Learning Task-Specific Initialization for Effective Federated Continual Fine-Tuning of Foundation Model Adapters

Danni Peng∗†, Yuan Wang∗†, Huazhu Fu∗‡, Qingsong Wei∗†, Yong Liu∗†, Rick Siow Mong Goh∗†
∗Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), Singapore
†{dannip, wang_yuan, wei_qingsong, liuyong, gohsm}@ihpc.a-star.edu.sg
‡hzfu@ieee.org

Abstract—As large models demonstrate their power across a wide range of applications, the federated learning (FL) community has also begun to seek solutions for leveraging these large models in a communication- and computation-efficient manner. In light of this, fine-tuning of lightweight adapters has emerged as a promising solution for adopting large models in FL. Another real-world challenge concerns with non-static data streams encountered by local clients, requiring continuous adapter fine-tuning to accommodate new tasks. In this work, we propose a method for effective continual adapter fine-tuning in FL (FedCAF), aimed at enhancing a client’s local learning on new tasks. Specifically, FedCAF employs both cross-task and cross-client knowledge transfer to generate an informed, task-specific initialization. By learning a set of attentive weights to combine past task models from all clients, FedCAF produces task-specific initializations that effectively enable better and faster task learning. On the large-scale cross-domain dataset DomainNet, we show that FedCAF significantly outperforms several competitive personalized and continual learning baselines under both class-incremental and domain-incremental settings.

Index Terms—Federated Continual Learning, Adapter Fine-Tuning, Knowledge Transfer, Task-Specific Initialization

I. INTRODUCTION

Amidst the pervasive use of computing devices and heightened privacy concerns in today’s big data era, federated learning (FL) [1], [2] has emerged as a solution that enables collaborative training across multiple clients (e.g., personal devices, private institutions) without infringing on data privacy. Typically, clients conduct training locally and upload their models to a central server for aggregation. The server then sends the aggregated model back to the clients for the next round of updates. During this process, only model checkpoints are communicated between parties, while the data remains exclusively with the local clients for data privacy. However, this FL paradigm is not without its constraints, such as limited bandwidth and computational power on the client side.

Recently, large models have demonstrated exceptional performance in a wide range of applications [3], [4]. To leverage the powerful representations, a prevalent strategy now involves fine-tuning a large, extensively pre-trained foundation model (FM) [5]–[7] on the downstream tasks instead of training from scratch. However, in the context of FL, performing full parameters fine-tuning of large models can be prohibitive considering the sheer size of FMs, which can lead to high communication costs in transporting the model, and also significantly increase the computational burdens on the clients when performing full fine-tuning locally [8]. Fortunately, the advent of various lightweight adapters [9]–[11] provides a more feasible solution for tuning the large models in FL setting. An adapter is a small neural network module that can be inserted at multiple layers of a pre-existing foundation model. During fine-tuning, only the adapter is made trainable to adapt the foundation model to the downstream tasks while the bulky foundation backbone is kept frozen, which effectively reduces the training costs. In the context of FL, a direct adoption is to perform adapter fine-tuning on the client side and send only the updated adapters to the server for aggregation [12], [13].

Another key consideration in real-world applications of FL is that clients continuously receive new data or encounter new tasks. During this process, the local data distribution may shift from the one learned previously [14]. This further complicates...
the problem, as data heterogeneity now exists along both temporal and spatial dimensions. Figure 1 illustrates adapter fine-tuning in this federated continual learning (FCL) scenario. Here, each client encounters a sequence of tasks locally over time and continually fine-tunes the adapter for each new task. A handful of recent works have explored methods to handle catastrophic forgetting in FCL [15]–[17], which mainly focus on maintaining performance on old tasks while learning new tasks. However, another important aspect – forward transfer – is largely overlooked in FCL research. Instead of merely maintaining old tasks’ performance, forward transfer aims to improve learning of new tasks, resulting in either faster learning or better final performance, by utilizing knowledge from old tasks (as illustrated in Figure 2a). This can be particularly helpful in situations where the data available for learning a new task for a client is limited (e.g., a hospital handling new diseases with few data points) or when a well-adapted model needs to be generated quickly (e.g., due to limited computational resources, or in urgent situations such as dealing with a fast-spreading new disease).

When considering the continual problem within FL, utilizing knowledge from other clients can also be beneficial for a client’s local task learning. Cross-client transfer (as shown in Figure 2c) has been one of the key focuses in personalized FL (pFL) [18]–[20] – a branch that prioritizes the client’s local performance and shares knowledge among different clients through the FL system to enhance personalized learning. In this work, we aim to leverage knowledge transfer across both tasks and clients (as shown in Figure 2c), harnessing useful information from both temporal and spatial dimensions for more effective client’s adapter fine-tuning on new tasks.

Generally, knowledge transfer can be achieved via distillation [21], gradient-based methods [22], or architecture-based methods [17], [23]. In this work, we consider incorporating useful knowledge into the model initialization to achieve effective continual adapter fine-tuning in FL, a method we term FedCAF. Specifically, in our approach, task-specific models obtained at each client are sent to the server and aggregated for cross-task knowledge amalgamation. Before the learning of each new task, the aggregated models are sent back to the client to undergo a fast pre-learning stage, where a set of attentive weights to combine the aggregated models is learned towards the client’s new task objective. The learned weights are then used to combine the aggregated models to form the initialization for adapter fine-tuning on the new task. On the extensive cross-domain dataset DomainNet [24], we construct two standard continual settings: class-incremental and domain-incremental [25], and demonstrate that our proposed FedCAF outperforms several competitive baselines, achieving better and faster task learning by leveraging cross-task and cross-client knowledge through the learned initialization.

II. RELATED WORKS

A. Foundation Model Fine-Tuning in Federated Learning

As the benefits of large models become increasingly prominent, studies have been dedicated to leveraging the power of these models in the FL setting [8], [26]. ViT-FL [27] first showed that using transformer-based foundation models like ViT [5] handles data heterogeneity in FL better than the convolutional networks. However, full parameters fine-tuning is often not feasible in real-world FL with edge devices. Hence, a handful of recent works have explored the potential of adopting parameter-efficient fine-tuning methods like adapter fine-tuning and prompt tuning in FL, where only a small number of task-specific parameters are tuned. FedCLIP [12] tunes and aggregates only the adapters that are applied to the image encoder of CLIP [7]. FedPrompt [28] and PromptFL [13] instead aggregate only the updated prompt tokens attached to the text inputs. In this work, we further consider adapter fine-tuning in FL with continual adaptation to new tasks.

B. Federated Continual Learning (FCL)

Similar to general continual learning [14], recent FCL works targeting unforgetting can be broadly categorized into three groups. Firstly, regularization-based methods like FedCurv [29] and FLwF [16] typically maintain the performance of old tasks and global model by explicitly constraining the model updates with a regularization or distillation term. Replay-based approaches preserve past knowledge by reusing past examples while learning new tasks [30], or leveraging generative methods [15]. Architecture-based approaches, on the other hand, assign isolated model parameters to different tasks and reuse model parameters of past tasks [17]. Despite the increased attention on applying FL in continual scenarios, existing works mainly focus on maintaining performance on old tasks, while the potential of leveraging knowledge from old tasks to improve the current task remains largely unexplored. This, however, could be a promising research direction for further reducing computation costs for edge users.

C. Personalized Federated Learning (pFL)

To better handle data heterogeneity across clients, pFL works focus on improving clients’ local performance while leveraging relevant knowledge from other clients, shared through the FL platform. Generally, two lines of methods have been developed to enhance clients’ local performance. The first approach focuses on learning a global model such that the clients can easily fine-tune to achieve good personalized performance from the global model [31]. Another approach directly optimizes the local client’s models by employing personalized aggregation of other client’s models, weighted using distance-based metrics [18]–[20]. In this work, we also leverage task-specific attentive weights for model aggregation. Instead, we adopt a learning-based strategy to optimize the attentive weights for the new task objective, which effectively improves new task learning in the continual setting.

III. METHODOLOGY

A. Problem Formulation

In a standard FL setup with $k$ clients and a central server, each client $i \in \{1, \ldots, k\}$ owns its private dataset $D_i$. Traditional FL aims to learn a global model $\theta$ that optimizes performance over all $k$ clients’ data: $\min_{\theta} \sum_{i=1}^{k} L(\theta; D_i)$, where
\( \mathcal{L} \) is some arbitrary loss function. To address the data heterogeneity problem (non-iid or unbalanced) across different clients, personalized FL (pFL) adopts a more flexible objective and learns \( k \) personalized models \( \{\theta_i, \cdots, \theta_k\} \), each optimized for a client’s local data: \( \min \{\theta_i, \cdots, \theta_k\} \sum_{i=1}^{k} \mathcal{L}(\theta_i; D_i) \).

Federated continual learning (FCL) further assumes that data heterogeneity exists not only across clients but also along time dimension. That is, the clients’ data is not static and they continuously receive new data with potentially evolving distributions. Formally, each client \( i \) experiences a local stream of tasks \( \{T_i^1, T_i^2, T_i^3, \cdots\} \) and learns a tasks in a sequential manner. Each task \( T_i^t \in D_i \) of client \( i \) can be considered as a subset drawn from the local dataset \( D_i \). Let \( X_i^t \) and \( Y_i^t \) denote the feature space and label space of \( T_i^t \) data, the tasks experienced locally by a client may have non-overlapping label spaces, i.e., \( Y_i^t \cap Y_j^t = \emptyset, \forall j \neq t \) (class-incremental), or shifting distributions in feature space, i.e., \( P_{X_i^t}(x) \neq P_{X_j^t}(x), \forall j \neq t \) (domain-incremental).

While most works in FCL aim to maintain performance on all previous tasks, i.e., \( \min \{\theta_i, \cdots, \theta_k\} \sum_{i=1}^{k} \sum_{j=1}^{t} \mathcal{L}(\theta_i; T_i^j) \), in this work, we focus on learning the new task more effectively. That is, at period \( t \), we aim to learn personalized models \( \{\theta_i^1, \cdots, \theta_i^k\} \) on new tasks \( \{T_i^1, \cdots, T_i^t\} \) across the \( k \) clients:

\[
\min_{\theta_1, \cdots, \theta_k} \sum_{i=1}^{k} \mathcal{L}(\theta_i; T_i^t). \tag{1}
\]

### B. Foundation Model Adapter Fine-Tuning

Generally, an adapter is a small neural network inserted at various layers of a large foundation model. Adapter fine-tuning involves tuning only the adapter parameters on the specific tasks while keeping the foundation backbone frozen.

Formally, let \( F^*(\cdot) \) denote an off-the-shelf transformer-based foundation model (e.g., ViT [5], CLIP [7]), and \( g_\omega(\cdot) \) denote the adapter module parameterized by \( \omega \). Given an input \( x \), the adapter can be applied at the input level [9] or at the layer outputs [11] (i.e., \( F^*(g_\omega(x)) \) or \( g_\omega(F^*(x)) \)), or as a parallel processor where the outputs of both functions are combined [11] (i.e., \( F^*(x) + g_\omega(x) \)). Without loss of generality, we represent the model of applying \( g_\omega(\cdot) \) to \( F^*(\cdot) \) as \( f_\omega(\cdot) \) collectively, where only \( \omega \) is trainable. Moreover, the final classifier layer, denoted by \( h_\psi(\cdot) \), also needs to be trained to adapt to the specific task. Hence, given a new task \( T_i^t \) of client \( i \), our goal is to tune both the adapter and the classifier \( \theta_i^t = [\omega_i^t, \psi_i^t] \), such that when applied to the fixed foundation backbone, they yield the best performance on the new task. Overall, our adapter fine-tuning objective is as follows:

\[
\min_{[\omega_i^1, \psi_i^1; \cdots; \omega_i^k, \psi_i^k]} \sum_{i=1}^{k} \mathcal{L}(\omega_i^t, \psi_i^t; T_i^t), \tag{2}
\]

where \( \mathcal{L}(\omega_i^t, \psi_i^t; T_i^t) := \sum_{(x, y) \in T_i^t} l(h_\psi(f_\omega(x)), y) \).

### C. Learning Task-Specific Initialization for Effective Continual Adapter Fine-Tuning (FedCAF)

In this section, we describe our FedCAF which learns task-specific initialization for effective adapter fine-tuning in FL. Overall, learning the task initialization consists of two steps before each round of new task learning: 1) model aggregation at server, and 2) learning the attentive weights for task-specific initialization at local clients.

#### a) Model Aggregation at Server: At each client \( i \), the series of tasks is learned in a sequential manner. After learning each task, the task-specific adapter and classifier will be sent to the server for client-specific aggregation of all the seen tasks. Suppose we are at the completion of learning task \( T_i^{t-1} \) at client \( i \). The task-specific adapter and classifier \( \theta_i^{t-1} = (\omega_i^{t-1}, \psi_i^{t-1}) \) are sent to the server and aggregated with parameters of all the seen tasks to obtain a set of client-specific aggregated adapters \( \{\bar{\omega}_1, \cdots, \bar{\omega}_k\} \) and classifiers \( \{\bar{\psi}_1, \cdots, \bar{\psi}_k\} \) as follows:

\[
\bar{\omega}_i = \sum_{j=1}^{t-1} p_i^j \cdot \omega_j^t, \quad \bar{\psi}_i = \sum_{j=1}^{t-1} p_i^j \cdot \psi_j^t, \tag{3}
\]

where \( p_i^t = \frac{|T_i^t|}{\sum_{k=1}^{t} |T_i^k|} \) is the data size weighted aggregation weight for task \( j \) parameters of client \( i \). This step amalgamates cross-task knowledge within each client.

#### b) Learning Attentive Weights for Task-Specific Initialization at Local Clients: Before learning the new task \( T_i^t \), client \( i \) first downloads the set of client-specific aggregated adapters \( \{\bar{\omega}_1, \cdots, \bar{\omega}_k\} \) and classifiers \( \{\bar{\psi}_1, \cdots, \bar{\psi}_k\} \) from the server, as obtained in the previous step. Next, we introduce two sets of learnable attentive weights \( \alpha = [\alpha_1, \cdots, \alpha_k] \in \mathbb{R}^k \) and \( \beta = [\beta_1, \cdots, \beta_k] \in \mathbb{R}^k \), associated with the set of adapters and the set of classifiers respectively. These attentive weights are used to combine the client-specific aggregated models to form a good initialization for the new task.

Finding the optimal attentive weights is non-trivial. In this work, we propose learning the attentive weights such that the model combined using these weights performs optimally on the new task objective. Specifically, let \( \bar{\omega}(\alpha) = \sum_{i=1}^{k} \alpha_i \cdot \bar{\omega}_i \) and \( \bar{\psi}(\beta) = \sum_{i=1}^{k} \beta_i \cdot \bar{\psi}_i \) represent the combined adapter and classifier. The attentive weights are learned by optimizing the combined model on the new task \( T_i^t \):

\[
\alpha_i^t, \beta_i^t = \arg \min_{\alpha, \beta} \mathcal{L}(\bar{\omega}(\alpha), \bar{\psi}(\beta); T_i^t). \tag{4}
\]

We apply gradient descent to update \( \alpha \) and \( \beta \) as follows:

\[
\alpha_i^t \leftarrow \alpha_i^t - \eta_i \nabla_{\alpha_i} \mathcal{L}(\bar{\omega}(\alpha), \bar{\psi}(\beta); T_i^t), \quad \beta_i^t \leftarrow \beta_i^t - \eta_\beta \nabla_{\beta} \mathcal{L}(\bar{\omega}(\alpha), \bar{\psi}(\beta); T_i^t), \tag{5}
\]

where \( \nabla_{\alpha} \mathcal{L}(\alpha) \) and \( \nabla_{\beta} \mathcal{L}(\beta) \) are simply \( [\bar{\omega}_1, \cdots, \bar{\omega}_k] \) and \( [\bar{\psi}_1, \cdots, \bar{\psi}_k] \), respectively.

As a result, the learned attentive weights \( \alpha_i^t \) and \( \beta_i^t \) determines how to leverage the knowledge from other clients’ previous tasks to facilitate the learning of client \( i \)’s new task \( T_i^t \). Note that this attentive weight learning process is conducted before the actual task learning. To minimize computational overhead, it is designed to be much shorter in duration compared to the actual task learning (e.g., it is performed for only 1 or 2 epochs in our experiments).
In the following, we first introduce our experimental setup and then discuss the experimental results.

A. Experimental Setup

a) Dataset and Settings:

We conduct our experiments on the large-scale cross-domain DomainNet dataset [24]. It comprises real-world images from 345 classes across 6 different domains: Clipart, Infograph, Painting, Quickdraw, Real, and Sketch. We sample 10% for our experiments, resulting in around 60k examples.

b) Implementation details:

For the foundation model, we use ViT-B/16 [5] pre-trained on ImageNet21k. For the adapter, we adopt LoRA [11] which applies low-rank decomposition to the Q and V matrices of each attention block. We set the rank to be 48. All fine-tuning is conducted on NVIDIA A100 GPUs with 40GB memory.

For each new task, we fine-tune the adapter and the 15-way classifier for 20 epochs with a learning rate 0.005. Note that global communication occurs only at the start of each new task learning for sharing the aggregated adapters and classifiers.

For our FedCAF, the attentive weights $\alpha_i$ and $\beta_t$ learned for 1 epoch at the start of each task learning, with initial value as $\frac{1}{k}$. We set the learning rates for both to be 0.05.

c) Baselines and Metrics:

To evaluate the effectiveness of our FedCAF initialization, we first introduce five naive baselines: 1) Rand initializes the adapter and the classifier (denoted collectively as $\theta$ for simplicity) randomly for each new task; 2) GlobalAvg aggregates $\theta$ of all the seen tasks from all clients to form the initialization; 3) ClientAvg aggregates $\theta$ of all the seen tasks of a client to create the initialization for that client’s new task; 4) FedAvg aggregates $\theta$ from only the last tasks of all clients to form the initializations; 5) ClientCont directly uses $\theta$ from the last task of the client as the initialization for that client’s new task.

We further include 3 pFL methods modified for our continual setting and 2 FCL methods. For the pFL baselines, PerFedAvg [31] aggregates a global initialization that optimizes performance after one step gradient descent; FedFomo [19] employs personalized aggregation weights to generate the initialization based on the difference in loss of a client’s model on its local dataset and those of the others; and FedAMP [18] generates aggregation weights based on the distance between model parameters. For a fair comparison, all the pFL methods are used to generate the initialization only before the actual task learning. For the FCL baselines, FedCurv [29] constrains model updates with a regularization term, while FLwF [16] regularizes model updates via distillation. Note that like most of the existing FCL works, both FedCurv and FLwF are designed to defy forgetting of old tasks.

To evaluate the performance of new task learning, we employ two metrics: the final accuracy (ACC), attained at the last epoch of task learning, and the learning curve area (LCA), computed by averaging the accuracy across all epochs throughout the task learning [32]. These two metrics effectively measure how well and how quickly a task is learned. For each compared method, we run 3 trials with different random seeds and report the mean and standard deviation.

B. Experimental Results

Tables I and II present the results for class-incremental and domain-incremental settings respectively. The ACC and LCA reported are averaged over all tasks and all clients. For all experiments, we conduct 3 trials and report the mean and SD. We also report the improvements over the Rand baseline.
C. Class-Incremental Results

Firstly, we examine the class-incremental performance as shown in Table I. We observe that GlobalAvg and FedAvg, which have access to cross-client knowledge, generally outperform ClientAvg and ClientCont. This underscores the importance of leveraging global information to generate a good initialization. The ClientCont baseline performs especially poorly, as using a 15-way classifier from a completely different task is not helpful for learning another task. ClientAvg, which aggregates a set of different classifiers, alleviates this problem.

For the pFL baselines, we notice that Per-FedAvg performs similarly to GlobalAvg in terms of ACC but with a slightly higher LCA. This is because Per-FedAvg employs a global initialization that optimizes for better personalized performance after a one-step gradient update, resulting in a faster learning process. As a personalized aggregation approach, FedFomo outperforms FedAMP, which implies that assigning weights to models based on local task performance is more effective. For the FCL methods, FedCurv and FLwF, we observe that they perform poorly in new task learning. This is because they are both designed to maintain the performance of old tasks instead of optimizing for new task. The results indicate that merely maintaining performance on old tasks does not necessarily lead to knowledge transfer to new tasks. Specially designed mechanism is required to achieve effective transfer.

Finally, for our proposed FedCAF, we observe that it significantly outperforms all the baselines. This can be attributed to its ability to achieve a fine balance between GlobalAvg and ClientAvg by automatically learning the attentive weights. Moreover, it proves superior to the strong personalized aggregation baseline FedFomo, demonstrating the advantages of directly optimizing the attentive weights for the new task objective. Figure 4 shows the sequential learning performance of 23 class-incremental tasks for 6 clients. Here, we compare FedCAF with Rand, GlobalAvg and ClientAvg. We can see that it consistently learns better and faster than the baselines.

D. Domain-Incremental Results

For the domain-incremental results in Table II, we see that the client-specific methods ClientAvg and ClientCont perform better than GlobalAvg and FedAvg. This is because in this setting, each client learns the same 15-way classification task. Reusing classifiers of the same client benefits learning of new task which involves the same 15 classes from a new domain. For the pFL and FCL baselines, most of the trends observed here are similar to those in the class-incremental setting. However, here we observe better performance for FedFomo and FedAMP, as they probably assign greater weights to tasks based on the model’s performance on local task instead of optimizing for new task. The results indicate that merely maintaining performance on old tasks does not necessarily lead to knowledge transfer to new tasks. Specially designed mechanism is required to achieve effective transfer.

Furthermore, in the domain-incremental setting, each client learns the same 15-way classification task. Reusing classifiers of the same client benefits learning of new task which involves the same 15 classes from a new domain. For the pFL and FCL baselines, most of the trends observed here are similar to those in the class-incremental setting. However, here we observe better performance for FedFomo and FedAMP, as they probably assign greater weights to models that are similar to a client’s local model, achieving similar effects to ClientAvg and ClientCont methods. For FedCurv and FLwF, the results are also more satisfactory here.
for the same reason, as drawing the updated model closer to the previous task model of the same client helps retain the useful information from the same classification task.

Nevertheless, our proposed FedCAF still performs the best in this challenging setting, implying the effectiveness of learning-based cross-client transfer in this cross-client label-skew scenario, where there seems to be no shareable knowledge among clients. Figure 5 shows the sequential learning performance of 6 domain-incremental tasks for 23 clients.

V. CONCLUSION

In this work, we propose FedCAF for effective federated continual fine-tuning of adapters. This method incorporates cross-task and cross-client knowledge into the task-specific initialization by learning a set of attentive weights. On DomainNet, we demonstrate the effectiveness of our method under two standard continual settings.

ACKNOWLEDGMENT

This Research is supported by the RIE2025 Industry Alignment Fund – Industry Collaboration Project (IAF-ICP) (Award No: I2301E0020), administered by A*STAR.

REFERENCES


