FEDGP: BUFFER-BASED GRADIENT PROJECTION FOR CONTINUAL FEDERATED LEARNING

Shenghong Dai[1], **Yicong Chen**[1], **Jy-yong Sohn**[2], **S M Iftekharul Alam**[3], **Ravikumar Balakrishnan**[3],

Suman Banerjee[4], **Nageen Himayat**[3], **Kangwook Lee**[1]

[1]Department of Electrical and Computer Engineering,
University of Wisconsin-Madison, Madison, WI 53706, USA
{sdai37, ychen2229, kangwook.lee}@wisc.edu

[2]Department of Applied Statistics, Yonsei University, Seoul, Korea

[3]Intel Labs, Santa Clara, CA 95054, USA

[4]Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI 53706, USA

ABSTRACT

Continual Federated Learning (CFL) is essential for enabling real-world applications where multiple decentralized clients adaptively learn from continuous data streams. A significant challenge in CFL is mitigating catastrophic forgetting, where models lose previously acquired knowledge when learning new information. Existing works on this issue either make unrealistic assumptions about the availability of task boundaries or heavily rely on surrogate samples. To address this limitation, we introduce a buffer-based Gradient Projection method (FedGP). This method tackles catastrophic forgetting by leveraging local buffer samples and aggregated buffer gradients, thus preserving knowledge across multiple clients. Our method is compatible with various existing continual learning and CFL techniques, enhancing their performance in the CFL context. Our experiments on standard benchmarks show consistent performance improvements across diverse scenarios. For example, on a task-incremental learning setting with CIFAR100, our method can help increase the accuracy up to 27%. Our code is available at https://github.com/shenghongdai/FedGP/.

1 INTRODUCTION

Federated Learning (FL) is a machine learning technique that facilitates collaborative model training among a large number of users while keeping data decentralized for privacy and efficient communication. In real-world applications, models trained via FL need the flexibility to continuously adapt to new data streams without forgetting past knowledge. This is critical in a variety of scenarios, such as autonomous vehicles, which must adapt to changes in the surroundings like new buildings or vehicle types without losing proficiency in previously encountered contexts. These real-world considerations make it essential to integrate FL with continual learning (CL) (Shmelkov et al., 2017; Chaudhry et al., 2018; Thrun, 1995; Aljundi et al., 2017; Chen & Liu, 2018; Aljundi et al., 2018), thereby giving rise to the concept of Continual Federated Learning (CFL).

The biggest challenge in CFL, as in CL, is *catastrophic forgetting*, where the model gradually shifts its focus from old data to new data and unintentionally discards previously acquired knowledge. Initial attempts to mitigate catastrophic forgetting in CFL incorporated existing CL solutions at each client of FL, such as replaying previous task data or penalizing the updates of weights that are crucial for

A preliminary version of this work was presented at the Federated Learning Systems (FLSys) Workshop @ Sixth Conference on Machine Learning and Systems, June 2023.

preserving the knowledge from earlier tasks. However, recent works (Bakman et al., 2023; Ma et al., 2022; Yoon et al., 2021) have observed that this naïve approach cannot fully mitigate the problem due to two reasons: (i) small-scale devices participating in FL only have limited buffer size to store the data from previous tasks, (ii) data distributions are not identical across clients in FL. Moreover, existing methods developed for CFL suffer from several limitations. These include scalability issues as the number of tasks grows (Yoon et al., 2021; Venkatesha et al., 2022), the need for significant effort in generating or collecting surrogate data (Ma et al., 2022), and significant communication overhead (Yao & Sun, 2020). A crucial constraint shared by all these methods is that they require explicit task boundaries. Mitigating catastrophic forgetting in practical scenarios where fixed task boundaries are absent throughout the training process, known as *general continual learning* (Buzzega et al., 2020), remains an important open question.

To address these existing challenges of CFL, we introduce a method called buffer-based Gradient Projection, which we dub FedGP. Our approach, illustrated in Fig. 1, involves two key components:

1. **Global Buffer Gradients**: Each client k computes the local buffer gradient g_{ref}^k of the global model with respect to its local buffer data. All local buffer gradients are then securely averaged to obtain aggregated buffer gradient g_{ref}.
2. **Local Gradient Projection**: In the next round, each client k updates its local model such that the direction for the model update does not conflict with aggregated buffer gradient g_{ref} from the previous round, ensuring each client preserves past information from all clients.

Importantly, our FedGP method is designed to be fully compatible with (i) general continual learning settings, when task boundary is unknown, and (ii) secure aggregation techniques (Bonawitz et al., 2017). Secure aggregation ensures that while clients share gradients or model updates, the individual data remains private (Bakman et al., 2023).

Our contributions: We introduce a new method for CFL, called FedGP. This method utilizes information from previous tasks across clients to effectively mitigate catastrophic forgetting, without having access to task boundaries or surrogate samples. Furthermore, FedGP can seamlessly integrate with existing CFL techniques to enhance performance.

We conduct comprehensive experiments to demonstrate the effectiveness of FedGP across various standard image classification benchmarks and a text classification task on the sequential-YahooQA dataset (Zhang et al., 2015; Mehta et al., 2021). FedGP consistently improves accuracy and reduces forgetting on top of existing CFL baselines across diverse benchmark datasets. Further, we evaluate the robustness of our method considering various buffer sizes, communication frequency, asynchronous environments, and different numbers of tasks and users.

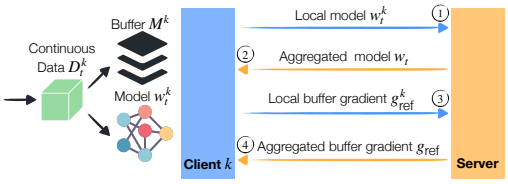


Figure 1: An overview of our proposed method, FedGP. In each round t, client k receives data D_t^k and trains a local model w_t^k. To address catastrophic forgetting, a portion of the incoming data is stored in a buffer $\mathcal{M}^k$. Given the aggregated model w_t provided by the central server, each client computes the gradient with respect to w_t using its buffer data $\mathcal{M}^k$. The server securely aggregates the local buffer gradients from all clients to obtain an aggregated buffer gradient g_{ref}, which will guide the local model update for each client k in the subsequent round.

2 Related Work

Prior work related to our paper falls into three categories: Continual Learning (CL), Federated Learning (FL), and Continual Federated Learning (CFL).

2.1 Continual Learning (CL)

CL addresses the problem of learning multiple tasks consecutively using a single model. Catastrophic forgetting (McCloskey & Cohen, 1989; Ratcliff, 1990; French, 1999), where a classifier trained for

a current task performs poorly on previous tasks, is a major challenge. Existing approaches can be categorized into regularization-based, architecture-based, and replay-based methods.

Regularization-based methods Some CL methods add a regularization term in the loss used for the model update; they penalize the updates on weights that are important for previous tasks. EWC (Kirkpatrick et al., 2017), SI (Zenke et al., 2017), Riemannian Walk (Chaudhry et al., 2018) are methods within this category. EWC uses Fisher information matrix to evaluate the importance of parameters for previous tasks. Besides, LwF (Li & Hoiem, 2017) leverages knowledge distillation to preserve outputs on previous tasks while learning the current task.

Architecture-based methods A class of CL methods assigns a subset of model parameters to each task, so that different tasks are learned by different parameters. This class of methods is also known as parameter isolation methods. Some methods including PNN (Rusu et al., 2016) and DEN (Yoon et al., 2017) uses dynamic architectures where the architecture changes dynamically as the number of tasks increases. These methods have issues where the number of required parameters grows linearly with the number of tasks. To tackle this issue, fixed network are used in the recent methods including PackNet (Mallya & Lazebnik, 2018), HAT (Serra et al., 2018) and PathNet (Fernando et al., 2017). SupSup (Wortsman et al., 2020) and DualNet (Pham et al., 2021) are the latest SOTA methods.

Replay-based methods To avoid catastrophic forgetting, a class of CL methods employs a replay buffer to save a small portion of the data seen in previous tasks and reuse it in the training of subsequent tasks. One of the early works in this area is ER (Ratcliff, 1990; Robins, 1995). In more contemporary studies, iCaRL (Rebuffi et al., 2017) stores exemplars of data from previous tasks and adds distillation loss for old exemplars to mitigate the forgetting issue. Deep Generative Replay (Shin et al., 2017) retains the memories of the previous tasks by loading the synthetic data generated by GANs without replaying the actual data for the previous tasks. GSS (Aljundi et al., 2019) optimally selects data for replay buffer by maximizing the diversity of samples in terms of the gradient in the parameter space. GEM (Lopez-Paz & Ranzato, 2017) and its variant A-GEM (Chaudhry et al., 2019) leverage an episodic memory that stores part of seen samples for each task to prevent forgetting old knowledge. Similarly, OGD (Farajtabar et al., 2020) stores gradients as opposed to actual data, providing a reference in projection. More recent work include GDumb (Prabhu et al., 2020), BiC (Wu et al., 2019), DER++, and Co^{2}L (Cha et al., 2021). Despite its simplicity, replay-based techniques have shown great performances on multiple benchmarks (Mai et al., 2022; Parisi et al., 2019). FedGP leverages a replay-based method that alleviates forgetting by reusing some data from previous tasks.

General continual learning Prior works on CL often rely on the information about the task boundaries. For example, some replay-based methods perform specific steps specifically at task boundaries, some regularization-based methods store network responses at these boundaries; architecture-based methods update the model architecture after one task is finished. However, when dealing with some data in practical settings, task boundaries are not clearly defined. This scenario, where sequential tasks are learned continuously without explicit knowledge of task boundaries, is referred to as *general continual learning* (Buzzega et al., 2020; Aljundi et al., 2019; Chaudhry et al., 2019). To address general continual learning, replay-based methods can utilize reservoir sampling (Vitter, 1985), which allows sampling throughout the training rather than relying on task boundaries. In our work, we specifically focus on general continual learning with reservoir sampling, particularly in the context of federated learning setups.

2.2 Federated Learning (FL)

FL enables collaborative training of a model with improved data privacy (Kairouz et al., 2021; Lim et al., 2020; Zhao et al., 2018; Konečnỳ et al., 2016). FedAvg (McMahan et al., 2017) is a widely used FL algorithm, but most existing methods (Li et al., 2020; Shoham et al., 2019; Karimireddy et al., 2020; Li et al., 2019; Mohri et al., 2019) assume static data distribution over time, ignoring temporal dynamics.

2.3 Continual Federated Learning (CFL)

CFL tackles the problem of learning multiple consecutive tasks in the FL setup. FedProx (Li et al., 2020) and FedCurv (Shoham et al., 2019) aim to preserve previously learned tasks, while FedWeIT (Yoon et al., 2021) and NetTailor (Venkatesha et al., 2022) prevent interference between

irrelevant tasks. Other methods including CFeD (Ma et al., 2022), FedCL (Yao & Sun, 2020), and GLFC (Dong et al., 2022) use surrogate datasets, importance weights, or class-aware techniques to distill the knowledge obtained from previous tasks. However, existing CFL methods suffer from several limitations, *e.g.*, not scalable as the number of tasks increases (Yoon et al., 2021; Venkatesha et al., 2022), requiring a surrogate dataset (Ma et al., 2022) or additional communication overhead (Yao & Sun, 2020), and not applicable to general continual setting that does not have fixed task boundaries. Our method successfully overcomes these obstacles, enhancing the efficacy of CFL.

3 PRELIMINARIES

We focus on finding a single classifier f (having model parameter w) that performs well on T tasks. We assume that at time slot $t \in [T]$, the classifier is only allowed to be trained for task t, where we define $[N] \coloneqq \{1, \cdots, N\}$ for a positive integer N. We assume the feature-label samples (x_t, y_t) for task t are drawn from an unknown distribution D_t. The optimization problem for CL at time $\tau \in [T]$ is written as

$$\min_{w} \sum_{t=1}^{\tau} \mathbb{E}_{(x_t,y_t)\sim D_t} \left[\ell\left(y_t, f\left(x_t; w\right)\right)\right], \tag{1}$$

where ℓ is the loss function, and $f(x_t; w)$ is the output of classifier f with parameter w, for input x_t. We consider a practical scenario where we do not have enough storage to save all the data seen for the previous task ($t < \tau$); instead, we employ a replay buffer $\mathcal{M}$ that selectively stores a subset of data. We use the buffer data as a proxy to summarize past samples and refine the model updates. We constrain the model updates in a way that the average loss for the data in buffer $\mathcal{M}$ does not increase. Given the model $w_{\tau-1}$ trained on previous tasks, the constrained optimization problem at time $\tau \in [T]$ is represented as:

$$\begin{aligned} &\min_{w} \mathbb{E}_{(x_\tau,y_\tau)\sim D_\tau} \left[\ell\left(y_\tau, f\left(x_\tau; w\right)\right)\right] \\ &\text{s.t.} \quad \mathbb{E}_{(x_b,y_b)\sim D_b} \left[\ell\left(y_b, f\left(x_b; w\right)\right)\right] \leq \mathbb{E}_{(x_b,y_b)\sim D_b} \left[\ell\left(y_b, f\left(x_b; w_{\tau-1}\right)\right)\right], \end{aligned} \tag{2}$$

where D_b is a uniform distribution over the samples in buffer $\mathcal{M}$, and (x_b, y_b) are sampled from this distribution D_b. The optimization problem in Eq. 2 can be reformulated for various CL methods as below. First, some methods including DER (Buzzega et al., 2020) use regularization techniques to find the model parameter w that minimizes the loss with respect to the local replay buffer $\mathcal{M}$ as well as current samples. For a given regularization coefficient γ, the optimization problem for CL with replay buffers at time $\tau \in [T]$ is:

$$\min_{w} \mathbb{E}_{(x_\tau,y_\tau)\sim D_\tau} \left[\ell\left(y_\tau, f(x_\tau; w)\right)\right] + \gamma \mathbb{E}_{(x_b,y_b)\sim D_b} \left[\ell\left(y_b, f(x_b; w)\right)\right]. \tag{3}$$

Second, some other methods, including A-GEM (Chaudhry et al., 2019), attempt to approximately implement the constraints of Eq. 2 by considering the gradients with respect to the current/buffer data. Specifically, the constraint promotes the alignment of the gradient with respect to the current batch of data (x_τ, y_τ) and that for the buffer data (x_b, y_b) sampled from the distribution D_b. This optimization problem at time $\tau \in [T]$ is formulated as:

$$\begin{aligned} &\min_{w} \mathbb{E}_{(x_\tau,y_\tau)\sim D_\tau} \left[\ell\left(y_\tau, f\left(x_\tau; w\right)\right)\right] \\ &\text{s.t.} \quad \mathbb{E}_{(x_\tau,y_\tau)\sim D_\tau,(x_b,y_b)\sim D_b} \left[\left\langle \nabla_w \left[\ell\left(y_\tau, f\left(x_\tau; w\right)\right)\right], \nabla_w \left[\ell\left(y_b, f\left(x_b; w\right)\right)\right]\right\rangle\right] \geq 0 \end{aligned} \tag{4}$$

For the continual *federated* learning (CFL) setup where the data is owned by K clients, we use the superscript $k \in [K]$ to denote each client, *i.e.*, client k samples the data from D_t^k at time t and employs a local replay buffer $\mathcal{M}^k$. In the case of using FedAvg (McMahan et al., 2017), each round of the CFL is operated as follows. First, each client $k \in [K]$ performs multiple iterations of local updates with D_t^k with the assistance of replay buffer $\mathcal{M}^k$. Second, once the local training is completed, each client sends the model updates to the central server. Finally, the central server aggregates the model updates and transmits them back to clients.

4 FEDGP

We introduce a method FedGP that is compatible with various CFL techniques, significantly enhancing their performance in the CFL context. Our approach draws inspiration from A-GEM (Chaudhry

et al., 2019), which projects the gradient with respect to its own historical data. Building upon this idea, we utilize the global buffer gradient, which is the average buffer gradient across all clients, as a reference to project the local gradient. This allows us to take advantage of the collective experience of multiple clients and mitigate the risk of forgetting previously learned knowledge in FL scenarios.

Algorithm 1 FedAvg ServerUpdate with FedGP

Initialize random w^k, and set $\mathcal{M}^k = \{\}$, $g_{\text{ref}} = \text{None}$
for each task $t = 1$ **to** T **do**
  for each communication $r = 1$ **to** R **do**
    $w^k \leftarrow \texttt{ClientUpdate}(t, w^k, g_{\text{ref}}),\ \forall k$
    $w \leftarrow \text{SecAgg}\left(w^k\right)$
    $g^k_{\text{ref}} \leftarrow \texttt{ComputeBufferGrad}(w, \mathcal{M}^k),\ \forall k$
    $g_{\text{ref}} \leftarrow \text{SecAgg}\left(g^k_{\text{ref}}\right)$
  end for
end for
Return w, the final global model

Algorithm 2 $\texttt{ClientUpdate}(t, w, g_{\text{ref}})$ at client k

Input: Task index t, model w, buffer gradient g_{ref}
Load the dataset $\mathcal{D}^k_t$, local buffer $\mathcal{M}^k$
Initialize $n = 0$ at the first task
for $(x, y) \in \mathcal{D}^k_t$ **do**
  $g = \nabla_w \left[\ell(y, f(x; w))\right]$
  $\tilde{g} \leftarrow g - \text{proj}_{g_{\text{ref}}} g \cdot \mathbf{1}(g^\top_{\text{ref}} g \leq 0)$
  $w \leftarrow w - \alpha\tilde{g}$ for some learning rate α
  $\mathcal{M}^k \leftarrow \texttt{ReservoirSampling}(\mathcal{M}^k, (x, y), n)$
  $n \leftarrow n + 1$
end for
Return w to server

Algorithm 3 $\texttt{ComputeBufferGrad}(w, \mathcal{M}^k)$

Input: global model w, local buffer $\mathcal{M}^k$
$(x_1, y_1) \ldots (x_m, y_m) \leftarrow$ random samples from $\mathcal{M}^k$
$g = \frac{1}{m} \sum_{i=1}^{m} \nabla_w \left[\ell(y_i, f(x_i; w))\right]$
Return g to server

As a replay-based method, FedGP maintains a local buffer on each client, which is a memory buffer randomly storing a subset of data sampled from old tasks. The local buffer at client k is denoted by $\mathcal{M}^k$. As the continuous data is loaded to the client, it keeps updating the buffer so that $\mathcal{M}^k$ becomes a good representative of old tasks.

Algorithm 1 provides the overview of our method in CFL setup, including the process of sharing information (model and buffer gradient) between the server and each client. For each new task $t \in [T]$, the server first aggregates the local models w^k from client $k \in [K]$, getting a global model w. Afterwards, the server aggregates the local buffer gradient g^k_{ref} (the gradient computed on the global model w with respect to the local buffer $\mathcal{M}^k$) from client $k \in [K]$ to obtain a global buffer gradient g_{ref}. It is worth noting that the term "aggregation" in this context refers to the averaging of locally computed values across all clients. Such aggregation can be securely performed by the central server using secure aggregation (Bonawitz et al., 2017), which is denoted as "SecAgg" in Algorithm 1. Note that here we have two functions used at the client side, `ClientUpdate` and `ComputeBufferGrad`, which are given in Algorithm 2 and 3, respectively. `ClientUpdate` shows how client k updates its local model for task t. The client first loads the global model w and the global buffer gradient g_{ref} which are received from the server in the previous round. It also loads the local buffer $\mathcal{M}^k$ storing a subset of samples for previous tasks, and the data $\mathcal{D}^k_t$ for the current task. For new batch of data $(x, y) \in \mathcal{D}^k_t$, the client computes the gradient $g = \nabla_w \ell(y, f(x; w))$ for the model w. The client then compares the direction of g with the direction of the global buffer gradient g_{ref} received from the server. When the angle between g and g_{ref} is greater than 90°, it implies that while using the direction of g as a reference for gradient descent may improve performance on the current task, but at the cost of degrading performance on previous tasks. To retain the knowledge on the previous tasks, we do the following: whenever g and g_{ref} are having a negative inner product, we project the gradient g onto the global buffer gradient (which can be considered as a reference) g_{ref} and remove this component from g, *i.e.*, define

$$\tilde{g} = g - \frac{g^T g_{\text{ref}}}{g^T_{\text{ref}} g_{\text{ref}}} g_{\text{ref}} \cdot \mathbf{1}(g^\top_{\text{ref}} g \leq 0), \tag{5}$$

following the idea suggested in (Chaudhry et al., 2019). As illustrated in Fig. 2, this projection helps prevent the model updates along the direction that is harming the performance on previous tasks.

After gradient projection, the client updates its local model w by applying the gradient descent step with the updated gradient $\tilde{g}$. Finally, the client updates the contents of the buffer $\mathcal{M}^k$ by using the reservoir sampling (Vitter, 1985) written in Algorithm 4 in the Appendix. Reservoir sampling selects a random sample of $|\mathcal{M}^k|$ elements from a local input stream, while ensuring that each element has an equal probability of being included in the sample. One of the advantages of this method is that it does not require any prior knowledge of the size of the data stream. Once the updated local models $\{w^k\}_{k=1}^K$ are transmitted to the server, the global model w is securely updated on the server side, and transmitted to each client. Then, each client k computes the local buffer gradient (*i.e.*, the gradient of the model w with respect to the samples in the local buffer $\mathcal{M}^k$) as shown in Algorithm 3 `ComputeBufferGrad`.

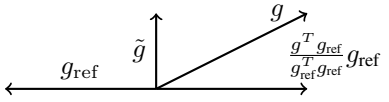


Figure 2: Illustration of the gradient projection in Eq. 5. If the angle between the gradient update g and global buffer gradient (considered as a reference) g_{ref} is larger than 90°, we project g on g_{ref} to minimize the interference and merely update along the directions of $\tilde{g}$ that is orthogonal to g_{ref}.

After each client computes the local buffer gradient g_{ref}^k, the server allows the use of secure aggregation to combine these local buffer gradients and update the global buffer gradient g_{ref}. Secure aggregation is a well-established technique in FL that ensures the server learns nothing about individual clients' data beyond their aggregated sum. This compatibility with secure aggregation enhances the privacy safeguards of our proposed approach, effectively minimizing the risk of data leakage from individual clients. The aforementioned process takes place between each communication and is repeated R times within each task. After traversing T tasks, the final global model w is obtained, as shown in Algorithm 1.

Note that the pseudocode describes the FL+FedGP process. FedGP is designed to be compatible with various CFL techniques. Details of the combination are elaborated upon in the Appendix D.

5 Experiments

In this section, we assess the efficacy of our method, FedGP, in combination with various CFL baselines, under non-IID data distribution across clients. To evaluate these methods, we conduct experiments on image classification tasks for benchmark datasets including rotated-MNIST (Lopez-Paz & Ranzato, 2017), permuted-MNIST (Goodfellow et al., 2013), sequential-CIFAR10, and sequential-CIFAR100 (Lopez-Paz & Ranzato, 2017) datasets, as well as a text classification task (Mehta et al., 2021) on sequential-YahooQA dataset (Zhang et al., 2015). We also explore FedGP on an object detection task on a streaming CARLA dataset (Dai et al., 2023; Dosovitskiy et al., 2017) in Appendix B. All experiments were conducted on a Linux workstation equipped with 8 NVIDIA GeForce RTX 2080Ti GPUs and averaged across five runs, each using a different seed. For further details and additional results, please refer to Appendix A.

5.1 Image Classification

5.1.1 Settings

Evaluation Datasets. We evaluate our approach on three CL scenarios: domain incremental learning (domain-IL), class incremental learning (class-IL), and task incremental learning (task-IL). For domain-IL, the data distribution of each class changes across different tasks. We use the rotated-MNIST and permuted-MNIST datasets for domain-IL, where each task rotates the training digits by a random angle or applies a random permutation. We create $T = 10$ tasks for domain-IL experiments.

For class-IL and task-IL, we partition the set of classes into disjoint subsets and assign each subset to a particular task. For instance, in our image classification experiments for class-IL and task-IL, we divide the CIFAR-100 dataset (with $C = 100$ classes) into $T = 10$ subsets, each of which contains the samples for $C/T = 10$ classes. Each task $t \in [T]$ is defined as the classification of images from each subset $t \in [T]$. The difference between class-IL and task-IL is that in the task-IL setup, we assume the task identity t is given at inference time. That is, the model f predicts among the $C/T = 10$ classes corresponding to task t. The class-IL and task-IL settings for CIFAR-10 are defined by splitting the CIFAR-10 dataset into $T = 5$ tasks, with each task having two unique classes.

In the FL setup, we assume that the data distribution is non-IID across the different clients. Once we define the data for each task, we split it among K clients in a non-IID manner. For the rotated-MNIST or permuted-MNIST dataset, each client receives samples for two MNIST digits. To create a sequential-CIFAR10 or sequential-CIFAR100 dataset, we use the Latent Dirichlet Allocation (LDA) (Hsu et al., 2019). This algorithm partitions the dataset among multiple clients by assigning samples of each class to different clients based on the probability distribution $p \sim \text{Dir}(\alpha)$, where $\alpha = 0.3$. Communication of models and buffer gradients occurs whenever all clients complete E local epochs training.

Architecture and Hyperparameters. For the rotated-MNIST and permuted-MNIST dataset, we use a simple CNN architecture (McMahan et al., 2017), and split the dataset into $K = 10$ clients. Each client performs local training for $E = 1$ epoch between communications, and we set the number of communication rounds as $R = 20$ for each task. For the sequential-CIFAR10 and sequential-CIFAR100 dataset, we use a ResNet18 architecture, and divide the dataset into $K = 10$ clients. Each client trains for $E = 5$ epochs between communications, and uses $R = 20$ rounds of communication for each task. During local training, Stochastic Gradient Descent (SGD) is employed with a learning rate of 0.01 for MNIST and 0.1 for CIFAR datasets. Unless otherwise noted, our studies used a 200 buffer size, a negligible storage concern for edge device like iPhone.

Baselines. We evaluate the performance improvement of FedGP on three types of baselines: 1) *FL*, the basic FedAvg which trains only on the current task without considering performance on previous tasks; 2) *FL+CL*, which is FedAvg (FL) with continual learning solutions applied to clients; and 3) *CFL*, which represents the existing Continual Federated Learning methods.

CL methods include A-GEM (Chaudhry et al., 2019), which aligns model gradients for buffer and incoming data; DER (Buzzega et al., 2020), utilizing network output logits for past experience distillation; iCaRL (Rebuffi et al., 2017), which adds current task nearest-mean-of-exemplars to a memory buffer via herding and counters representation deterioration with a self-distillation loss term; and L2P (Wang et al., 2022), a state-of-the-art approach that instructs pre-trained models to sequentially learn tasks using prompt pool memory spaces.

CFL methods we tested are FedCurv (Shoham et al., 2019), which avoids updating past task-critical weights; FedProx (Li et al., 2020), introducing a proximal weight for global model alignment; CFeD (Ma et al., 2022), using surrogate dataset-based knowledge distillation; and GLFC (Dong et al., 2022), a tripartite method to counteract forgetting: 1) clients retain old task data and incorporate it with a normalization factor during new task training, 2) clients save the prior model, compute the KL divergence loss between the new and old model outputs (from the last layer), and 3) an additional proxy server is used to gather perturbed gradients for client sample generation. As for FedWeIT (Yoon et al., 2021), we believe it might not serve as a suitable benchmark given its focus on personalized federated learning (FL) without a global model accuracy to contrast with.

Note that CFeD, GLFC, and iCaRL require task boundaries during training. They exploit task changes to snapshot the network, with iCaRL further relying on these for memory buffer updates. For specific parameters and implementation unique to each method, please refer to Appendix A.7.

Performance Metrics. We assess the performance of the global model on the test dataset, which is a union of the test data for all previous tasks. The average accuracy (measured after training on task t) is denoted as $\text{Acc}_t = \frac{1}{t}\sum_{i=1}^{t} a_{t,i}$, where $a_{t,i}$ is evaluated on task i after training up to task t. Additionally, we measure a performance metric called *forgetting* in Appendix A.2, which is defined as the difference between the best accuracy obtained throughout the training and the current accuracy (Chaudhry et al., 2018). This metric measures the model's ability to retain knowledge of previous tasks while learning new ones. The forgetting at task t is defined as: $\text{Fgt}_t = \frac{1}{t-1}\sum_{i=1}^{t-1} \max_{j=1,\cdots,t-1}(a_{j,i} - a_{t,i})$. We also computed the Backward transfer (BWT) and Forward transfer (FWT) metrics (Lopez-Paz & Ranzato, 2017). See Appendix A.5 for details.

5.1.2 RESULTS

Table 1 presents the average accuracy Acc_T of various methods on image classification benchmark datasets measured upon completion of the final task T. For each setting, we compare the performance of an existing method with/without FedGP. We observe that the proposed methods (represented by "w/ FedGP") nearly always improves the base methods ("w/o FedGP") across the different datasets

Table 1: Average accuracy Acc_T (%) on standard benchmark datasets. '-' indicates experiments we were unable to run, because of compatibility issues (e.g. GLFC and iCaRL in Domain-IL) or the absence of surrogate (e.g. CFeD on MNIST). The results, averaged over 5 random seeds, demonstrate the benefits of our proposed method in combination with all baselines. A buffer size of 200 is utilized whenever methods require it. Note that FL+L2P needs additional pretrained ViT.

	rotated-MNIST (*Domain-IL*)		**sequential-CIFAR10** (*Class-IL*)		**sequential-CIFAR10** (*Task-IL*)	
Method	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP
FL (McMahan et al., 2017)	$68.02^{\pm 3.1}$	$79.46^{\pm 4.1}$ (↑**11.44**)	$17.44^{\pm 1.3}$	$18.02^{\pm 0.6}$ (↑**0.58**)	$70.58^{\pm 4.0}$	$80.83^{\pm 2.0}$ (↑**10.25**)
FL+A-GEM (Chaudhry et al., 2019)	$68.34^{\pm 5.6}$	$74.74^{\pm 2.3}$ (↑**6.40**)	$17.82^{\pm 0.9}$	$19.44^{\pm 0.9}$ (↑**1.62**)	$77.14^{\pm 3.1}$	$83.16^{\pm 1.6}$ (↑**6.02**)
FL+DER (Buzzega et al., 2020)	$57.73^{\pm 3.6}$	$81.33^{\pm 3.3}$(↑**23.60**)	$18.44^{\pm 3.7}$	$30.94^{\pm 3.8}$ (↑**12.50**)	$69.34^{\pm 3.2}$	$77.99^{\pm 0.8}$ (↑**8.65**)
FL+iCaRL (Rebuffi et al., 2017)	-	-	$28.54^{\pm 3.8}$	$33.92^{\pm 3.0}$ (↑**5.38**)	$80.85^{\pm 2.9}$	$80.09^{\pm 4.1}$ (↓**0.76**)
FL+L2P (Wang et al., 2022)	$80.90^{\pm 3.3}$	$85.05^{\pm 0.7}$ (↑**4.15**)	$28.61^{\pm 1.0}$	$81.86^{\pm 7.2}$ (↑**53.25**)	$98.49^{\pm 0.1}$	$98.63^{\pm 0.3}$ (↑**0.14**)
FedCurv (Shoham et al., 2019)	$68.21^{\pm 2.6}$	$80.53^{\pm 4.3}$ (↑**12.32**)	$17.36^{\pm 0.7}$	$17.86^{\pm 0.5}$ (↑**0.50**)	$67.77^{\pm 1.4}$	$81.28^{\pm 1.1}$ (↑**13.51**)
FedProx (Li et al., 2020)	$67.79^{\pm 3.2}$	$78.74^{\pm 4.1}$ (↑**10.95**)	$16.67^{\pm 2.7}$	$17.97^{\pm 0.8}$ (↑**1.30**)	$69.57^{\pm 6.5}$	$81.23^{\pm 1.3}$ (↑**11.66**)
CFeD (Ma et al., 2022)	-	-	$16.30^{\pm 4.6}$	$24.07^{\pm 8.5}$ (↑**7.77**)	$77.35^{\pm 4.6}$	$79.30^{\pm 5.7}$ (↑**1.95**)
GLFC (Dong et al., 2022)	-	-	$41.42^{\pm 1.3}$	$41.61^{\pm 1.3}$ (↑**0.19**)	$81.84^{\pm 2.1}$	$82.87^{\pm 1.0}$ (↑**1.03**)
	permuted-MNIST (*Domain-IL*)		**sequential-CIFAR100** (*Class-IL*)		**sequential-CIFAR100** (*Task-IL*)	
FL	$25.92^{\pm 2.1}$	$34.23^{\pm 2.7}$ (↑**8.31**)	$8.76^{\pm 0.1}$	$17.08^{\pm 1.8}$ (↑**8.32**)	$47.74^{\pm 1.2}$	$74.71^{\pm 0.9}$ (↑**26.97**)
FL+A-GEM	$33.43^{\pm 1.4}$	$39.09^{\pm 3.5}$ (↑**5.66**)	$8.90^{\pm 0.1}$	$19.53^{\pm 1.3}$ (↑**10.63**)	$63.84^{\pm 0.8}$	$74.84^{\pm 0.5}$ (↑**11.00**)
FL+DER	$19.79^{\pm 1.7}$	$38.81^{\pm 2.0}$ (↑**19.02**)	$13.32^{\pm 1.6}$	$22.96^{\pm 3.6}$ (↑**9.64**)	$57.71^{\pm 1.2}$	$65.57^{\pm 1.9}$ (↑**7.86**)
FL+iCaRL	-	-	$21.76^{\pm 1.1}$	$27.44^{\pm 1.2}$ (↑**5.68**)	$69.91^{\pm 0.7}$	$72.83^{\pm 0.5}$ (↑**2.92**)
FL+L2P	$66.98^{\pm 4.6}$	$69.15^{\pm 3.1}$ (↑**2.17**)	$23.12^{\pm 1.7}$	$46.16^{\pm 0.4}$ (↑**23.04**)	$94.46^{\pm 0.4}$	$94.91^{\pm 0.2}$ (↑**0.45**)
FedCurv	$26.00^{\pm 2.4}$	$35.21^{\pm 5.1}$ (↑**9.21**)	$8.92^{\pm 0.1}$	$16.67^{\pm 0.9}$ (↑**7.76**)	$49.14^{\pm 1.6}$	$74.64^{\pm 0.7}$ (↑**25.49**)
FedProx	$25.92^{\pm 2.5}$	$35.60^{\pm 4.7}$ (↑**9.68**)	$8.75^{\pm 0.2}$	$16.92^{\pm 1.4}$ (↑**8.17**)	$47.05^{\pm 3.2}$	$73.95^{\pm 0.8}$ (↑**26.89**)
CFeD	-	-	$13.76^{\pm 1.2}$	$26.66^{\pm 0.3}$ (↑**12.9**)	$51.41^{\pm 1.0}$	$72.20^{\pm 0.9}$ (↑**20.79**)
GLFC	-	-	$13.18^{\pm 0.4}$	$13.47^{\pm 0.7}$ (↑**0.29**)	$49.78^{\pm 0.8}$	$49.20^{\pm 1.2}$ (↓**0.58**)

and scenarios, as seen from the upward arrows indicating performance improvements. Additional results are obtained for forgetting performance Fgt_T given in Table 8 in Appendix A.2. Moreover, the performance of FedGP is analyzed progressively across tasks in Appendix A.1.

Remarkably, even a simple integration of the basic baseline, FL, with FedGP surpassed the performance of most FL+CL and CFL baselines. For instance, in the sequential-CIFAR100 experiment, FL with FedGP (17.08% class-IL, 74.71% task-IL) outperformed a majority of the baselines. Specifically, it exceeds the performance of the two advanced CFL baselines: GLFC (13.18% class-IL, 49.78% task-IL) and CFeD (13.76% class-IL, 51.41% task-IL). This underscores the substantial capability of our method in the CFL setting. Importantly, FedGP can achieve competitive performance even without utilizing information about task boundaries, unlike CFeD, GLFC, and iCaRL.

We also note that the FL+L2P method consistently exhibited the highest accuracy, largely due to the utilization of a pretrained Vision Transformer (ViT) (Dosovitskiy et al., 2020; Zhang et al., 2022) and an extensively prolonged runtime. Yet, our approach still managed to achieve significant performance augmentation on top of it.

Table 2: Impact of the buffer size on Acc_T (%)

		rotated-MNIST (*Domain-IL*)		**sequential-CIFAR100** (*Class-IL*)		**sequential-CIFAR100** (*Task-IL*)	
Buffer Size	Method	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP
200	FL+A-GEM	$68.34^{\pm 5.6}$	$74.74^{\pm 2.3}$ (↑**6.40**)	$8.90^{\pm 0.1}$	$19.53^{\pm 1.3}$ (↑**10.63**)	$63.84^{\pm 0.8}$	$74.84^{\pm 0.5}$ (↑**11.00**)
500		$70.18^{\pm 8.7}$	$78.74^{\pm 3.2}$ (↑**8.56**)	$8.87^{\pm 0.1}$	$25.89^{\pm 0.9}$ (↑**17.02**)	$64.38^{\pm 1.4}$	$79.35^{\pm 0.5}$ (↑**14.97**)
5120		$69.97^{\pm 3.2}$	$79.17^{\pm 4.3}$ (↑**9.20**)	$8.85^{\pm 0.1}$	$33.30^{\pm 2.5}$ (↑**24.45**)	$64.99^{\pm 1.5}$	$84.52^{\pm 0.3}$ (↑**19.53**)
200	FL+DER	$57.73^{\pm 3.6}$	$87.13^{\pm 1.1}$ (↑**29.40**)	$13.32^{\pm 1.6}$	$22.96^{\pm 3.6}$ (↑**9.64**)	$57.71^{\pm 1.2}$	$65.57^{\pm 1.9}$ (↑**7.86**)
500		$60.00^{\pm 7.2}$	$88.83^{\pm 1.6}$ (↑**28.83**)	$15.44^{\pm 1.5}$	$34.87^{\pm 1.7}$ (↑**19.43**)	$60.79^{\pm 1.2}$	$73.53^{\pm 1.1}$ (↑**12.74**)
5120		$58.63^{\pm 3.9}$	$89.46^{\pm 1.2}$ (↑**30.83**)	$18.89^{\pm 1.0}$	$45.76^{\pm 3.8}$ (↑**26.87**)	$62.77^{\pm 1.5}$	$83.41^{\pm 1.3}$ (↑**20.64**)

Effect of Buffer size. Table 2 reports the performances of baseline CL methods (A-GEM and DER) with/without FedGP for different buffer sizes, ranging from 200 to 5120. For all different datasets and all IL settings, increasing the buffer size further improves the advantage of applying FedGP, by providing more data for replay and mitigating forgetting. We are assuming that every client has the same buffer size. If the buffer sizes are not equal during model training, clients with bigger buffers

Table 3: Effect of communication on Acc_T (%). Our method outperforms baselines when utilizing equivalent communication overhead.

Method	R-MNIST	P-MNIST	S-CIFAR10		S-CIFAR100	
	Domain-IL	*Domain-IL*	*Class-IL*	*Task-IL*	*Class-IL*	*Task-IL*
FL	$68.02^{\pm 3.1}$	$25.92^{\pm 2.1}$	$17.44^{\pm 1.3}$	$70.58^{\pm 4.0}$	$8.76^{\pm 0.1}$	$47.74^{\pm 1.2}$
FL w/ FedGP ($2\times$ comm overhead)	$\mathbf{79.46^{\pm 4.1}}$	$34.23^{\pm 2.7}$	$\mathbf{18.02^{\pm 0.6}}$	$\mathbf{80.83^{\pm 2.0}}$	$\mathbf{17.08^{\pm 1.8}}$	$\mathbf{74.71^{\pm 0.9}}$
FL w/ FedGP (equalized comm overhead)	$75.63^{\pm 3.9}$	$\mathbf{35.91^{\pm 4.2}}$	$16.65^{\pm 1.0}$	$78.79^{\pm 2.8}$	$13.62^{\pm 0.6}$	$73.96^{\pm 0.4}$

might add more diverse data, which could make the model biased. A possible solution is to use a reweighting algorithm, which we plan to explore in the future.

Effect of communication frequency. Compared with baseline methods, FedGP has extra communication overhead for transmitting the buffer gradients from each client to the server. This means that the required amount of communication is doubled for FedGP. To compare baselines and FedGP under the same communication overhead constraint, we consider reducing the communication frequency of FedGP by a factor of two, while keeping the computation unchanged. Table 3 compares three methods: (i) the FL baseline, (ii) the FL baseline with FedGP, and (iii) the FL baseline with FedGP with the communication frequency reduced by half. Note that (i) and (iii) have the same amount of communication overhead, while (ii) has 2x communication overhead compared with (i). One can confirm that FedGP (with equalized comm overhead) improves the accuracy of FL baseline without additional communication resources. For example, applying FedGP on the sequential-CIFAR100 improves the task-IL accuracy over 26% (from 47.74% to 73.96%), without additional overhead.

Effect of computation overhead. Computation overhead is also an important aspect to consider and we have conducted experiment on the actual wall-clock time measurements. Taking a CIFAR100 experiment as an example, the running time for 200 epochs for FedAvg on our device is 4068.97s. When FedGP, which is built on top of FedAvg, was used, it ran for an additional 293.26s. This indicates that it ran 7.2% longer over the same 200 epochs. The time consumed by FedGP can be divided into two parts: (i) computing the global reference gradient after each FedAvg, and (ii) projecting the gradient. In the above experiment, the reference gradient computation was done 200 times, taking a total of 49.07s, and the gradient projection was performed on 109,471 batches, which is 68.38% of the total batches, taking a total of 244.19s. Overall, FedGP is computationally efficient and affordable for edge devices.

Table 4: Acc_T (%) for asynchronous task boundaries on the sequential-CIFAR100 dataset.

Method	*Class-IL*	*Task-IL*
FL	$16.22^{\pm 1.2}$	$59.04^{\pm 1.7}$
FL+A-GEM	$16.92^{\pm 1.0}$	$69.41^{\pm 1.3}$
FL+A-GEM+FedGP	$30.74^{\pm 1.5}$	$\mathbf{77.70^{\pm 0.4}}$
FL+DER	$31.95^{\pm 2.6}$	$68.28^{\pm 1.5}$
FL+DER+FedGP	$\mathbf{36.29^{\pm 1.0}}$	$72.02^{\pm 0.7}$

Asynchronous task boundaries. In our previous experiments, we assumed synchronous task boundaries where clients finish tasks at the same time. However, in many real-world scenarios, different clients finish each task asynchronously. Motivated by this practical setting, we conducted experiments in an asynchronous task boundary setting on sequential-CIFAR100. For every $R = 20$ communications, instead of traversing all data allocated for the current task, each client traverses exactly 500 samples allocated to it, irrespective of whether these samples come from the same task. Consequently, some clients might finish a task faster than others and move on to the next task. Thus, during each global communication, clients could be working on different tasks. This setup more closely aligns with our general continual learning settings, when the task boundary is unknown. Table 4 shows the accuracy of each method averaged over T tasks after finishing all training, under the asynchronous setting. Similar to the synchronous case, FedGP improves the accuracy of baseline methods including A-GEM and DER. Notably, we have a better performance in the asynchronous setting (see Table 4) compared with the synchronous setting (see Table 1). This might be because, in the asynchronous setting, some clients receive new tasks earlier than others, which allows the model to be exposed to more diverse data for each round, thus reducing the forgetting effect.

Effect of the number of tasks. We have conducted experiments with different number of tasks for each dataset. For CIFAR100, we experimented with task numbers 5 and 10, while for CIFAR10 we tested with task numbers 2 and 5. Our results in Table 5 consistently demonstrate that the FedGP algorithm provides a significant improvement in performance across all these different

task numbers. An interesting observation is that as the number of tasks increases, FedGP have better performance improvement to baseline. This is because a higher number of tasks increases the likelihood of data distribution shifts and therefore the problem of catastrophic forgetting becomes more prominent. As such, FedGP, designed to handle this issue, has more opportunities to improve the learning process in such scenarios. This might also partly explain why, in Table 1, FedGP shows a generally higher improvement over the baselines on the sequential-CIFAR100 dataset compared to the sequential-CIFAR10.

Table 5: Average accuracy Acc_T (%) across various task numbers.

(# of Task, # of Classes per Task)	**sequential-CIFAR10** (*Class-IL*) FL	**sequential-CIFAR10** (*Class-IL*) FL w/ FedGP	**sequential-CIFAR10** (*Task-IL*) FL	**sequential-CIFAR10** (*Task-IL*) FL w/ FedGP
(2, 5)	$43.53^{\pm 0.8}$	$44.05^{\pm 0.8}$ (↑**0.52**)	$75.54^{\pm 0.6}$	$77.52^{\pm 0.8}$ (↑**1.98**)
(5, 2)	$17.44^{\pm 1.3}$	$18.02^{\pm 0.6}$ (↑**0.6**)	$70.58^{\pm 4.0}$	$80.83^{\pm 2.0}$ (↑**10.25**)
	sequential-CIFAR100 (*Class-IL*) FL	**sequential-CIFAR100** (*Class-IL*) FL w/ FedGP	**sequential-CIFAR100** (*Task-IL*) FL	**sequential-CIFAR100** (*Task-IL*) FL w/ FedGP
(5, 20)	$16.49^{\pm 0.3}$	$22.71^{\pm 0.9}$ (↑**6.22**)	$50.60^{\pm 0.9}$	$69.41^{\pm 0.8}$ (↑**18.81**)
(10, 10)	$8.76^{\pm 0.1}$	$17.08^{\pm 1.8}$ (↑**8.32**)	$47.74^{\pm 1.2}$	$74.71^{\pm 0.9}$ (↑**26.97**)

Effect of the number of users. We also conducted experiments to assess scalability by increasing the client count to $K = 20$. Table 6 shows the results for $K = 20$ clients. This results demonstrates that FedGP consistently improves the performance of baselines, across different number of clients.

Table 6: The Acc_T (%) performance measured when we have $K = 20$ users. Similar to the results for $K = 10$ in Table 1, our method improves the performance of baselines.

	rotated-MNIST (*Domain-IL*)		**sequential-CIFAR10** (*Class-IL*)		**sequential-CIFAR10** (*Task-IL*)	
Method	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP
FL	$62.45^{\pm 8.5}$	$76.01^{\pm 4.6}$ (↑**13.56**)	$16.44^{\pm 1.4}$	$15.82^{\pm 1.7}$ (↓**0.62**)	$68.18^{\pm 5.3}$	$73.45^{\pm 4.3}$ (↑**5.27**)
FedCurv	$62.57^{\pm 8.3}$	$76.46^{\pm 4.1}$ (↑**13.89**)	$17.31^{\pm 0.6}$	$14.64^{\pm 3.1}$ (↓**2.67**)	$67.33^{\pm 3.3}$	$70.31^{\pm 3.7}$ (↑**2.98**)
FedProx	$62.14^{\pm 8.6}$	$75.84^{\pm 4.4}$ (↑**13.70**)	$16.37^{\pm 1.1}$	$16.15^{\pm 1.3}$ (↓**0.22**)	$66.24^{\pm 1.4}$	$74.79^{\pm 3.9}$ (↑**8.55**)
FL+A-GEM	$67.66^{\pm 8.0}$	$78.10^{\pm 3.6}$ (↑**10.44**)	$16.15^{\pm 1.9}$	$17.36^{\pm 0.8}$ (↑**1.21**)	$72.39^{\pm 3.4}$	$80.61^{\pm 2.6}$ (↑**8.22**)
FL+DER	$57.33^{\pm 3.2}$	$87.84^{\pm 1.5}$ (↑**30.51**)	$17.13^{\pm 2.3}$	$19.18^{\pm 3.7}$ (↑**2.05**)	$70.82^{\pm 1.9}$	$77.04^{\pm 2.5}$ (↑**6.22**)
	permuted-MNIST (*Domain-IL*)		**sequential-CIFAR100** (*Class-IL*)		**sequential-CIFAR100** (*Task-IL*)	
FL	$20.26^{\pm 1.6}$	$20.67^{\pm 4.7}$ (↑**0.41**)	$8.61^{\pm 0.1}$	$17.47^{\pm 1.1}$ (↑**8.86**)	$50.00^{\pm 1.6}$	$76.29^{\pm 0.8}$ (↑**26.29**)
FedCurv	$20.25^{\pm 1.9}$	$23.30^{\pm 5.7}$ (↑**3.05**)	$8.93^{\pm 0.0}$	$19.42^{\pm 1.1}$(↑**10.49**)	$49.83^{\pm 1.4}$	$79.58^{\pm 0.6}$ (↑**29.75**)
FedProx	$20.19^{\pm 1.4}$	$23.78^{\pm 5.2}$ (↑**3.59**)	$8.88^{\pm 0.1}$	$18.86^{\pm 1.0}$ (↑**9.98**)	$50.86^{\pm 1.2}$	$78.19^{\pm 0.9}$ (↑**27.33**)
FL+A-GEM	$24.43^{\pm 2.1}$	$23.29^{\pm 3.8}$ (↓**1.14**)	$8.62^{\pm 0.1}$	$19.58^{\pm 1.2}$ (↑**10.96**)	$63.02^{\pm 0.6}$	$76.23^{\pm 0.6}$ (↑**13.21**)
FL+DER	$17.89^{\pm 1.3}$	$46.17^{\pm 3.0}$ (↑**28.28**)	$11.53^{\pm 0.5}$	$26.64^{\pm 2.8}$ (↑**15.11**)	$57.00^{\pm 1.4}$	$69.42^{\pm 1.0}$ (↑**12.42**)

Additionally, we evaluated a real-world scenario where only a random subset of clients participates in training during each round. Detailed information and results are available in Appendix A.4

5.2 TEXT CLASSIFICATION

In addition to image classification, we also extended the evaluation of our method on text classification task (Mehta et al., 2021). For this purpose, we utilized the YahooQA (Zhang et al., 2015) dataset which comprises texts (questions and answers), and user-generated labels representing 10 different topics. Similar to the approach taken with the CIFAR10 dataset, we partitioned the YahooQA dataset into 5 tasks, where each task consisted of two distinct classes. Within each task, we used LDA to partition data across 10 clients in a non-IID manner. To conduct the experiment, we employed a pretrained DistilBERT (Sanh et al., 2019) with linear classification layer. We freeze the DistilBERT model and only fine-tune the additional linear layer. The results of this experiment can be found in Table 7. We can observe that FedGP consistently enhances the accuracy (Acc_T) over baselines, particularly in class-IL scenarios.

Table 7: Average classification accuracy Acc_T (%) on split-YahooQA dataset.

Method	**sequential-YahooQA** (*Class-IL*) w/o FedGP	**sequential-YahooQA** (*Class-IL*) w/ FedGP	**sequential-YahooQA** (*Task-IL*) w/o FedGP	**sequential-YahooQA** (*Task-IL*) w/ FedGP
FL	$17.86^{\pm 0.6}$	$30.67^{\pm 4.4}$(↑**12.81**)	$80.87^{\pm 1.2}$	$88.04^{\pm 1.4}$(↑**7.17**)
FL+A-GEM	$20.86^{\pm 0.3}$	$47.02^{\pm 1.9}$(↑**26.16**)	$87.29^{\pm 1.3}$	$90.20^{\pm 0.2}$(↑**2.91**)
FL+DER	$43.64^{\pm 2.1}$	$54.28^{\pm 1.3}$(↑**10.64**)	$89.57^{\pm 0.2}$	$90.48^{\pm 0.2}$(↑**0.91**)

6 Conclusion

In this paper, we present FedGP, a novel method of using buffer data for mitigating the catastrophic forgetting issues in CFL. Specifically, we use the gradient projection method to prevent model updates that harm the performance on previous tasks. Our empirical results on benchmark datasets (rotated-MNIST, permuted-MNIST, sequential-CIFAR10 and sequential-CIFAR100) and on a text classification dataset show that FedGP improves the performance of existing CL and CFL methods.

Acknowledgements

This work is funded by Intel/NSF joint grant CNS-2003129. Suman Banerjee and Shenghong Dai were also supported in part through the following US National Science Foundation(NSF) grants: CNS-1838733, CNS-1719336, CNS-1647152, and CNS-1629833, and an award from the US Department of Commerce with award number 70NANB21H043. Kangwook Lee was also supported by NSF Award DMS-2023239, and the University of Wisconsin-Madison Office of the Vice Chancellor for Research and Graduate Education with funding from the Wisconsin Alumni Research Foundation. Yicong Chen was supported by the 2023 Wisconsin Hilldale Undergraduate Research Fellowship.

References

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert Gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3366–3375, 2017.

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 139–154, 2018.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019.

Yavuz Faruk Bakman, Duygu Nur Yaldiz, Yahya H Ezzeldin, and Salman Avestimehr. Federated orthogonal training: Mitigating global catastrophic forgetting in continual federated learning. *arXiv preprint arXiv:2309.01289*, 2023.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, 2017.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930, 2020.

Hyuntak Cha, Jaeho Lee, and Jinwoo Shin. Co^{2}L: Contrastive continual learning. In *Proceedings of the IEEE/CVF International conference on computer vision*, pp. 9516–9525, 2021.

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)*, pp. 532–547, 2018.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. In *ICLR*, 2019.

Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018.

Shenghong Dai, SM Iftekharul Alam, Ravikumar Balakrishnan, Kangwook Lee, Suman Banerjee, and Nageen Himayat. Online federated learning based object detection across autonomous vehicles in a virtual world. In *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, pp. 919–920. IEEE, 2023.

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*, pp. 10164–10173, 2022.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Conference on robot learning*, pp. 1–16. PMLR, 2017.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3762–3773. PMLR, 2020.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. PathNet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3 (4):128–135, 1999.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pp. 5132–5143. PMLR, 2020.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114 (13):3521–3526, 2017.

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.

Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. *arXiv preprint arXiv:1905.10497*, 2019.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

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

Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):2031–2063, 2020.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Yuhang Ma, Zhongle Xie, Jue Wang, Ke Chen, and Lidan Shou. Continual federated learning based on knowledge distillation. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, volume 3, 2022.

Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, Hyunwoo Kim, and Scott Sanner. Online continual learning in image classification: An empirical survey. *Neurocomputing*, 469:28–51, 2022.

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*, pp. 7765–7773, 2018.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

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*, pp. 1273–1282. PMLR, 2017.

Sanket Vaibhav Mehta, Darshan Patil, Sarath Chandar, and Emma Strubell. An empirical investigation of the role of pre-training in lifelong learning. *arXiv preprint arXiv:2112.09153*, 2021.

Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. Agnostic federated learning. In *International Conference on Machine Learning*, pp. 4615–4625. PMLR, 2019.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.

Quang Pham, Chenghao Liu, and Steven Hoi. DualNet: Continual learning, fast and slow. *Advances in Neural Information Processing Systems*, 34:16131–16144, 2021.

Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. GDumb: A simple approach that questions our progress in continual learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pp. 524–540. Springer, 2020.

Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. iCaRL: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

G Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, et al. OpenFL: An open-source framework for federated learning. *arXiv preprint arXiv:2105.06413*, 2021.

Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2): 123–146, 1995.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pp. 4548–4557. PMLR, 2018.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

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*, pp. 3400–3409, 2017.

Neta Shoham, Tomer Avidor, Aviv Keren, Nadav Israel, Daniel Benditkis, Liron Mor-Yosef, and Itai Zeitak. Overcoming forgetting in federated learning on non-IID data. *arXiv preprint arXiv:1910.07796*, 2019.

Sebastian Thrun. Is learning the n-th thing any easier than learning the first? *Advances in neural information processing systems*, 8, 1995.

Yeshwanth Venkatesha, Youngeun Kim, Hyoungseob Park, Yuhang Li, and Priyadarshini Panda. Addressing client drift in federated continual learning with adaptive optimization. *Available at SSRN 4188586*, 2022.

Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 139–149, 2022.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184, 2020.

Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 374–382, 2019.

Xin Yao and Lifeng Sun. Continual local training for better initialization of federated models. In *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 1736–1740. IEEE, 2020.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

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*, pp. 12073–12086. PMLR, 2021.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pp. 3987–3995. PMLR, 2017.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

Zizhao Zhang, Han Zhang, Long Zhao, Ting Chen, Sercan Ö Arik, and Tomas Pfister. Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 3417–3425, 2022.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-IID data. *arXiv preprint arXiv:1806.00582*, 2018.

A SUPPLEMENTARY RESULTS

In this section, we furnish additional experimental outcomes that serve to further bolster the findings of our primary investigation.

A.1 PROGRESSIVE PERFORMANCE OF FEDGP ACROSS TASKS

Fig. 3 depicts the average accuracy Acc_t measured at task $t = 1, 2, \cdots, 10$ and the average forgetting Fgt_t measured at task $t = 2, 3, \cdots, 10$. The accuracy of FedAvg rapidly drops as different tasks are given to the model, as expected. FedCurv and FedProx perform similarly to FedAvg, while A-GEM and DER partially alleviate forgetting, resulting in higher accuracies and reduced forgetting compared to FedAvg. Combining these baselines with FedGP lead to significant performance improvements, which allows the solid lines in the accuracy plot consistently remain at the top. For example, for the experiment on task-IL for sequential-CIFAR100, the accuracy measured at task 5 (denoted by Acc_5) is 55.37% for FedProx, while 71.12% for FedProx+FedGP. These results demonstrate that FedGP effectively mitigates forgetting and enhances existing methods in CFL.

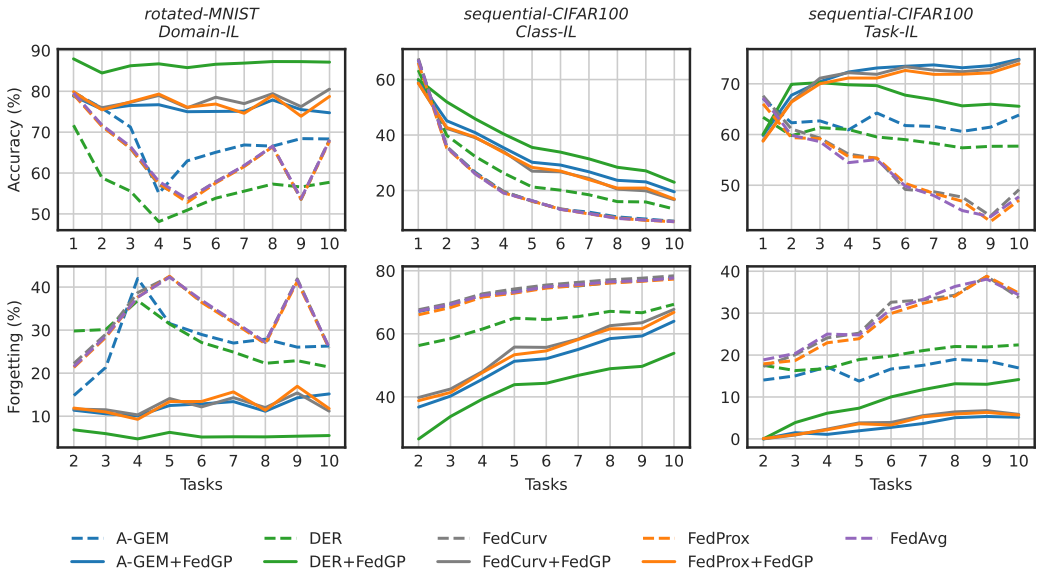


Figure 3: Evaluating accuracy (↑) and forgetting (↓) in multiple datasets with and without FedGP using a buffer size of 200. The solid lines indicate the results obtained with our method, while the dotted lines represent the results obtained without our method. The results show a significant improvement in accuracy as well as reduced forgetting for all settings.

A.2 FORGETTING ANALYSIS ACROSS DATASETS

We present the complementary information to Table 1 in Table 8, illustrating the extent of Fgt_T observed across multiple benchmark datasets. Our method exhibits exceptional effectiveness in mitigating forgetting. Remarkably, it demonstrates consistent performance across all datasets and baselines, making it a versatile solution.

In line with the presentation of forgetting in Table 8, we present the forgetting analysis when the number of clients is set to 20 in Table 9. Notably, our method exhibits consistent and impressive performance across varying numbers of users. It consistently proves its effectiveness regardless of the specific user count, showcasing its robustness and reliability.

Table 8: Average forgetting Fgt_T (%) (lower is better) on benchmark datasets at the final task T.

	rotated-MNIST (*Domain-IL*)		**sequential-CIFAR10** (*Class-IL*)		**sequential-CIFAR10** (*Task-IL*)	
Method	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP
FL	$25.98^{\pm 3.2}$	$11.66^{\pm 2.7}$(↓**14.32**)	$80.69^{\pm 3.6}$	$78.62^{\pm 4.3}$(↓**2.07**)	$15.37^{\pm 4.8}$	$4.49^{\pm 1.9}$(↓**10.88**)
FedCurv	$25.80^{\pm 2.4}$	$11.18^{\pm 2.7}$(↓**14.62**)	$80.90^{\pm 6.6}$	$79.85^{\pm 3.9}$(↓**1.05**)	$19.37^{\pm 4.8}$	$4.77^{\pm 1.6}$(↓**14.60**)
FedProx	$25.74^{\pm 3.1}$	$11.76^{\pm 2.9}$(↓**13.98**)	$84.35^{\pm 2.4}$	$80.24^{\pm 2.5}$(↓**4.11**)	$18.24^{\pm 4.9}$	$4.17^{\pm 1.0}$(↓**14.07**)
FL+A-GEM	$26.30^{\pm 5.7}$	$15.18^{\pm 2.4}$(↓**11.12**)	$82.18^{\pm 6.6}$	$80.38^{\pm 2.5}$(↓**1.80**)	$10.00^{\pm 3.0}$	$4.15^{\pm 0.7}$(↓**5.85**)
FL+DER	$21.42^{\pm 4.0}$	$5.51^{\pm 1.2}$(↓**15.91**)	$60.98^{\pm 14.6}$	$47.88^{\pm 7.2}$(↓**13.10**)	$6.34^{\pm 4.9}$	$2.73^{\pm 1.3}$(↓**3.61**)
	permuted-MNIST (*Domain-IL*)		**sequential-CIFAR100** (*Class-IL*)		**sequential-CIFAR100** (*Task-IL*)	
FL	$43.47^{\pm 5.3}$	$21.40^{\pm 4.9}$(↓**22.07**)	$77.69^{\pm 0.5}$	$67.02^{\pm 2.3}$(↓**10.67**)	$34.38^{\pm 1.6}$	$5.39^{\pm 0.8}$(↓**28.99**)
FedCurv	$42.88^{\pm 5.0}$	$22.85^{\pm 3.5}$(↓**20.03**)	$78.40^{\pm 0.9}$	$67.75^{\pm 0.8}$(↓**10.65**)	$33.71^{\pm 2.2}$	$5.86^{\pm 0.7}$(↓**27.85**)
FedProx	$42.59^{\pm 5.6}$	$20.77^{\pm 5.6}$(↓**21.82**)	$77.35^{\pm 0.4}$	$66.81^{\pm 2.2}$(↓**10.54**)	$34.79^{\pm 3.6}$	$5.69^{\pm 0.9}$(↓**29.10**)
FL+A-GEM	$35.61^{\pm 5.3}$	$24.05^{\pm 2.4}$(↓**11.56**)	$77.97^{\pm 0.7}$	$63.99^{\pm 2.0}$(↓**13.98**)	$16.92^{\pm 1.1}$	$5.16^{\pm 0.5}$(↓**11.76**)
FL+DER	$45.33^{\pm 5.0}$	$34.71^{\pm 5.0}$(↓**10.62**)	$69.37^{\pm 1.7}$	$53.84^{\pm 6.7}$(↓**15.53**)	$22.43^{\pm 0.7}$	$14.16^{\pm 1.7}$(↓**8.27**)

Table 9: The Fgt_T (%) (lower is better) performance measured when we have $K = 20$ users.

	rotated-MNIST (*Domain-IL*)		**sequential-CIFAR10** (*Class-IL*)		**sequential-CIFAR10** (*Task-IL*)	
Method	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP
FL	$31.00^{\pm 9.5}$	$13.45^{\pm 3.6}$ (↓**17.55**)	$82.62^{\pm 3.1}$	$73.39^{\pm 4.5}$ (↓**9.23**)	$17.93^{\pm 2.7}$	$6.14^{\pm 4.9}$ (↓**11.79**)
FedCurv	$30.73^{\pm 9.3}$	$12.97^{\pm 3.8}$ (↓**17.76**)	$79.55^{\pm 3.8}$	$75.38^{\pm 5.3}$ (↓**4.17**)	$18.19^{\pm 3.0}$	$9.14^{\pm 3.1}$ (↓**9.05**)
FedProx	$31.04^{\pm 9.7}$	$13.31^{\pm 3.4}$ (↓**17.73**)	$82.94^{\pm 1.1}$	$78.67^{\pm 4.2}$ (↓**4.27**)	$20.60^{\pm 2.6}$	$8.52^{\pm 3.0}$ (↓**12.08**)
FL+A-GEM	$25.22^{\pm 8.8}$	$11.02^{\pm 3.0}$ (↓**14.20**)	$82.39^{\pm 2.4}$	$80.25^{\pm 4.1}$ (↓**2.14**)	$12.29^{\pm 2.2}$	$4.00^{\pm 2.4}$ (↓**8.29**)
FL+DER	$28.93^{\pm 6.6}$	$5.18^{\pm 1.1}$ (↓**23.75**)	$55.10^{\pm 9.8}$	$60.90^{\pm 3.8}$ (↑**5.80**)	$3.20^{\pm 1.6}$	$2.71^{\pm 1.7}$ (↓**0.49**)
	permuted-MNIST (*Domain-IL*)		**sequential-CIFAR100** (*Class-IL*)		**sequential-CIFAR100** (*Task-IL*)	
FL	$24.27^{\pm 5.2}$	$8.67^{\pm 7.0}$ (↓**15.60**)	$73.05^{\pm 0.5}$	$62.71^{\pm 0.9}$ (↓**10.34**)	$27.07^{\pm 1.7}$	$2.48^{\pm 0.7}$ (↓**24.59**)
FedCurv	$24.02^{\pm 5.4}$	$8.10^{\pm 5.4}$ (↓**15.92**)	$80.07^{\pm 0.5}$	$68.58^{\pm 1.1}$ (↓**11.49**)	$34.63^{\pm 1.7}$	$3.48^{\pm 0.6}$ (↓**31.15**)
FedProx	$23.01^{\pm 5.7}$	$5.93^{\pm 5.1}$ (↓**17.08**)	$79.46^{\pm 0.5}$	$68.40^{\pm 0.9}$ (↓**11.06**)	$32.82^{\pm 1.4}$	$4.13^{\pm 0.7}$ (↓**28.69**)
FL+A-GEM	$22.12^{\pm 4.9}$	$9.45^{\pm 5.4}$ (↓**12.67**)	$72.97^{\pm 1.1}$	$60.27^{\pm 1.3}$ (↓**12.70**)	$12.54^{\pm 1.3}$	$2.66^{\pm 0.2}$ (↓**9.88**)
FL+DER	$32.26^{\pm 1.1}$	$27.30^{\pm 4.2}$ (↓**4.96**)	$67.07^{\pm 0.8}$	$47.74^{\pm 3.8}$ (↓**19.33**)	$19.78^{\pm 1.7}$	$8.67^{\pm 1.4}$ (↓**11.11**)

A.3 Extended analysis on the influence of buffer size

In the main body of our study, we examine the influence of different buffer sizes on the performance metric Acc_T, utilizing rotated-MNIST and sequential-CIFAR100 datasets. To further augment our analysis, we have included two additional datasets in Table 10, incorporating various buffer sizes. By evaluating Acc_T (where higher values indicate better performance), we discovered that our proposed method, referred to as FedGP, consistently enhances the average accuracy across these two datasets.

Table 10: Impact of the buffer size on Acc_T (%)

		permuted-MNIST (*Domain-IL*)		**sequential-CIFAR10** (*Class-IL*)		**sequential-CIFAR10** (*Task-IL*)	
Buffer Size	Method	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP	w/o FedGP	w/ FedGP
200	FL+A-GEM	$33.43^{\pm 1.4}$	$39.09^{\pm 3.5}$ (↑**5.66**)	$17.82^{\pm 0.9}$	$19.44^{\pm 0.9}$ (↑**1.62**)	$77.14^{\pm 3.1}$	$83.16^{\pm 1.6}$ (↑**6.02**)
500	FL+A-GEM	$33.35^{\pm 1.0}$	$42.45^{\pm 6.9}$ (↑**9.10**)	$18.39^{\pm 0.2}$	$20.34^{\pm 0.6}$ (↑**1.95**)	$78.43^{\pm 3.0}$	$85.95^{\pm 0.6}$ (↑**7.52**)
5120	FL+A-GEM	$32.72^{\pm 1.4}$	$40.07^{\pm 2.5}$ (↑**7.35**)	$16.41^{\pm 2.6}$	$20.64^{\pm 2.2}$ (↑**4.23**)	$73.89^{\pm 3.3}$	$86.82^{\pm 1.5}$ (↑**12.93**)
200	FL+DER	$19.79^{\pm 1.7}$	$43.43^{\pm 0.9}$ (↑**23.64**)	$18.44^{\pm 3.7}$	$30.94^{\pm 3.8}$ (↑**12.50**)	$69.34^{\pm 3.2}$	$77.99^{\pm 0.8}$ (↑**8.65**)
500	FL+DER	$19.17^{\pm 1.6}$	$43.38^{\pm 2.4}$ (↑**24.21**)	$20.81^{\pm 3.6}$	$29.78^{\pm 4.3}$ (↑**8.97**)	$71.17^{\pm 1.5}$	$74.98^{\pm 3.5}$ (↑**3.81**)
5120	FL+DER	$18.57^{\pm 1.4}$	$44.68^{\pm 2.4}$ (↑**26.11**)	$34.75^{\pm 2.2}$	$42.38^{\pm 4.5}$ (↑**7.63**)	$78.22^{\pm 2.3}$	$81.94^{\pm 1.7}$ (↑**3.72**)

A.4 Random sampling

We implement a more realistic federated learning environment by applying uniform sampling techniques to randomly select the participating clients in each round. We conduct experiments on CIFAR100. A total of 50 clients is set up, and during each communication, only a random 50% of the clients participate in training. As can be seen, even in such a scenario, where our algorithm cannot update the reference gradient using the local buffer from all clients, there is still an improvement in performance using our algorithm.

Table 11: Average accuracy Acc_T (%) with 50 clients and 50% client sampling rate, for sequential-CIFAR100

Method	*Class-IL*	*Task-IL*
FL	7.46 ± 0.08	43.85 ± 1.33
FL+FedGP	9.34 ± 0.31 ($\uparrow$ 1.88)	65.76 ± 0.48 ($\uparrow$ 21.91)

A.5 Backward and forward transfer metrics

Our method outperforms FedAvg (FL) in both Backward and Forward Transfer metrics across the sequential-CIFAR10 and sequential-CIFAR100 datasets, as shown in the Table 12.

Table 12: Backward and Forward Transfer ($\uparrow$) Results for sequential-CIFAR100 and sequential-CIFAR10

Metric	Dataset	Methods	*Class-IL*	*Task-IL*
Backward	CIFAR100	FL	-78.11	-36.52
Backward	CIFAR100	FL+FedGP	-72.24 ($\uparrow$ 5.87)	-3.68 ($\uparrow$ 32.84)
Backward	CIFAR10	FL	-78.78	-14.48
Backward	CIFAR10	FL+FedGP	-78.55 ($\uparrow$ 0.23)	-0.60 ($\uparrow$ 13.88)
Forward	CIFAR100	FL	16.98	16.98
Forward	CIFAR100	FL+FedGP	17.16 ($\uparrow$ 0.18)	17.48 ($\uparrow$ 0.50)
Forward	CIFAR10	FL	12.75	12.74
Forward	CIFAR10	FL+FedGP	12.98 ($\uparrow$ 0.23)	12.99 ($\uparrow$ 0.25)

A.6 Effect of different curriculum.

We evaluate how the performance of FedGP changes when we shuffle the order of tasks in the continual learning. We randomly shuffle the sequential-CIFAR100 task order and label them as curriculum 1 to 4, as shown in the Table 13. Regardless of the different curriculum, FL+FedGP outperforms FedAvg.

Table 13: Average accuracy Acc_T (%) across randomized curriculum in sequential-CIFAR100.

Curriculum	Methods	*Class-IL*	*Task-IL*
1	FL	8.15	46.25
1	FL+FedGP	12.10 ($\uparrow$ 3.95)	72.69 ($\uparrow$ 26.44)
2	FL	8.46	47.56
2	FL+FedGP	14.37 ($\uparrow$ 5.91)	73.19 ($\uparrow$ 25.63)
3	FL	8.82	45.04
3	FL+FedGP	12.58 ($\uparrow$ 3.76)	74.71 ($\uparrow$ 29.67)
4	FL	7.87	43.87
4	FL+FedGP	14.85 ($\uparrow$ 6.98)	73.74 ($\uparrow$ 29.87)

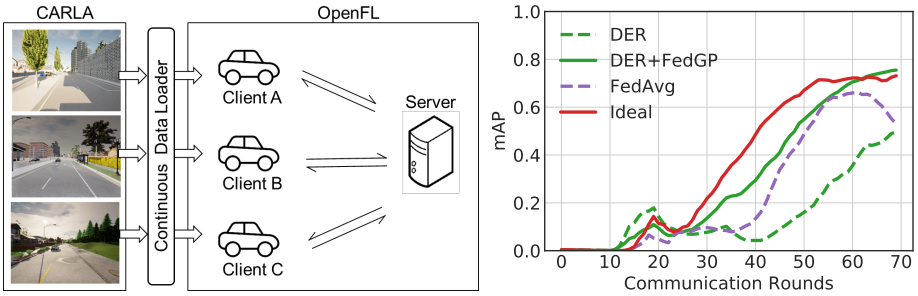
A.7 Additional hyperparameters for specific methods

In addition to the hyperparameters discussed in the main paper, additional method-specific hyperparameters are outlined in Table 14.

Table 14: Additional hyperparameters for specific methods.

Method	**Parameter**	**Values**
FL+DER	Regularization Coefficient	sequential-CIFAR10 (0.3), Others (1)
FL+L2P	Communication Round R	rotated-MNIST (5), permuted-MNIST (1), sequential-CIFAR10 (20), sequential-CIFAR100 (20)
CFeD	Surrogate Dataset Note: No server distillation included.	sequential-CIFAR10 (CIFAR100), sequential-CIFAR100 (CIFAR10)

B OBJECT DETECTION



(a) Framework for automotive data evaluation. (b) Object detection performance comparison.

Figure 4: (a) The data loader continuously supplies data from CARLA camera outputs to individual FL clients. Each client trains on its local data and updates its buffer to retain old knowledge. (b) The result shows the object detection performance comparison between Ideal, FedAvg, DER, and DER+FedGP on a realistic CARLA dataset.

Here we test FedGP on realistic streaming data (Dai et al., 2023) which leverage two open source tools, an urban driving simulator (CARLA (Dosovitskiy et al., 2017)) and a FL framework (OpenFL (Reina et al., 2021)). As shown in Fig. 4a, CARLA provides OpenFL with a real-time collection of continuous streaming vehicle camera output data and automatic annotation about object detection. This streaming data capture the spatio-temporal dynamics of data generated from real-world applications. After loading data of vehicles from CARLA, OpenFL performs collaborative training over multiple clients.

We evaluate the solutions to the forgetting problem by spawning two vehicles in a virtual town. During the training of the tinyYOLO (Redmon & Farhadi, 2017) object detection model, we use a custom loss that combines classification, detection and confidence losses. In order to quantify the quality of the incremental model trained by various baselines, we report a common metric, namely, mean average precision (mAP). This metric assesses the correspondence between the detected bounding boxes and the ground truth, with higher scores indicating better performance. To calculate mAP, we analyze the prediction results obtained from pre-collected driving snippets of vehicular clients. These driving snippets are gathered by navigating the town over a duration of 3000 simulation seconds.

For those experiments on realistic CARLA streaming data, we compare the performances of Ideal, FedAvg, DER and DER+FedGP. In the Ideal scenario, the client possesses sufficient memory to retain all data from prior tasks, enabling joint training on all stored data. The last two methods are equipped with buffer size of 200. We train for 70 communication rounds and each round continues for about 200 simulation seconds. The results are presented in Fig. 4b. Note that at communication round 60, one client gets on the highway, which incurs a domain shift. One can confirm that the performance of FedAvg degrades in such domain shift scenario, whereas DER and DER+FedGP maintain the accuracy. Moreover, FedGP nearly achieves the performance of the ideal scenario with infinite buffer size, demonstrating the effectiveness of our method.

C ADDITIONAL ALGORITHMS

In this section, we present the pseudocode for the `ReservoirSampling` algorithm (see Algorithm 4). In the initial phase, when the buffer is not yet full (i.e., $n \leq |\mathcal{M}^k|$), `ReservoirSampling` stores each new sample (x, y) in the buffer. After the buffer is full, the algorithm determines two things: (1) whether it should replace an element in the buffer with the new sample, and (2) which element in the buffer it will replace.

Algorithm 4 `ReservoirSampling`$(\mathcal{M}^k, (x, y), n)$ (Vitter, 1985) at client k

Input: local buffer $\mathcal{M}^k$, incoming data (x, y) and the number of observed samples n
if $n \leq |\mathcal{M}^k|$ **then**
  Add data (x, y) into local buffer $\mathcal{M}^k$
else
  $i \leftarrow \text{Uniform}\{1, 2, \cdots, n\}$
  if $i \leq |\mathcal{M}^k|$ **then**
    $\mathcal{M}^k[i] \leftarrow (x, y)$
  end if
end if
Return $\mathcal{M}^k$, the updated local buffer

Algorithm 5 DER `ClientUpdate` at client k

Input: Task index t, model w, buffer gradient g_{ref}
Load the dataset $\mathcal{D}_t^k$, local buffer $\mathcal{M}^k$
Initialize $n = 0$ at the first task
for $(x, y) \in \mathcal{D}_t^k$ **do**
  $z \leftarrow h(x; w)$ where $f(x; w) \coloneqq \sigma(h(x; w))$
  $(x', z', y') \leftarrow \mathcal{M}^k$
  $\ell_{\text{reg}} \leftarrow \lambda \|z' - h(x'; w)\|_2^2$
  $g = \nabla_w [\ell(y, f(x; w)) + \ell_{\text{reg}}]$
  $\tilde{g} \leftarrow g - \text{proj}_{g_{\text{ref}}} g \cdot \mathbf{1}(g_{\text{ref}}^\top g \leq 0)$
  $w \leftarrow w - \alpha \tilde{g}$ for some learning rate α
  `ReservoirSampling`$(\mathcal{M}^k, (x, z, y), n)$
  $n \leftarrow n + 1$
end for
Return w to server

Algorithm 6 A-GEM `ClientUpdate` at client k

Input: Task index t, model w, buffer gradient g_{ref}
Load the dataset $\mathcal{D}_t^k$, local buffer $\mathcal{M}^k$
Initialize $n = 0$ at the first task
for $(x, y) \in \mathcal{D}_t^k$ **do**
  $g_c = \nabla_w [\ell(y, f(x; w))]$
  $(x', y') \leftarrow \mathcal{M}^k$
  $g_b = \nabla_w [\ell(y', f(x'; w))]$
  $g \leftarrow g_c - \text{proj}_{g_b} g_c \cdot \mathbf{1}(g_b^\top g_c \leq 0)$
  $\tilde{g} \leftarrow g - \text{proj}_{g_{\text{ref}}} g \cdot \mathbf{1}(g_{\text{ref}}^\top g \leq 0)$
  $w \leftarrow w - \alpha \tilde{g}$ for some learning rate α
  `ReservoirSampling`$(\mathcal{M}^k, (x, y), n)$
  $n \leftarrow n + 1$
end for
Return w to server

D CONTINUAL LEARNING METHODS WITH FEDGP

We provided the pseudocode for Algorithm 2 modifications when implementing FL+DER+FedGP and FL+A-GEM+FedGP, respectively presented in Algorithm 5 and Algorithm 6. Other FL+CL and CFL methods are also combined with FedGP in a similar manner.

Algorithm 5 incorporates Dark Experience Replay (DER) into the local update process on client $k \in [K]$.When the server sends the global model w to client k, the client calculates the output logits or pre-softmax response z. In addition, the client samples past data (x', y') and the corresponding logits z' from the buffer $\mathcal{M}^k$. To address forgetting, the regularization term considers the Euclidean distance between the sampled output logits and the current model's output logits on buffer data. The gradient g is then refined using this regularization term to minimize the discrepancy between the current and past output logits, thereby mitigating forgetting. The following steps are the same as in the main text.

Algorithm 6 combines with A-GEM, applying gradient projection twice. First, the client computes the gradient g_c with respect to the new data from $\mathcal{D}_t^k$. After replaying previous samples (x', y') stored in the local buffer $\mathcal{M}^k$, the client computes the gradient g_b with respect to this buffered data. If these gradients differ significantly in terms of their direction, the client projects g_c onto g_b to remove interference.