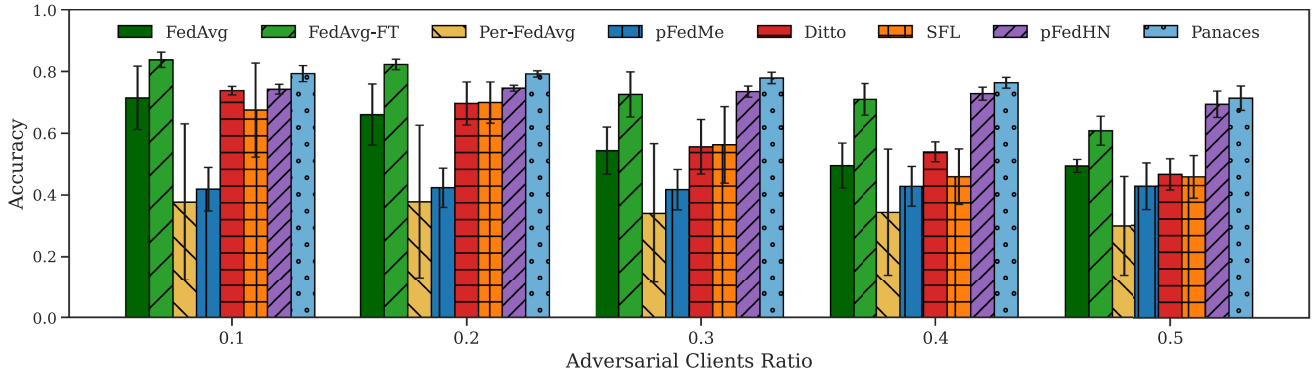
	CompCars(↑)	METR-LA(↓)	PEMS-BAY(↓)
Panacea-MLP	85.7±2.1	3.2±1.7	3.6±4.9
Panacea-GN	**89.1±1.2**	3.2±1.4	1.2±0.4
Panacea	88.2±1.2	**1.0±0.2**	**0.4±0.4**

Ablation study on *Panacea* architecture design. Value in **bold** denotes the best result.

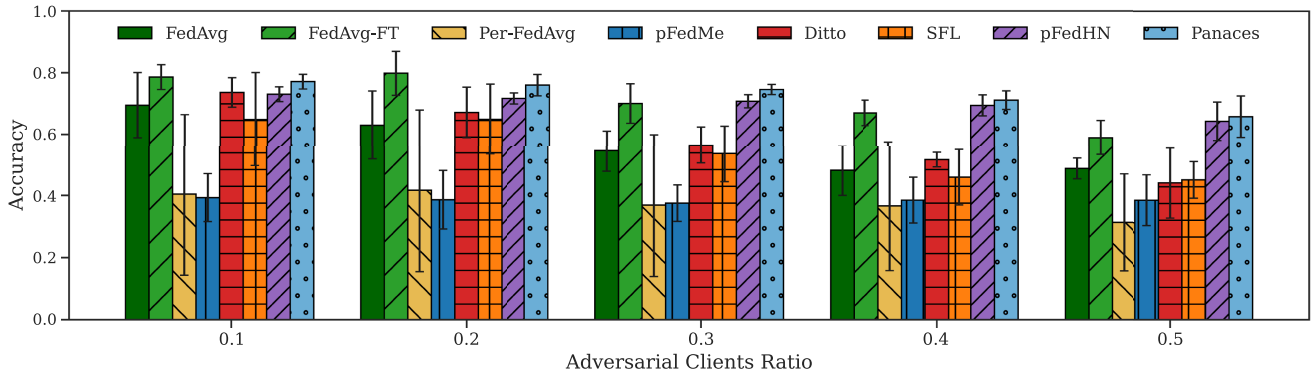
C. Ablation Study

Ablation on Panacea*architecture design:* To illustrate the necessity of incorporating the graph generator module, we evaluate a variant of our framework with the GNN encoder and the MLP module only, called Panacea-*GN* and Panacea-*MLP* respectively. The client embedding dimension was fixed to 100 for all datasets. The results are shown in Table IV. We can observe that *Panacea* achieves significantly better performance than *Panacea*-GN for the forecasting tasks on two road network traffic datasets, indicating the necessity of incorporating the graph generator module. In contrast, *Panacea*-GN performs slightly better than *Panacea* for the regression task on TPT-48 and image classification on CompCars. It is interesting that the performance gap between *Panacea* and *Panacea-GN* is related to the number of clients. CompCars have a smaller number of clients (30) and show little gap between *Panacea* and *Panacea-GN*. On the other hand, Panacea is significantly better than *Panacea*-GN on datasets with more clients (like from 60 to 325). We hypothesize that when the graph is small, our framework might overfit the knowledge of the graph which can hurt the performance of the clients.

Ablation on local steps of clients/server: We also examine the effect of performing local optimization steps for the clients and server. Fig. 3(f)–(i) shows the performance on TPT-48



(a) Test accuracy of benign clients under model poisoning attack



(b) Test accuracy of unseen clients under model poisoning attack

Fig. 4. Compare final performance under various attack ratios on CompCars.

and CompCars throughout the training process for both clients and the server. Specifically, Fig. 3(f) and (h) compare training using the chain rule with various K_c while fixing the server step $K_s = 10$. Using our proposed update rule, i.e., making multiple local update steps yields significant improvements in convergence speed and final accuracy when setting $K_c = 50$. Fig. 3(g) and (i) compare training the graph hypernetwork using the chain rule with various K_s while fixing the client step $K_c = 50$. When setting $K_s = 1$, the performance is poor. We observe that *Panacea* achieves the best in convergence speed and predictive performance when $K_s = 10$. As stated above, we set $K_c = 50$ and $K_s = 10$ while comparing with the baselines.

D. Robustness Evaluation Under Various Attacks

To evaluate the robustness of our method against adversarial attacks in federated learning, we conduct robustness evaluations over model poisoning and backdoor attacks, respectively.

Model poisoning attacks: For the model poisoning attack, an attacker manipulates the training process by uploading incorrect gradients. We implement this attack using the *local model poisoning* [43] strategy, where the adversarial gradient $\mathbf{g}'$ is computed as: $g' = -\lambda g$, where λ is set to 0.5in our experiments. Fig. 4 depicts the predictive performance of all baselines under various model poisoning attack ratios of malicious clients, from 10% to 50%. We can observe *Panacea* consistently outperforms most of the baselines even under a very high attack ratio of model poisoning attacks, verifying its resilience to model poisoning attacks.

TABLE V
COMPARISON OF DIFFERENT ALGORITHMS PERFORMANCE UNDER BACKDOOR ATTACKS WITH AND WITHOUT SIMPLE-TUNING

Algorithm	Original		+Simple-Tuning	
	ASR (↓)	Acc (↑)	ASR (↓)	Acc (↑)
FedAvg [4]	91.0	84.3	67.2	**83.2**
Ditto [23]	84.1	**90.9**	66.2	82.4
Panacea	**49.0**	76.0	**47.1**	82.5

ASR denotes attack success rate (the lower is better), and Acc denotes accuracy (the higher is better).

Backdoor attacks: We choose *Blended* [44] as a backdoor attack method, which is the most effective attack method reported in [45]. We randomly choose one client as the attacker and put the hello-kitty pattern on 50% of training samples with the blending ratio α of 0.2. We set the target label $y_t = 1$ for blended samples. We adopt Simple-Tuning [45] for all baselines, which is one of the most effective defense methods against backdoor attacks. Specifically, we reinitialize the parameters of the classifier layer and retrain it on the client with local data. As Table V shows, with Simple-Tuning, all baselines exhibit better robustness — with a lower attack success rate (ASR)

under the backdoor attacks. *Panacea* achieves 82.5% accuracy while keeping the attack success rate at 47.1%, which is substantially lower than the other baselines, whose ASR remains above 66%.

Limitations: Our work sheds light on and facilitates the development of personalized federated learning based on graph hypernetworks. However, *Panacea* assumes that the graph relation of clients is prior; one limitation is that graph relation may not be explicitly observed. Moreover, though our framework can enhance the predictive performance, generalization capability, and robustness in one go, how incorporating graph relation into the federated learning procedure impact the fairness metrics of learned models is unknown. Specifically, how do clients with biased datasets in our framework impact the experienced fairness of other clients on their local distributions? We leave the above issues as our future work.

V. Concluding Remarks

We introduced a new federated learning framework called *Panacea* for generating personalized models based on federated graph hypernetworks. The core of *Panacea* is the composition of a graph hypernetwork and a graph generator. *Panacea* has several advantages: it is agnostic to the target network architecture, can reinforce the collaboration among adjacent clients and generate personalized models across various application domains with better predictive performance, enjoys better generalization capability to unseen clients during training, is more resilient to malicious clients under label-flipping attacks. Extensive experiments on synthetic and real-world datasets have demonstrated the rationality and effectiveness of using *Panacea*, resulting in a new state-of-the-art for generating personalized models in federated learning settings.

References

[1] S. AbdulRahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani, "A survey on federated learning: The journey from centralized to distributed on-site learning and beyond," *IEEE Internet Things J.*, vol. 8, no. 7, pp. 5476–5497, Apr. 2021.

[2] P. Voigt and A. Von dem Bussche, "The EU general data protection regulation (GDPR)," in *A Practical Guide*, 1st Ed., Cham, Switzerland: Springer, 2017.

[3] P. Kairouz et al., "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, no. 1/2, pp. 1–210, 2021.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[5] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Gener. Comput. Syst.*, vol. 115, pp. 619–640, 2021.

[6] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, "Towards personalized federated learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 12, pp. 9587–9603, Dec. 2023.

[7] A. Shamsian, A. Navon, E. Fetaya, and G. Chechik, "Personalized federated learning using hypernetworks," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 9489–9502.

[8] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[9] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proc. Int. Conf. Learn. Representations*, 2018.

[10] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," in *Proc. Int. Conf. Learn. Representations*, 2020.

[11] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of FedAvg on Non-IID data," in *Proc. Int. Conf. Learn. Representations*, 2019.

[12] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Mach. Learn. Syst.*, 2020, pp. 429–450.

[13] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with Non-IID data," 2018, *arXiv: 1806.00582*.

[14] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5132–5143.

[15] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning: A meta-learning approach," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 3557–3568.

[16] F. Chen, M. Luo, Z. Dong, Z. Li, and X. He, "Federated meta-learning with fast convergence and efficient communication," 2018, *arXiv: 1802.07876*.

[17] Y. Jiang, J. Konevcny, K. Rush, and S. Kannan, "Improving federated learning personalization via model agnostic meta learning," 2019, *arXiv: 1909.12488*.

[18] M. Khodak, M.-F. F. Balcan, and A. S. Talwalkar, "Adaptive gradient-based meta-learning methods," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 5915–5926.

[19] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 3557–3568.

[20] C. T Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with moreau envelopes," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 21394–21405.

[21] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4424–4434.

[22] O. Marfoq, G. Neglia, A. Bellet, L. Kameni, and R. Vidal, "Federated multi-task learning under a mixture of distributions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 15434–15447.

[23] T. Li, S. Hu, A. Beirami, and V. Smith, "Ditto: Fair and robust federated learning through personalization," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 6357–6368.

[24] C. Ma, V. Smith, M. Jaggi, M. Jordan, P. Richtárik, and M. Takác, "Adding vs. averaging in distributed primal-dual optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1973–1982.

[25] C. Zhang, M. Ren, and R. Urtasun, "Graph HyperNetworks for neural architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019.

[26] A. Brock, T. Lim, J. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through HyperNetworks," in *Proc. Int. Conf. Learn. Representations*, 2018.

[27] E. Nachmani and L. Wolf, "Molecule property prediction and classification with graph hypernetworks," 2020, *arXiv: 2002.00240*.

[28] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "The limitations of federated learning in sybil settings," in *Proc. 23rd Int. Symp. Res. Attacks, Intrusions Defenses*, San Sebastian, USENIX Association, 2020, pp. 301–316.

[29] C. Meng, S. Rambhatla, and Y. Liu, "Cross-node federated graph neural network for spatio-temporal data modeling," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2021, pp. 1202–1211.

[30] C. He et al., "FedGraphNN: A federated learning system and benchmark for graph neural networks," in *Proc. Workshop Graph Neural Netw. Syst.*, 2021.

[31] F. Chen, G. Long, Z. Wu, T. Zhou, and J. J. Jiang, "Personalized federated learning with a graph," in *Proc. Int. Joint Conf. Artifical Intell.*, 2022, pp. 2575–2582.

[32] J. Liu, P. Dolan, and E. R. Pedersen, "Personalized news recommendation based on click behavior," in *Proc. 15th Int. Conf. Intell. User Interfaces*, 2010, pp. 31–40.

[33] X. Peng, Y. Li, and K. Saenko, "Domain2Vec: Domain embedding for unsupervised domain adaptation," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2020, pp. 756–774.

[34] Z. Xu, H. He, G.-H. Lee, B. Wang, and H. Wang, "Graph-relational domain adaptation," in *Proc. Int. Conf. Learn. Representations*, 2021.

[35] L. Yang, P. Luo, C. Change Loy, and X. Tang, "A large-scale car dataset for fine-grained categorization and verification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3973–3981.

[36] "Climate change in the contiguous United States," 2020. [Online]. Available: https://github.com/washingtonpost/data-2C-beyond-the-limit-usa

[37] R. Vose et al., "Gridded 5KM GHCN-Daily Temperature and Precipitation Dataset (nCLIMGRID) Version 1," Maximum Temperature, Minimum Temperature, Average Temperature, and Precipitation, 2014.
[38] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, "Exploiting shared representations for personalized federated learning," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2021, pp. 2089–2099.
[39] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2014, pp. 1724–1734.
[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[41] T. maintainers and contributors, "TorchVision: PyTorch's computer vision library," 2016. [Online]. Available: https://github.com/pytorch/vision
[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015.
[43] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to {Byzantine-Robust} federated learning," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 1605–1622.
[44] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," 2017, *arXiv: 1712.05526*.
[45] Z. Qin, L. Yao, D. Chen, Y. Li, B. Ding, and M. Cheng, "Revisiting personalized federated learning: Robustness against backdoor attacks," in *Proc. 29th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2023, pp. 4743–4755.

Wanyu Lin (Member, IEEE) received the BEngr degree from the School of Electronic Information and Communications, Huazhong University of Science and Technology, China, the MPhil degree from the Department of Computing, The Hong Kong Polytechnic University, and the PhD degree from the Department of Electrical and Computer Engineering, University of Toronto. Her research interests include collaborative AI, trustworthy AI, and AI for Science. She has served as associate editor for *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*. She is a member of ACM.

Hao Lan received the BE and ME degrees from the School of Telecommunications Engineering, Xidian University, in 2015 and 2018, respectively, and the PhD degree from the Department of Electrical & Computer Engineering, University of Toronto. His research interests include distributed machine learning, deep reinforcement learning, graph machine learning, and model interpretability.

Hao He received the BS degree from the School of Electrical Engineering and Computer Science, Peking University, China, and the PhD degree from the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology. His research focuses on the application of machine learning to healthcare, with a particular emphasis on sleep science. He is a member of the Society for Neuroscience and a recipient of the Takeda Fellowship and the Barbara J. Weedon Fellowship.

Baochun Li (Fellow, IEEE) received the BEngr degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995, and the MS and PhD degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000. Since 2000, he has been with the Department of Electrical and Computer Engineering, University of Toronto, where he is currently a professor. He holds the Bell Canada Endowed Chair in computer engineering since August 2005. His research interests include cloud computing, distributed systems, datacenter networking, and wireless systems. He was the recipient of the *IEEE Communications Society* Leonard G. Abraham Award in the Field of Communications Systems in 2000. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the *IEEE Communications Society*, and a recipient of the University of Toronto McLean Award. He is a member of ACM.