Traceable Federated Continual Learning

Qiang Wang[1]*, Bingyan Liu[1]*†, Yawen Li[2]
[1]School of Computer Science, Beijing University of Posts and Telecommunications
[2]School of Economics and Management, Beijing University of Posts and Telecommunications
q.wang@bupt.edu.cn, bingyanliu@bupt.edu.cn, warmly0716@126.com

Abstract

Federated continual learning (FCL) is a typical mechanism to achieve collaborative model training among clients that own dynamic data. While traditional FCL methods have been proved effective, they do not consider the task repeatability and fail to achieve good performance under this practical scenario. In this paper, we propose a new paradigm, namely Traceable Federated Continual Learning (TFCL), aiming to cope with repetitive tasks by tracing and augmenting them. Following the new paradigm, we develop ***TagFed****, a framework that enables accurate and effective* ***T****racing,* ***aug****mentation, and* ***Fed****eration for TFCL. The key idea is to decompose the whole model into a series of marked sub-models for optimizing each client task, before conducting group-wise knowledge aggregation, such that the repetitive tasks can be located precisely and federated selectively for improved performance. Extensive experiments on our constructed benchmark demonstrate the effectiveness and efficiency of the proposed framework. We will release our code at:* https://github.com/P0werWeirdo/TagFCL.

1. Introduction

With the popularity of federated learning (FL) [26, 27, 33], more and more researchers are devoted to applying FL to a variety of applications such as object detection [29], medical image analysis [28], and recommendation systems [39]. Despite being effective, most of the existing FL methods assume that client data are stable and will not be changed over time. However, in real-world applications, data may come continuously and we cannot predict the data variability in advance. The research community has noticed this problem and proposes a new concept called federated continual learning (FCL) [8, 43], where each local client receives data in a streaming manner and a central server attempts to achieve federation in such a dynamic environment.

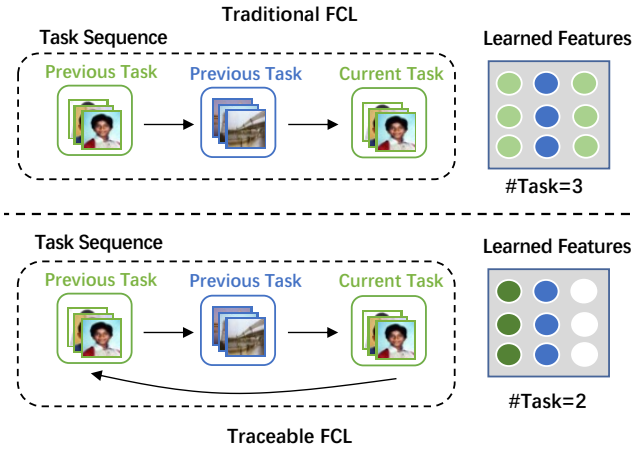

Figure 1. Comparison of traditional FCL and traceable FCL in one client. Here different colors denote different features. The white color means that the neuron has no feature (i.e., unused). The darker color indicates the stronger feature representation ability.

We notice that current FCL solutions pay more attention to the typical problems existed in CL such as catastrophic forgetting [8]. However, all of them obey the following unpractical data setting: *the upcoming task data are completely different from previous tasks*. We argue that in real-world scenarios, similar or even identical task data have a large possibility to appear many times. **On the one hand**, the client itself is full of uncertainty. When the FCL process runs for a long time, it is inevitable to encounter repetitive tasks, especially for the federated scenario that may include a large number of clients. **On the other hand**, previous tasks may have limited data in that time and cannot achieve good performance. Therefore, the developers have the motivation to further improve the performance by collecting more data for that task in the future. In a word, the repetitive task sequence is commonly seen and should be regarded as a default setting.

Unfortunately, to the best of our knowledge, existing FCL techniques do not take the task repeatability into con-

*The first two authors contributed equally.
†Bingyan Liu is the corresponding author.

sideration and fail to achieve good performance under the condition (see in Section 3.1). To fill the gap, in this paper, we propose a new paradigm, namely **Traceable Federated Continual Learning (TFCL)**, which aims at *tracing the previous repetitive data features to the current one and augmenting them before federation, in order to further boost the performance.* As shown in Figure 1, for each client, instead of treating each incoming task as equal like the traditional FCL does, TFCL checks whether the current task is a repetitive task, which will be traced and connected to the previous target task for further processing before federation. Intuitively, compared to the traditional continual learning in the client side, the proposed paradigm decreases the number of tasks since the repetitive task is not considered as a new task, which reduces the loss caused by new task learning and benefits the overall performance.

Achieving TFCL is non-trivial and requires overcoming the following key challenges. First, the client cannot store all the previous task data due to the limited storage, which means that the trained model in the current state is the only resource we can utilize to infer and augment the previous features. Considering the poor interpretability of neural networks, it is challenging to accurately trace and optimize the corresponding data features. Second, the task repeatability in different clients may be irregular, which suggests that we cannot directly federate them because in a certain timeline, the feature distribution of repetitive tasks among clients might be highly heterogeneous. Therefore, a novel federation scheme is needed to cope with the heterogeneous situation.

In this paper, we address the aforementioned challenges by proposing **TagFed**, a framework that enables accurate and effective **T**racing, **aug**mentation, and **Fed**eration for TFCL. The rationale behind TagFed is to (1) decompose the whole model into a series of marked sub-models to selectively optimize different client task; (2) conduct group-wise knowledge aggregation for efficient federation. Specifically, to tackle the first challenge, we present *traceable task learning*, where the model pruning technique [10, 22] is utilized to iteratively generate sub-models for different client tasks. Note that each new task can only use the neurons that are not occupied for previous tasks to learn the corresponding features. Besides, we mark the weight of each task and save a weight copy of previous repetitive tasks. This allows selective retraining of specific weights for the current repetitive task, while still utilizing weight copies for non-repetitive tasks. In this way, when a repetitive task comes, we can accurately locate its corresponding features by the marked neurons, which will be easily retrained as augmentation, without introducing extra learning parameters like a new task.

TagFed addresses the second challenge by designing *group-wise knowledge aggregation*, whose key idea is to construct several model groups in the server and federate knowledge in a client-server transfer manner. For similar or identical repetitive tasks, their client knowledge is sequentially transferred to a same server model (i.e., a group) as aggregation and the server model then puts the aggregated knowledge back to each client. Here we do not use the head-to-head aggregation methods (e.g., FedAvg [33]) because the model parameters of previous tasks are frozen and cannot be modified.

Considering that there is no available benchmark to conduct TFCL, we manually construct a benchmark based on public datasets (details in the appendix), where we simulate the repeatability range and degree for different clients. We evaluate our framework on the benchmark and compare it to the traditional FCL baselines. The results show that our approach significantly outperforms other methods on accuracy performance by a large margin while using roughly 50% communication cost. In addition, we conduct a series of in-depth theoretical analyses (details in the appendix) and empirical experiments to prove the effectiveness of the proposed framework.

This paper makes the following contributions:

- We propose a new FCL paradigm, called Traceable Federated Continual Learning (TFCL), where previous tasks can be traced and further optimized to deal with the repetitive task sequence in an FL system. To the best of our knowledge, this is the first attempt in the literature to study and explore TFCL.
- We design and implement TagFed, a framework to enable accurate and effective TFCL. Through feature tracing, sub-model augmentation, and group-wise knowledge federation, TagFed can achieve higher model accuracy at a small communication cost.
- Extensive experiments on our constructed benchmark demonstrate the superiority of TagFed over other baselines.

2. Related Work

2.1. Continual Learning

Continual learning (CL) aims to learn deep models from the data that gradually come in a sequence during the training phase. Recent continual learning approaches mainly include two categories: task-based CL, where all class data come but are from a different dataset for each phase [3–5]; class-based CL, which means that each phase receives the data of a new set of classes from an identical dataset [2, 13, 14]. The latter one is typically called class-incremental learning (CIL) and our work follows this data setting. Specifically, most of the related methods pay more attention to solving the problem of forgetting old data. For instance, Mallya *et al.* [31] packed multiple tasks into a single model by pruning the unimportant neurons. Douil-

lard *et al.* [9] proposed a spatial-based distillation loss applied throughout the model and a representation comprising multiple proxy vectors for each object class to enhance the performance. In our work, we are devoted to extending traditional CL under the federation scenario and propose a new paradigm, namely *traceable federated continual learning*, to address the task repeatability problem in a streaming data sequence. Note that Zhu *et al.* [46] also discussed the recurrence of tasks. However, they focused on periodical distribution shift, which is different from ours that is specifically designed for repetitive tasks without any periodical nature.

2.2. Federated Learning

Federated learning (FL) has been proposed as a privacy-preserving distributed machine learning approach that enables training on a large corpus of decentralized user data [11, 33, 35, 40, 41]. Much effort has been made in this hot research topic, including attacks/defense on FL [1, 19, 34, 42, 45], personalized FL [25, 38, 44], data heterogeneity on FL [24, 32, 37], communication-efficient FL [15, 16], etc. Recently a few works have begun to explore FL on dynamic client data, which are called *federated continual learning (FCL)*. For example, Yoon *et al.* [43] proposed FedWeIT, a framework that can achieve effective FCL by alleviating the interference between incompatible tasks and boosting the positive knowledge transfer across clients during learning. Dong *et al.* [8] proposed a practical scenario, called Federated Class-Incremental Learning (FCIL), where each local client collects data continuously with its own preference and allows new clients with unseen new classes to join in the FL training at any time. Luopan *et al.* [30] proposed FedKNOW, aiming at an accurate and scalable federated continual learning via a novel signature task knowledge. Different from the above FCL approaches assuming that the task sequence among clients is non-repetitive, our framework manages to effectively cope with the task repeatability and achieve traceable FCL.

3. Problem Formulation

3.1. Federated Continual Learning

Generally, the standard FCL process is composed of the following key steps: (1) The client side copes with the streaming tasks with a specially designed loss for typical continual learning [20, 36]; (2) At a timestep, the continually learned models in each client are uploaded to a central server, where different model aggregation methods are implemented to generate an improved global model. We iterate this pipeline many rounds until model convergence. Compared to traditional federated learning, FCL assumes that the overall system runs in a dynamic data environment, which is more challenging to achieve effective federation. Recent solutions [8, 43] attempt to address the client interference and catastrophic forgetting problem that are widely appeared in FCL, without considering the data repeatability in a task sequence.

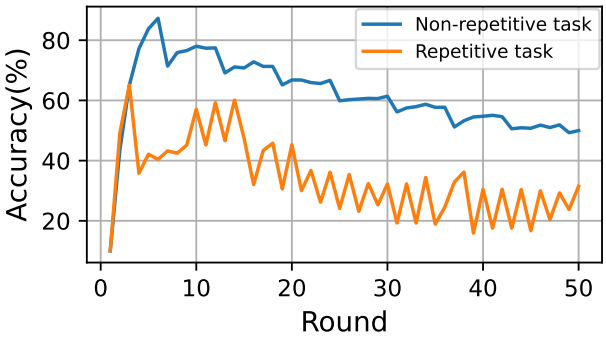

Figure 2. Performance of the traditional FCL method GLFC on the non-repetitive and repetitive task settings. Here the accuracy refers to the whole task performance as the learning continues.

To observe the influence of the repetitive task environment, we conduct a preliminary experiment, where we employ a state-of-the-art FCL method GLFC [8] to evaluate the performance on the non-repetitive and repetitive task settings respectively. Here CIFAR100 [17] is used and partitioned to simulate 10 clients and we run GLFC for 50 federated rounds. As shown in Figure 2, we can clearly see that after introducing the repetitive tasks, the performance degradation is obvious, which means that the typical FCL methods fail to achieve effective local task learning and global knowledge aggregation.

3.2. Traceable Federated Continual Learning

Motivated by the poor performance of current FCL solutions in the repetitive task environment, we attempt to introduce a new paradigm, *Traceable Federated Continual Learning*, to achieve effective FCL under the repetitive data stream. Specifically, our data setting differs previous ones from two aspects: (1) For a certain time, the incoming task has ever appeared in the previous task sequence; (2) Besides the task repeatability, the incoming task in different clients can also be various.

Formally, assuming there are N local clients, each of which contains a task sequence $S_t^i = \{s_1^i, s_2^i, ..., s_n^i\}$ at the time t. Here i denotes the i_{th} client and n is the number of the task sequence. If $S_{t-k}^i = S_t^i$, we believe a repetitive task appears at the time t and should be traced back to the time $t-k$ for reprocessing. Based on the symbols, we define the goal of *Traceable Federated Continual Learning* as follows.

Definition 3.1. (Traceable Federated Continual Learning (TFCL)). *Suppose the current model M_{cur} has learned*

$P = \{p_1, p_2, ..., p_q\}$ *tasks, which are reflected by corresponding features* $F = \{f_1, f_2, ..., f_q\}$. *Here* q *is the number of previous tasks. When the incoming task* $p_j \in P$, *the goal of TFCL is to accurately trace and connect the corresponding data features* f_j *to* p_j *for further processing, and then selectively federate the various clients for an improved model* M_{new}.

In the following sections, we will describe a series of designed techniques in detail for accomplishing our objective.

4. Framework Design

4.1. Overview

We design and develop TagFed, a framework to achieve TFCL via tracing, strengthening and selectively federating previous certain features, such that the learning process will not be interfered by the repetitive task. Figure 3 depicts the key modules of TagFed, which can be briefly summarized as follows.

- *Traceable Task Learning (TTL)*: This module takes advantage of the pruning technique to obtain incremental sub-models for incoming tasks. When a repetitive task comes, instead of processing it as a new task, the designed module traces the corresponding features of previously marked sub-models and further retrains the parameters for learning more sufficient features (the darker neurons in the figure). The learned model in each client is then masked to leave the current data features for later federation.
- *Group-wise Knowledge Aggregation (GKA)*: This module constructs a client-server knowledge transfer pipeline as federation. The key message between the two ends is a series of data features and logits, which comes from different clients. Considering the variety of the uploaded data features, we develop multiple groups in which tasks with similar or identical features are fed into a same server model as knowledge aggregation. The output of the aggregated server model is finally transferred back to corresponding clients to provide more valuable knowledge.

The two modules will be operated many times until we achieve desirable performance. In the remainder of the section, we describe in detail our approach for implementing each module.

4.2. Traceable Task Learning

The basic idea for our *Traceable Task Learning (TTL)* is employing a sub-model to train each task data and freezing the used weights during new task training. When a new task comes, we compare the label distribution with previous ones that have been reserved to figure out whether it is a repetitive task.

Principle of sub-model generation. Assume that all clients begin with a shared model M_{ini} and the i_{th} client is learning for task p_i. The initial model is first trained via its local dataset and then each client performs gradual pruning [21, 23, 47] on the model, which removes a portion of the weights to obtain a sub-model and fine-tunes the sub-model iteratively to maintain accuracy. Here the size of each sub-model is based on the task complexity and can be set by each client. After obtaining the sub-model, we keep its weights unalterable to avoid forgetting and enable learning until a repetitive task p_i comes to the i_{th} client in the future. According to the principle of the sub-model generation, in the following parts, we respectively illustrate how to cope with non-repetitive and repetitive new tasks.

Incremental training for non-repetitive new tasks. Suppose current model M_{cur} has learned $P = \{p_1, p_2, ..., p_{q-1}\}$ tasks, task p_q is coming and $p_q \neq p_i, i = 1, 2, ..., q-1$. The model weights preserved for task p_1 to task p_{q-1} are denoted as $W^P_{1:q-1}$ (i.e., a series of sub-models). The pruned weights associated with task p_{q-1} are denoted as W^E_{q-1} that can be used to learn new tasks. We apply a learnable mask $B_q \in \{0, 1\}^D$ to pick the old weights $W^P_{1:q-1}$, where D is the dimension of W^E_{q-1}. The picked weights are represented as $B_q \odot W^P_{1:q-1}$, where $\odot$ is the element-wise product between B and $W^P_{1:q-1}$. We then learn a real-valued mask $\hat{B}_q$ and applies a threshold for binarization to generate B_q. Specifically, when training the binary mask B_q, we update the real-valued mask $\hat{B}_q$ in the backward pass and quantize it with a threshold on $\hat{B}_q$. Hence, the model can pick a part of $W^P_{1:q-1}$ to re-use for the new task via the mask. Besides, the weights W^E_{q-1} can be used for task p_q. The mask B and W^E_{q-1} are optimized together on the training data of task p_q with the loss function via back-propagation. Then we will gradually prune the model once again. When the ratio of weights for task p_q meets the task requirement, they are represented as W^P_q and frozen to get rid of the old task's forgetting. This incremental training process will be iterated many times until there are no non-repetitive new tasks.

Traceable augmentation for repetitive new tasks. Suppose current model M_{cur} has learned $P = \{p_1, p_2, ..., p_{q-1}\}$ tasks, task p_q is coming and $p_q = p_i = ... = p_j$, $q>i>...>j$. The model weights preserved for task p_i and p_j are denoted as W^P_i and W^P_j. The real value mask of task p_i is represented as $\hat{B}_i$. We achieve traceable augmentation by (1) saving a copy of W^P_j to prevent the influence from other tasks; (2) retraining the real value mask $\hat{B}_i$ and W^E_i together on the training data of task p_q with the loss function via back-propagation and finally generating $\hat{B}_q$ and W^P_q. Note that B_q depending on $\hat{B}_q$ picks a part of $W^P_{1:q-1}$ and until the next repetitive task p_q comes, we will apply W^P_q to process task p_q. Otherwise,

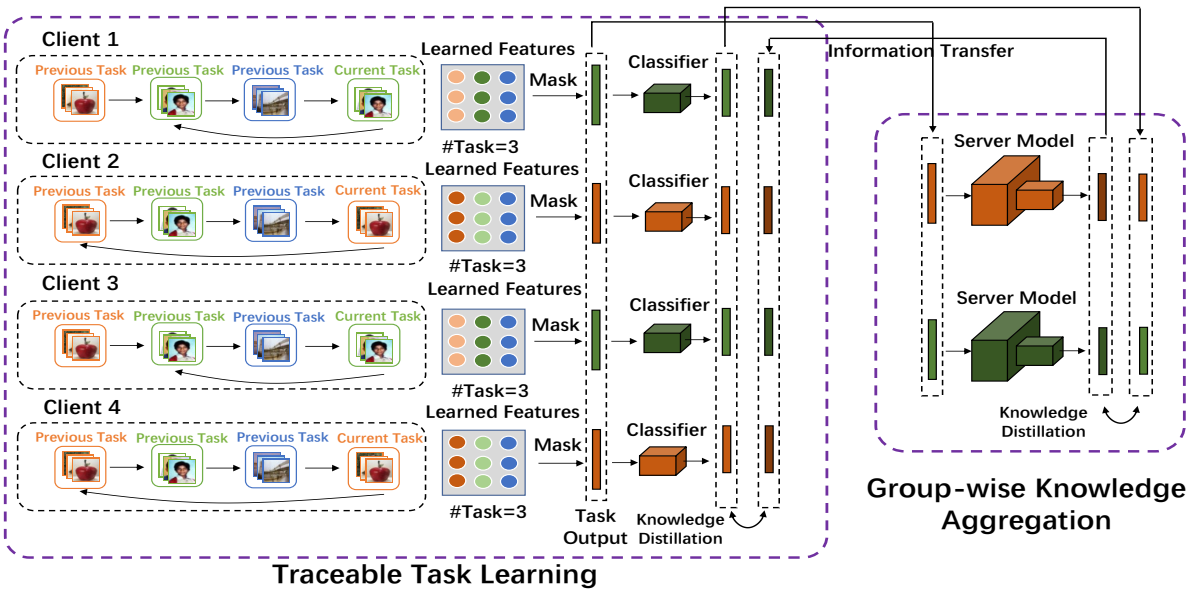

Figure 3. Overview of the proposed TagFed framework. It mainly includes two modules: *traceable task learning* and *group-wise knowledge aggregation*. Here different colors represent different features and the darker the color is, the more sufficient features the neuron holds.

we will apply W_j^P to keep the accuracy of other tasks.

Difference to other CL methods. Existing CL methods can only pinpoint the relevant weights and fine-tune them, which may affect the performance of non-repetitive tasks, as these tasks also rely on the frozen weights. In our approach, we devise a copying mechanism to store the weights frozen previously, along with marking their repetition frequency. This enables selective utilization for both repetitive and non-repetitive tasks.

4.3. Group-wise Knowledge Aggregation

Group-wise Knowledge Aggregation (GKA) serves as the second step to achieve effective federation among clients. We cannot use the traditional aggregation approach in FL (e.g., FedAvg [33]) because they attempt to change the weights of the whole model, which is infeasible for our client model that is divided into several sub-models, whose weights are frozen to avoid forgetting previous tasks when learning new tasks. Therefore, we construct a client-server bridge for information transfer and introduce group-wise knowledge distillation to accomplish aggregation.

Information transfer. In our framework, clients require extracting the feature maps after a specific hidden layer and related logits in their local models, which will be sent to the server during training. Note that the feature maps may leak users' confidential information, we can add some designed noise to further ensure privacy (details in the appendix). The server side will then use different models to aggregate knowledge from uploaded information according to current features of the client task and send the corresponding predicted value back to clients as the aggregated knowledge. Based on the knowledge, clients will further retrain and distill their models for improved performance.

Server-side knowledge distillation. In our TFCL, the feature distribution of repetitive tasks among clients can exhibit significant heterogeneity within a given timeframe. This circumstance has driven us to create distinct groups to facilitate selective knowledge distillation rather than blindly distilling them together. Given a series of feature maps and logits, we first put the feature maps into different groups based on the logits and construct a server model for each group. The number of server models depends on the number of learning tasks. In this way, the knowledge of identical tasks can be clustered for effective aggregation, getting rid of the interference from other irrelevant tasks. Concretely, for each group, the server loss function is defined as:

$$\begin{aligned} \ell_s &= \alpha_s \ell_{CE} + \beta_s \ell_{KD}\left(\boldsymbol{z}_s, \boldsymbol{z}_c^{(k)}\right) \\ &= \alpha_s \ell_{CE} + \beta_s D_{KL}\left(\boldsymbol{p}_k \| \boldsymbol{p}_s\right) \end{aligned} \tag{1}$$

where ℓ_{CE} is the cross-entropy loss between the predicted values and the ground truth labels. D_{KL} is the Kullback Leibler (KL) Divergence function. $\boldsymbol{z}_s$ and $\boldsymbol{z}_c^{(k)}$ are the predicted values of server models and client models. $\boldsymbol{p}_k^i = \frac{\exp\left(z_c^{(k,i)}/T\right)}{\sum_{i=1}^{C}\exp\left(z_c^{(k,i)}/T\right)}$ and $\boldsymbol{p}_s^i = \frac{\exp\left(z_s^i/T\right)}{\sum_{i=1}^{C}\exp(z_s^i/T)}$, where C is the number of clients and T is a temperature parameter. They are the probabilistic prediction of the k_{th} client model and the server model. α_s and β_s are the hyperparameters to control the ratio of the cross-entropy loss and the Kullback

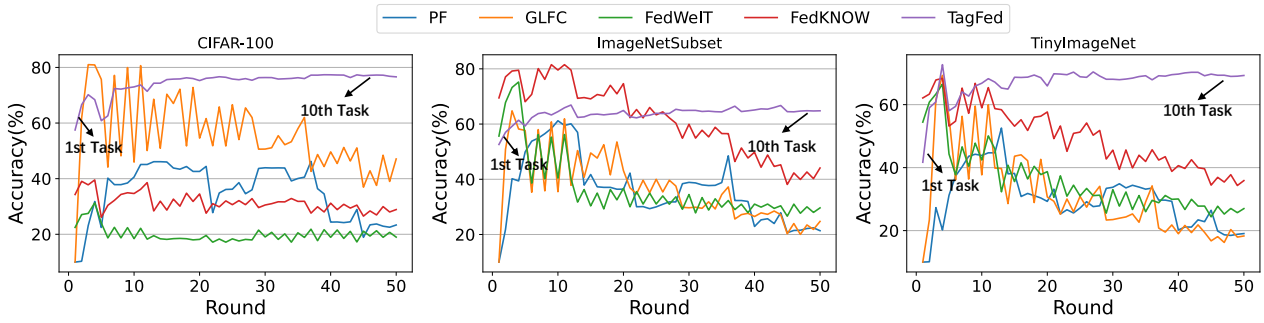

Figure 4. Convergence comparison on different datasets. Here the final accuracy of different methods is represented at the location of "10th Task" since we run 10 tasks in this FCL process. *PF* denotes *PackNet+FedAvg*.

Leibler Divergence. Note that in each federation round, the server will receive the message sent from clients and sequentially distill the corresponding server model based on Eq. 1 for each group.

Client-side knowledge distillation. Different from the distillation on the server side, the client-side knowledge distillation does not need to maintain several models for training. Instead, they connect to the corresponding server model based on the logits and utilize its aggregated knowledge to improve the local performance. For each client, the optimization loss can be defined as:

$$\begin{aligned} \ell_c^{(k)} &= \alpha_c \ell_{CE} + \beta_c \ell_{KD}\left(\boldsymbol{z}_s, \boldsymbol{z}_c^{(k)}\right) \\ &= \alpha_c \ell_{CE} + \beta_c D_{KL}\left(\boldsymbol{p}_s \| \boldsymbol{p}_k\right) \end{aligned} \tag{2}$$

where most of the symbols have been formally defined in Eq. 1 except α_c and β_c, which are another two hyperparameters to control the loss in the client end. As a result, each client can benefit from the valuable knowledge aggregated by the corresponding server model.

Besides, we would like to highlight that only the parameters of tasks that initially appear are saved (roughly 10%, 10.53M), which is an acceptable consumption for clients. Regarding the server-side distillation, it is commonly believed that the server resources are abundant, and the associated storage cost at the server level is not a bottleneck.

5. Experiments

5.1. Experimental Setup

Benchmark. Since there is no available benchmark to evaluate the performance of our framework, we manually construct one with three datasets that are widely used in the field of federated continual learning: CIFAR-100 [17], ImageNetSubset [7] and TinyImageNet [18]. As stated in Section 2.1, our settings mainly follow class-incremental learning (CIL), where each class is regarded as a task.

To make a fair comparison, we first follow the typical client settings used in FCL [8] as a base. Next, for the task repeatability settings, we divide federating rounds into ordinary rounds and backtracking rounds. Client models will learn the current task in an ordinary round and learn repetitive tasks in a backtracking round. For each backtracking round, the incoming tasks of every client might be different. The backtracking degree is represented by two hyperparameters, *backtracking region* and *backtracking time*. Backtracking region is an integer *r*, which means that the last *r* tasks will appear in this backtracking process. Backtracking time is an integer *t*, which means there are *t* backtracking rounds after an ordinary round. We apply the settings to all three datasets as the repetitive task simulation.

Baselines. To the best of our knowledge, there are few works targeting federated continual learning and most of the baselines just combine continual learning and federated learning directly. Here we select one representative combination: PackNet [31] + FedAvg [33]. Besides, we choose FedWeIT [43], GLFC [8] and FedKNOW [30] as baselines, which are state-of-the-art methods specially designed for federated continual learning. In the appendix, we provide a detailed description of these baselines.

Implementation details. We now describe the standard implementation of TagFed, which is used throughout our experiments unless otherwise specified. The concrete parameter settings are as follows: We employ ResNet-18 [12] as the backbone and extract feature maps after the second BasicBlock. The hyper-parameter α_s and α_c are set to 0.9 in Eq. 1 Eq. 2. β_s and β_c are set to 0.1. The distillation temperature is set to 5. We use SGD as the optimizer for training model parameters, and the learning rate is set to 0.2 with a momentum of 0.9. We use Adam as the optimizer for training masks, and the learning rate is set to 1e-4. The sparsity of each layer is set to 0.1 to hold all tasks into a model, with the purpose of making a fair comparison. All of the experiments are conducted for roughly 80 federating rounds to guarantee convergence. Finally, we run each experiment 3 times and average them as the reported results. All our experiments are simulated and conducted in a server that has 4 GeForce GTX 3090 GPUs, 48 Intel

Table 1. Effect of the proposed modules of different task numbers on CIFAR-100. *Ours-w/oModule* denotes the performance of our framework without using the Module. Here "Task Scale" refers to the number of tasks.

Task Scale	2	4	6	8	10	Avg
Ours-w/oTTL	69.20%	59.05%	55.62%	55.65%	56.05%	59.11%
Ours-w/oGKA	71.00%	73.43%	75.25%	76.21%	75.76%	74.33%
Ours	**71.40%**	**75.39%**	**76.36%**	**77.18%**	**78.35%**	**75.73%**

Xeon CPUs, and 128GB memory. We implement TagFed in Python with PyTorch, and all the experiments are conducted on a ResNet-18 architecture [12], which is pre-trained with the ImageNet dataset [6].

5.2. Performance Comparison

Accuracy comparison on our benchmark. As shown in Figure 4, we can clearly see that the proposed framework outperforms other methods by a large margin in the final accuracy. This demonstrates that by traceable task learning and group-wise knowledge aggregation, we can greatly boost the accuracy performance for the task repeatability scenario. Besides, it is worth noting that: (1) *GLFC* performs well at the beginning while degrading dramatically as new tasks come, which indicates that the repetitive tasks disturb its original learning state. This phenomenon also appears in *FedWeIT* and *FedKNOW* except the CIFAR-100 dataset. The reason may be that the two methods can easily become overfitting when the data scale is small such as CIFAR-100; (2) It is obvious that *PackNet+FedAvg* obtains poor performance in most of the learning process. We believe this is due to the incompatibility between the weight-level aggregation and the dynamic data settings; (3) In addition to the final accuracy, the convergence speed of TagFed is faster than others, which means that our framework can quickly adapt to a new repetitive or non-repetitive task.

Ablation study. TagFed mainly includes two key components: traceable task learning (TTL) and group-wise knowledge aggregation (GKA). Here we conduct ablation studies to explore whether each of them is beneficial to the final performance. Table 1 summarizes the results, where we record the corresponding accuracy performance of different task scales on CIFAR-100. From the table, we can find that both the two modules are essential to cope with the TFCL scenario.

Task-specific performance. In addition to the overall performance on all tasks, we are also interested in the detailed performance on each task. Figure 5 records the task-specific performance on the TinyImageNet dataset, where we set 10 tasks and each of which includes 20 classes. From the figure, we can observe that the accuracy on different tasks is various for all the methods. Specifically, *PackNet+FedAvg* has low performance on the first several tasks

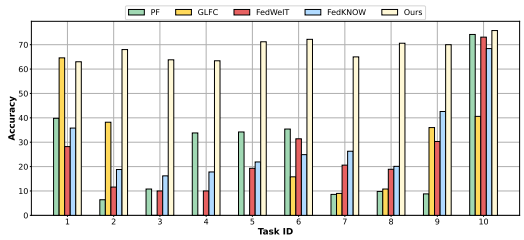

Figure 5. The task-specific performance on the TinyImageNet dataset.

Table 2. Results on the noise-introduced situation with CIFAR-100. Here *PF* denotes *PackNet+FedAvg*.

Task scale	2	3	5
PF	45.30%	42.96%	31.66%
GLFC	50.10%	54.57%	62.02%
FedWeIT	18.24%	18.17%	18.86%
FedKNOW	36.17%	34.50%	31.87%
Ours(1/dFIL = 5)	70.30%	73.06%	73.38%
Ours(1/dFIL = 1)	70.50%	73.86%	73.60%
Ours(1/dFIL = 0.5)	71.20%	72.93%	74.10%
Ours(w/o noise)	71.40%	75.27%	75.99%

while achieving excellent performance on the final task. We believe this is because *PackNet+FedAvg* pays more attention to the current task and federates them without considering the task forgetting problem, which will obtain poor accuracy in the previous tasks. Other FCL baselines, however, despite addressing catastrophic forgetting, fail to deal with the repetitive tasks and might introduce a large new task learning loss as the repetitive task increases. As a result, they cannot reach good performance and even for some tasks, the accuracy becomes zero. The results on other datasets can be found in the appendix.

Noise-introduced performance. Because our pipeline requires exchanging the feature maps between the clients and the server, we introduced a designed noise to further ensure privacy. Here we adopted *diagonal Fisher information leakage* (dFIL) as the privacy metric. The detailed noise setting can be found in the appendix. As shown in Table 2, although the performance of ours decreases slightly compared to the version without noise, it still significantly sur-

Table 3. The number of parameters needed for communication and computation on different methods.

Method	Uploading Cost	Offloading Cost	Computational Cost
PackNet+FedAvg	11.70MB	11.70MB	6.33 MB * 40 rounds
GLFC	11.80MB	11.70MB	11.50 MB * 45 rounds
FedWeIT	9.70MB	26.20MB	14.70 MB * 21 rounds
FedKNOW	11.70MB	11.70MB	11.50 MB * 32 rounds
Ours	10.30MB	0.05MB	11.50 MB * 15 rounds

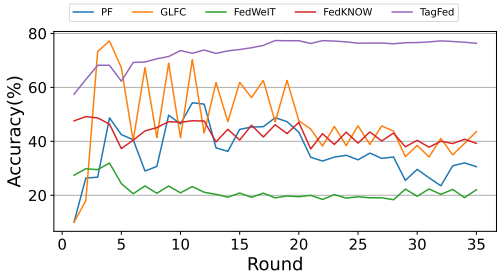

Figure 6. Performance on large-scale clients.

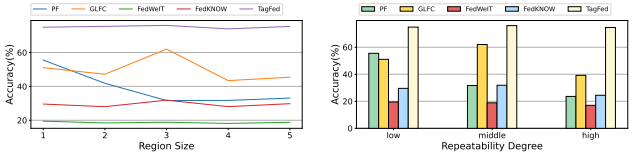

Figure 7. (a) Performance on the non-iid condition. (b) Effect of the Repeatability Degree.

passes other baselines, which validates the effectiveness of TagFed under the privacy-preserving scenario.

Performance on the non-iid condition. The non-iid degree can be represented by the backtracking region, which decides the number of previous tasks to trace. The larger region we backtrack, the higher heterogeneity we are in. Here we set the region size from 1 to 5, with CIFAR-100 running for 5 tasks to observe the performance. As shown in Figure 7 (a), TagFed can achieve consistently better performance regardless of the heterogeneity degree.

Performance on large-scale clients. In above mentioned experiments, we only operate the FL system with 10 clients. To explore whether the proposed approach can be extended to a large-scale scenario, we add the number of clients to 30 and implement each method respectively. The experiments are conducted on CIFAR-100. As illustrated in Figure 6, TagFed surpasses other baselines by a large margin as we increase the number of tasks, which is consistent with our previous analysis.

5.3. Efficiency of TagFed

In this subsection, we study the efficiency of TagFed, where we record the uploading, offloading, and training parameters as the cost measurement. Table 3 demonstrates the results. For the uploading cost, we can see that the difference among these methods is not obvious, which means that the cost of uploaded feature maps is comparable to the whole model. For the offloading cost, TagFed can show a great advantage over other methods since it only needs the computed prediction vector, whose dimension is extremely small. Besides, we can see that TagFed does not result in increased computation cost as most training parameters have been frozen in the later FL rounds. Moreover, due to its fast convergence speed, we can complete FL with fewer rounds, further reducing the overall computation cost. In general, our framework can save roughly 50% communication cost compared to others while saving a little computational cost.

5.4. Effect of the Repeatability Degree

This subsection explores the effect of the repeatability degree on the final performance. Here we partition the repeatability degree into three levels: *low repeatability*, *middle repeatability*, and *high repeatability*, in terms of tracing rounds and regions (details in the appendix). We evaluate different methods on CIFAR-100 with five tasks. As shown in Figure 7 (b), unlike other baselines that are significantly influenced by the degree of repeatability, TagFed is not affected and achieves improved performance. This further validates the effectiveness of TagFed.

6. Conclusion

In this paper, we propose Traceable Federated Continual Learning (TFCL), where repetitive tasks can be accurately and effectively traced and processed to boost the performance. We design and implement TagFed, a framework to achieve TFCL through feature tracing, sub-model augmentation, and group-wise knowledge federation. Experiments on our simulated benchmark demonstrate the effectiveness and efficiency of TagFed.

Acknowledgments

This work was partly supported by the National Natural Science Foundation of China (62302054), the Fundamental Research Funds for the Central Universities, and the Research Innovation Fund for College Students of Beijing University of Posts and Telecommunications.

References

[1] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643. PMLR, 2019. 3

[2] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 233–248, 2018. 2

[3] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018. 2

[4] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *ICLR*, 2019.

[5] Guy Davidson and Michael C Mozer. Sequential mastery of multiple visual tasks: Networks naturally learn to learn and forget to forget. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9282–9293, 2020. 2

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 7

[7] Jinhong Deng, Wen Li, Yuhua Chen, and Lixin Duan. Unbiased mean teacher for cross-domain object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4091–4101, 2021. 6

[8] Jiahua Dong, Lixu Wang, Zhen Fang, Gan Sun, Shichao Xu, Xiao Wang, and Qi Zhu. Federated class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10164–10173, 2022. 1, 3, 6

[9] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *European Conference on Computer Vision*, pages 86–102. Springer, 2020. 3

[10] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015. 2

[11] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018. 3

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6, 7

[13] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 831–839, 2019. 2

[14] Xinting Hu, Kaihua Tang, Chunyan Miao, Xian-Sheng Hua, and Hanwang Zhang. Distilling causal effect of data in class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3957–3966, 2021. 2

[15] Shaoxiong Ji, Wenqi Jiang, Anwar Walid, and Xue Li. Dynamic sampling and selective masking for communication-efficient federated learning. *arXiv preprint arXiv:2003.09603*, 2020. 3

[16] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016. 3

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 3, 6

[18] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015. 6

[19] Suyi Li, Yong Cheng, Yang Liu, Wei Wang, and Tianjian Chen. Abnormal client behavior detection in federated learning. *arXiv preprint arXiv:1910.09933*, 2019. 3

[20] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017. 3

[21] Bingyan Liu, Yao Guo, and Xiangqun Chen. Wealthadapt: A general network adaptation framework for small data tasks. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2179–2187, 2019. 4

[22] Bingyan Liu, Yuanchun Li, Yunxin Liu, Yao Guo, and Xiangqun Chen. Pmc: A privacy-preserving deep learning model customization framework for edge computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(4):1–25, 2020. 2

[23] Bingyan Liu, Yifeng Cai, Yao Guo, and Xiangqun Chen. Transtailor: Pruning the pre-trained model for improved transfer learning. In *Proceedings of the AAAI conference on artificial intelligence*, pages 8627–8634, 2021. 4

[24] Bingyan Liu, Yifeng Cai, Ziqi Zhang, Yuanchun Li, Leye Wang, Ding Li, Yao Guo, and Xiangqun Chen. Distfl: Distribution-aware federated learning for mobile scenarios. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(4):1–26, 2021. 3

[25] Bingyan Liu, Yao Guo, and Xiangqun Chen. Pfa: Privacy-preserving federated adaptation for effective model personalization. In *Proceedings of the Web Conference 2021*, pages 923–934, 2021. 3

[26] Bingyan Liu, Yifeng Cai, Hongzhe Bi, Ziqi Zhang, Ding Li, Yao Guo, and Xiangqun Chen. Beyond fine-tuning: Efficient and effective fed-tuning for mobile/web users. In *Proceedings of the ACM Web Conference 2023*, pages 2863–2873, 2023. 1

[27] Bingyan Liu, Nuoyan Lv, Yuanchun Guo, and Yawen Li. Recent advances on federated learning: A systematic survey. *arXiv preprint arXiv:2301.01299*, 2023. 1

[28] Quande Liu, Cheng Chen, Jing Qin, Qi Dou, and Pheng-Ann Heng. Feddg: Federated domain generalization on medical

image segmentation via episodic learning in continuous frequency space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1013–1023, 2021. 1

[29] Yang Liu, Anbu Huang, Yun Luo, He Huang, Youzhi Liu, Yuanyuan Chen, Lican Feng, Tianjian Chen, Han Yu, and Qiang Yang. Fedvision: An online visual object detection platform powered by federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 13172–13179, 2020. 1

[30] Yaxin Luopan, Rui Han, Qinglong Zhang, Chi Harold Liu, and Guoren Wang. Fedknow: Federated continual learning with signature task knowledge integration at edge. *ICDE*, 2023. 3, 6

[31] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018. 2, 6

[32] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619*, 2020. 3

[33] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017. 1, 2, 3, 5, 6

[34] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019. 3

[35] Seungeun Oh, Jihong Park, Praneeth Vepakomma, Sihun Baek, Ramesh Raskar, Mehdi Bennis, and Seong-Lyun Kim. Locfedmix-sl: Localize, federate, and mix for improved scalability, convergence, and latency in split learning. In *Proceedings of the ACM Web Conference 2022*, pages 3347–3357, 2022. 3

[36] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *Proceedings of the IEEE international conference on computer vision*, pages 3400–3409, 2017. 3

[37] Khe Chai Sim, Petr Zadrazil, and Françoise Beaufays. An investigation into on-device personalization of end-to-end automatic speech recognition models. *arXiv preprint arXiv:1909.06678*, 2019. 3

[38] Chengxu Yang, Qipeng Wang, Mengwei Xu, Zhenpeng Chen, Kaigui Bian, Yunxin Liu, and Xuanzhe Liu. Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data. In *Proceedings of the Web Conference 2021*, pages 935–946, 2021. 3

[39] Liu Yang, Ben Tan, Vincent W Zheng, Kai Chen, and Qiang Yang. Federated recommendation systems. In *Federated Learning*, pages 225–239. Springer, 2020. 1

[40] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019. 3

[41] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018. 3

[42] Xue Yang, Yan Feng, Weijun Fang, Jun Shao, Xiaohu Tang, Shu-Tao Xia, and Rongxing Lu. An accuracy-lossless perturbation method for defending privacy attacks in federated learning. In *Proceedings of the ACM Web Conference 2022*, pages 732–742, 2022. 3

[43] Jaehong Yoon, Wonyong Jeong, Giwoong Lee, Eunho Yang, and Sung Ju Hwang. Federated continual learning with weighted inter-client transfer. In *International Conference on Machine Learning*, pages 12073–12086. PMLR, 2021. 1, 3, 6

[44] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. Salvaging federated learning by local adaptation. *arXiv preprint arXiv:2002.04758*, 2020. 3

[45] Ziqi Zhang, Yuanchun Li, Bingyan Liu, Yifeng Cai, Ding Li, Yao Guo, and Xiangqun Chen. Fedslice: Protecting federated learning models from malicious participants with model slicing. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 460–472. IEEE, 2023. 3

[46] Chen Zhu, Zheng Xu, Mingqing Chen, Jakub Konečnỳ, Andrew Hard, and Tom Goldstein. Diurnal or nocturnal? federated learning of multi-branch networks from periodically shifting distributions. In *International Conference on Learning Representations*, 2021. 3

[47] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017. 4