ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240715

FedMTL: Privacy-Preserving Federated Multi-Task Learning

Pritam Sen, Cristian Borcea

New Jersey Institute of Technology, Newark, New Jersey, USA
{ps37,borcea}@njit.edu

Abstract. Multi-task learning (MTL) enables simultaneous learning of related tasks, enhancing the generalization performance of each task and facilitating faster training and inference on resource-constrained devices. Federated Learning (FL) can further enhance performance by enabling collaboration among devices, effectively leveraging distributed data to improve model performance, while ensuring that the raw data remains on the respective devices. However, conventional FL is inadequate for handling MTL models trained on different sets of tasks. This paper proposes FedMTL, a new FL aggregation technique that handles task heterogeneity across users. FedMTL generates personalized MTL models based on task similarities, which are determined by analyzing the parameters for the task-specific layers of the trained models. To prevent privacy leakage through these model parameters and to protect the privacy of the task types, FedMTL employs low-overhead algorithms that are adaptable to existing techniques for secure aggregation. Extensive experiments on three datasets demonstrate that FedMTL performs better than state-of-the-art approaches. Additionally, we implement the FedMTL aggregation algorithm using secure multi-party computation, showing that it can achieve the same accuracy with the plaintext version while preserving privacy.

1 Introduction

The growing volume of data generated by smart phones and IoT devices enables them to train models for many tasks. Multi-task learning (MTL) [5] proves particularly useful for mobile and IoT devices that require local model training to uphold privacy (i.e., avoid sending data to a server for centralized training), as the overhead of MTL is comparatively lower on resource-constrained devices than training individual models for each task. For example, a device can collect audio signals and employ MTL models for tasks such as speech recognition, speaker identification, and emotion detection. Autonomous vehicles, likewise, can capture video data to train MTL models for lane detection, recognizing traffic signs, predicting pedestrian intent, and detecting obstacles. Text data gathered by mobile devices can also be used to train MTL models for sentiment analysis, text summarization, spam detection, and named entity recognition.

Although these models can be trained independently by each device, collaboration among devices can further improve model performance by allowing them to learn from each others' models trained on similar or different sets of tasks. However, privacy becomes a concern when gathering raw data from devices for centralized learning or when clients exchange data for distributed learning. Federated learning (FL) [27] allows collaborative training across clients while keeping client data locally. Unfortunately, conventional FL is not applicable to MTL scenarios, wherein clients do not collectively train models for similar sets of tasks. In this paper, we focus on designing an algorithm that allows clients to obtain personalized MTL models in an FL setting while safeguarding the confidentiality of user data.

Prior studies [34, 20] on federated MTL assume each client performs only one task, and tune the local models through methods such as (a) minimizing the parameter differences between models or (b) clustering models into distinct groups to generate average models for each group. However, these approaches face challenges when clients work on different sets of tasks due to: (i) Heterogeneity in model structures arising from differences in the set of tasks, and (ii) Difficulty of clustering models, as a client model with a specific set of tasks may yield a similar score to models trained by others on different subsets of tasks, making it difficult to assign a unique cluster ID. Therefore, it is crucial to design an aggregation algorithm that considers the heterogeneity of tasks executed by each client. Furthermore, the algorithm needs to ensure the privacy of the model parameters to prevent leakage of the training datasets [14] and of the tasks executed by each client. While various privacy-preserving techniques exist for secure aggregation [3, 13], they cannot be applied directly because of the following challenges: (a) The heterogeneity of model structures may leak the number and types of tasks in the dataset, and (b) Analyzing encrypted model parameters and applying complex algorithms on encrypted data incurs high computation and communication overhead.

To address these issues, we propose **FedMTL**, a novel algorithm to support MTL in the FL setting. FedMTL uses new protocols to enhance the capabilities of each client model by enabling collaboration among models from the participating clients based on similarities in the tasks they execute. FedMTL generates personalized MTL models based on task similarities, which are determined by analyzing the parameters for the task-specific layers of the trained models. FedMTL assumes that the supported tasks have unique IDs and the clients are aware of the task IDs for which the model has been trained. Furthermore, it uses a hard parameter sharing-based architecture [5] for local MTL models. This architecture involves sharing hidden layers for the related tasks while maintaining task-specific output layers. The architecture enables the MTL model to understand the overall data structure while gaining expertise in multiple tasks. The aggregator of FedMTL applies layer-wise aggregation policies and computes different sets of weights on clients' model parameters to improve the personalization of the MTL models. FedMTL aims to develop a low resource-consumption approach, enabling straightfor-

ward integration with established privacy-preserving techniques for secure aggregation. In particular, the secure version of FedMTL aims to preserve the privacy of: (a) the parameters of the client's models, (b) the types of the tasks executed by each client, and (c) the size of the training dataset. To the best of our knowledge, FedMTL is the first framework that supports privacy-preserving MTL in FL settings.

To evaluate the effectiveness of FedMTL, we implement the proposed FL aggregation algorithm and privacy-preserving protocols using PyTorch and conduct experiments on benchmark datasets. The results show that FedMTL outperforms baseline approaches when each client trains an MTL model for distinct sets of two tasks. Furthermore, FedMTL demonstrates its adaptability in cases where clients' MTL models are trained on different numbers of tasks — a scenario not directly supported by the state-of-the-art approaches. Additionally, we conduct ablation studies that demonstrate the significance of our aggregation algorithm compared to alternative methods. We use CrypTen [17] to implement a secure version of FedMTL aggregation and achieve the same accuracy performance as the plain text while preserving the privacy of client data.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 defines the FedMTL aggregation problem, and Section 4 describes the FedMTL aggregation algorithm. Section 5 presents the privacy-preserving protocols for FedMTL. Section 6 shows the system and security analysis of FedMTL, and Section 7 presents the experimental evaluation of FedMTL. The paper concludes in Section 8.

2 Related Work

Multi-task learning (MTL) [5, 26, 8, 28] enhances task performance and reduces training and inference time by simultaneously learning related tasks. To learn the generic patterns of data, as well as task-specific features, several MTL architectures [36, 31] adopt hard parameter sharing, which utilizes the same hidden layers for all tasks and incorporates task-specific output layers. However, these approaches typically assume a centralized setup, where the model trainer has full access to all tasks and data. Our research addresses scenarios where each client has a small amount of data for training an MTL model focused on a subset of all the supported tasks in the system, exploring collaboration among clients to enhance the performance of the MTL models.

Federated learning (FL) [27] enables collaboration among clients without sharing private datasets. In FL, each client trains a model using its private data, and uploads the model to a central server for aggregation with models from other clients. To address data heterogeneity, several works [22, 15] aim to learn a personalized model for each client, enhancing compatibility with highly non-IID clients. FedFomo [40] allows each client to compute personalized aggregation weights by minimizing validation loss using model information from other clients, leading to high computational and communication costs, as well as privacy concerns. The work in [15] introduces a clustering algorithm to represent relationships among clients and aggregates models within client groups. FedMSplit [7] enables federated training on multi-modal distributed data, taking into account modality incongruity across clients while assuming the task remains consistent. Despite addressing data heterogeneity, all these methods overlook the task heterogeneity across clients.

To address task heterogeneity, MOCHA [34] employs an optimization algorithm to fit related models trained for separate tasks. FedU [12] formulates the federated MTL problem using Laplacian regularization to explicitly leverage the relationships among the models. Since these approaches update models for all clients simultaneously, it is necessary to re-run the algorithms to support a newly added client, leading to waste in bandwidth and computational resources. FedEM [25] enables each client to learn personalized models as mixtures of several component models. The work in [4] employs a clustering-based training approach to address task heterogeneity by enabling each client to infer its similarity with others by comparing their layer-wise weight updates sent to the server, and then determining how to aggregate weights for selected similar clients. In contrast to our work, all these prior studies assume each client executing a single task.

Although MAS [43] enables clients to train local models for multiple tasks, it assumes the clients have data and labels for all possible tasks. MAS allows the server to pick a subset of tasks, and the clients train models for that specific subset, ensuring task consistency across all clients. In contrast, FedMTL assumes that each client holds data for only a subset of tasks, which may differ among clients, making it a more realistic scenario. Furthermore, FedMTL does not require collecting all clients' models for aggregation, and it enhances personalized MTL models by leveraging the similarities in task-specific parameters among participating clients' models.

To avoid data leakage [42, 39] from uploaded model parameters, several studies [35, 41, 33] propose secure aggregation of local models. These studies employ techniques such as secure multi-party computation (SMPC) [9], homomorphic encryption (HE) [30], or a combination of both. However, none of these works are designed to support MTL where clients' model architectures are different, each working on a different set of tasks. In our work, we introduce an algorithm designed for computing personalized MTL models with minimal overhead, which has the flexibility to integrate with established privacy-preserving techniques for secure aggregation.

3 FedMTL Aggregation Problem Definition

The goal of FedMTL is to design FL aggregation protocols that enhance personalized MTL models, without imposing heavy computational burdens on devices. We consider a set of tasks $\mathbf{U} = \{T_1, \ldots, T_K\}$ supported by FedMTL, where the total number of supported tasks, $K = |\mathbf{U}|$. Each task T_k, $k \in \{1, \ldots, K\}$, has a unique integer ID. We consider a hard-sharing architecture for the multi-task models, with the model parameters represented as $\mathbf{W} = \{\mathbf{W}^S, \mathbf{W}^T\}$, where $\mathbf{W}^S$ and $\mathbf{W}^{T_k}$ represent the parameters for the shared layers and task T_k, respectively (shown in Fig. 1).

In each round, FedMTL picks a set of clients $\mathbf{C}$, where the number of clients is $N = |\mathbf{C}|$. Each client $C_i \in \mathbf{C}$ trains an MTL model for a subset of tasks $\mathbf{U}_i \subseteq \mathbf{U}$ using data $D_i = \{\mathbf{X}_i, \mathbf{Y}_i\}$, where X_i represents the input features and $\mathbf{Y}_i$ denotes the list of labels for $K_i = |\mathbf{U}_i|$ number of tasks, as $\mathbf{Y}_{ik}$ is the label for task $T_k \in \mathbf{U}_i$. The set of tasks executed by each client can be different. Training on local data, each client obtains the local model $\mathbf{W}_i$, which consists of the parameters of the shared layer $\mathbf{W}_i^S$ and the parameters of the task-specific layers (task layers) $\mathbf{W}_i^T$. We use $\mathbf{W}_i^T[j]$ to represent the parameters for j-th task and $\mathbf{W}_i^{T_k}$ to represent the parameters for the task with ID k, i.e., T_k.

This work focuses on the aggregation protocol employed by the server to generate personalized MTL models for each client. As shown in Fig. 1, the aggregator $\mathbf{A}$ collects the model parameters from N clients $\mathbf{W}_i, i \in \{1, \ldots, N\}$ and aggregates these parameters to get the personalized model parameters $\mathbf{W}_i^* = \{\mathbf{W}_i^{S^*}, \mathbf{W}_i^{T^*}\}$, where $\mathbf{W}_i^{S^*}$ is the aggregated model parameters of the shared layer and $\mathbf{W}_i^{T_k^*}$ is the aggregated layer parameters for task T_k.

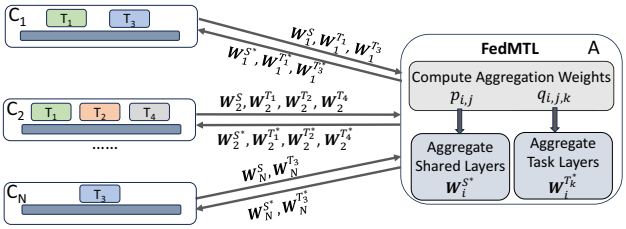


Figure 1. FedMTL Problem Setup

In the case of personalized FL, the optimization problem can be represented as in Eq. 1, where the function $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$ denotes the expected loss over the data distribution of C_i.

$$\forall i, \mathbf{W}_i^* = \arg\min_{w_i} f_i(\mathbf{W}_i) \tag{1}$$

To achieve this, we define $p_{i,j}$ as the weights of client C_i to aggregate the parameters of the shared layers from the local model of client C_j, as shown in Eq.2. Additionally, we define $q_{i,j,k}$ as the weights to aggregate the parameters of task layers of C_i using the parameters of the local model from C_j for task T_k, as shown in Eq.3. For conventional FL that supports only single-task learning, $\mathbf{W}_i^S$ contains $\mathbf{W}_i^{T_k}$ and Eq. 2 is similar to the vanilla FedAvg if $p_{i,j} = 1$.

$$\mathbf{W}_i^{S^*} = \frac{\sum_{j=1}^{N} p_{i,j} \mathbf{W}_j^S}{\sum_{j=1}^{N} p_{i,j}} \tag{2}$$

$$\mathbf{W}_i^{T_k^*} = \frac{\sum_{j=1}^{N} q_{i,j,k} \mathbf{W}_j^{T_k}}{\sum_{j=1}^{N} q_{i,j,k}} \tag{3}$$

The FedMTL aggregation problem is to determine the weights $p_{i,j}$ and $q_{i,j,k}$ that enhance personalized MTL models in an FL setting.

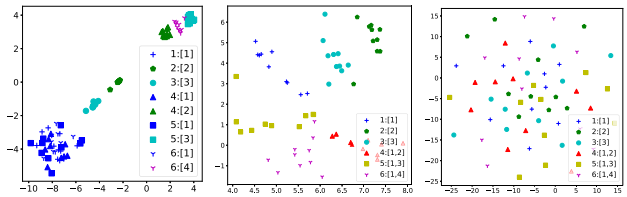
4 FedMTL Algorithm

This section first presents our analysis (Section 4.1) of the characteristics of the parameters across different layers of the local MTL models and the similarities of the models trained by the users executing different sets of tasks. This analysis provides insights into the rationale behind our algorithm's design by highlighting the similarities in parameters at different layers of the local models. Then, we introduce the aggregation algorithm (Section 4.2) of FedMTL that improves the personalized MTL models by leveraging similarities between task-specific parameters to aggregate local models.

4.1 MTL Model Parameter Analysis

To inform the design of FedMTL, we consider at MTL model running over the CelebA dataset [24] and randomly partition it among 60 clients using the symmetric Dirichlet distribution. We consider 4 tasks, where each task T_k, $k \in \{1, \ldots, 4\}$ consists of detecting 5 different face attributes. The clients are grouped into 6 groups of equal size, where each group G_i, $i \in \{1, \ldots, 6\}$, trains an MTL model for the i-th set of tasks from the list: $[[T_1], [T_2], [T_3], [T_1, T_2], [T_1, T_3], [T_1, T_4]]$. As local MTL models, we use the LeNet [19] architecture as the base, with a task-specific linear layer for each task. We then collect the trained model parameters after executing 20 epochs and visualize those weight updates by t-SNE [38] in Fig. 2.

We analyze the parameters of the MTL models in terms of (i) task layers and (ii) shared layers. Fig. 2(a) shows the parameters of the task layers mapped into 2-dimensions, where the label $G_i : [T_k]$ represents the parameters $\mathbf{W}^{T_k}$ for the task T_k of the clients in group G_i. We observe that the parameters of the task layers are very similar for the clients performing the same set of tasks. Such similarity can even be across task groups; for example, parameters for task T_1 from all groups of clients are very close, even though the task is executed in conjunction with different tasks.



(a) (b) (c)

Figure 2. t-SNE visualization of 60 MTL model parameters: (a) task layers, (b) top shared layers, (c) bottom shared layers

We analyze the shared layers $\mathbf{W}^{S_l}$, $1 \leq l \leq L$, of the MTL models, where L is the number of shared layers. For a linear layer in the LeNet model, which is closer to the task layer (i.e., $l \rightarrow L$), the parameters exhibit similarity among clients within the same group (as shown in Fig. 2(b)), as these models are trained for the same set of tasks. For some clients in groups G_4, G_5, G_6, these parameters are close to the parameters of the clients in group G_1, since there is a common task T_1 in all of these groups. Fig. 2(c) shows the parameters of the convolution layer, which is the bottom layer (i.e., $l \rightarrow 1$) in the shared layers of the LeNet model. For these bottom layers, the parameters do not show any specific pattern. This analysis leads us to the following insight for the design of FedMTL: although the shared layers learn generic patterns of data, some layers are influenced more by the tasks, as these parameters are significantly updated by the gradients computed based on the losses associated with each task. Our experimental results in Section 7 further demonstrate that the parameters in task-specific layers are similar across clients performing the same task. Therefore, FedMTL should treat the shared layers and task-specific layers separately, prioritizing the parameters in the task-specific layers to evaluate similarity across clients' models.

4.2 FedMTL Aggregation

When dealing with MTL in FL settings, the model parameters $\mathbf{W}_i$, cannot be directly aggregated due to the heterogeneity in model architecture arising from variations in the number and types of tasks executed by the clients. Therefore, we consider aggregating the shared layers and task layers separately. Although the shared layers from all clients have the same structure, identical weights $p_{i,j}$ should not be assigned for aggregation in Eq. 2, as the shared layers are learned differently by clients executing different sets of tasks. Regarding the task layers, the i-th client's task layers, denoted as $\mathbf{W}_i^{T_k}$ for task T_k, can be improved by learning from the task layers $\mathbf{W}_j^{T_k}$, $j \in \mathcal{N}(C)$, where $\mathcal{N}(C)$ is the subset of clients executing T_k. Based on the observations presented in Section 4.1, we propose employing the FedMTL algorithm to aggregate the local models, thereby enhancing both the task layers and the shared layers to create a personalized MTL model for each client. In each round, FedMTL first computes the similarity score for each pair of clients by analyzing the task layers of each client's MTL model, and then uses the similarity scores to compute the aggregation weights $p_{i,j}$ and $q_{i,j,t}$ to aggregate the shared and the task layers as in Eq. 2 and 3, respectively.

Computation of Similarity Scores. To compute the similarity score for a pair of clients (C_i, C_j), FedMTL compares the parameters of the task layers and identifies the one-to-one mapping between the task layers of the two clients that maximizes the overall score. The overall similarity score also captures the similarity in the shared layers; if the task layers are similar, there is a high likelihood that the shared layers are as well. FedMTL analyzes the parameters of the task layers instead of relying on matching task IDs, as data from different domains can be used for the same task, and thus, does not guarantee similarities in the model parameters.

For a pair of clients, FedMTL first computes the cosine similarities of all pairs of tasks between these clients, denoted as $A_{i,j} \in \mathbb{R}^{K_i \times K_j}$, using Eq. 4. Then, FedMTL uses a standard Hungarian algorithm [18], H_{Score}, to find the one-to-one task mapping with maximum cumulative score $H_{i,j}$ (Eq. 5). The similarity score for client C_i with respect to clients C_j, denoted as $S_{i,j} \in [0, 1]$, is computed by dividing the score $H_{i,j}$ by the number of tasks K_i, as shown in Eq. 6. Here, $S_{i,j}$ might not necessarily be the same as $S_{j,i}$, since the number of tasks executed by the clients can differ.

$$A_{i,j}[m][n] = \cos(\mathbf{W}_i^T[m], \mathbf{W}_j^T[n]); \quad m \in \{1, \dots K_i\}, n \in \{1, \dots K_j\} \tag{4}$$

$$H_{i,j} = H_{Score}(A_{i,j}) \tag{5}$$

$$S_{i,j} = H_{i,j}/K_i \tag{6}$$

Aggregation of the MTL models. To obtain the personalized MTL model for a client C_i, FedMTL assigns varying weights to all clients' MTL models based on the similarity score, as calculated in Eq. 6. To mitigate adverse effects arising from highly dissimilar models belonging to another client C_j, FedMTL sets $S_{i,j} = 0$ if the score $S_{i,j}$ falls below a threshold value Z_i, effectively excluding models that differ significantly from aggregation. The threshold value Z_i is adaptable, $0 \leq Z_{min} \leq Z_i \leq Z_{max} \leq 1$, allowing clients to select their threshold based on the current model's ($\mathbf{W}_i$) performance. In our experiments, initially, the threshold value is set low, $Z_i = Z_{min}$, to encourage learning from a diverse set of clients. As a client participates in multiple rounds and achieves improved model performance, Z_i is programmatically increased in each round by $(1 - Z_{min})/R$, where R is the total number of rounds. This adjustment facilitates learning from more similar models, thereby reducing the need for substantial changes in the parameters, and can be continued until the model's accuracy converges on the test dataset. The updated score is expressed as in Eq. 7.

$$S_{i,j} = \begin{cases} 0 & \text{if } S_{i,j} < Z_i \\ S_{i,j} & \text{otherwise} \end{cases} \tag{7}$$

For C_i, FedMTL computes the aggregation weights $p_{i,j}$, using Eq. 8, and then uses $p_{i,j}$ in Eq. 2 to compute $\mathbf{W}_i^{S^*}$ to aggregate the shared layers from all clients $C_j, j \in \{1, \dots, N\}$. Here, D_j denotes the size of the training dataset of client C_j, and $S_{i,j}$ represents the pairwise similarity scores of the client models, which are used to compute the aggregation weights. Eq. 8 is equivalent to the aggregation weights for FedAvg, if $S_{i,j} = 1$.

$$p_{i,j} = S_{i,j} \times D_j \tag{8}$$

Similarly, FedMTL computes the weights $q_{i,j,k}$ using $S_{i,j}$ as in Eq. 9 and uses $q_{i,j,k}$ in Eq. 3 to obtain the parameters of the task layers $\mathbf{W}_i^{T_k}$ for task T_k of C_i, by aggregating the parameters of the corresponding task $\mathbf{W}_j^{T_k}$ from C_j.

$$q_{i,j,k} = \begin{cases} S_{i,j} \times D_j & \text{if } T_k \in \mathbf{U}_j \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

FedMTL uses the same aggregation weights for all parameters in a model, such that the scaled model captures a similar parameter distribution as the local model. Following this approach, the personalized model of a client is the linear combination of all the models of users executing a similar subset of tasks.

5 Privacy-Preserving FedMTL

Improved performance through personalization in FedMTL comes with a trade-off in privacy, as the aggregator needs access to information about the tasks executed by each client and their local model parameters. To preserve the privacy of client data in FL, we can use an established cryptographic technique $\mathcal{E}$, such as SMPC or HE, for secure aggregation. We assume that fundamental operations (e.g., addition, multiplication, comparison, etc.) are supported in $\mathcal{E}$. However, it is required to design secure protocols which the clients invoke to protect the numbers/types of the tasks and the model parameters before uploading them to the server. Furthermore, server-side protocols must be developed to implement the aggregation algorithm within the specified cryptographic domain. In this section, we specifically explore secret sharing [11, 6] in SMPC, as the cryptographic technique to design a secure FedMTL system. The secret-sharing scheme allows sharing of a secret x among P servers, such that the servers can use their shares to compute a function, while each server learns nothing about the secret. Next, we describe the threat model, the overview of the privacy-preserving FedMTL, and the secure protocols to be followed at the client and server sides.

5.1 Threat Model

We consider an honest-but-curious adversary in a P-party SMPC settings, where each of the P servers honestly follows the protocols, but may individually attempt to learn clients' private data. We assume that the application developer is responsible for setting up the distributed trust, and the parties (client and servers) will communicate using a secure channel.

In our threat model, we consider that at most $P - 1$ servers can collude to learn clients' sensitive data and global statistics. A secure version of FedMTL guarantees the following privacy properties:

- Individual model parameters of the clients are not revealed to anyone other than the source client itself.
- IDs of the tasks executed by each client are protected.
- Size of the training dataset used by each client is protected.
- Secure aggregation of encrypted models produces correct personalized models, achieving the same level of accuracy as if the aggregation were done using plain text data.

5.2 System Overview

Figure 3 shows the secure FedMTL system with P aggregators and N clients. The aggregators communicate with each other to compute the required parameters encrypted in secret-shared format, ensuring privacy against the threat model. In additive (or arithmetic) secret sharing (A-SS) [11, 6], P values $(x_1, \dots, x_P)$ are chosen uniformly at random, subject to the requirement that $\sum_{i=1}^{P} x_i = x \pmod L$, where $L = 2^l$ represents l-bit integers. This can be done

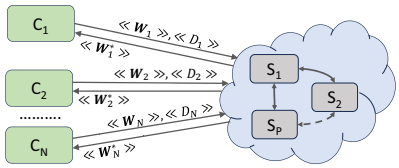


Figure 3. Secure FedMTL system using SMPC

by choosing $x_1, \ldots, x_{P-1} \in \mathbb{Z}_L$ uniformly at random, and then setting $x_P = x - \sum_{i=1}^{P-1} s_i \pmod L$. The reconstruction algorithm simply adds all the shares modulo L, that is, $x = (\sum_{p \in P}[x]_p) \pmod L$. We denote the sharing of x across the parties $p \in P$ by $\langle\langle x \rangle\rangle = \left\{ \langle\langle x \rangle\rangle_p \right\}_{p \in P}$, where $\langle\langle x \rangle\rangle_p$ indicates p 's share of x. Fundamental operations are already supported in A-SS; certain functions, such as addition, can be performed locally, while others, such as multiplication and comparison, require communication between servers.

To preserve the data privacy in FedMTL, each client C_i uploads the parameters of the shared layers, task layers, and size of the dataset in encrypted format to the aggregator servers as $\langle\langle \mathbf{W}_i^S \rangle\rangle$, $\langle\langle \mathbf{W}_i^T \rangle\rangle$ and $\langle\langle D_i \rangle\rangle$ respectively. The servers invoke secure protocols to compute $\langle\langle \mathbf{W}_i^{S^*} \rangle\rangle$ and $\langle\langle \mathbf{W}_i^{T^*} \rangle\rangle$ which are downloaded and decrypted by C_i to get the personalized model $\mathbf{W}_i^* = \{\mathbf{W}_i^{S^*}, \mathbf{W}_i^{T^*}\}$ in plain text.

Since each of the P aggregator servers can monitor the computation's control flow, it is essential to use oblivious operations that avoid data-dependent control flow. This ensures the security of the input, output, and intermediate results throughout the aggregation process. The secure aggregation protocols in FedMTL provides the security guarantees described in Section 5.1 to ensure that honest clients no longer have to put their complete trust in all servers for privacy. As long as one server is functioning correctly, privacy is guaranteed.

5.3 Secure Aggregation Protocols

FedMTL employs secure protocols designed to ensure the privacy of client data during the FL workflow. Next, we present the workflow (Algorithm 1) and the protocols for the clients and the servers.

Model initialization. To participate in the privacy-preserving FedMTL workflow, a client requests the aggregator to get the list of supported tasks $\mathbf{U}$, and the complete MTL model architecture with the shared layers $\mathbf{W}^S$ and task layers $\mathbf{W}^{T_k}$, $1 \leq k \leq K = |\mathbf{U}|$, initialized with $\mathbf{W}^{S^{(0)}}$ and $\mathbf{W}^{T_k{}^{(0)}}$. The maximum number of parameters for the task layers is $dt_m = \max_{T_k \in \mathbf{U}} |\mathbf{W}^{T_k}|$. Since these parameters are not confidential, a client can connect to any aggregator to receive this data in plain text.

The client C_i selects a set of tasks $\mathbf{U}_i \subseteq \mathbf{U}$, and create the local MTL model $\mathbf{W}_i^{(0)} = \{\mathbf{W}_i^{S^{(0)}}, \mathbf{W}_i^{T^{(0)}}\}$, where $\mathbf{W}_i^{T^{(0)}}$ contains the initial parameters $\mathbf{W}_i^{T_k{}^{(0)}}$ for $T_k \in \mathbf{U}_i$.

Client side data preparation. At each round, the aggregator randomly selects a subset of clients $\mathbf{C}$, $N = |\mathbf{C}|$, as in traditional FL. Then, it invokes the protocol $\mathcal{F}_{SU}$ (defined below) for each client C_i, initiating the client to train the MTL model using its training dataset of size D_i. After training the model for a predefined number of epochs as in traditional FL, C_i gets the model $\mathbf{W}_i = \{\mathbf{W}_i^{S^{(r)}}, \mathbf{W}_i^{T^{(r)}}\}$ and sets the threshold Z_i in it's r-th communication round. Then, C_i follows the steps below to upload its private data for secure aggregation.

- $\mathcal{F}_{SU}$ encrypts the parameters of the shared layers as $\langle\langle \mathbf{W}_i^S \rangle\rangle$, thereby securing all the values in the vector $\mathbf{W}_i^S$.

Algorithm 1 Secure FedMTL Aggregation

Input: Communication Round R, number of tasks K, initial model parameters $\mathbf{W}^{(0)} = \{\mathbf{W}^{S^{(0)}}, \mathbf{W}^{T_k{}^{(0)}}\}$, $1 \leq k \leq K$
Output: MTL models $\mathbf{W}_i^*$ for C_i

1: **for** r = 1, . . . , R **do**
2: Randomly selects a subset of clients $\mathbf{C}$, $N = |\mathbf{C}|$
3: **for** client $C_i \in \mathbf{C}$ **do**
4: $\langle\langle \mathbf{W}_i^S \rangle\rangle, \langle\langle \mathbf{W}_i^{T'} \rangle\rangle, \langle\langle \mathbf{M}_i \rangle\rangle, \langle\langle D_i \rangle\rangle, \langle\langle Z_i \rangle\rangle = C_i.\mathcal{F}_{SU}()$
5: $\langle\langle \mathbf{W}_i^{T''} \rangle\rangle = \langle\langle \mathbf{M}_i \rangle\rangle^\intercal \times \langle\langle \mathbf{W}_i^{T'} \rangle\rangle$
6: **end for**
7: $\langle\langle \mathbf{S} \rangle\rangle = \mathcal{F}_H(\langle\langle \mathbf{W}^{T''} \rangle\rangle)$, where $\langle\langle \mathbf{W}^{T''}[i] \rangle\rangle = \langle\langle \mathbf{W}_i^{T''} \rangle\rangle$
8: **for** client $C_i \in \mathbf{C}$ **do**
9: $\langle\langle \mathbf{W}_i^{S^*} \rangle\rangle = \mathcal{F}_S(\langle\langle \mathbf{W}^S \rangle\rangle, \langle\langle \mathbf{S} \rangle\rangle, \langle\langle \mathbf{D} \rangle\rangle)$
10: $\langle\langle \mathbf{W}_i^{T^{**}} \rangle\rangle = \mathcal{F}_T(\langle\langle \mathbf{W}^{T''} \rangle\rangle, \langle\langle \mathbf{S} \rangle\rangle, \langle\langle \mathbf{D} \rangle\rangle)$
11: $\langle\langle \mathbf{W}_i^{T^*} \rangle\rangle = \langle\langle \mathbf{M}_i \rangle\rangle \times \langle\langle \mathbf{W}_i^{T^{**}} \rangle\rangle$
12: $\mathbf{W}_i^* = C_i.\mathcal{F}_{SR}(\langle\langle \mathbf{W}_i^{S^*} \rangle\rangle, \langle\langle \mathbf{W}_i^{T^*} \rangle\rangle)$
13: **end for**
14: **end for**

- As $\mathbf{W}_i^{T_k}$ may vary in size for different tasks $T_k \in \mathbf{U}_i$, $\mathcal{F}_{SU}$ ensures uniform size dt_m by padding zeros at the end, thereby protecting the type of the task.
- To protect the number of tasks, $\mathcal{F}_{SU}$ generates fake task-specific vectors, each of size dt_m with zero value, and appends those vectors to $\mathbf{W}_i^T$ to generate $\mathbf{W}_i^{T'} \in \mathbb{R}^{K_i' \times dt_m}$, where $K_i' = |\mathbf{W}_i^{T'}|$, $K_i \leq K_i' \leq K$.
- $\mathcal{F}_{SU}$ generates a mapping $\mathbf{M}_i \in \{0,1\}^{K_i' \times K}$ that maps the index of each task-specific parameter $\mathbf{W}_i^{T'}$ to the ID of that task. Thus, $\mathbf{M}_i[j][k] = 1$, if $\mathbf{U}_i[j] = T_k$, otherwise 0.
- $\mathcal{F}_{SU}$ uploads $\langle\langle \mathbf{W}_i^S \rangle\rangle$, $\langle\langle \mathbf{W}_i^{T'} \rangle\rangle$, $\langle\langle \mathbf{M}_i \rangle\rangle$, $\langle\langle D_i \rangle\rangle$, $\langle\langle Z_i \rangle\rangle$.

In this way, $\mathcal{F}_{SU}$ protects the dataset size, the actual number and type of the executed tasks, and the associated model parameters for both the shared and task layers.

Server side aggregation. At each round, FedMTL computes the task-specific parameter from C_i for all T tasks, $\mathbf{W}_i^{T''} \in \mathbb{R}^{K \times dt_m}$ by performing matrix multiplication of $\mathbf{M}_i^\intercal$ and $\mathbf{W}_i^{T'}$, where $\mathbf{M}_i^\intercal$ is the transpose of matrix $\mathbf{M}_i$. Using matrix multiplication in the secret-sharing scheme, the aggregator servers get the share of $\langle\langle \mathbf{W}_i^{T''} \rangle\rangle$ for all clients (Line 5). Then, for each pair of clients (i,j), $1 \leq i, j \leq N$, it uses the secret-sharing version of the Hungarian algorithm $\mathcal{F}_H$ and computes $\langle\langle \mathbf{S}_{i,j} \rangle\rangle$ following Eq. 5, 6, 7. Next, the aggregator servers invoke: (i) $\mathcal{F}_S$, to securely compute Eq. 8 and 2 to get the parameters of the shared layers $\langle\langle \mathbf{W}_i^{S^*} \rangle\rangle$ (Lines 9), (ii) $\mathcal{F}_T$, to securely compute Eq. 9 and 3 to get the parameters of the task layers $\langle\langle \mathbf{W}_i^{T^{**}} \rangle\rangle$ and mapped into the parameters for the required tasks $\langle\langle \mathbf{W}_i^{T^*} \rangle\rangle$ for C_i using the matrix $\langle\langle M_i \rangle\rangle$ (Lines 10-11). $\mathcal{F}_H$, $\mathcal{F}_S$ and $\mathcal{F}_T$ are computed securely using standard techniques for addition, multiplication and comparison operations in A-SS domain.

Retrieving personalized models. Client C_i invokes $\mathcal{F}_{SR}$ (Line 12) to download $\langle\langle \mathbf{W}_i^{S^*} \rangle\rangle$ and $\langle\langle \mathbf{W}_i^{T^*} \rangle\rangle$ from the aggregator and decrypts them to obtain the updated model $\mathbf{W}_i^* = \{\mathbf{W}_i^{S^*}, \mathbf{W}_i^{T^*}\}$. Subsequently, the client can proceed with training the model for the upcoming round.

5.4 Secure FedMTL Example

Fig. 4 illustrates the privacy-preserving FedMTL aggregation for 2 clients in a single round. Client C_1 trains the model for tasks T_1

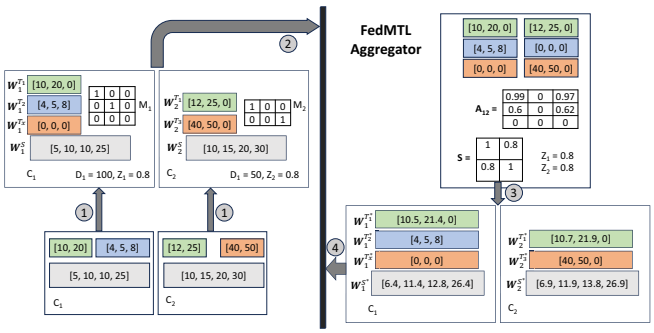


Figure 4. Example of privacy-preserving FedMTL aggregation of the MTL models from two clients C_1 and C_2

and T_2, while client C_2 trains the model for tasks T_1 and T_3. Subsequently, for each client C_i, FedMTL invokes the $\mathcal{F}_{SU}$ protocol to prepare the data for upload, which includes the parameters of the shared layers $\mathbf{W}_i^S$, parameters of the task layers $\mathbf{W}_i^{T_k}$, task-mapping $\mathbf{M}_i$, dataset size D_i, and threshold value Z_i, as shown in step ① (Fig. 4). For each task layer, clients pad the parameter vector with zero values such that all task layers have the same number of parameters, thereby protecting the types of the tasks. Additionally, a client can add one or more fake tasks to protect the actual number of tasks for which the MTL model is trained. For instance, C_1 adds a fake task $\mathbf{W}_1^{T_x}$ in. Fig. 4. In step ②, $\mathcal{F}_{SU}$ encrypts all the values and uploads the encrypted data to the aggregator servers.

The FedMTL aggregator invokes $\mathcal{F}_H$ to calculate the similarities $\mathbf{A}_{i,j}$ between the task layers of C_i and C_j. It then employs the Hungarian assignment algorithm and uses the threshold value Z_i to compute the similarity matrix $\mathbf{S}$. In step ③, the similarity matrix $\mathbf{S}$ is employed by protocols $\mathcal{F}_S$ and $\mathcal{F}_T$ to calculate the aggregation weights $p_{i,j}$ and $q_{i,j,t}$ for aggregating the shared and task layers of the clients' models, respectively. Finally, in step ④, the aggregated model parameters are downloaded by the clients.

6 System Overhead and Security Analysis

This section presents first the overhead analysis of FedMTL without privacy protection, and then the overhead and security analysis of the privacy-preserving version of FedMTL.

6.1 Overhead Analysis of FedMTL without Privacy

The FedMTL aggregation incurs similar computation and communication overhead as state-of-the-art approaches that analyze model parameters at the server to provide personalized models. We consider N number of clients participating in training, with the model parameters denoted as $d = ds + K \times dt_m$, where ds is the number of parameters in the shared layer, dt_m is the maximum number of parameters in the task-specific layer and K is the number of tasks. In FedMTL, there is no additional computational overhead at the client side other than the training of the MTL models. For aggregation, the computational complexity at the server side is $O(N^2(d + K^2 \times dt_m + K^3))$. Typically, $d \gg dt_m$ and when T is small, the complexity becomes $O(N^2 d)$, which is similar to FedAMP [16]. For FedFomo [40] and MOCHA [34], the complexity at the server is $O(Nd)$; however, both approaches offload additional computation onto clients. Other state-of-the-art approaches, such as FedAvg [27], FedProx [21], and pFedMe [37], have a lower computational cost of $O(Nd)$, but they do not perform well in the case of task-heterogeneity.

In FedMTL, each client uploads the model parameters, dataset size, task-mapping, and threshold value to the aggregator. The dataset size, task-mapping, and threshold value are very small and can be represented in a few bytes. Therefore, the amount of data (including upload and download) that each client needs to transmit per communication round in FedMTL is approximately $2 \cdot \Delta$, where Δ represents the size of the model. FedMTL's overhead is lower compared to state-of-the-art approaches, such as FedFomo, MOCHA, FedAMP, FedDWA, etc. The overhead of FedMTL is the same as that of FedAvg, FedProx, pFedMe. However, FedMTL outperforms them in terms of accuracy while demonstrating the capability of aggregation even when clients train MTL models on different sets of tasks.

6.2 Overhead of Secure FedMTL

The overhead associated with the secure aggregation of FedMTL depends on the cryptographic technique used to protect the privacy of the data. We present the overhead analysis in the case of using SMPC for privacy-preserving FedMTL as discussed in Section 5.

In secure FedMTL, each client C_i incurs negligible computational overhead in preparing the data for uploading, as it only involves preparing the task-map of size $K_i' \times K$ and updating the K_i' number of vectors of size dt_m that contain the parameters of the task-layers. At the server side, the computational complexity is $O(N^2 \times (d + K^2 \times dt_m + K^3))$, which is the same as the plain text in asymptotic notation. However, in a secret-shared domain that utilizes Beaver triples [2] to evaluate multiplication operations, one multiplication in plain text is equivalent to four multiplications and three additions in A-SS format.

In secure FedMTL using SMPC, each client needs to transmit approximately $2 \times P \times \Delta$ of data (including upload and download) per communication round. Here, Δ represents the size of the model, and P is the number of aggregator servers. The same amount of data transmission is required for other state-of-the-art approaches, such as FedAvg, when using SMPC for secure aggregation.

To execute the secure protocols in FedMTL, aggregator servers need to communicate with each other. For example, when comparing two B-bit numbers, the standard implementation requires $log(B)$ rounds of communication. FedMTL protocols, such as, $\mathcal{F}_S$ and $\mathcal{F}_T$, involve only multiplication operations, and can be computed in a single round. The computation of similarity scores using the $\mathcal{F}_H$ protocol involves calculating the score for each pair of clients; thus, the simple implementation requires a large number of communication rounds. However, by processing data in batches, the number of communication rounds can be reduced.

6.3 Security Analysis of Secure FedMTL

FedMTL employs the standard additive secret-shared (A-SS) approach to upload the clients' private data to the aggregator servers. Thus, data at rest (model parameters, dataset size, task-mapping threshold value) is information-theoretically secure [10] against the threat model following Axiom 1. For aggregation, FedMTL uses existing A-SS protocols, which are secure following Axiom 2. After the axioms, we present two theorems to show that Secure FedMTL preserves the privacy of client data throughout the FL workflow.

Axiom 1. A value x is information-theoretically secure in additive secret-shared format even if $P - 1$ out of P parties collude.

Axiom 2. There exist secure protocols for fundamental operations (arithmetic operations, comparisons, sorting) that preserve the privacy of the input and output of each operation.

Theorem 1. $\mathcal{F}_{SU}$ secures each client's uploaded data to the aggregator servers, protecting the parameters of the trained MTL model and the number and types of executed tasks from attacks described in the threat model.

Proof. In FedMTL, each value in the vectors representing the parameters of the shared layers and task-specific layers is encrypted in A-SS format. Therefore, the parameters of the trained MTL model are secure, following Axiom 1. Since the size of the vectors for each task-layer is the same, the adversary cannot identify the type of the task by observing the volume of data. Additionally, since each client can add fake task-layers, the adversary cannot learn the actual number of executed tasks by observing the number of task-layers. Thus, the parameters, as well as the number and types of tasks, are protected against the threat model.

Theorem 2. The private data of each client is protected against the threat model throughout the execution of the secure protocols ($\mathcal{F}_H$, $\mathcal{F}_S$, $\mathcal{F}_T$) for FedMTL aggregation.

Proof. In FedMTL aggregation, $\mathcal{F}_H$ computes the similarity score for each pair of clients, involving addition, element-wise multiplication, matrix-multiplication, and comparison operations in the secret-shared domain. Since the similarity score is calculated over all possible pairs of tasks, one from each client, it does not leak the number or types of tasks. Finally, $\mathcal{F}_S$ and $\mathcal{F}_T$ compute the aggregated parameters using the matrix-multiplication operation. Since these operations are secure in the secret-shared domain following Axiom 2, the clients' private data is protected against the threat model throughout the FedMTL workflow.

7 Evaluation

The evaluation has several goals: demonstrate the feasibility of FedMTL when clients execute different numbers and types of tasks, compare the performance of FedMTL with state-of-the-art approaches, conduct ablation studies to assess the performance of the proposed algorithm compared to alternative approaches, and demonstrate the performance of the privacy-preserving version of FedMTL in terms of accuracy and overhead.

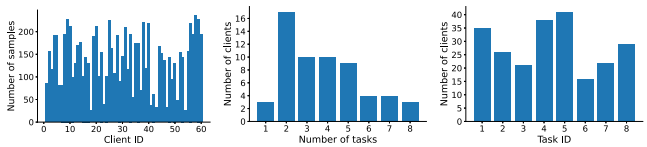
7.1 Experimental Setup

Dataset description, Data and Task distribution. We evaluate FedMTL on two face attribute datasets (CelebA and LFWA) [23] and one indoor scene dataset, NYUD2 [29], which are commonly used to evaluate MTL models. For CelebA/LFWA, we consider $K = 8$ tasks, each involving the classification of $L = 5$ face attributes. For NYUD2, we consider $K = 4$ tasks, each involving $L = 3$ categories for pixel-wise semantic segmentation. Additionally, to assess FedMTL's effectiveness in scenarios with both data and task heterogeneity, we merge CelebA and LFWA to form a unified face attribute dataset, referred to as FaceA.

For each dataset, we select a number of samples D and distribute the samples among N clients using a symmetric Dirichlet distribution. We set $N = 60, 80$ and 20 for CelebA/LFWA, FaceA, and NYUD respectively. Each client C_i splits its dataset into training, validation, and test sets with a distribution of 70%, 15%, and 15%, respectively. In FedMTL, C_i can select any number of tasks to participate in the FL workflow. Then, to choose tasks from the set $\mathbf{U}$, where $K = |\mathbf{U}|$, within the dataset, each client C_i (where $i \in 1, \ldots, N$) selects a random number of tasks K_i such that $1 \leq K_i \leq K$. Subsequently, the client randomly picks K_i tasks from the task-set $\mathbf{U}$. The values for N, D, and K for the datasets are shown in Table 1.

Table 1. Dataset parameters

Dataset	D	N	K	L
CelebA	12,000	60	8	5
LFWA	9,000	60	8	5
FaceA	16,000	80	8	5
NYUD2	1,449	20	4	3



(a) (b) (c)

Figure 5. Sample distribution of tasks among $N = 60$ clients: (a) Number of samples per client, (b) Distribution of clients across different numbers of tasks, (c) Distribution of clients across different tasks

Table 2. Number of parameters in MTL models

Model Architecture	#Parameters in the shared layers	#Parameters in a task-layer
LeNet	1,628,210	2,505
MobileNet_v2	2,223,872	6,405
SegNet	24,943,296	37,123

We consider two different task-distributions: (a) D_1: where each client selects $K_i = 2$ tasks, (b) D_2: each client selects K_i tasks where $1 \leq K_i \leq K$.

To provide an illustration for one dataset, Figure 5 shows the number of samples per client and task distributions for the CelebA dataset using a random seed.

Implementation Details. For the CelebA/LFWA/FaceA datasets, we use LeNet [19] and MobileNetv2 [32] model architectures. For NYUD2 we use SegNet model architecture [1]. For client C_i, we replace the last layer of each model with K_i task layers, where each task layer is a linear layer with L number of neurons for CelebA/LFWA/FaceA and a sequence of convolution layers with L output channels for NYUD2. Table 2 shows the number of shared parameters and task-layer parameters in each model.

We evaluate the performance of FedMTL by comparing it with state-of-the-art approaches, implemented using the open source library PFLlib [1] and use the default hyper-parameters.

To implement secure FedMTL aggregation, we employ CrypTen [17], which provides APIs for creating arithmetic shares of private data and supports $\mathcal{P}$-party SMPC computations. We use 64 bits ($B = 64$) to represent values in A-SS format. The experiments are conducted on a 3.4GHz Intel Core i7, with parties running in separate processes. We implemented the Secure FedMTL prototype to demonstrate the feasibility of the proposed system using CrypTen's basic APIs, without focusing on latency optimizations. It is possible to reduce both the latency and the overall overhead through pre-computation and parallel data processing.

Comparison methods. We compare FedMTL with the following approaches: (i) FedAvg [27], which is the vanilla FL technique, (ii) FedProx [21], which employs a proximal term to formulate the clients' optimization objectives to mitigate the adverse influence of heterogeneity on FL (iii) MOCHA [34], which considers each client as a separate task and applies MTL with model similarity penalization, (iv) pFedMe [37], which uses a regularized loss function to optimize the personalized model w.r.t each client's local data distribution, (v) FedFomo [40], which computes personalized aggre-

[1] https://github.com/TsingZ0/PFLlib

Table 3. Comparison of FedMTL with Baselines

Dataset #	CelebA	LFWA	FaceA	NYUD2	
	Acc (%)	Acc (%)	Acc (%)	mIoU (%)	PixAcc (%)
FedMTL	**83.3**	**70.9**	**78.3**	**37.2**	**74.4**
FedAvg	76.0	64.7	68.5	28.9	62.0
FedProx	76.4	64.7	69.3	30.4	63.9
MOCHA	81.7	67.9	76.9	34.9	71.1
pFedMe	76.2	64.7	69.2	30.3	64.2
FedFomo	80.9	68.9	76.1	35.1	72.0
FedAMP	81.9	68.4	77.1	35.4	71.3

gation weights via minimizing the validation loss on each client based on the model information collected from other clients, and (vi) FedAMP [16], which employs federated attentive message passing to facilitate collaboration among similar clients.

Training and Aggregation Settings. For the experiments, we assume 100% client participation, although FedMTL would still work in the case of partial participation. The number of local training epochs is set to 1, and the number of global communication rounds is set to 20. We employ mini-batch SGD as the local optimizer in all approaches. The batch size for each client is set to 32 for CelebA/LFWA/FaceA, and 2 for NYUD2. For aggregation, we vary the threshold Z from 0.75 to 0.95 linearly. We conduct tests for all methods over 3 runs and report the average results.

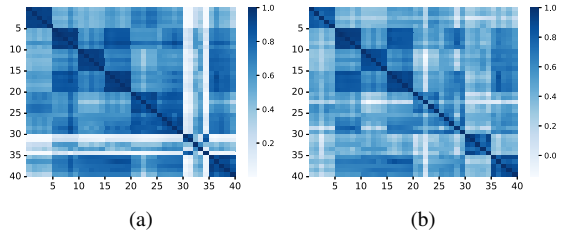
Evaluation Metrics. For CelebA/LFWA/FaceA, we report the average test accuracy of the personalized MTL models across all participant devices. For NYUD2, we report mIoU and pixel accuracy for the semantic segmentation task.

7.2 Results and Analysis

Comparison with Baselines. We compare FedMTL with state-of-the-art approaches when clients are executing the same number of tasks (using task-distribution D_1). The model architecture remains the same across clients, which is required for the baseline approaches. The results presented in Table 3 show that FedMTL achieves better performance compared to other methods for all datasets. FedAvg, FedProx, and pFedMe achieve low accuracy because they aggregate local models without considering task heterogeneity, and the accuracy is impacted by the aggregation of model parameters from conflicting tasks. FedFomo and FedAMP manage to tackle this adverse effect by applying regularization or personalized aggregation weights. In contrast, FedMTL assigns aggregation weights based on similarities in task layers, enabling clients to achieve higher accuracy. FedMTL avoids the complex client-side analysis in FedFomo and FedAMP, reducing computation and communication overhead in resource-constrained devices.

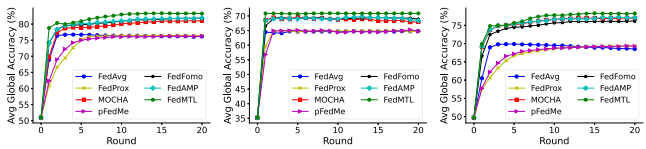
Data and Task Heterogeneity. We consider $N = 80$ clients, where C_i can use data samples from either CelebA or LFWA within FaceA. The tasks $\mathbf{U}$, $K = |\mathbf{U}| = 8$, are distributed among the N clients according to D_1. The results for the FaceA dataset, presented in Table 3, indicate that FedMTL outperforms other approaches by effectively handling both data and task heterogeneity.

To analyze in more detail, we create 2 groups, where the group G_1 and G_2 have data from CelebA and LFWA respectively. There are 20 clients in each group. Each group is divided into 4 sub-groups where the clients in each sub-group G_{ig}, $i \in \{1, 2\}$, $g \in \{1, 2, 3, 4\}$ train MTL model locally for two tasks: $[T_a, T_b]$, where $a = 2g - 1, b = 2g, 1 \leq a, b \leq 8$. As shown in Fig. 6(a), the parameters of the task layers of the clients in the same group G_{ig} are very similar. Although G_{1g} and G_{2g} work on the same tasks, the task layers are not very similar since the datasets are different. Since FedMTL assigns weights based on the similarities of task-specific parameters, it enables the clients to aggregate models considering task and domain heterogeneity, thus performing better compared to other approaches.



(a) (b)

Figure 6. Similarity scores: (a) task-layers (b) shared-layers



(a) (b) (c)

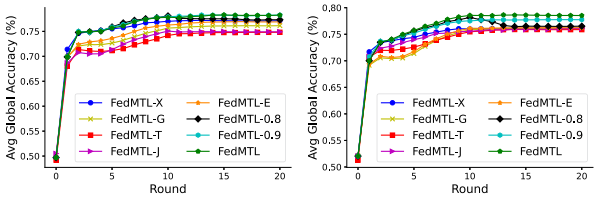
Figure 7. Test accuracy during training for 3 datasets: (a) CelebA, (b) LFWA, (c) FaceA

Efficiency. To evaluate the efficiency of FedMTL aggregation algorithm, we record the evolution of average test accuracy over global communication rounds for CelebA, LFWA, and FaceA datasets, with tasks distributed according to the D_1 distribution. As illustrated in Figure 7, FedMTL achieves higher accuracy than other state-of-the-art approaches and converges within a few rounds.

Different Numbers of Tasks. We evaluate the effectiveness of FedMTL in the case of task heterogeneity, where each client works on a different number of tasks (D_2 task-distribution). The task distributions are presented in Figure 5. We do not compare the results with existing techniques, as they cannot support this scenario directly. FedMTL works well in addressing task heterogeneity and achieves 82.5% and 70.5% accuracy for CelebA and LFWA respectively, and 36.7% mIoU for NYUD2.

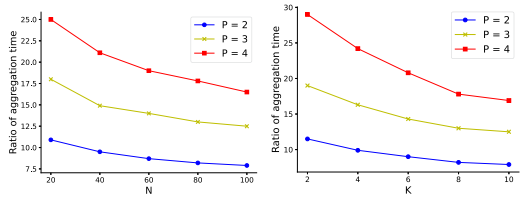
Ablation Studies. We conduct ablation studies to compare different versions of FedMTL in scenarios where clients execute varying sets of tasks. Specifically, we consider the following versions:

- FedMTL-X: To aggregate the shared layers from all clients, it computes the aggregation weights $p_{i,j}$ for C_i by setting $S_{i,j} = 1$ in Eq. 8, and ignores the task layers from other clients by setting $q_{i,j,t} = 1$ if $j = i$, otherwise $q_{i,j,t} = 0$.
- FedMTL-G: To aggregate the shared layers and task layers of task T_t for C_i, it computes the aggregation weights by setting $S_{i,j} = 1$ in Eq. 8 and 9.
- FedMTL-T: To aggregate the shared layers from all clients, it computes $p_{i,j}$ for C_i by setting $S_{i,j} = 1$ in Eq. 8, and aggregates the task-layers by computing the similarity scores following Eq. 4 to Eq. 7 and setting $q_{i,j,t}$ values using Eq. 9 as in FedMTL.
- FedMTL-J: It computes the similarity between two client's models based on the IDs of tasks, instead of task-specific parameters. The similarity score for a pair of clients (C_i, C_j) is computed by $J_{i,j} = \frac{|\mathbf{U}_i \cap \mathbf{U}_j|}{|\mathbf{U}_i \cup \mathbf{U}_j|}$, $J_{i,j} \in [0, 1]$, where $\mathbf{U}_i$ and $\mathbf{U}_j$ are the set of tasks executed by C_i and C_j respectively. It computes the aggregation weights using Eq. 8 and 9, where $S_{i,j} = J_{i,j}$.
- FedMTL-E: For C_i, it only considers the clients executing the same set of tasks. The similarity score for a pair of clients (C_i, C_j) is computed based on the types of tasks as $E_{i,j} = (\mathbf{U}_i == \mathbf{U}_j)$,



(a) (b)

Figure 8. Average global accuracy for FaceA dataset with (a) D_1 task distribution (b) D_2 task distribution



(a) (b)

Figure 9. Ratio of aggregation time between the secure and plain-text versions of FedMTL in P-party setting for varying (a) number of participating clients N and (b) number of tasks K.

$E_{i,j} \in \{0, 1\}$. It computes the aggregation weights using Eq. 8 and 9, where $S_{i,j} = E_{i,j}$.

- FedMTL-Z: It applies our proposed algorithm to aggregate the models based on the similarity scores from the task-specific parameters and uses a fixed threshold value Z.

Comparison with Alternative Approaches. Figure 8 shows the average accuracy per round for FaceA with different task distributions. It illustrates that our approach, FedMTL, utilizing the similarities of task layers, achieves the highest accuracy, 78.3% and 78.5% for D_1 and D_2 task distribution, respectively. FedMTL outperforms FedMTL-X and FedMTL-G since these approaches fail to learn the task layers from the participating clients effectively. FedMTL-T aggregates the task-layers based on the similarity score computed using parameters of the task-layers, and aggregates the shared layers as in conventional FedAvg. However, this approach fails to perform well because shared layers are generic for all clients and may be adversely affected by client models trained on dissimilar sets of tasks. While FedMTL-J and FedMTL-E take into account task heterogeneity, they are adversely affected when clients execute the same task from a different domain. This occurs because the similarity scores are measured based on task ID, potentially assigning identical weights to model parameters from different domains.

Effect of hyper-parameter. We also examine the impact of the threshold value Z used as a hyper-parameter in our similarity score algorithm. Figure 8 shows the average global test accuracy for FaceA dataset using FedMTL-Z with $Z = 0.8$ (FedMTL-0.8) and $Z = 0.9$ (FedMTL-0.9). By setting a constant threshold value, FedMTL-0.8 and FedMTL-0.9 either allow learning from many clients, potentially affecting personalization negatively, or limit learning from a few highly similar models. Figure 8 also shows that FedMTL achieves the highest accuracy as it increases the threshold Z in each round.

Secure Version of FedMTL. In the privacy-preserving version of FedMTL, we encrypt the model parameters, dataset size, and task mapping before uploading the data to the P-aggregators. We evaluate the accuracy of the secure FedMTL for CelebA, LFWA, FaceA and NYUD2 datasets using task-distribution D1. For each dataset, we get the same accuracy performance as the plain text version. As shown in Figure 9, the overhead of privacy-preserving FedMTL w.r.t. the plain text version decreases as the number of clients N or number of tasks T increases. This is due to the fact that, while both versions incur increased computation overhead as N or T increases, the number of communication rounds among the computing parties in the secure version of FedMTL remains constant. For the aggregation of MTL models using the LeNet architecture, the amount of data transferred from a client to each aggregator server is around $12.5MB$. However, as the number of aggregator servers P increases, the communication overhead also rises due to increased communication among the computing parties. Given that FedMTL aggregation does not require real-time processing, it remains feasible for real-world scenarios. For example, with $P = 3$, $N = 80$, $K = 8$, the secure aggregation of MTL models using the LeNet architecture takes around 45 seconds. About 71% of the total time is spent on aggregating shared parameters ($\mathcal{F}_S$) because there are significantly more shared parameters compared to parameters in the task layers. However, it is possible to reduce this overhead through pre-computation and parallel data processing.

To evaluate secure FedMTL considering network delay and bandwidth restrictions, we performed an experiment with 3 AWS instances (t2.micro, US-East-1 region) as the computing parties. For the MTL models using the LeNet architecture, FedMTL takes around 52 seconds to complete the secure aggregation of models from $N = 80$ clients. This indicates that network latency has minimal impact on the overall aggregation time.

8 Conclusion

We presented FedMTL, a novel FL aggregation technique that enables clients to obtain improved personalized MTL models for their sets of tasks by collaborating with other clients. FedMTL computes aggregation weights for each client by analyzing the parameters of task-specific layers in MTL models and applies a layer-wise aggregation policy on the models from participating clients. The FedMTL aggregation can be integrated with established privacy-preserving techniques for secure aggregation, thus guaranteeing the privacy of clients' private data. The experimental results demonstrated that FedMTL outperforms state-of-the-art FL aggregation approaches and can work in cases where clients are involved in different sets of tasks. We also implemented a secure version of FedMTL using secret-sharing SMPC, which achieves the same accuracy performance as plain text while preserving the privacy of client data. In future work, we intend to investigate the performance of FedMTL in cases involving varying data sizes across tasks within a single client. Additionally, we plan to explore scenarios where clients do not explicitly share their task IDs with the aggregator.

Acknowledgement

This research was supported by the National Science Foundation (NSF) under Grants No. NS 2237328 and DGE 2043104.

References

[1] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12): 2481–2495, 2017.

[2] D. Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991. doi: 10.1007/3-540-46766-1_34.

[3] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1175–1191, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468. doi: 10.1145/3133956.3133982. URL https://doi.org/10.1145/3133956.3133982.

[4] R. Cai, X. Chen, S. Liu, J. Srinivasa, M. Lee, R. Kompella, and Z. Wang. Many-task federated learning: A new problem setting and a simple baseline. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 5037–5045, June 2023. doi: 10.1109/CVPRW59228.2023.00532.

[5] R. Caruana. Multitask learning. *Machine Learning*, 28, 07 1997. doi: 10.1023/A:1007379606734.

[6] D. Chaum, I. Damgård, and J. van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 87–119. Springer, 1987. doi: 10.1007/3-540-48184-2_7.

[7] J. Chen and A. Zhang. Fedmsplit: Correlation-adaptive federated multi-task learning across multimodal split networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 87–96, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393850. doi: 10.1145/3534678.3539384. URL https://doi.org/10.1145/3534678.3539384.

[8] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR, 2018.

[9] R. Cramer, I. B. Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015.

[10] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure mpc for dishonest majority – or: Breaking the spdz limits. In J. Crampton, S. Jajodia, and K. Mayes, editors, *Computer Security – ESORICS 2013*, pages 1–18, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40203-6.

[11] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. *IACR Cryptology ePrint Archive*, 2011:535, 01 2011. doi: 10.1007/978-3-642-32009-5_38.

[12] C. T. Dinh, T. T. Vu, N. H. Tran, M. N. Dao, and H. Zhang. A new look and convergence rate of federated multitask learning with laplacian regularization. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–11, 2022. doi: 10.1109/TNNLS.2022.3224252.

[13] Y. Dong, X. Chen, L. Shen, and D. Wang. Eastfly: Efficient and secure ternary federated learning. *Computers & Security*, 94:101824, 2020.

[14] J. Geng, Y. Mou, F. Li, Q. Li, O. Beyan, S. Decker, and C. Rong. Towards general deep leakage in federated learning. *arXiv preprint arXiv:2110.09074*, 2021.

[15] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran. An efficient framework for clustered federated learning. *Advances in Neural Information Processing Systems*, 33:19586–19597, 2020.

[16] Y. Huang, L. Chu, Z. Zhou, L. Wang, J. Liu, J. Pei, and Y. Zhang. Personalized cross-silo federated learning on non-iid data. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 7865–7873, 2021.

[17] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten. Crypten: Secure multi-party computation meets machine learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 4961–4973. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/2754518221cfbc8d25c13a06a4cb8421-Paper.pdf.

[18] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998. doi: 10.1109/5.726791.

[20] R. Li, F. Ma, W. Jiang, and J. Gao. Online federated multitask learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 215–220, 2019. doi: 10.1109/BigData47090.2019.9006060.

[21] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[22] J. Liu, J. Wu, J. Chen, M. Hu, Y. Zhou, and D. Wu. Feddwa: Personalized federated learning with dynamic weight adjustment. In E. Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 3993–4001. International Joint Conferences on Artificial Intelligence Organization, 8 2023. doi: 10.24963/ijcai.2023/444. URL https://doi.org/10.24963/ijcai.2023/444. Main Track.

[23] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3730–3738, 2015. doi: 10.1109/ICCV.2015.425.

[24] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[25] O. Marfoq, G. Neglia, A. Bellet, L. Kameni, and R. Vidal. Federated multi-task learning under a mixture of distributions. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15434–15447. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/82599a4ec94aca066873c99b4c741ed8-Paper.pdf.

[26] B. McCann, N. S. Keskar, C. Xiong, and R. Socher. The natural language decathlon: Multitask learning as question answering, 2019. URL https://openreview.net/forum?id=B1lfHhR9tm.

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

[28] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3994–4003, 2016.

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

[30] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999.

[31] L. Pascal, P. Michiardi, X. Bost, B. Huet, and M. A. Zuluaga. Maximum roaming multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9331–9341, 2021.

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

[33] J. Shu, T. Yang, X. Liao, F. Chen, Y. Xiao, K. Yang, and X. Jia. Clustered federated multitask learning on non-iid data with enhanced privacy. *IEEE Internet of Things Journal*, 10(4):3453–3467, 2023. doi: 10.1109/JIOT.2022.3228893.

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

[35] J. So, B. Guler, and A. Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, PP:1–1, 01 2021. doi: 10.1109/JSAIT.2021.3054610.

[36] A. C. Stickland and I. Murray. BERT and pals: Projected attention layers for efficient adaptation in multi-task learning. *CoRR*, abs/1902.02671, 2019. URL http://arxiv.org/abs/1902.02671.

[37] C. T. Dinh, N. Tran, and J. Nguyen. Personalized federated learning with moreau envelopes. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21394–21405. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f4f1f13c8289ac1b1ee0ff176b56fc60-Paper.pdf.

[38] L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

[39] Z. Ying, Y. Zhang, and X. Liu. Privacy-preserving in defending against membership inference attacks. In *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, pages 61–63, 2020.

[40] M. Zhang, K. Sapra, S. Fidler, S. Yeung, and J. M. Alvarez. Personalized federated learning with first order model optimization. *arXiv preprint arXiv:2012.08565*, 2020.

[41] Z. Zhang, J. Li, S. Yu, and C. Makaya. Safelearning: Secure aggregation in federated learning with backdoor detectability. *IEEE Transactions on Information Forensics and Security*, 18:3289–3304, 2023. doi: 10.1109/TIFS.2023.3280032.

[42] L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. *Advances in neural information processing systems*, 32, 2019.

[43] W. Zhuang, Y. Wen, L. Lyu, and S. Zhang. Mas: Towards resource-efficient federated multiple-task learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 23414–23424, 2023.