Personalized Federated Learning With Server-Side Information

JAEHUN SONG, MIN-HWAN OH, AND HYUNG-SIN KIM (Member, IEEE)
Graduate School of Data Science, Seoul National University, Seoul 08826, Republic of Korea
Corresponding author: Hyung-Sin Kim (hyungkim@snu.ac.kr)

This work was supported in part by Creative-Pioneering Researchers Program through Seoul National University, and in part by NA VER.

ABSTRACT Personalized Federated Learning (FL) is an emerging research field in FL that learns an easily adaptable global model in the presence of data heterogeneity among clients. However, one of the main challenges for personalized FL is the heavy reliance on clients’ computing resources to calculate higher-order gradients since client data is segregated from the server to ensure privacy. To resolve this, we focus on a problem setting where the server may possess data independent of clients’ data – a prevalent problem setting in various applications, yet relatively unexplored in the existing literature. Specifically, we propose FedSIM, a new method for personalized FL that actively utilizes such server data to improve meta-gradient calculation in the server for increased personalization performance. Experimentally, we demonstrate through various benchmarks and ablations that FedSIM is superior to existing methods in terms of accuracy, more computationally efficient by calculating the full meta-gradients in the server, and converges up to 34.2% faster.

INDEX TERMS Federated learning, personalization, meta-learning.
the central server in most FL methods. Previous methods in personalized FL suggest that the main server needs only to aggregate and average the optimized client weights to update the global model. Here, the resourceful server is idle for most of the training process while the resource-constrained clients are busy optimizing their local models. In contrast to previous work, we instead attempt to improve personalized FL performance by actively coordinating all the resources available in the entire FL framework.

Therefore, this paper considers a variant of the personalized FL problem where the server contains its own data. We denote this problem setting as **Personalized FL with Server Data**. **Server data** is defined as data used to create and test a model in the server before initiating the FL process and can be available in various application domains. For example, hospitals and healthcare providers may test the validity of models using their records before implementing large-scale FL for training patient-wise predictions based on more privacy-sensitive individual records. In a predictive text application, an initial predictive model can be trained at the server with common phrases or words before implementing large-scale FL for each client’s mobile device in a predictive text application. In addition, an autonomous driving company gathers data in various road conditions to train a model but can utilize FL to improve the model for each driver. However, most FL methods use such server data only for creating an initial model, and disregard it during the FL process.

To this end, we propose a new method to estimate computationally-heavy client-specific meta-gradients in the server using server data. To elaborate, our method attempts to optimize Eq. (1) by utilizing both the computational and data resources of the server to augment model performance. An illustration of this process is depicted in Figure 1.

We summarize our main contributions as follows.

- We propose a novel method **FedSIM** for Personalized Federated Learning with Server Data. To our knowledge, **FedSIM** is the first personalized FL method that efficiently utilizes the server’s computational and information resources to compute the estimates of full meta-gradients with no additional client computation compared to conventional FL.

  - The key components of our proposed method include (i) a custom loss with $L_2$ regularization for local optimization, (ii) the approximation of first-order meta gradients for each client by using the differences between personalized model parameters and global model parameters, (iii) the approximation of second-order meta gradients for each client using server data without explicitly computing Hessian matrices.

  - The empirical evaluations demonstrate that **FedSIM** effectively improves model performance **even when the server has a relatively small amount of data** compared to the entire dataset ($\leq 5\%$), or when the distribution of server data weakly represents that of non-i.i.d. data for each client.

  - We show that **FedSIM** outperforms existing methods in personalized FL. In standardized FL benchmarks proposed in [8], **FedSIM** is up to 2.57\% more accurate and requires 34.2\% fewer communication rounds for convergence.

**II. RELATED WORK**

**A. FEDERATED LEARNING**

Federated learning has rapidly evolved in various aspects [12], with both empirical analyses [13] and theoretical guarantees [14] showing that FL models exhibit similar performance to models trained in centralized data centers even when data does not leave clients. In particular, there have been several works on the various aspects of FL, including methods of reducing communication costs through quantization [15], [16] or adaptive gradient upload rounds [15], [17], [18], and convergence analyses with well-defined lower bounds [14], [17], [19].

A fundamental problem of FL is accuracy degradation due to training the model with non-i.i.d. data across clients [20]. This problem is significant because heterogeneous data distributions are standard in practice [13] and thus investigated in several studies [21], [22], [23] with solutions including normalized federated updates [24] and computing stochastic gradients in mini-batches [25].

**B. PERSONALIZED FEDERATED LEARNING**

Personalized Federated Learning is a personalized variant of federated learning that aims to improve model performance in non-i.i.d. data settings. Examples include using Moreau envelopes [10], model interpolation [26], statistical processes [27] and generative models [28]. Some recent works focus primarily on transfer learning-based personalization [29], [30], [31] for improving heterogeneous performance using post-adaptation methods.

In particular, **FedMeta** [7] and Per-FedAvg [6] consider building upon the Model-Agnostic Meta-Learning (MAML) formulation [5], and study the empirical and theoretical success of the framework in a federated environment. However,
these approaches require the resource-constrained clients to execute full Hessian calculations locally, thereby significantly increasing client-side computation and memory overhead. Several other works aim to decrease this computational bottleneck by disregarding second-order calculations [11], inspired by first-order gradient-based meta-learning approaches as in [32] while sacrificing model performance.

In contrast, FedSIM aims to calculate computationally heavy meta gradients at the server using server data to mitigate accuracy degradation due to disregarding the Hessians without additional computational burden on clients. A table illustrating the differences between personalized FL methods is shown in Table 1.

### III. FEDERATED LEARNING WITH SERVER INFORMATION META-LEARNING (FedSIM)

#### A. PROBLEM: PERSONALIZED FL WITH SERVER DATA

In conventional FL, there are $n$ clients in a federated environment that tries to find a global model $\theta$ by optimizing the following problem:

$$\min_{\theta \in \mathbb{R}^d} f(\theta) := \frac{1}{n} \sum_{i=1}^{n} f_i(\theta)$$

(2)

where $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$ ($i = 1, \ldots, n$) denotes the expected loss over the data distribution of client $i$ such that

$$f_i(\theta) = \mathbb{E}_{D_i \sim p_i}[\ell_i(\theta; D_i)]$$

(3)

where $D_i$ is a random data sample drawn from client $i$’s data distribution $p_i$ and $\ell_i(\theta; D_i)$ is the loss corresponding with this data sample w.r.t. a global model parameter $\theta$.

In contrast to Eq. (2), we learn an adaptable global model $\theta$ in a federated environment, by formulating and solving a bi-level (server- and client-side) problem defined as

$$\min_{\theta \in \mathbb{R}^d} F(\theta) := \frac{1}{n} \sum_{i=1}^{n} F_i(\theta)$$

(4)

$$F_i(\theta) = \min_{\phi_i \in \mathbb{R}^d} \tilde{f}_i(\phi_i) := f_i(\phi_i) + \frac{\lambda}{2} \|\phi_i - \theta\|_2^2$$

(5)

where $\phi_i$ denotes the personalized model of client $i$ and $\lambda$ is a regularization parameter. Note that instead of the conventional loss in Eq.(3), we define a different loss function $\tilde{f}_i(\phi_i)$ that includes an $L_2$ regularization term such that its gradient becomes

$$\nabla_{\phi_i} \tilde{f}_i(\phi_i) = \nabla_{\phi_i} f_i(\phi_i) + \lambda (\phi_i - \theta)$$

(6)

This custom loss is in accordance with ideas from [10], [33] such that the personalized parameters $\phi_i$ are encouraged to tend towards the global parameters $\theta$, which improves the convergence of the global model in non-i.i.d. data settings.

Lastly, we make a practical assumption not included in previous work on personalized FL: the server has its own data with distribution $p_s$ independent of clients’ data. Here, we also assume that the proportion of server data is small compared to the entire dataset.

### B. FedSIM: A FL FRAMEWORK FOR SERVER UTILIZATION

FedAvg in [9]), a centralized server computes a global model by averaging models from decentralized devices. At each round $t$, the server samples a client subset $S_t$ of size $m$ to optimize the global model $\theta_{t-1}$. Each client $i \in S_t$ updates $\theta_{t-1}$ with its private data $D_i \sim p_i$ using gradient decent for $E$ epochs and uploads the optimized model $\phi_i$ back to the server. Finally, the server updates the global model to $\theta_t$ by averaging $\phi_i$ received from $S_t$. It is important to note that $D_i$ was never shared between the clients and the server.

The main contribution of this work comes from allocating the calculation of $\nabla_{\theta} F_i(\theta)$ to optimize Eq.(4) between the clients and the server such that the server can calculate meta-gradients for multiple tasks without sharing data. To this end, FedSIM follows the same principles as FL but with additional computation for meta-gradients in the server to learn an easily adaptable global model.
calculating local gradient updates at each local epoch.

### 2) SERVER-SIDE ALGORITHM

The client’s main goal is to learn a personalized model $\phi_i$ by calculating local gradient updates at each local epoch $e$ as

$$\phi_{i,e} = \phi_{i,e-1} - \alpha \nabla_{\phi_i}(\phi_{i,e-1})$$  \hspace{1cm} (7)

To calculate $\nabla_{\phi_i}(\phi_{i,e-1})$ in practice, we use an unbiased estimate $\nabla_{\phi_i}(\phi_{i,e-1}; D_i)$ by sampling a mini-batch of data $D_i$ from distribution $p_i$. This process is illustrated in Algorithm 1.

### Algorithm 1 FedSIM: Client-Side

**Require:** Step size $\alpha$, regularization strength $\lambda$, client data distribution $p_i$

**ClientUpdate($i, \theta$):** // Run on client $k$

```
\phi_{i,0} \leftarrow \theta

for each local epoch $e$ from 1 to $E$ do
   Sample a mini-batch $D_i$ from distribution $p_i$
   Calculate $\phi_{i,e}$ using $D_i$ with Eq.(7)
end for

Return $\phi_{i,E}$ to server
```

1) CLIENT-SIDE ALGORITHM

The client’s main goal is to learn a personalized model $\phi_i$ by calculating local gradient updates at each local epoch $e$ as

$$\phi_{i,e} = \phi_{i,e-1} - \alpha \nabla_{\phi_i}(\phi_{i,e-1})$$  \hspace{1cm} (7)

To calculate $\nabla_{\phi_i}(\phi_{i,e-1})$ in practice, we use an unbiased estimate $\nabla_{\phi_i}(\phi_{i,e-1}; D_i)$ by sampling a mini-batch of data $D_i$ from distribution $p_i$. This process is illustrated in Algorithm 1.

### Algorithm 2 FedSIM: Server-Side

**Require:** Step size $\beta$, $\delta$, server data distribution $p_s$

**Initialize $\theta_0$**

**for each round $t = 1, 2, \ldots$ do**

```
Sample a mini-batch $D_s$ from distribution $p_s$

for each client $i \in S_t$ in parallel do
   $\phi_i \leftarrow \text{ClientUpdate}(i, \theta_{t-1})$ (Algorithm 1)
   Calculate $v_i = \nabla_{\phi_i}(\phi_i)$ with Eq.(9)
   Calculate $d_i = \nabla^2_{\phi_i}(\phi_i)v_i$ using $D_i$ with Eq.(10)
   Calculate meta-gradient $\nabla_{\phi_i}(\theta_{t-1}) = v_i - \delta d_i$
   Update $\phi_i \leftarrow \phi_i - \beta \nabla_{\phi_i}(\theta_{t-1})$
end for

$\theta_t \leftarrow 1/m \sum_{i \in S_t} \phi_i$
```

2) SERVER-SIDE ALGORITHM

The server then attempts to optimize Eq.(4) for multiple communication rounds in Algorithm 2. In each round $t$, the central server (i) samples $m$ clients, (ii) calculates meta-gradients $\nabla_{\phi_i}(\theta_{t-1})$ for each of these clients using local model $\phi_i$ and server data $D_s$, and (iii) updates the global model from $\theta_{t-1}$ to $\theta_t$ using these meta-gradients.

As shown in [34], the gradient of Eq.(5) w.r.t. $\theta$ with the local loss function $f_i(\phi_i)$ can be written as

$$\nabla_{\phi_i}(\theta_{t-1}) = (I + \frac{1}{\lambda} \nabla_{\phi_i}^2(\phi_i))^{-1} \nabla_{\phi_i}(\phi_i)$$  \hspace{1cm} (8)

Note that $\nabla_{\phi_i}(\theta_{t-1})$ is not dependent on the original meta-model $\theta_{t-1}$, while corresponding with the personalized model $\phi_i$. This characteristic comes from the regularization term in Eq. (5) [34]. Since the meta-gradient $\nabla_{\phi_i}(\theta_{t-1})$ is decoupled from $\theta_{t-1}$, the server approximates the meta-gradient without requiring a history of client $i$’s local updates. This allows clients to utilize multi-step gradient descent for local optimization.

We can see from Eq.(8) that the calculation of $\nabla_{\phi_i}(\theta_{t-1})$ requires two terms:

1) A first-order gradient $v_i = \nabla_{\phi_i}(\phi_i)$
2) A Hessian-vector product $d_i = \nabla^2_{\phi_i}(\phi_i)v_i$

Unlike previous meta-learning approaches to personalized FL, we propose to calculate both $v_i$ and $d_i$ using the server, without requiring additional information or computation from clients.

3) FIRST-ORDER META-GRADIENT

As in Per-FedAvg, the first-order meta-gradient $v_i$ ideally requires a client-specific query dataset $D_i^q$ to calculate an unbiased estimate $\nabla_{\phi_i}(\theta_{t-1}; D_i^q)$. However, in FedSIM, since the server does not have the required client data, we instead approximate $v_i$ by using the weight difference between a personalized model $\phi_i$ and global model $\theta_t$ such that

$$v_i = \nabla_{\phi_i}(\phi_i) \approx \theta - \phi_i$$  \hspace{1cm} (9)

The intuition behind this method comes from the fact that the derivative of $\nabla_{\phi_i}(\phi_i)$ in Eq.(6) at a stationary point $\phi_i$ becomes sufficiently small.

A possible alternative to calculate $v_i$ at the server is to sample a query dataset $D_i^q$ from server data distribution $p_s$ and calculate $\nabla_{\phi_i}(\theta_{t-1}; D_i^q)$. A potential drawback in this approach is that $D_i^q$ does not come from the data distribution of client $i$. Our ablation study in Section IV shows that the weight difference approximation is superior to direct calculation using server data.

4) SECOND-ORDER META-GRADIENT

To calculate $d_i$, instead of separately computing the Hessian $\nabla_{\phi_i}(\phi_i)$, we approximate the entire Hessian-vector product $d_i$ by using Hessian-free estimation [35] as follows:

$$d_i = \nabla^2_{\phi_i}(\phi_i)v_i \approx \frac{\nabla_{\phi_i}(\phi_i + \delta v_i) - \nabla_{\phi_i}(\phi_i - \delta v_i)}{2\delta}$$  \hspace{1cm} (10)

This approximation produces an error of at most $\rho\delta \|v_i\|^2$, where $\rho$ is the parameter for Lipschitz continuity of the Hessian of $f$ [35].

Ideally, calculating unbiased estimates for the two first-order gradients $\nabla_{\phi_i}(\phi_i + \delta v_i)$ and $\nabla_{\phi_i}(\phi_i - \delta v_i)$ in Eq.(10) requires additional client-specific data. We take an alternative approach since the server does not have client data. The server samples $D_s$ from its own data distribution $p_s$ and calculates $\nabla_{\phi_i}(\phi_i + \delta v_i; D_s)$ and $\nabla_{\phi_i}(\phi_i - \delta v_i; D_s)$.

Note that we reuse the same dataset $D_s$ to calculate $d_i$ for all clients in $S_t$. Given that the server data distribution $p_s$ is likely to be different from each client’s data distribution, the quality of $d_i$ calculated using $D_s$ may not be
ideal. Nevertheless, we hypothesized that using the non-ideal second-order terms would improve performance over disregarding the second-order terms altogether. The effectiveness of this approximation will be empirically evaluated in Sections IV and V.

5) KEY DIFFERENCES
Compared to Per-FedAvg [6], a recent meta-learning method for personalized ML, FedSIM does not calculate meta-gradients, which is a computationally expensive operation, at resource-constrained clients but at the server. Moreover, FedSIM is not restricted to a one-step gradient update when calculating $\phi_i$ at clients but allows multi-step updates. pFedMe [10] and FedProx [33] are similar to FedSIM in that the client-side problem in Eq.(5) includes a regularization term, and each client utilizes multi-step gradient descent to obtain its optimized model $\phi_i$. On the other hand, FedSIM enables meta-learning without more client computation. Most importantly, FedSIM actively utilizes the server to aggregate personalized models and calculate computationally heavy meta-gradients using server data.

C. KEY COMPONENTS FOR FedSIM
The key components of the FedSIM framework can be summarized as follows:

- **Custom loss for local optimization**: Each client adds an $L_2$ regularization term to its loss function as in Eq.(5) when optimizing a global model locally. This decouples meta gradient calculation (at the server) from local optimization history (at the clients).

- **First-order meta gradient calculation using weight differences**: Despite the existence of server data, the server calculates the first-order gradient $v_i$ using (client-specific) weight differences as in Eq.(9) instead of using the server data.

- **Second-order meta gradient calculation using server data**: The server calculates second-order gradient $d_i$ in a Hessian-Free way as in Eq.(10). The approximation requires the two terms calculated using server data $D_s \sim p_i$ as $\nabla_{v_i} f_i(\phi_i + \delta v_i; D_s)$ and $\nabla_{v_i} f_i(\phi_i - \delta v_i; D_s)$.

With these components, FedSIM ensures that data remains on the client while also ensuring that the calculation and communication done on the client is no more intensive than that done during standard federated learning.

IV. EXPERIMENTS
The goal of our experiments is to evaluate (i) the performance of FedSIM compared with existing methods on personalized FL with non-i.i.d. client data, (ii) the convergence and computational overhead of FedSIM, and (iii) the effectiveness of the three key components for FedSIM. All our experiments were simulated using a school server comprising four NVIDIA RTX 3900 GPUs and two Intel Xeon Silver CPUs. Each simulation was run for 500 rounds, with the exception of the CIFAR-100 image classification task, which was run for 2000 communication rounds. We extensively utilized the TensorFlow Federated (TFF) package, an open-source framework for calculations on decentralized data built to work on top of TensorFlow. Here, TFF has been developed to facilitate open research and experimentation with federated learning. To the best of our knowledge, this section is the most comprehensive empirical study on personalized FL.

A. EXPERIMENTAL DESIGN

1) BENCHMARKS
We compare FedSIM with other personalized FL methods based on optimization-based meta-learning, FedMeta [7], Fed-Reptile [11], Per-FedAvg (FO) [6], pFedMe [10], and also standard FL methods not focused on personalization including FedAvg [9] and FedProx [33]. Note that since FL is a newly growing research field, existing work has used their own benchmarks to evaluate their respective methodologies, with their own methods of splitting data in a non-i.i.d. manner, which made it difficult to provide a fair comparison in performance.

To mitigate this problem, the authors of FedML [8] opened a research library including benchmarks for federated learning. Thus, we use four non-i.i.d. datasets, Federated EMNIST [36], CIFAR-100 [37], Shakespeare [9], and Stack-Overflow [38], and train a standardized neural network for each dataset with experiments constructed as suggested in [8]. While three of these datasets are naturally partitioned with a non-i.i.d. distribution, CIFAR-100 is partitioned using Pachinko Allocation Method as in [39]. The exact specifications of the dataset are summarized in the appendix in Table 2.

2) SERVER DATA SIMULATION
For each dataset, we randomly sample 5% of the non-i.i.d. data partitions and reserve them as server data, while using the remaining 95% partitions as client datasets. It is important to note that while all the methods use server data for training an initial model, only FedSIM uses server data during the actual FL process. Given that FedSIM takes advantage of server data, we also experimented with different amounts of server data.

Furthermore, when running Per-FedAvg (FO) and FedMeta, 80% and 20% of each client’s training data are

| Table 2. Non-i.i.d. datasets and model architectures for federated learning benchmark [8]. |
|-----------------------------------------------|
| **Datasets** | **# of training samples** | **Non-i.i.d. partition method** | **# of partitions for clients** | **# of partitions reserved server data** | **Baseline model architecture** |
| Federated EMNIST | 671585 | realistic | 3230 | 0–170 | CNN (2 Conv + 2 FC) |
| CIFAR-100 | 50000 | Pachinko | 475 | 0–25 | ResNet-18 + group normalization |
| Shakespeare | 16068 | realistic | 680 | 0–35 | RNN (2 LSTM + 1 FC) |
| Stack-Overflow | 135818730 | realistic | 325354 | 0–17123 | RNN (1 LSTM + 2 FC) |

J. Song et al.: Personalized Federated Learning With Server-Side Information
allocated as the client’s support and query datasets, respectively, for local calculation of meta gradients.

3) TRAINING PROCESS
Training is carried out with \( m = 10 \) as in [33] and [39], such that \( m \) clients are randomly sampled in each round to perform local optimization. The test accuracy for each communication round is evaluated every round by sampling \( m \) clients, deploying the current global model, fine-tuning (personalization) each client’s training data, and finally averaging the validation accuracy of all clients. At the end of each communication round, we calculate validation performance by sampling \( m \) clients, where each client is first fine-tuned using standard training with no custom loss and tests on its test dataset using its own fine-tuned model. Note that we use \( \lambda = 1 \) and \( \delta = 0.25 \). The final test accuracy is calculated in the same manner using the entire client population.

Thus, each communication round can be summarized as follows:
1) Sampling phase: where several clients are chosen from the entire client pool, each with their own unique data randomly sampled from the training dataset
2) Training phase: where the model is trained to adapt to each unique client quickly
3) Testing phase: where the model is tested on a new client with data from the test dataset

A summary of the hyperparameters we used for each dataset is given in Table 3.

Note that since FedProx and FedAvg do not provide a personalization step, we add an update step to simulate personalization of the global model.

### TABLE 3. Summary of hyperparameters used for each task.

| Hyperparameters | FedEMNIST | CIFAR-100 | Shakespeare | StackOverflow |
|-----------------|-----------|-----------|-------------|---------------|
| Client Optimizer| Adam | Adam | Adam | Adam |
| Client Schedule | x | x | x | x |
| Client Learning Rate | 0.01 | 0.01 | 0.01 | 0.01 |
| Server Optimizer | SGD | SGD | SGD | SGD |
| Server Schedule | Linear Decay | Linear Decay | Linear Decay | Linear Decay |
| Server Learning Rate | 0.25 | 0.25 | 0.25 | 0.25 |
| Batch Size | 20 | 20 | 4 | 16 |

### TABLE 4. Best performance of each method on each dataset.

| Methodologies | FedEMNIST | CIFAR-100 | Shakespeare | StackOverflow |
|---------------|-----------|-----------|-------------|---------------|
| FedAvg update | 88.04 | 51.82 | 57.55 | 27.24 |
| FedAvg + update | 85.66 | 41.49 | 54.28 | 25.21 |
| pFedMe | 87.28 | 52.31 | 59.02 | 27.91 |
| Per-FedAvg (PO) | 86.17 | 50.89 | 55.34 | 27.95 |
| FedMeta | 89.77 | 46.29 | 51.46 | 26.72 |
| FedRepli | 85.84 | 51.96 | 55.85 | 27.29 |
| FedSIM (ours) | 91.51 ± 0.273 | 53.09 ± 0.104 | 60.81 ± 0.329 | 28.17 ± 0.171 |

### B. METHOD PERFORMANCE

1) EFFECTS OF SERVER DATA PROPORTION

Figure 2 shows the average test accuracy of various personalized FL methods with varying amounts of server data. Note that while the amount of server data varies from 0 to 5%, 95% of the entire dataset is always allocated as clients. The results show that more server data results in better accuracy in all methods, implying that training an initial model using server data improves performance. FedAvg shows the worst performance since it does not train an adaptable (personalizable) model. Although the performance of the other five conventional methods varies by data settings, FedSIM always provides the best accuracy once server data is given.

In particular, with 5% server data, our method’s performance exceeds all other values in every dataset. As shown
in Table 4, when comparing the best values in each dataset, \textit{FedSIM} provides $0.22 - 2.57\%$ higher accuracy than the next best methods. This verifies that \textit{FedSIM}'s meta-gradient computation is an effective way for using server data during the FL process even when server data is not representative of the entire dataset.

Note that server data is not ideal since conventional MAML requires task (client)-specific datasets for meta gradients. However, our results suggest that if the client datasets are not given to the server due to privacy concerns, calculation of second-order meta gradients using the server data can be a good alternative rather than giving up the second-order terms as in \textit{Per-FedAvg} (FO), \textit{pFedMe}, and \textit{Fed-Reptile}. When there is no server data, \textit{FedSIM} cannot calculate Hessian estimates, essentially becoming the same as \textit{pFedMe}. In this setting, however, \textit{FedSIM} still outperforms both \textit{Fed-Reptile} and \textit{FedProx}, showing that the implementation of both a custom loss and first-gradient estimates results in more accurate meta-gradients by preventing local model divergence.

Furthermore, \textit{FedSIM} shows that utilizing the server to calculate meta gradients directly is more effective than simply averaging locally trained meta models as in \textit{FedMeta}. Note that \textit{FedMeta} enables each client to calculate full meta gradients including second-order terms on its client-specific dataset when optimizing its local model, which requires heavy computation on clients but is not effective for improving the global model.

2) EFFECTS OF LOCAL EPOCHS

Figure 3 shows the test accuracy of the same methods with 5\% server data and varying local epochs $E$. While all the methods show better accuracy as $E$ increases, \textit{FedSIM} experiences remarkable improvement when $E$ increases from 1 to 5 and regularly outperforms all the other methods when $E \geq 5$. In each dataset, the best accuracy value is given by \textit{FedSIM} with $E = 20$, showing a $1.09 - 2.57\%$ increase in accuracy compared to the second highest values.

C. RESOURCE EFFICIENCY

Next, we evaluate the resource efficiency of \textit{FedSIM} regarding local computation and communication overhead. Figure 4 plots test accuracy as communication round ($t$) increases. While \textit{FedSIM} achieves the highest accuracy in all cases, it achieves the next best accuracy in $34.2\%$, $11.38\%$, $19.44\%$, and $20.07\%$ fewer communication rounds for each respective dataset due to the use of more accurate meta gradients for model updates. Since all the methods require the same communication overhead in each round (i.e., dissemination of $\theta$ and aggregation of $\phi_i$), fewer rounds entail less communication overhead.

Figure 5 shows the average client computation time for local optimization in each round. The computation time of \textit{FedMeta} (or \textit{Per-FedAvg}) quickly increases with local epochs due to local calculation of second-order meta gradients. \textit{Per-FedAvg} (FO) ignores the second-order terms but still calculates first-order meta-gradients locally, resulting in the second-longest computation time. On the other hand, \textit{FedSIM} shows a modest increase in client computation time as local epoch increases, similar to \textit{FedAvg} that does not provide personalization since meta gradients are calculated at the server. Overall, \textit{FedSIM} not only trains a more accurate model but also does so resource-efficiently.
D. ABLATION STUDIES

Given that the distribution of server data is dissimilar to that of each client’s data using server data without caution may result in performance degradation. To this end, we evaluate the extent to which each key component of FedSIM contributes to its performance, namely the (i) loss function, (ii) first-order (FO) meta gradient calculation, and (iii) second-order (SO) meta gradient calculation. We made three variants of FedSIM, FedSIM-var1 that uses basic loss function without $L_2$ regularization, FedSIM-var2 that calculates FO meta gradients using server data instead of weight difference (i.e., $\nabla \phi_i(\phi_i; D_q s)$ where $D_q s \sim p_s$), and FedSIM-var3 that disregards SO meta gradients.

Table 5 shows the performance of these variants. Comparison with FedSIM-var3 verifies that although calculating SO meta gradients using client-independent server data is not theoretically ideal, using the SO terms still results in significantly better performance than relying only on FO meta gradients. The FedSIM-var2 case shows, however, that the non-ideal server data causes severe performance degradation when used for FO meta gradient calculation; using (client-specific) weight differences is a better choice in the case of calculating FO meta gradients. In addition, FedSIM-var1 proves that using a custom loss to decouple local optimization history from meta gradients and calculating meta gradients based on $\phi_i$ (locally optimized model) rather than $\theta$ (previous meta-model) result in more useful meta gradients. Overall, the results verify that each critical component of FedSIM highly impacts model accuracy.

V. DATA DISSIMILARITY ANALYSIS

We can see in practical FL scenarios that although clients may have non-i.i.d. data, the data distributions of clients are not entirely unrelated. Thus, prior work such as FedProx [33], Per-FedAvg [6], and pFedMe [10], perform convergence analyses on the global model by assuming that both data distributions and local gradients have bounded dissimilarity among clients. FedSIM makes a similar assumption that both server data distributions and meta gradients calculated using server data have bounded dissimilarity. Thus, in this section, we conduct experiments to evaluate the effect of distributional deviation between server and client data.

First, we investigate data dissimilarity between the server and clients in non-i.i.d. data settings with varying amount of server data. Next, we empirically observe that the variance of data dissimilarity between the clients and the server is an appropriate measure of model performance in FedSIM.

A. DISTRIBUTION COMPARISON

To investigate data dissimilarity, we randomly sample a small percentage of data from two image datasets, Federated-EMNIST and CIFAR-100, as in Section IV-A to simulate server data. The average image distribution of the server data is then compared to each of the remaining clients, using a Structural Similarity Index (SSIM) [40] to compare the image data. Thus, each comparison produces SSIM($i, j$) for $i \in D_s$ and $j \in D$ where $D$ is the set of all clients in a dataset and $D_s$ is the set of server data. This process is repeated many times for each proportion of server data, resulting in boxplots in Figure 6.

Figure 6 shows different trends of data similarity in the two datasets. Regarding CIFAR-100, there is a noticeable increase in SSIM with more server data. This is contrary to SSIM in EMNIST, which remain fairly consistent. We hypothesize that this is due to the fact that EMNIST is a relatively simple dataset, not only represented in grayscale but also consisting of handwritten letters that hardly differ by client, which leads to fast saturation of data similarity with only a small amount of server data. On the other hand, CIFAR-100 provides far more diverse images which require more server data such that data similarity can converge (albeit at a lower SSIM...
than EMNIST), which is more representative of real-world images.

Despite the differences, in both datasets, the general trend of the variance of the similarity metrics decreases with more server data. In addition, SSIM is higher than 0.75 even when server data proportion is 1%, showing that server and client data distributions are not entirely unrelated.

### B. RELATIONSHIP WITH PERFORMANCE

We then analyze the impact of average image similarity on model performance, as seen in Figure 7. This figure shows the relationship between FedSIM performance on the EMNIST and CIFAR-100 datasets and the variance in image similarity (SSIM) between the server data and each of the client data. Data points were collected by training the models various times with different amounts of server data. Then, the server data used during each learning process is used to calculate image similarity with the remaining client data. Finally, we measure the variance of image similarity for each data point, and plot it with respect to the distribution of model accuracy.

Here, the middle of each box represents the mean variance in image similarity for each server data proportion, while the error bars represent the variance in test accuracy. The graphs show that there is a negative correlation between SSIM variance and model performance. More server data results in better accuracy due to its correlation with SSIM variance. This implies that even with the same amount of server data, model performance can depend on the method by which the server dataset is constructed.

### VI. CONCLUSION

This paper investigates a practical problem setting of FL, personalized federated learning with server data. Previous works on personalized FL either calculated all second-order gradients in the client, incurring significant computational overhead in resource-constrained devices, or disregarded the gradients, leading to decreased personalized performance. Instead, this paper proposes to utilize the best of both worlds by actively using the server during the FL training process.

To enable this, we adopt the meta-learning process to create FedSIM where meta-gradients are calculated using the server to improve model performance and reduce client computational overhead. We show that FedSIM solves the proposed FL problem by first performing local optimization using a custom loss function with a regularization term and then using server data with these locally optimized models to calculate higher order gradients. We also provide a variety of numerical experiments and ablations to illustrate the performances of our method compared with existing methods in personalized FL. Finally, we present empirical analyses on the distribution of server data and its impact on performance.

Although our method provides a viable method of personalized FL using the server, future directions for this work can include finding better methods of utilizing the server data to further improve performance on heterogeneous data distributions. Furthermore, deriving mathematical proofs for the convergence of FedSIM can be effective in proving the effectiveness of server data in finding good initial conditions on latent representations of client data. In this respect, exploring the statistical implications of server data on the calculation of meta-gradients can also be helpful for future iterations of FedSIM.

While we focus on personalized FL and meta-learning, we believe that this work opens up an interesting avenue in the FL regime that investigates how a powerful server and its data can contribute to the federated learning process effectively.

### APPENDIX A

#### EXPERIMENT DETAILS

**A. EMNIST**

EMNIST [41] consists of images of digits and upper and lower case English characters, with 62 total classes. The federated version of EMNIST [36] partitions the digits by their author. The dataset has natural heterogeneity stemming from the writing style of each person. We perform a character recognition task using this dataset, with a full description of the model in Table 6.

Federated EMNIST is partitioned in a manner such that 3,400 individuals constitute a separate client, with each client having an individual training dataset and a testing dataset. Thus, a testing round for EMNIST consists of sampling a user, training on the user’s handwriting style, and testing on the individual testing dataset for that particular user.

**B. CIFAR-100**

CIFAR-100 consists of images with RGB channels of $32 \times 32$ pixels each. An unsigned int8 represents each pixel. As is standard with CIFAR datasets, we perform preprocessing on the training images. For training images, we augment the data by performing a random horizontal flip. We then scale the pixel values so each value lies between $[0, 1]$. Finally, we train a modified ResNet-18 model, where group normalization layers replace the batch normalization layers.

Our model trains on a federated version of CIFAR-100 as proposed in [39], where the authors apply a two-step latent Dirichlet allocation (LDA) process by first randomly partitioning the data to reflect the “coarse” and “fine” labels structure of CIFAR-100 by using the Pachinko Allocation Method (PAM), and finally creating a federated dataset using LDA with a parameter of 0.1. Using this method, the authors of [39] generate a training dataset consisting of 500 clients and a testing dataset comprised of 100 clients.

Although we trained our model using 500 training clients, we needed to slightly modify the testing dataset to allow fine-tuning of the model when deployed. Therefore, for each test client, we split the client dataset into a fine-tuning dataset and a validation dataset consisting of 80 and 20% of the data, respectively. When testing, we sample ten clients from the test client space, optimize the models on the fine-tuning dataset for each client, and evaluate the models using each client’s respective validation datasets.
learned 96-dimensional space. It then feeds the embedded words. Our RNN model embeds these sequences into a single LSTM layer, followed by two densely connected layers with a softmax activation at the end. The model architecture is in Table 8.

### REFERENCES

[1] P. R. Center. (2021). Mobile Phone Ownership Over Time. [Online]. Available: https://www.pewresearch.org/internet/fact-sheet/mobile/

[2] I. H. Sarker, Y. B. Abushark, and A. I. Khan, “ContextPCA: Predicting context-aware smartphone apps usage based on machine learning techniques,” *Symmetry, vol. 12, no. 4, p. 499, Apr. 2020.*

[3] F. Ahamed and F. Farid, “Applying Internet of Things and machine-learning for personalized healthcare: Issues and challenges,” in *Proc. Int. Conf. Mach. Learn. Data Eng. (iCMLDE), Dec. 2018,* pp. 19–21.

[4] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, “Federated learning for mobile keyboard prediction,” Tech. Rep., 2019.

[5] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[6] 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.*, vol. 33, 2020, pp. 3557–3568.

[7] F. Chen, M. Luo, Z. Dong, Z. Li, and X. He, “Federated meta-learning with fast convergence and efficient communication,” Tech. Rep., 2019.

[8] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, X. Zhu, J. Wang, L. Shen, P. Zhao, Y. Kang, Y. Liu, R. Raskar, Q. Yang, M. Annavaram, and S. Avestimehr, “FedML: A research library and benchmark for federated machine learning,” 2020, arXiv:2007.13518.

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

[10] C. T. Dinh, N. Tran, and J. Nguyen, “Personalized federated learning with Moreau envelopes,” in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS),* vol. 33, 2020, pp. 21394–21405. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/f4f1f13c8289ac1b1ee0ff17%6b56fc60-Paper.pdf

[11] Y. Jiang, J. Konecný, K. Rush, and S. Kannan, “Improving federated learning personalization via model agnostic meta learning,” Tech. Rep., 2019.

[12] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions,” *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020.

[13] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, “Towards federated learning at scale: System design,” in *Proc. Mach. Learn. Syst.*, vol. 1, 2019, pp. 374–388.

[14] F. Hanzely, S. Hanzely, S. Horváth, and P. Richtarik, “Lower bounds and optimal algorithms for personalized federated learning,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 2304–2315.

[15] M. M. Amiri, D. Gunduz, S. R. Kulkarni, and H. V. Poor, “Federated learning with quantized global model updates,” Tech. Rep., 2020.

[16] J. Sun, T. Chen, G. Giannakis, and Z. Yang, “Communication-efficient distributed learning via lazily aggregated quantized gradients,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 3370–3380.
