SacFL: Self-Adaptive Federated Continual Learning for Resource-Constrained End Devices

Zhengyi Zhong, Weidong Bao, Ji Wang, Jianguo Chen, Lingjuan Lyu, *Senior Member, IEEE*, and Wei Yang Bryan Lim, *Member, IEEE*

***Abstract*—The proliferation of end devices has led to a distributed computing paradigm, wherein on-device machine learning models continuously process diverse data generated by these devices. The dynamic nature of this data, characterized by continuous changes or *data drift*, poses significant challenges for on-device models. To address this issue, continual learning (CL) is proposed, enabling machine learning models to incrementally update their knowledge and mitigate *catastrophic forgetting*. However, the traditional centralized approach to CL is unsuitable for end devices due to privacy and data volume concerns. In this context, federated CL (FCL) emerges as a promising solution, preserving user data locally while enhancing models through collaborative updates. Aiming at the challenges of limited storage resources for CL, poor autonomy in task shift detection, and difficulty in coping with new adversarial tasks in the FCL scenario, we propose a novel FCL framework named self-adaptive federated CL (SacFL). SacFL employs an encoder–decoder architecture to separate task-robust and task-sensitive components, significantly reducing storage demands by retaining lightweight task-sensitive components for resource-constrained end devices. Moreover, SacFL leverages contrastive learning to introduce an autonomous data shift detection mechanism, enabling it to discern whether a new task has emerged and whether it is a benign task. This capability ultimately allows the device to autonomously trigger CL or attack defense strategy without additional information, which is more practical for end devices. Comprehensive experiments conducted on multiple text and image datasets, such as Cifar100 and THUCNews, have validated the effectiveness of SacFL in both class-incremental and domain-incremental scenarios. Furthermore, a demo system has been developed to verify its practicality.**

***Index Terms*—Adversarial attack, data shift, federated continual learning (CL), self-adaptive ability.**

I. Introduction

IN RECENT years, the rapid development of end devices has given rise to a distributed intelligent computing paradigm. Within this framework, these devices generate vast amounts of data, including images, text, and audio. Over time, the collected data undergoes continuous changes, a phenomenon known as *data drift*. Training a subsequent task with a model previously trained on a different task results in a significant decline in performance on the original task. This phenomenon is known as *catastrophic forgetting* [1]. One of the primary challenges in training machine learning models is to enhance their capacity for continual learning (CL) or to mitigate the rate of forgetting.

The predominant approach in CL primarily focuses on centralized scenarios [2], where user data generated by end devices is transmitted to a central node for training. However, this approach has become increasingly unsuitable for portable devices. On the one hand, user data is often highly privacy-sensitive, and directly transferring this data to remote servers poses a significant threat to user privacy [3], [4]. On the other hand, the effectiveness of models relies on extensive datasets, but the data volume available on individual end devices is inadequate to fully support the training of robust models. Therefore, in the context of distributed end devices, it is crucial to address how to enable multiple end devices to collaboratively learn continually while ensuring the privacy of client data. Federated learning (FL) [5] has emerged as a promising solution to these challenges. FL uploads model updates to remote servers while preserving users' data locally, thereby enhancing the learning process of models in a distributed manner. Building upon this premise, our study aims to explore CL methods based on FL.

Unlike the centralized approach, federated CL (FCL) requires each end device to perform CL, which introduces three distinct challenges as follows.

1) *C1:* Using conventional CL methods requires retaining entire or major segments of past models or preserving a large amount of historical data, imposing considerable storage demands on end devices. However, the inherent hardware limitations result in scarce storage resources, leading to a significant storage burden on resource-constrained end devices in FL.
2) *C2:* Conventional CL methods typically require external intervention to notify the model of task changes or data drift, lacking inherent mechanisms to detect data drift and adaptively adjust the learning process. This is impractical in distributed end device scenarios, where numerous autonomous devices, such as surveillance

Received 28 May 2024; revised 23 December 2024 and 1 March 2025; accepted 26 April 2025. Date of publication 16 May 2025; date of current version 4 September 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62372486, in part by the Pearl River Talent Plan under Grant 2023QN10X579, and in part by the Natural Science Foundation of Guangdong Province under Grant 2023A1515011179. *(Corresponding author: Ji Wang.)*

Zhengyi Zhong, Weidong Bao, and Ji Wang are with the Laboratory for Big Data and Decision, National University of Defense Technology, Changsha 410073, China (e-mail: zhongzhengyi20@nudt.edu.cn; wdbao@nudt.edu.cn; wangji@nudt.edu.cn).

Jianguo Chen is with the School of Software Engineering, Sun Yat-sen University, Guangzhou 510275, China (e-mail: chenjg33@mail.sysu.edu.cn).

Lingjuan Lyu is with Sony AI, Shinagawa 141-8610, Japan (e-mail: lingjuanlvsmile@gmail.com).

Wei Yang Bryan Lim is with the College of Computing and Data Science, Nanyang Technological University, Singapore 639798 (e-mail: bryan.limwy@ntu.edu.sg).

Digital Object Identifier 10.1109/TNNLS.2025.3565827

cameras, operate without external intervention, making conventional CL methods unsuitable.

3) *C3:* Conventional CL methods typically assume that new data is benign. However, in the context of FL, a distributed environment where data on end devices is uncontrollable, it is difficult to prevent malicious clients from introducing adversarial data during the CL process, which can potentially disrupt the global historical knowledge. Current methods cannot continuously monitor adversarial data and defend against such attacks.

To address the above challenges, we design an encoder–decoder architecture that splits the model into a task-robust encoder and a lightweight task-sensitive decoder based on the variation of tasks. Only the decoder is preserved for historical tasks, while the encoder model is shared among tasks and clients. This approach not only facilitates knowledge transfer both in temporal and spatial dimensions but also effectively alleviates the resource burden to end devices. Meanwhile, inspired by contrastive learning, we compare the distances between the encoders before and after updates to determine if data drift occurs. If the distance exceeds a certain threshold, it indicates data drift, triggering the CL mechanism and allowing end devices to update knowledge in a self-adaptive manner. These approaches avoid the need for extra information (e.g., task ID and data label) and facilitate federated continual learning with self-adaptive ability. Furthermore, once task changes are monitored, we further consider whether the new tasks are benign or not. We propose adversarial task monitoring and defense methods, enabling clients to autonomously assess whether a new task is adversarial and take corresponding defense measures to mitigate the impact of the attack. This approach enhances the adaptability of clients in FCL under adversarial environments.

In summary, the contributions are as follows.

1) Breaking the conventional assumption of centralized CL by proposing an FCL method called self-adaptive federated CL (SacFL). This method effectively integrates knowledge from resource-constrained devices while simultaneously reducing the resource requirements of CL.
2) We introduce a data shift detection method that enables end devices to autonomously trigger the CL mechanism without relying on extra information or sharing data with the server. This innovation significantly enhances the self-adaptive capability of model training on end devices while safeguarding privacy.
3) To address the potential issue of encountering new adversarial data during the CL, an adversarial task detection method and defense strategy are proposed, enhancing the adaptability of SacFL in adversarial environments.
4) We validate the effectiveness of SacFL using multiple image and text datasets, including FashionMNIST, Cifar10, Cifar100, and THUCNews. Evaluations are performed in both class-incremental and domain-incremental scenarios. In addition, we conduct experiments on a demo system, further confirming its superiority.

II. Related Work

A. Continual Learning

Current CL methods can be divided into three main categories: regularization-based approach, replay-based approach, and architecture-based approach [1].

The regularization-based approach aims to balance the model performance between new and old tasks by adding regularization terms during the training process of new tasks, thus preventing catastrophic forgetting. Specifically, regularization can be applied at both the parameter and function level. At the parameter level, the importance of model parameters is computed to identify the parameters that contribute significantly to the computation results. Penalty regularization terms are then added to these parameters, allowing them to retain knowledge from old tasks [6]. In addition, freezing certain important parameters or reducing their learning rate can be regarded as variants of this regularization method. At the function level, knowledge distillation [7] is commonly used to preserve old knowledge [8]. When complete data for the old tasks are not available, inference can be performed using incremental data, additional unlabeled data, or generated data [9]. Furthermore, when only partial data for previous tasks are accessible, data replay and knowledge distillation can be combined to enhance performance [2].

The replay-based approach has three primary subdirections: experience replay, generative replay, and feature replay [10], [11]. Experience replay involves constructing a replay buffer to store a small amount of historical data, which is then replayed during the training of subsequent tasks to enhance the model's learning ability [12]. In addition to experience replay, generative replay involves generating data using generative models. Instead of replaying old samples, generated data is used to retain memory throughout the CL process [13], [14]. Feature replay, on the other hand, replays the features of old data by utilizing feature extractors [15], [16].

The above methods are based on parameter sharing between different tasks. In contrast, the architecture-based approach takes a different approach by implementing separate model structures for different tasks at the architectural level, achieving parameter isolation between tasks to avoid catastrophic forgetting. Typical methods include parameter allocation, model decomposition, and modular networks. Parameter allocation involves freezing key parameters for each task using masks, while the remaining parameters are used for training subsequent tasks [17], [18]. Model decomposition decomposes the model into task-sharing and task-specific components, where the task-specific model expands as the number of tasks increases [19]. On the other hand, modular networks establish a subnetwork for each incremental task; however, this may incur significant memory overhead [20].

Currently, the majority of CL methods are developed under the assumption that new data is reliable, and research on the robustness of CL is very limited [21]. Reference [21] is the first to investigate the vulnerability of CL models to adversarial attacks. It employs a replay-based approach, enhancing the robustness of CL by training on boundary samples selected from both old and new tasks.

Reference [22] enhances resistance to adversarial attacks by training the model on robust features derived from the original data. However, these methods are considered preemptive defenses. This article focuses on remedial measures, specifically how to identify the adversarial new samples during the CL and how to mitigate harms.

B. Federated Learning

FL was proposed by Google in 2016 [5] as a way to transfer model parameters instead of data, reducing the privacy leakage risk in traditional cloud computing. FL can be categorized into three types: horizontal FL, vertical FL [3], and transfer FL [4], [23]. Horizontal FL is currently a research hotspot and focuses on several areas.

1) *Personalization:* Under the FL framework, clients' personalized demands can be categorized into data heterogeneity, system heterogeneity, and task heterogeneity [24], [25]. Techniques used in this area include adding user context [26], meta-learning [27], transfer learning [28], knowledge distillation [29], and base+personalization layers [30].
2) *Federated Mechanism:* The naive algorithm of FL is FedAvg [5], yet it often produces biased models in distributed computing. Therefore, researchers have proposed improvements to aggregation algorithms, such as FedBCD [31], SAFL [32], FedProx [33], and FedMA [34], taking into account factors like client fairness and heterogeneity. In addition to single-layer centralized aggregation, there are also approaches targeting multilayer learning architectures, such as HierFAVG [35], HFEL [36], FLEE [37], and ACFL [38].
3) *Communication:* Communication is an important concern in the field of FL [39], as the transmission of gradients or model parameters between clients and servers is often done wirelessly and can be highly unstable [26]. Gradient compression [40] is a commonly used method to solve this problem.

C. Federated CL

In recent years, the issue of catastrophic forgetting in clients within the FL framework has increasingly attracted the attention of researchers [41]. Some scholars have proposed combining the concepts of FL and CL to develop an FCL framework [42]. Yang et al. [42] systematically review the two scenarios—synchronous and asynchronous—that exist in FCL, and analyze the causes of catastrophic forgetting from both spatial and temporal dimensions. This work further clarifies the differences between FCL and traditional CL. For class-incremental problems, Dong et al. [43] proposed a novel global-local forgetting compensation model, GLFC, which weakens catastrophic forgetting as much as possible from both global and local perspectives, ultimately enabling FL to train a globally incremental model. Qi et al. [44] proposed the FedCIL framework, which combines generative methods to use an ACGAN generator to replay synthetic data from previous distributions, thus alleviating catastrophic forgetting. Zhang et al. [45] presented TARGET to remember historical experience via knowledge distillation in class-incremental scenarios. For domain-incremental problems, Li et al. [46] selected cached samples based on the importance of local samples and their relevance to the global dataset, using sample replay to overcome catastrophic forgetting. Huang et al. [47] proposed a federated cross correlation and CL method. To address heterogeneity issues, this method utilizes unlabeled public data for communication and constructs cross correlation matrices to learn generalizable representations under domain shift. At the same time, for catastrophic forgetting, knowledge distillation is used in local updates to provide interdomain and intradomain knowledge effectively without leaking participants' privacy. In addition, some work can be applied to both class increment and domain increment scenarios. Yoon et al. [48] proposed a new FCL framework called FedWeIT. This framework decomposes the local model parameters of each client into dense base parameters and sparse task-adaptive parameters to enable more efficient communication. Jiang et al. [49] focuses on mitigating catastrophic forgetting in global models and proposes a method called federated orthogonal training (FOT) to ensure orthogonal relationships between tasks. Jiang et al. [50] proposed an FL architecture called fed-speech for the federated multispeaker TTS system. This architecture uses progressive pruning masks to separate parameters to preserve speaker characteristics while applying selective masks to effectively reuse knowledge within tasks. Ma et al. [51] presented the CFeD method based on knowledge distillation technology, which extracts old knowledge from the surrogate dataset through the construction of pseudo-labels and knowledge distillation. In addition, some scholars have investigated client drift caused by the nonindependent and identical distribution between clients during FCL [52].

Summary: Our work differs from previous research in the following aspects.

1) SacFL can automatically monitor changes in data and trigger CL mechanisms without requiring extra information.
2) In addition to identifying new tasks, SacFL can also automatically discern whether a task is adversarial and activate defense mechanisms. This capability has not been considered in other works yet.
3) Different from methods like knowledge distillation, SacFL only requires storing a lightweight task-sensitive decoder, effectively reducing storage overhead on end devices.

III. Proposed Method: SacFL

A. Motivation

During the CL process, as data shifts, the last several layers of deep models (e.g., fully connected layers) change significantly, whereas the preceding layers exhibit minimal variation. Using the FashionMNIST dataset as an example, we construct a LeNet neural network comprising convolutional layers, activation layers, max pooling layers, and fully connected layers. Both convolutional and fully connected layers contain two types of parameters: weights and biases. In the context of CL, we divide the ten classes of data into five

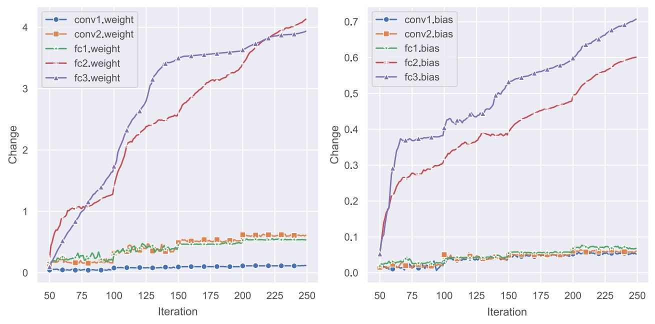


Fig. 1. Changes of parameters in different model layers during the training process. It is worth noting that each task is trained for 50 iterations, and there is no need to calculate the changes in model parameters for the zeroth task. Therefore, the abscissa in the figure starts from 50. The vertical axis represents the difference between specific layer parameters and the corresponding layer parameters after training the zeroth task.

tasks, each task comprising two classes: {0,1}, {2,3}, {4,5}, {6,7}, and {8,9}. Each task is trained for 100 iterations, with the initialization model for each subsequent task derived from the previous one. By observing the parameter changes between consecutive tasks, we can discern the impact of task transitions on the model. In our experiment, we project multidimensional model parameters onto 2-D graphs and use the Euclidean distance between these parameter graphs to represent changes in the model layers. The resulting curve graphs (see Fig. 1) illustrate the changes in weights (left) and biases (right). From these graphs, we can see that the weight and bias changes of the final fully connected layer are the most pronounced as tasks shift.

B. Framework and Pipeline

1) Framework: The framework of SacFL is depicted in Fig. 2. Based on the sensitivity of model parameters to task changes, we divide the on-device model M into a task-robust encoder E and a task-sensitive decoder D, i.e., $M = E \circ D$. The parameters of the encoder demonstrate relative stability across diverse tasks, while the decoder shows high variability in response to task-specific dynamics. SacFL constructs an encoder pool, a decoder pool, and a proxy history data pool on the server. The encoder pool stores global encoders for history tasks. In subsequent iterations, these history encoders are incorporated into aggregations to release catastrophic forgetting. The decoder pool stores decoders for clients' history tasks, and the proxy history data pool stores client history data collected from public sources. These two pools facilitate the monitoring of adversarial tasks. All three pools evolve as the number of tasks increases. In addition, SacFL also builds a small decoder pool for each client to store history task decoders, enabling rapid local access for computation.

2) Pipeline: When no task changes occur, similar to traditional FL, clients train the encoder and decoder using cross-entropy loss. A key difference in SacFL is that the client monitors local data drift by tracking changes in the encoder's output after one local training epoch in each iteration. Once data drift is detected, the local task is considered to have changed, and the decoder from the previous task is pushed to the local decoder pool to store history knowledge. Simultaneously, the trained encoder and decoder are sent to the encoder pool and decoder pool on the server. The server's history decoder pool and proxy history data are used to determine whether the new task is an adversarial task. If the new task is identified as adversarial, the attack defense strategy is implemented locally, and a robust Krum [53] aggregation method is applied at the server to mitigate the attack's impact until a new task is detected. It is important to note that the decoder for adversarial tasks is not stored in the decoder pool. When a task changes, only the encoder is transferred between tasks, and the corresponding decoder needs to be reinitialized at the beginning of each new task. If the user has an inference request, the relevant history decoder is retrieved from the local decoder pool and combined with the current encoder to perform computation, effectively preventing catastrophic forgetting.

The proposed method offers several advantages as follows.

1) The decoder typically consists of the final few layers or even a single layer. Compared to methods that store most of the history models on end devices, this approach occupies significantly less storage space, leading to substantial improvements in storage efficiency.
2) By dividing the model into task-robust and task-sensitive layers, the task-robust layers are transferred across different tasks, ensuring the sharing of common knowledge. Meanwhile, maintaining a separate decoder for each task preserves task independence, thereby reducing interference between tasks.
3) The design of data drift and adversarial task detection methods enables the timely detection of task changes and self-adaptive defense against adversarial attacks. These methods enhance the client's self-adaptive CL.

C. Training Process

Assuming there are K clients, i.e., end devices, in the FL framework. Each client faces T CL tasks, which can be represented as $\{0, \dots, t, \dots, T\}$ with I_t federated iterations for task t. The total number of iterations is $\mathbb{I} = \sum_{t=1}^{T} I_t$. The client models are denoted by M, and the set of all client models is $\{M_1, M_2, \dots, M_K\}$.

Generally speaking, the FL process can be divided into four stages: server distribution, client local training, client upload, and server aggregation. Here, we will mainly focus on local training and server aggregation. During the client local training stage, the number of local epochs is N in each round. When client k faces task t, the trained model M_k^t is obtained. Assuming the learning rate of the client is η, the client's local training process can be represented as follows:

$$M_k^t(i,n) = M_k^t(i,n-1) - \eta \nabla F_k^t\left(M_k^t(i,n-1)\right), \quad n = 1,\dots,N \tag{1}$$

where $M_k^t(i,n)$ represents the model obtained from the nth local epoch of client k in the ith iteration of task t. $F_k^t(M_k^t(i,n-1))$ denotes the loss function of the model $M_k^t(i,n-1)$ when client k faces task t. Before the client starts local training, the

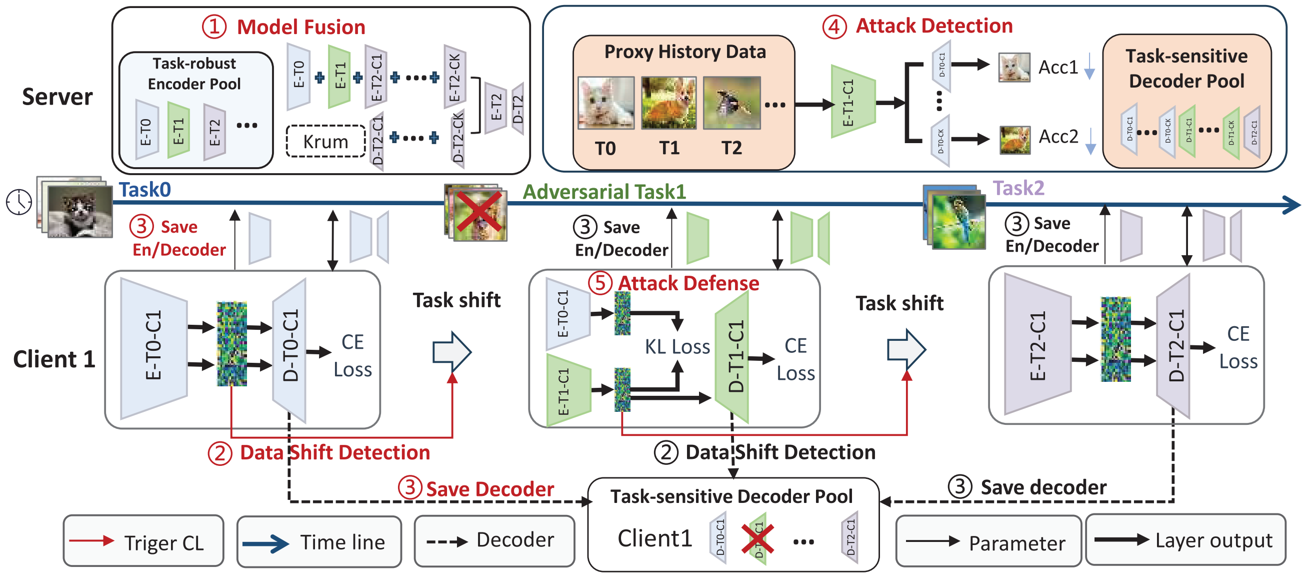


Fig. 2. Framework of SacFL. When no task change occurs, the client trains the encoder and decoder using the classical FL approach, with the exception of performing data drift detection during each iteration. If data drift is detected, the decoder from the previous task is pushed to the local decoder pool. At the same time, the updated encoder and decoder are uploaded to the server to determine whether the new task is adversarial. If an adversarial task is detected, local attack defense mechanisms and Krum aggregation are activated to mitigate the impact of the attack, continuing until the next task is identified.

model parameters obtained from the server side are $M_k^t(i,0)$, and they can be represented as

$$M_k^t(i,0) = E_k^t(i,0) \circ D_k^t(i,0) \tag{2}$$

where

$$E_k^t(i,0) = \frac{\sum_{j=0}^{t-1} E_\cdot^j(I_j,N) + E_\cdot^t(i,0)}{t+1} \tag{3}$$

$$E_\cdot^t(i,0) = \sum_{k=1}^{K} \frac{\mathrm{DS}_k^t}{\sum_{k=1}^{K}\mathrm{DS}_k^t} E_k^t(i-1,N) \tag{4}$$

$$D_k^t(i,0) = D_\cdot^t(i,0) = \sum_{k=1}^{K} \frac{\mathrm{DS}_k^t}{\sum_{k=1}^{K}\mathrm{DS}_k^t} D_k^t(i-1,N). \tag{5}$$

$E_k^t(i,0)$ undergoes a two-stage fusion. The first stage is spatial fusion [refer to (4)]. In (4), $E_\cdot^t(i,0)$ is the globally aggregated encoder obtained after $i-1$ iterations at current task t, which is the weighted sum of $E_k^t(i-1,N)$. DS_k^t is the data size of client k during task t. The second stage is temporal fusion [refer to (3)]. In (3), $E_\cdot^{t-1}(I_{t-1},N)$ is the globally aggregated encoder after I_{t-1} iterations of task $t-1$ which is stored in Task-robust Pool. Note that when clients are facing the first task and there are no previous tasks, the training process of encoder is similar to traditional FL steps. Equation (5) illustrates the training process of decoders. When clients encounter a new task, they reinitialize the decoders and then update them in a regular FL manner. After one specific task training is completed, its corresponding lightweight decoder is stored in task-sensitive pools. In the above CL process, the shared knowledge contained in different tasks is inherited between generations of encoders. Only one encoder needs to be stored in the clients to inherit the common knowledge of historical tasks, while a memory-efficient branch is dedicated to storing task-sensitive knowledge. This approach significantly reduces end devices' storage requirements and promotes long-term CL.

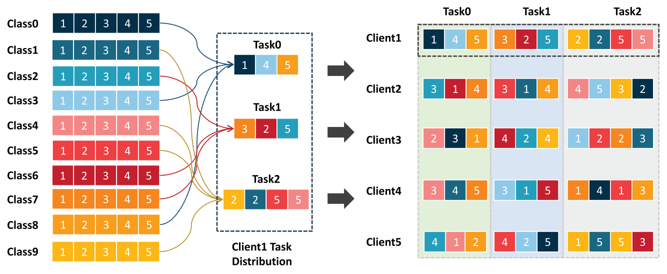
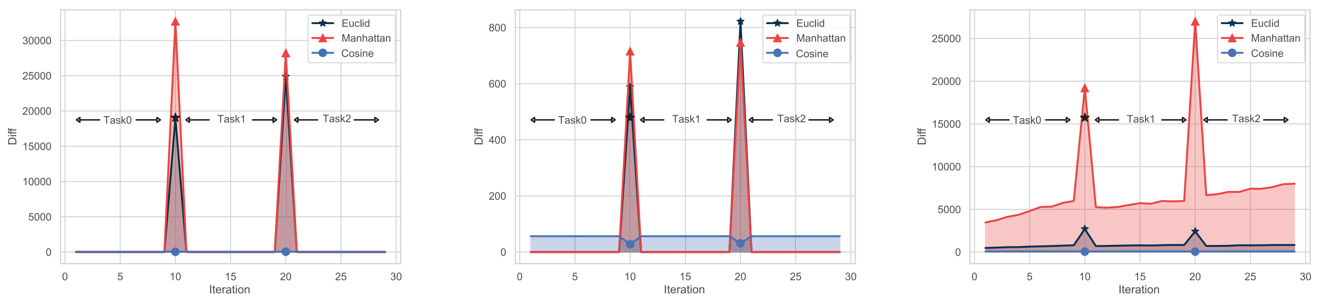


Fig. 3. Class-incremental data setting. Taking five clients as an example, the data of each class is divided into five parts. Through random selection, Client 1 extracts the first, fourth, and fifth parts from the data labeled 0, 3, and 8 as the data for Task 0. Subsequent tasks are generated in the same manner.

It is worth noting that in the process of CL, the structure of $E_\cdot^t(\cdot,\cdot)$ and $E_\cdot^{t-1}(\cdot,\cdot)$ remains the same, but the structure of $D_\cdot^t(\cdot,\cdot)$ and $D_\cdot^{t-1}(\cdot,\cdot)$ does not always remain consistent. For example, in class-incremental tasks, when the model encounters more classes, the branch structure of the model will automatically expand to adapt to the new task, resulting in a significant change in $D_\cdot^t(\cdot,\cdot)$ structure.

D. Data Drift Detection

The traditional method for data drift detection relies on data comparison or performance observation. However, these methods require a considerable amount of memory to store historical data or labeled data, which is not friendly for resource-constrained end devices. Meanwhile, in the context of SacFL, the method proposed in Section III-C is a model-based CL technique that does not store historical data; the client only retains data for the current task. Therefore, inspired by contrastive learning [54], we propose a memory-efficient and label-free data drift detection method. Data drift can



(a) (b) (c)

Fig. 4. Change of the feature values extracted by the encoder. (a) Fa-MNIST, task number = 3. (b) Cifar10, task number = 3. (c) THUCNews, task number = 3.

be detected by comparing the encoders' outputs before and after local learning on clients. The specific approach is as follows: after the server distributes the aggregated encoder to clients, each client performs one round of local training using its local data. To measure the difference between the encoders before and after local training, a certain number of current task data are picked and input into the above two models. If the change value exceeds a certain threshold, it indicates significant differences in the features extracted by the two models from the same data. We can then conclude that substantial model changes and data drift have occurred on the client.

In this process, it is important to note that we use the difference between the output features of encoders to detect data shifts. Since the encoder is less sensitive to data alterations compared to the decoder, data drift is only identified when the encoder's output undergoes substantial changes, preventing misjudgments and improving the accuracy of data shift detection. Furthermore, experiments reveal that compared to commonly used Euclidean distance and Cosine distance, the Manhattan distance is more sensitive to variations in the encoder's output (as shown in Section V-B). From Fig. (4), we can see that the variations of Euclidean distance and Cosine distance are less than the Manhattan distance when the task shifts. However, when the task does not change, the value of the Manhattan distance remains nearly unchanged. Therefore, we employ the Manhattan distance to detect data drift. The calculation formula is as follows:

$$\mathrm{Diff} = \mathrm{Manhattan}\left(E_k^t(i,1)\left(\mathrm{DA}_k^t\right), E_k^t(i,0)\left(DA_k^t\right)\right) \tag{6}$$

where $E_k^t(i,1)$ represents the encoder parameters of the kth client after locally training one epoch during ith federated iteration when facing task t. DA_k^t refers to the data for task t of client k, and $E_k^t(i,1)(\mathrm{DA}_k^t)$ represents the data features extracted by inputting DA_k^t into the encoder $E_k^t(i,1)$. $E_k^t(i,0)$ is the received encoder of client k at the beginning of iteration i when facing task t. Similarly, $E_k^t(i,0)(\mathrm{DA}_k^t)$ denotes the extracted feature of $E_k^t(i,0)$ by inputting DA_k^t. This method is effective not only when the new data is benign, but also demonstrates its efficacy in adversarial tasks, as validated in Section V-B. It can accurately identify adversarial data as a new task. Through the data detection mechanism, end devices can automatically detect data changes and trigger CL, greatly enhancing the clients' self-adaptive capabilities.

E. Adversarial Attack Defense

In the process of CL, new tasks may involve adversarial examples aimed at attacking the model. Therefore, when a new task arises, it should be assessed first. Only when the new task's samples are benign should the CL mechanism be activated. If the new task consists of adversarial data, appropriate defense measures are needed to mitigate the impact on historical knowledge. Accordingly, we propose methods for adversarial task detection and adversarial attack defense.

1) Adversarial Task Detection: In SacFL, we construct a decoder pool for history tasks and a proxy history data pool on the server for adversarial task monitoring. Suppose client z detects a switch from task $j-1$ to task j using a data drift detection mechanism. The updated encoder $E_z^j(0,1)$ is uploaded to the server. Then, the updated encoder is combined with the decoders from the decoder pool $P_D = \{D_k^t(I_t,N) | k \in [0,K], t \in [0,j)\}$, respectively. After that, the corresponding proxy history data is fed into the model, generating the following outputs:

$$\mathrm{Opt}_k^t = \left(E_z^j(0,1)^\circ D_k^t(I_t,N)\right)\left(x_k^t, y_k^t\right) \quad \forall k \in [0,K] \quad \forall t \in [0,j-1]. \tag{7}$$

Based on the above outputs, the accuracy on all clients $k \in [0,K]$ and the corresponding historical tasks $t \in [0,j-1]$ is obtained, from which the performance degradation rate of the historical task caused by $E_z^j(0,1)$ is calculated

$$\mathrm{Degrade}_z^j = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{j}\sum_{t=0}^{j-1}\frac{\mathrm{Acc}_k^t - \widetilde{\mathrm{Acc}}_k^t}{\mathrm{Acc}_k^t}\right). \tag{8}$$

Acc_k^t represents the original accuracy of client k on task t. When $\mathrm{Degrade}_z^j$ exceeds a certain threshold, we consider the task to be adversarial. This is because, the encoder's parameter changes a little, and the historical decoder is used for testing, which, in principle, should not cause a significant degradation in performance on historical tasks. If a significant performance drop in the historical task still occurs, it indicates that the new task is adversarial, directly leading to a substantial change in the encoder's parameters.

2) Adversarial Attack Defense: To effectively defend against the above-mentioned attacks, we constrain the changes in the encoder during the training of adversarial tasks. Suppose client z detects task j as an adversarial task, while task $j-1$ is a benign task. In this case, the KL divergence between the output of $E_{\cdot}^{j-1}(I_{j-1}, E)$ and $E_z^j(i, e)$ is computed. By minimizing this value, the degree of performance degradation can be reduced. The formula for this calculation is as follows:

$$\mathcal{F}_{\text{ebd}} = \text{KL}\left(E_{\cdot}^{j-1}\left(I_{j-1}, E\right)\left(x_z^j, y_z^j\right), E_z^j\left(i, e\right)\left(x_z^j, y_z^j\right)\right). \quad (9)$$

At the same time, the cross-entropy loss should also be considered

$$\mathcal{F}_{\text{ce}} = \text{CE}\left(y_z^j, M_z^j\left(i, e\right)\left(x_z^j, y_z^j\right)\right). \quad (10)$$

Finally, we get the local training loss

$$\mathcal{F}_k^t = \alpha\mathcal{F}_{\text{ebd}} + (1-\alpha)\mathcal{F}_{\text{ce}}. \quad (11)$$

In addition, we also employ a more robust aggregation method, Krum [53], on the server to defend against adversarial attacks.

F. Algorithm

To elucidate the method described above, we provide an algorithmic explanation in Algorithm 1. The algorithm's inputs include the number of clients K, the total number of FL iterations $\mathbb{I}$, the number of local training rounds N, the data for each client (x_k^t, y_k^t), the learning rate η, the encoder pool P_E and decoder pool P_D on the server, and proxy history data pool P_{pd}. The final output is the global encoder and task-sensitive decoders.

Initially, the server initializes the task ID and $M_{\cdot}^t$ (see Algorithm 1, Lines 1 and 2), and then separates the model into encoder and decoder based on the layer changes with task shifts (see Algorithm 1, Line 3). The encoder shows low sensitivity to task variations, while the decoder is highly sensitive. Subsequently, the initialized encoder and decoder are distributed to the clients (see Algorithm 1, Line 5). Upon receiving the model, each client performs N rounds of local training. When the federated iteration count is greater than 1, each client checks for data shift (see Algorithm 1, Line 12) after one epoch of local training. If a task change is detected, the $E_k^t(i, 1)$ remains unchanged, but the $D_k^t(i, 1)$ is reinitialized, and the task-sensitive decoder pool is updated (Lines 15 and 16). After that, clients will upload $E_k^t(i, 1)$ to the server for further detection to determine whether it is an adversarial task (Lines 17–19). If it is identified as an adversarial task, local defensive training will be conducted using (11) (Lines 20 and 21). After N rounds of local training, the clients upload their encoder $E_k^t(I, N)$ and decoder $D_k^t(i, N)$ to the server (Line 22). At this stage, if data drift occurs on the client side, it is necessary to update both the iteration i of the current task and the task ID t (Lines 23–25). Then, the server selects different strategies to aggregate all encoders and decoders based on whether the clients are under attack. Finally, $E_{\cdot}^t(i+1, 0)$ and $D_{\cdot}^t(i+1, 0)$ are obtained (Lines 27–30) and the encoder pool is updated (Line 31).

Algorithm 1 SacFL

```
    Input: Clients' number $K$, learning rate $\eta$, federated
           round $\mathbb{I}$, clients' model $M_{\cdot}^0$ (composed by Encoder
           $E_{\cdot}^0$ and Decoder $D_{\cdot}^0$), local epoch $N$, client $k$'s
           data $(x_k^t, y_k^t)$, Decoder pool on the server $P_D$,
           Encoder pool $P_E$, Proxy history data pool $P_{pd}$
    Output: Task-robust Encoder, Task-sensitive Decoder
            Pool
 1  Initialize task ID $t = 0$;
 2  Initialize global model $M_{\cdot}^t$ on the server;
 3  Decompose model $M_{\cdot}^t$ into Encoder $E_{\cdot}^t$ and Decoder $D_{\cdot}^t$;
 4  for federated round $i = 1, \dots, \mathbb{I}$ do
 5      Server distribute $E_{\cdot}^t(i)$, $D_{\cdot}^t(i)$ to clients;
 6      // Local Training
 7      for client $k = 1, \dots, K$ do
 8          for local epoch $n = 1, \dots, N$ do
 9              $M_k^t(i, n) \leftarrow M_k^t(i, n-1) - \eta F_k^t((x_k^t, y_k^t),$
10              $M_k^t(i, n-1))$;
11              if $i > 1$ and $n = 1$ then
12                  $SHIFT \leftarrow DataDetection((x_k^t, y_k^t),$
13                  $E_k^t(i, 0), E_k^t(i, 1))$;
14                  if $SHIFT = True$ then
15                      Initialize $D_k^t(i, 1)$ as $D_k^{t+1}(0, 0)$;
16                      Push $D_k^t(i-1, N)$ to Decoder Pools
                        on the client $k$ and the server;
17                      Push $E_k^t(i, 1)$ to the server for attack
                        detection;
18                      $ATTACK \leftarrow AttackDetection(E_k^t$
19                      $(i, 1), P_{ph}, P_D)$;
20                      if $ATTACK = True$ then
21                          Local updating using Eq. (11)
22          Upload $E_k^t(i, N)$ and $D_k^t(i, N)$ to the server;
23      $i \leftarrow i + 1$;
24      if $SHIFT = True$ then
25          $t \leftarrow t + 1$, $i \leftarrow 1$
26      // Server Aggregation
27      if $ATTACK = True$ then
28          $E_{\cdot}^t(i+1, 0)$, $D_{\cdot}^t(i+1, 0)$ $\leftarrow$Aggregation using
            $P_E$ based on Krum;
29      else
30          $E_{\cdot}^t(i+1, 0)$, $D_{\cdot}^t(i+1, 0)$ $\leftarrow$Aggregation using
            $P_E$ based on Eq. (3) and Eq. (5);
31      Update $P_E$ with $E_{\cdot}^t(i+1)$.
```

IV. Theoretical Analysis

In SacFL, when there are no changes in the client, the convergence analysis is similar to that of FedAvg. However, differences arise when clients autonomously switch to different tasks. First, during the aggregation process, it is necessary not only to aggregate the current client models but also to integrate historical task models. Second, the autonomy of task switching among different clients leads to noticeable differences in distributions between clients. To demonstrate the convergence of SacFL in the context of CL, it is essential to establish the convergence of each subtask in this scenario. Therefore, we begin with an analysis of subtask t.

Aiming at the first difference, we regard all historical models as client models that do not participate in training but are solely involved in the aggregation process. It can be derived by following formulas:

$$E_k^t(i,0) = \frac{\sum_{j=0}^{t-1} E_\cdot^j(I_j,N) + E_\cdot^t(i,0)}{t+1} = \sum_{j=0}^{t-1}\frac{1}{t+1}E_\cdot^j(I_j,N) + \sum_{k=1}^{K}\frac{\mathrm{DS}_k^t}{(t+1)\sum_{k=1}^{K}\mathrm{DS}_k^t}E_k^t(i-1,N) \tag{12}$$

where the aggregated weight of historical models is $(1/t+1)$.

Aiming at the second difference, we have following assumptions and definition.

Assumption 1: For task t, 1) all clients participate in training; 2) F_k^t is Z^t-smooth and γ^t-convex; 3) the expected variance of client k's stochastic gradients is bounded by $(\beta_k^t)^2$; and 4) the expected value of the square of client k's stochastic gradients' norm is bounded by $(\rho^t)^2$.

Definition 1: Define ϕ^t as the heterogeneity degree of data shift, which is calculated as follows:

$$\phi^t = \tilde{F}^t - \sum_{j=1}^{K}\frac{\mathrm{DS}_k^t}{\sum_{k=1}^{K}\mathrm{DS}_k^t}\tilde{F}_k^t \tag{13}$$

where $\tilde{F}^t$ and $\tilde{F}_k^t$ are the minimum of F^t and F_k^t, respectively.

If we want to prove the global model on task t is convergent, then the following equation should be satisfied:

$$[F^t(M_\cdot^t(i))] - \tilde{F}^t \leq \text{an upper bound } \mathbb{B} \tag{14}$$

where I is the iteration number of task t and $\tilde{F}^t$ is the optimal loss value. When $\mathbb{B}$ decreases as the number of iterations i increases, it indicates that the global model is progressively approaching the optimal model for task t.

According to Assumption (2), Z^t-smooth function F_k^t possesses the following properties:

$$F_k^t\left(M_\cdot^t(i)\right) \leq F_k^t\left(\tilde{M}_\cdot^t\right) + \left(M_\cdot^t(i) - \tilde{M}_\cdot^t\right)^T \nabla F_k^t\left(\tilde{M}_\cdot^t\right) + \frac{Z^t}{2}\left\|M_\cdot^t(i) - \tilde{M}_\cdot^t\right\|^2 \tag{15}$$

where $\tilde{M}_\cdot^t$ is the parameter that minimizes the loss value, and its gradient is $\nabla F_k^t(\tilde{M}_\cdot^t) = 0$. Therefore, the above equation can be further transformed into

$$\mathbb{E}\left[F_k^t\left(M_\cdot^t(i)\right)\right] - F_k^t\left(\tilde{M}_\cdot^t\right) \leq \frac{Z^t}{2}\mathbb{E}\left\|M_\cdot^t(i) - \tilde{M}_\cdot^t\right\|^2. \tag{16}$$

In the above formula, $(Z^t/2)\mathbb{E}\|M_\cdot^t(i) - \tilde{M}_\cdot^t\|^2$ is $\mathbb{B}$ in (14). Since Z^t is a constant, we only need to prove that $\mathbb{E}\|M_\cdot^t(i) - \tilde{M}_\cdot^t\|^2$ decreases with the number of iterations i increases, in order to achieve global convergence. Based on Assumptions (3)–(5) and definitions, combined with Lemma 1–3 from [55], it can be derived that

$$\mathbb{E}\left\|M_\cdot^t(i) - \tilde{M}_\cdot^t\right\|^2 \leq \frac{\lambda^t}{\zeta^t + i} \tag{17}$$

where $\lambda^t = \max\{(\mu^2 G^t/(\mu\gamma^t)-1), (\zeta^t+1)\mathbb{E}\|M_\cdot^t(1) - \tilde{M}_\cdot^t\|^2\}$, μ is some value larger than $(1/\gamma^t)$, $\zeta^t = \max\{(8Z^t/\gamma^t), N\}$, and $G^t = \sum_{k=1}^{K}\left(DS_k^t/\sum_{k=1}^{K}DS_k^t\right)^2(\beta_k^t)^2 + 6Z^t\phi^t + 8(N-1)^2(\rho^t)^2$.

When $\mu = (2/\gamma^t)$, then,

$$\lambda^t = \max\left\{\frac{\mu^2 G^t}{\mu\gamma^t - 1}, \left(\zeta^t + 1\right)\mathbb{E}\left\|M_\cdot^t(1) - \tilde{M}_\cdot^t\right\|^2\right\} \leq \frac{\mu^2 G^t}{\mu\gamma^t - 1} + \left(\zeta^t + 1\right)\mathbb{E}\left\|M_\cdot^t(1) - \tilde{M}_\cdot^t\right\|^2 = \frac{4G^t}{(\gamma^t)^2} + \left(\zeta^t + 1\right)\mathbb{E}\left\|M_\cdot^t(1) - \tilde{M}_\cdot^t\right\|^2. \tag{18}$$

Combining (16)–(18), we can get

$$\mathbb{E}\left[F_k^t\left(M_\cdot^t(i)\right)\right] - F_k^t\left(\tilde{M}_\cdot^t\right) \leq \frac{Z^t}{\zeta^t + i}\left[\frac{2G^t}{(\gamma^t)^2} + \frac{\left(\zeta^t + 1\right)\mathbb{E}\left\|M_\cdot^t(1) - \tilde{M}_\cdot^t\right\|^2}{2}\right]. \tag{19}$$

From the above equation, it can be observed that for a single task t, as the number of iterations i increases, the loss values of the aggregated global model across various clients gradually decrease and approach the minimum value. Therefore, it can be concluded that SacFL converges for each task t within the framework of CL, leading to overall convergence in the CL process.

Furthermore, $G^t = \sum_{k=1}^{K}\left(DS_k^t/\sum_{k=1}^{K}DS_k^t\right)^2(\beta_k^t)^2 + 6Z^t\phi^t + 8(N-1)^2(\rho^t)^2$, we obtain

$$\mathbb{E}\left[F_k^t\left(M_\cdot^t(i)\right)\right] - F_k^t\left(\tilde{M}_\cdot^t\right) = \frac{12Z^{2t}}{(\zeta^t + i)(\gamma^t)^2}\phi^t + \frac{2Z^t\left(\sum_{k=1}^{K}\left(\frac{DS_k^t}{\sum_{k=1}^{K}DS_k^t}\right)^2\left(\beta_k^t\right)^2 + 8(N-1)^2\left(\rho^t\right)^2\right)}{(\gamma^t)^2\left(\zeta^t + i\right)} + \frac{\left(\zeta^t + 1\right)Z^t\mathbb{E}\left\|M_\cdot^t(1) - \tilde{M}_\cdot^t\right\|^2}{2\left(\zeta^t + i\right)}. \tag{20}$$

From the above equation, it can be seen that as the CL progresses, the tasks autonomously vary among different clients, leading to increased ϕ^t. This enhancement in heterogeneity results in a greater number of iterations required for convergence, thereby slowing down the convergence rate.

V. Experimental Verification

In the experimental section, we mainly focus on answering the following questions.

1) *Q1:* Under the FL framework, is the SacFL effective compared to mainstream CL methods when the client's task changes occur infrequently?
2) *Q2:* In scenarios where the client's task undergoes continuous changes, does SacFL maintain its advantages?
3) *Q3:* Apart from class-incremental learning, does SacFL retain its effectiveness when the clients' data experiences domain-incremental changes?
4) *Q4:* When a new task involves adversarial data, how can clients defend against them?
5) *Q5:* Can SacFL reduce resource consumption on end devices compared to other CL methods?
6) *Q6:* What is the impact of the data drift detection mechanism on model performance?

7) *Q7:* Does SacFL still perform well in the demo system from the real world?

The answers to the above questions correspond to Sections V-C–V-I, respectively. Our code is available at: https://github.com/Zhong-Zhengyi/SacFL-Code

A. Experimental Settings

1) Framework: To answer the above-mentioned questions, we design an FL framework consisting of 50 clients and one server. This framework is tailored for the cross-device scenario in FL, wherein a subset of clients participates in each iteration round. To minimize the consumption of client storage resources during CL, we utilize the last layer of the model as the decoder, while all preceding layers serve as the encoder in our experiments.

2) Datasets: The experimental image datasets encompass FashionMNIST, Cifar10 [56], and Cifar100 [56], featuring 10, 10, and 100 classes, respectively. In addition, the text dataset employed is THUCNews [57], comprising 14 categories of Chinese news data collected from Sina News RSS between 2005 and 2011. To cover cases where the number of classes between tasks is equal (task number =5) and unequal (task number =3), we select ten classes and randomly sample 5000 news from each class. These classes include lottery, stock, education, furnishment, technology, fashion, sports, game, social, and entertainment. Among these, 4000 are designated for training, while the remaining 1000 are reserved for testing.

3) CL Settings: To address the class-incremental problem, referring to the experimental setup of Qi et al. [58], we split the data classes into T parts, corresponding to the total number of tasks. For example, if there are three tasks and ten classes in total, each task comprises 3, 3, and 4 classes, respectively. More specifically, if the label set for the first task of client 1 is {0, 3, 8}, and for the second task, it is 7, 6, 2, thus, the third task comprises classes 9, 1, 5, and 4; in contrast, if there are five tasks, each task comprises two classes. It should be noted that in real-life scenarios, data classes across different clients may intersect. Therefore, to better simulate real-world situations, we randomly sample a specific number of data classes for each client in one task. Meanwhile, an equal amount of data from the same class is randomly distributed among the clients to prevent duplication. The detailed process is illustrated in Fig. 3. This approach ensures coverage of two data distribution scenarios between clients: iid and non-iid. In addition, to address the domain-incremental problem, we opt to introduce Gaussian noise and multiplicative noise to simulate domain-incremental scenarios. Similar to [51] and [48], we evaluate the effectiveness of CL by measuring the model's average testing accuracy on the current task and historical tasks. A lower accuracy indicates more severe catastrophic forgetting.

4) Baselines: The benchmarks consist of two categories: continual-based methods and traditional methods. The continual-based methods include CFeD [51], LwF-Fed [8], EWC-Fed [6], MultiHead-Fed [51], FCIL [43], and FedWeIT [48]. The traditional federated methods mainly include two classic algorithms in the FL field: **FedAvg** [5] and **FedProx** [33].

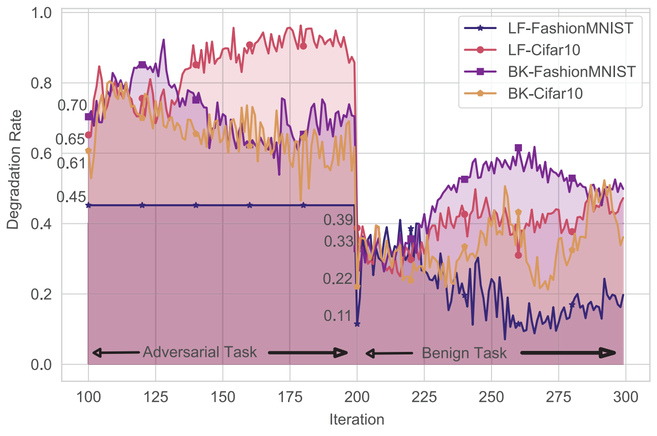


Fig. 5. Average degradation rate of historical tasks under adversarial attacks.

5) Hyper-Parameters: We selected Adam as the optimizer with a batch size of 32, a learning rate of 0.05 for FashionMNIST and THUNews, a learning rate of 0.01 for Cifar10/100, and a local training epoch number of 5. For the number of iterations of a single task, FahionMNIST and Cifar10 are 100, THUNews is 50, and Cifar100 is 50 or 100.

B. Data Drift and Adversarial Task Detection

1) Data Drift Detection: In real-world scenarios, data changes frequently happen without clear indicators. Hence, it's necessary to design an appropriate data drift detection mechanism to identify these changes and trigger the CL process. This section focuses on investigating threshold configurations for activating the CL mechanism. The goal is to equip SacFL with the capability to accurately detect dataset shifts, thus facilitating subsequent CL tasks. Fig. 4 illustrates the changes in encoder features for the FashionMNIST, Cifar10, and THUCNews datasets under the three-task scenario. In our experiment, we conducted ten federated rounds for each task. From the figures, it can be observed that the encoder's extracted features exhibit sharp fluctuations during task transitions, indicating significant changes. Specifically, with a total of three tasks, the Mahattan values of the encoder features for FashionMNIST increase from nearly 0 to over 20000, for Cifar10 from almost 0 to over 600, and for THUCNews from approximately 5000 to over 15000. Consequently, we set the threshold at 20000 for FashionMNIST, 600 for Cifar10, and 15000 for THUCNews. Following the first local training epoch at clients, if the change values of encoders' extracted features surpass the specified thresholds, we identify data shifts.

2) Adversarial Task Detection: During the CL process, when new samples are adversarial, they significantly degrade the performance on history tasks compared to general catastrophic forgetting. Therefore, after detecting data drift, it is necessary to further confirm whether the data is adversarial. This section validates the adversarial task detection mechanism using the FashionMNIST and Cifar10 datasets, under nontargeted attacks (label flipping) and targeted attacks (backdoor attacks). The number of tasks is 5, with task 1 being the adversarial data. By observing the decline of historical

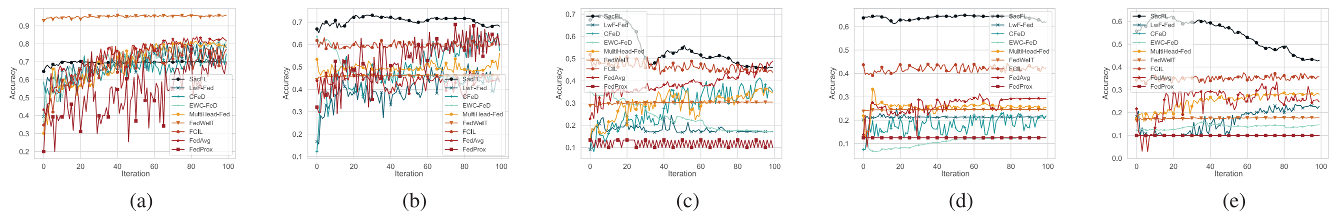


Fig. 6. FashionMNIST, task number =5. (a) Fa-MNIST, task 0. (b) Fa-MNIST, task 1. (c) Fa-MNIST, task 2. (d) Fa-MNIST, task 3. (e) Fa-MNIST, task 4.

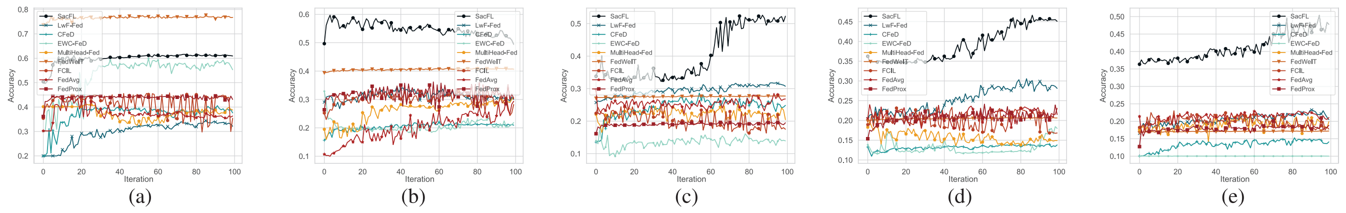


Fig. 7. Cifar10, task number =5. (a) Cifar10, task 0. (b) Cifar10, task 1. (c) Cifar10, task 2. (d) Cifar10, task 3. (e) Cifar10, task 4.

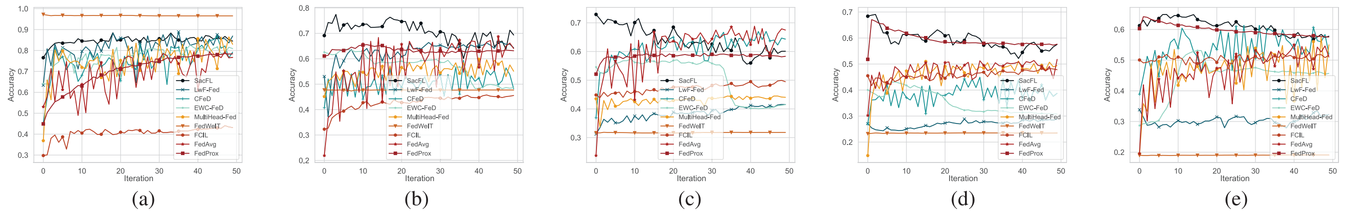


Fig. 8. THUCNews, task number =5. (a) THUCNews, task 0. (b) THUCNews, task 1. (c) THUCNews, task 2. (d) THUCNews, task 3. (e) THUCNews, task 4.

TABLE I

EXPERIMENTAL RESULTS. THE BOLDED ACCURACIES ARE THE OPTIMAL RESULTS, WHILE THE UNDERLINED ONES ARE SUBOPTIMAL RESULTS IN THE SAME SCENARIO. DUE TO SPACE LIMITATIONS, WE ONLY LISTED THE AVERAGE ACCURACY OF THE MODEL ON ALL HISTORICAL DATA AFTER THE TRAINING OF ALL TASKS, WITHOUT LISTING THE RESULTS OF INTERMEDIATE TASKS

		Continual-based Methods							**Traditional Methods**	
		SacFL	**CFeD**	**LwF-Fed**	**EWC-Fed**	**MH-Fed**	**FedWeIT**	**FCIL**	**FedAvg**	**FedProx**
Class	**FM-3**	0.64±2.91e-02	0.1±2.88e-04	0.32±1.81e-02	0.37±2.27e-03	0.48±1.07e-02	0.38±2.23e-03	**0.77**±1.25e-02	0.36±2.40e-03	0.1±2.16e-04
	FM-5	**0.43**±5.09e-02	0.1±3.53e-04	0.22±1.67e-03	0.15±5.18e-04	0.28±1.78e-03	0.18±1.19e-02	0.36±1.91e-03	0.25±1.77e-03	0.1±6.24e-18
	C10-3	0.3±2.45e-02	0.32±3.78e-03	0.33±2.86e-03	0.27±8.33e-04	0.33±1.59e-03	0.25±6.73e-03	**0.38**±3.79e-03	0.33±1.29e-03	0.31±3.99e-04
	C10-5	**0.47**±9.92e-03	0.14±1.78e-03	0.22±2.93e-03	0.1±2.10e-05	0.19±3.05e-03	0.17±6.28e-03	0.18±4.45e-03	0.21±1.27e-03	0.18±4.08e-03
	News-3	**0.68**±1.63e-02	0.57±1.22e-03	0.67±3.06e-03	0.61±2.40e-04	0.65±2.41e-03	0.37±2.36e-03	0.51±1.81e-03	0.65±8.13e-04	0.65±1.28e-03
	News-5	**0.58**±6.73e-02	0.57±1.07e-02	0.32±2.82e-03	0.46±1.27e-03	0.51±3.06e-02	0.19±7.64e-04	0.52±1.78e-03	0.53±6.42e-03	0.57±3.72e-04
	C100-10	**0.32**±2.74e-03	0.04±4.06e-04	0.11±6.57e-04	0.01±2.25e-05	0.06±4.99e-04	0.05±2.52e-03	0.09±1.64e-03	0.08±2.83e-04	0.07±2.49e-04
	C100-15	**0.38**±2.94e-03	0.05±1.23e-03	0.09±8.91e-04	0.01±7.80e-19	0.08±4.51e-04	0.05±3.46e-03	0.10±7.16e-04	0.08±1.57e-03	0.08±3.55e-04
	C100-20	**0.43**±4.11e-03	0.04±3.09e-04	0.05±3.34e-04	0.01±7.80e-19	0.05±5.37e-04	0.03 ±2.24e-03	0.06±1.54e-03	0.05±5.38e-04	0.05±3.77e-04
Domain	**C10**	**0.8**±1.22e-03	0.76±1.62e-03	0.77±1.18e-03	0.76±9.07e-04	0.79±6.20e-04	0.78±3.10e-03	–	0.79±5.20e-04	0.66±9.06e-04

knowledge, we can identify the adversarial task. As shown in Fig. 5, the vertical axis represents the average degradation rate of the model's performance on the proxy historical data. At the beginning of label flipping (iteration =100), the degradation rates for Cifar10 and FashionMNIST are 65% and 45%, respectively, which are higher than the benign tasks (iteration =200) with initial degradation rates of 39% and 11%. When the attack method is a backdoor attack (iteration =100), the initial degradation rates for Cifar10 and FashionMNIST are 61% and 70%, respectively, again exceeding that for benign tasks (iteration =200), which are 22% and 33%. Overall, regardless of the type of attack, the degradation rates at the beginning of attacks (iteration =100) are above 40%, while these initial degradation rates of benign new tasks (iteration =200) are below 40%. Thus, we set 40% as the threshold for detecting adversarial tasks. If the average degradation rate on historical tasks exceeds this threshold, it indicates that the task is adversarial.

C. Simple Class CL

This section focuses on the class-incremental scenario and applies the findings from Section V-B to simple class CL using the FashionMNIST, Cifar10, and THUCNews datasets. In this scenario, the data is relatively stable and undergoes shifts only a few times. Therefore, we consider scenarios with 3 and 5

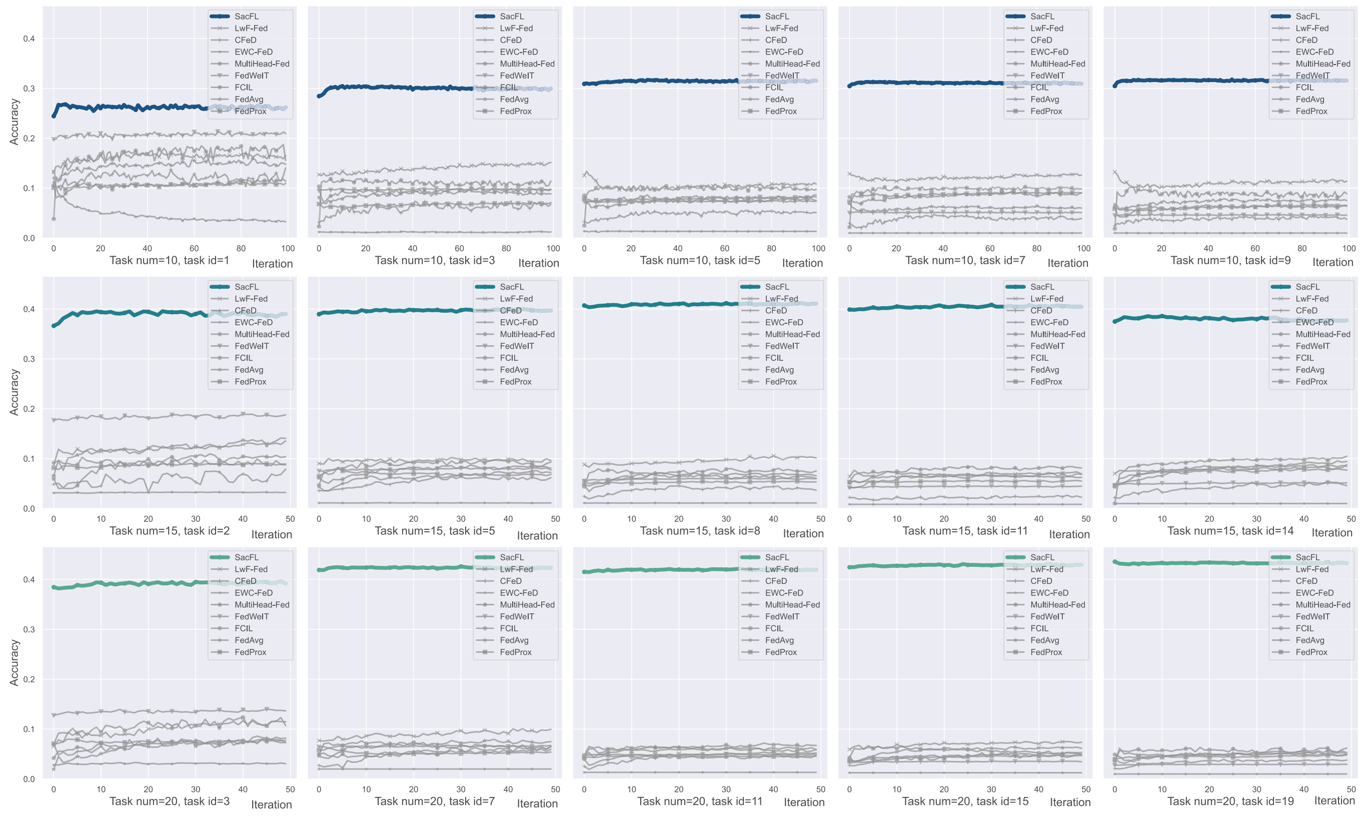


Fig. 9. Comparison of performance under Cifar100.

data shifts, corresponding to 3 and 5 tasks, respectively. In the experimental setup, we endeavored to ensure an equal distribution of the class number included in each task. Since each of the aforementioned datasets comprises 10 classes, with 3 tasks, the distribution is as follows: 3, 3, and 4 classes per task, respectively. When there are five tasks, as 10 is divisible by 5, each task contains 2 classes. The experimental results are listed in Table I and visualized in Figs. 6–8.

In the results figure, the horizontal axis represents the number of iterations for the current task, while the vertical axis indicates the average accuracy of the model on the testing data from both all historical tasks and the current task. The model for the subsequent task is initialized using the parameters from the previous task. From Table I, it can be observed that when the total number of tasks is 3, SacFL holds an optimal or near-optimal position across various datasets, being on par with most methods, yet it does not demonstrate a distinct advantage. However, when the total number of tasks increases to 5, the accuracy of SacFL in tasks 1–4 is significantly higher than that of other methods (see Figs. 6–8), both on the FashionMNIST and Cifar10 datasets. While most methods achieve only 20%–30% accuracy in Cifar10, SacFL attains 50%–60% accuracy. Notably, compared to the scenario with three tasks, the advantages of SacFL become more pronounced as the number of tasks increases to 5. We speculate that as the number of tasks increases, the superiority of SacFL gradually strengthens (verified in Section V-D). The reason behind this is that through monitoring the model layers' changes with tasks, we identify task-sensitive lightweight decoders and directly leverage the historical information they encapsulate. This ensures the integrity of historical task-related knowledge. Moreover, these lightweight task-sensitive decoders notably alleviate storage resource demands compared to storing the entire historical model. However, we also observe that in subsequent tasks, while SacFL maintains a significant advantage, there may be a slight decline. This is because, on individual datasets, when the average forgetting rate for historical tasks exceeds the learning rate for new tasks, the overall accuracy shows a decreasing trend. The reason behind this is that training in subsequent tasks can introduce minor alterations to the encoder, diminishing the coupling between the encoder and decoders from previous tasks. It is not guaranteed to occur. For example, there is a slight decrease in the FashionMNIST and THUCNews datasets, but a weak upward trend in the Cifar10 and Cifar100 datasets.

D. Sequential Class CL

In Section V-C, we focus on the scenario where the data remains relatively stable, namely, simple CL. However, in real-world scenarios, CL is a long-term endeavor, and the variations across merely 3 or 5 tasks are insufficient. It is necessary to validate SacFL in situations with more task variations. Therefore, in this section, we introduce the Cifar100 dataset to construct a larger number of tasks incorporating a wider range of data classes. Our aim is to assess the efficacy of SacFL in handling extensive task variations. Specifically, we test the performance under scenarios involving 10, 15, and 20

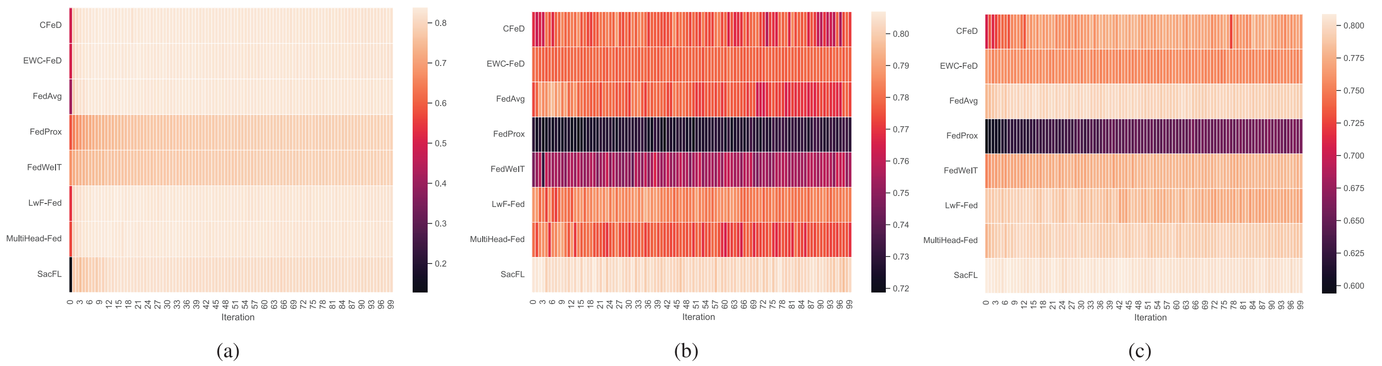


Fig. 10. Cifar10, task number =3, domain-incremental scenario. Each line represents the change in accuracy of a specific algorithm as the number of iterations increases. The lighter the color, the higher the accuracy. (a) Cifar10, task 0. (b) Cifar10, task 1. (c) Cifar10, task 2.

tasks. Due to space limitations, we only present a subset of the experimental results, as shown in Fig. 9.

In Fig. 9, the results are depicted for different numbers of tasks: when there are ten tasks, the outcomes for task 1, task 3, task 5, task 7, and task 9 are displayed; with 15 tasks, the results for task 2, task 5, task 8, task 11, and task 14 are shown; and when there are 20 tasks, the results for task 3, task 7, task 11, task 15, and task 19 are illustrated. It is evident that irrespective of whether the total number of tasks is 10, 15, or 20, traditional methods exhibit minimal effectiveness in sequential tasks, whereas the SacFL approach demonstrates a clear advantage and maintains stable convergence. This reaffirms the superior performance of SacFL in handling sequential tasks.

E. Domain CL

In the experiments in Sections V-C and V-D, validations are carried out under the class-incremental scenario. In addition to class increment, domain increment is also an important setting in CL. In the domain-incremental scenario, the labels of the data remain unchanged, but the data itself undergoes shifts. To simulate this scenario, we introduce Gaussian noise and multiplicative noise to the Cifar10 dataset, thus constructing domain-incremental datasets. Consequently, we obtain three tasks: task 0 for the original dataset, task 1 for Gaussian noise, and task 2 for multiplicative noise. The experimental results are illustrated in Fig. 10.

In Fig. 10, under the original data (task 0), the convergence results and speeds of SacFL are consistent with other methods, achieving an accuracy of 80%. However, upon introducing Gaussian noise to the Cifar10 dataset, all methods exhibit noticeable fluctuations. Except for SacFL, the performance of other methods significantly decreases. Notably, FedProx, which does not employ CL mechanisms, experiences the most significant decline. Furthermore, when the model is further exposed to the multiplicative noise dataset, SacFL's accuracy remains high. Therefore, based on the experimental results in Sections V-C–V-E, we conclude that SacFL performs well in both class-incremental and domain-incremental scenarios in CL.

F. CL Under Adversarial Attack

All the experiments above are under the assumption that new tasks are benign. However, it is inevitable that some clients are maliciously attacked in the real world. Based on the threshold obtained in Section V-B, we can accurately detect adversarial tasks and trigger the adversarial CL mechanism. In this section, we validate our approach using the Fashion-MNIST and Cifar10 datasets in the contexts of untargeted attacks (label flipping) and targeted attacks (backdoor attacks). The experimental setup assumes a class-incremental learning scenario with three tasks, where adversarial data appears in task 1, while tasks 0 and 3 contain benign samples. The final results are summarized in Table II. In Table II, we compare the proposed adversarial CL defense method, SacFL, against commonly used adversarial defense methods in FL (Krum [53], Median [59], and Trimmed_mean [59]) under the adversarial task (ID =1) scenario. The reported values represent the test accuracy of the model across all historical tasks. As shown in the table, SacFL outperforms other methods overall in terms of defense effectiveness. This demonstrates that SacFL is more effective in countering adversarial samples encountered during CL.

TABLE II
PERFORMANCE OF DIFFERENT STRATEGIES AGAINST ATTACKS

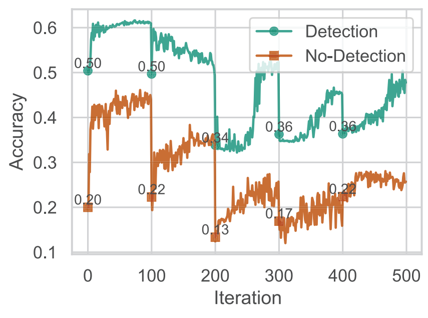
	Label Flipping		Backdoor Attack	
	Cifar10	**F-MNIST**	**Cifar10**	**F-MNIST**
SacFL	**0.48** ±3.35e-02	**0.15** ±4.71e-02	**0.41** ±3.32e-02	**0.38** ±3.81e-02
Krum	0.42 ±1.86e-02	0.11 ±3.43e-02	0.33 ±3.96e-02	0.30 ±5.88e-02
Median	0.43 ±2.74e-02	0.02 ±9.73e-03	0.39 ±4.02e-02	0.14 ±3.52e-02
Trim_m	0.47 ±1.91e-02	0.08 ±4.58e-02	0.37 ±3.21e-02	0.15 ±3.33e-02

TABLE III
COMPUTATION (DENOTED BY C) AND STORAGE (DENOTED BY S) OVERHEAD OF DIFFERENT CL METHODS

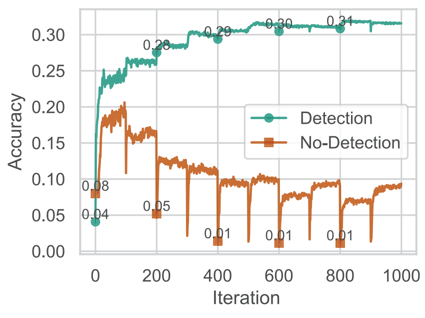
		SacFL	**CFeD**	**LwF-Fed**	**EWC-Fed**	**FCIL**	**FedWeIT**
LeNet	S	**4K**	177K	177K	177K	177K	171K
	C	5.02	5.22	**4.92**	7.18	9.91	19.44
Resnet18	S	**21K**	43.73M	43.73M	43.73M	43.73M	681K
	C	21.79	23.08	**18.75**	27.7	23.46	62.5
TextCNN	S	**5K**	10.51M	10.51M	10.51M	10.51M	301K
	C	**3.42**	4.08	4.34	4.43	6.57	6.23

G. Resource Analysis

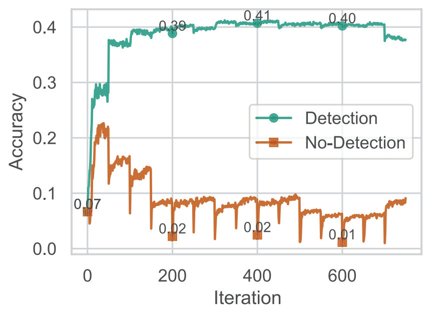
When considering the adaptation to limited resources on end devices, SacFL demonstrates significant advantages in both



(a)



(b)



(c)

Fig. 11. Ablation study of data shift detection component. (a) Cifar100, task number =5. (b) Cifar100, task number =10. (c) Cifar100, task number =15.

computational and storage efficiency compared to other methods, as illustrated in Table III. Especially in the storage aspect, traditional model-based FCL methods typically necessitate storing the entire model to preserve historical knowledge. In contrast, SacFL only maintains a lightweight decoder, thus reducing storage overhead. Taking the ResNet-18 model for Cifar10 as an example, other methods consume 43.73 MB/681 KB, while the lightweight decoder only occupies 0.19 MB, reducing by 99.9%; similarly, reductions of 97.7% for LeNet and 99.9% for TextCNN. Regarding computation resources, Table III displays the average time consumed per federated iteration when the total number of tasks is 3. We can conclude that compared to the average of other methods, SacFL reduces the computing time by 46.22%, 29.92%, and 33.33% for LeNet, ResNet18, and TextCNN, respectively. Therefore, SacFL consumes fewer resources overall and is more suitable for end devices with limited resources. It should be noted that the resource consumption of the Multihead method is not listed in the table since it undergoes significant structural changes in each task, making it incomparable to other methods.

H. Ablation Studies

In this section, we perform ablation validation on the data drift detection component. Due to space constraints, we specifically focus on validation within the class-incremental scenario involving a large number of classes, yielding results as depicted in Fig. 11. It can be observed that in the absence of data drift detection, the model's performance deteriorates with task transitions. However, upon integrating the data drift detection component, the model's performance just experiences only a brief decline after task changes, yet it recovers during subsequent training.

I. Demo System

In addition to validating SacFL in a simulation system, we also develop a distributed demo system, consisting of 5 mini computers NUC with CPU and a central server. The NUCs are equipped with Intel[1] Core[2] i7-10710U processors, 24 GB of RAM, and run on Ubuntu 18.04. The central server contains four NVIDIA GeForce RTX 3090 GPUs, and 128 GB of RAM, and operates on Ubuntu 22.04. All the NUCs are connected through the IEEE 802.11 wireless network. Leveraging the FashionMNIST dataset, we compare the performance of SacFL with that of typical CL methods such as EWC-Fed and the non-CL method FedAvg, as depicted in Fig. 12. In Fig. 12, the test results on all historical tasks for the five NUCs are presented after training. It can be observed that the SacFL model exhibits overwhelming superiority over the other two methods across all clients. Therefore, SacFL maintains its advantage in realistic distributed computing scenarios.

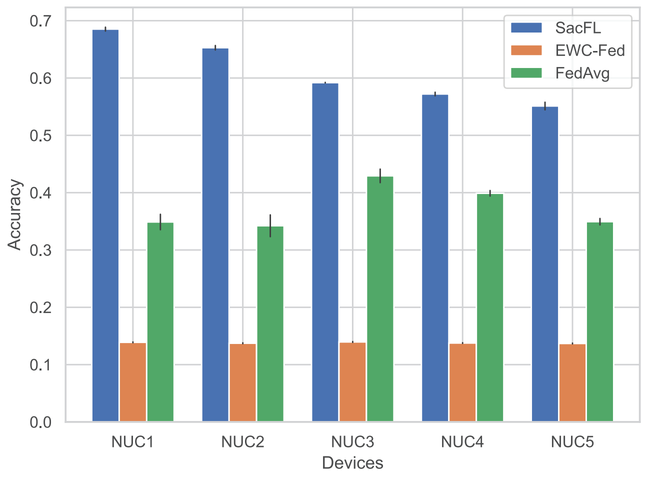


Fig. 12. Performance of SacFL on the demo system.

VI. Conclusion

This article addresses the problem of CL for resource-constrained end devices, proposing an FCL method called SacFL. SacFL identifies that the last few layers are highly sensitive to task variations. Based on this observation, the model is divided into a task-robust encoder and a task-sensitive lightweight decoder. By only storing the lightweight decoders instead of the whole model or historical data on end devices, the overhead of storage and computation resources can be effectively reduced. Moreover, a data shift detection mechanism based on contrastive learning is introduced to detect task changes. It can autonomously identify new tasks and determine whether they are adversarial. For benign tasks, it triggers the CL mechanism, while for adversarial tasks, it activates the

[1]Registered trademark.
[2]Trademarked.

attack-defense strategy. Experimental validations conducted on both image and text datasets yield five key conclusions: 1) SacFL demonstrates advantages over mainstream CL and conventional methods, particularly evident when encountering more frequent changes; 2) SacFL greatly reduces the storage and computing overhead on end devices, achieving a reduction ratio of up to 99.9%, especially in terms of storage resources; 3) beyond class-incremental scenarios, SacFL remains effective in domain-incremental scenarios; 4) in scenarios where the new task is malicious, its effectiveness in mitigating attacks exceeds that of common federated robust aggregation methods; and 5) except for the simulation system, SacFL is also effective in a real demo system, demonstrating its practicality.

References

[1] Y. Ge, Y. Li, S. Ni, J. Zhao, M. Yang, and L. Itti, "CLR: Channel-wise lightweight reprogramming for continual learning," in *Proc. ICCV*, Oct. 2023, pp. 18752–18762.

[2] A. Douillard, M. Cord, C. Ollion, T. Robert, and E. Valle, "PODNet: Pooled outputs distillation for small-tasks incremental learning," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 86–102.

[3] W. Y. B. Lim et al., "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, 3rd Quart., 2020.

[4] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, 2019.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, vol. 54, A. Singh and J. Zhu., Eds., Fort Lauderdale, FL, USA, 2017, pp. 1273–1282.

[6] K. James et al., "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 13, pp. 3521–3526, Mar. 2017.

[7] J. P. Gou, B. S. Yu, S. J. Maybank, and D. C. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, no. 31, pp. 1789–1819, Jul. 2021.

[8] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, Dec. 2017.

[9] P. Dhar, R. V. Singh, K. Peng, Z. Wu, and R. Chellappa, "Learning without memorizing," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 5138–5146.

[10] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," 2023, *arXiv:2302.00487*.

[11] Y. Li, Q. Li, H. Wang, R. Li, W. Zhong, and G. Zhang, "Towards efficient replay in federated incremental learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, vol. 17, Jun. 2024, pp. 12820–12829.

[12] A. Chaudhry et al., "On tiny episodic memories in continual learning," 2019, *arXiv:1902.10486*.

[13] M. Boschini, P. Buzzega, L. Bonicelli, A. Porrello, and S. Calderara, "Continual semi-supervised learning through contrastive interpolation consistency," *Pattern Recognit. Lett.*, vol. 162, pp. 9–14, Aug. 2022.

[14] Y. Xiang, Y. Fu, P. Ji, and H. Huang, "Incremental learning using conditional adversarial networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 6619–6628.

[15] X. Liu et al., "Generative feature replay for class-incremental learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2020, pp. 226–227.

[16] Z. Gong, K. Zhou, X. Zhao, J. Sha, S. Wang, and J.-R. Wen, "Continual pre-training of language models for math problem understanding with syntax-aware memory network," in *Proc. ACL*, Jan. 2022, pp. 5923–5933.

[17] A. Mallya, D. Davis, and S. Lazebnik, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 67–82.

[18] M. Xue, H. Zhang, J. Song, and M. Song, "Meta-attention for ViT-backed continual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, New Orleans, LA, USA, Jun. 2022, pp. 150–159.

[19] N. Mehta, K. J. Liang, V. K. Verma, and L. Carin, "Continual learning using a Bayesian nonparametric dictionary of weight factors," in *Proc. Int. Conf. Artif. Intell. Statist.*, Jan. 2020, pp. 100–108.

[20] R. Jathushan, H. Munawar, H. Salman, K. F. Shahbaz, and S. Ling, "Random path selection for incremental learning," 2019, *arXiv:1906.01120*.

[21] T. Bai, C. Chen, L. Lyu, J. Zhao, and B. Wen, "Towards adversarially robust continual learning," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2023, pp. 1–5.

[22] H. Khan, N. C. Bouaynaya, and G. Rasool, "Adversarially robust continual learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2022, pp. 1–8.

[23] W. Huang et al., "A federated learning for generalization, robustness, fairness: A survey and benchmark," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 12, pp. 9387–9406, Dec. 2024.

[24] Z. Zhong, J. Wang, W. Bao, J. Zhou, X. Zhu, and X. Zhang, "Semi-HFL: Semi-supervised federated learning for heterogeneous devices," *Complex Intell. Syst.*, vol. 9, no. 2, pp. 1995–2017, Apr. 2023.

[25] W. Huang, M. Ye, Z. Shi, and B. Du, "Generalizable heterogeneous federated cross-correlation and instance similarity learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 2, pp. 712–728, Oct. 2023.

[26] P. Kairouz et al., "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, nos. 1–2, pp. 1–210, Jun. 2021.

[27] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning: A meta-learning approach," 2020, *arXiv:2002.07948*.

[28] K. Wang, R. Mathews, C. Kiddon, H. Eichner, F. Beaufays, and D. Ramage, "Federated evaluation of on-device personalization," 2019, *arXiv:1910.10252*.

[29] D. Li and J. Wang, "FedMD: Heterogenous federated learning via model distillation," 2019, *arXiv:1910.03581*.

[30] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," 2019, *arXiv:1912.00818*.

[31] Y. Liu et al., "A communication efficient collaborative learning framework for distributed features," 2019, *arXiv:1912.11187*.

[32] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 4615–4625.

[33] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Mach. Learn. Syst.*, vol. 2, 2020, pp. 429–450.

[34] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," 2020, *arXiv:2002.06440*.

[35] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–6.

[36] S. Luo, X. Chen, Q. Wu, Z. Zhou, and S. Yu, "HFEL: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6535–6548, Oct. 2020.

[37] Z. Zhong, W. Bao, J. Wang, X. Zhu, and X. Zhang, "FLEE: A hierarchical federated learning framework for distributed deep neural network over cloud, edge, and end device," *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 5, pp. 1–24, Oct. 2022.

[38] H. Huang et al., "Active client selection for clustered federated learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 11, pp. 16424–16438, Jul. 2023.

[39] S. M. Shah and V. K. N. Lau, "Model compression for communication efficient federated learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 9, pp. 5937–5951, Sep. 2021.

[40] J. Konečný and P. Richtárik, "Randomized distributed mean estimation: Accuracy vs. communication," *Frontiers Appl. Math. Statist.*, vol. 4, p. 62, Dec. 2018.

[41] H. Yu et al., "Overcoming spatial–temporal catastrophic forgetting for federated class-incremental learning," in *Proc. 32nd ACM Int. Conf. Multimedia*, Oct. 2024, pp. 5280–5288.

[42] X. Yang, H. Yu, X. Gao, H. Wang, J. Zhang, and T. Li, "Federated continual learning via knowledge fusion: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 8, pp. 3832–3850, Feb. 2024.

[43] J. Dong et al., "Federated class-incremental learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2022, pp. 10164–10173.

[44] D. Qi, H. Zhao, and S. Li, "Better generative replay for continual federated learning," 2023, *arXiv:2302.13001*.

[45] J. Zhang, C. Chen, W. Zhuang, and L. Lyu, "TARGET: Federated class-continual learning via exemplar-free distillation," in *Proc. ICCV*, Oct. 2023, pp. 4759–4770.

[46] Y. Li, W. Xu, Y. Qi, H. Wang, R. Li, and S. Guo, "SR-FDIL: Synergistic replay for federated domain-incremental learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 11, pp. 1879–1890, Aug. 2024.

[47] W. Huang, M. Ye, and B. Du, "Learn from others and be yourself in heterogeneous federated learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 10143–10153.

[48] J. Yoon, W. Jeong, G. Lee, E. Yang, and S. J. Hwang, "Federated continual learning with weighted inter-client transfer," in *Proc. 38th Int. Conf. Mach. Learn.*, vol. 139, Jul. 2021, pp. 12073–12086.

[49] Y. F. Bakman, D. N. Yaldiz, Y. H. Ezzeldin, and S. Avestimehr, "Federated orthogonal training: Mitigating global catastrophic forgetting in continual federated learning," 2023, *arXiv:2309.01289*.

[50] Z. Jiang, Y. Ren, M. Lei, and Z. Zhao, "FedSpeech: Federated text-to-speech with continual learning," 2021, *arXiv:2110.07216*.

[51] Y. Ma, Z. Xie, J. Wang, K. Chen, and L. Shou, "Continual federated learning based on knowledge distillation," in *Proc. IJCAI*, Jul. 2022, pp. 2182–2188.

[52] Y. Venkatesha, Y. Kim, H. Park, Y. Li, and P. Panda, "Addressing client drift in federated continual learning with adaptive optimization," 2022, *arXiv:2203.13321*.

[53] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, Dec. 2017, pp. 118–128.

[54] Q. Li, B. He, and D. Song, "Model-contrastive federated learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Sep. 2021, pp. 10708–10717.

[55] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of FedAvg on non-IID data," 2019, *arXiv:1907.02189*.

[56] A. Krizhevsky and H. Geoffrey, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009. [Online]. Available: https://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf

[57] J. Li, M. Sun, and Z. Xian, "A comparison and semi-quantitative analysis of words and character-bigrams as features in Chinese text categorization," in *Proc. ACL*, Jan. 2006, pp. 545–552.

[58] D. Qi, H. Zhao, and S. Li, "Better generative replay for continual federated learning," 2023, *arXiv:2302.13001*.

[59] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. Int. Conf. Mach. Learn.*, Jul. 2018, pp. 5650–5659.

Zhengyi Zhong received the B.S. degree from the College of Systems Engineering, National University of Defense Technology, Changsha, China, in 2020, where she is currently pursuing the Ph.D. degree.

Her research interests include federated learning, continual learning, machine unlearning, and domain adaptation.

Weidong Bao received the Ph.D. degree in information systems from the National University of Defense Technology, Changsha, China, in 1999.

He is currently a Professor with the College of Systems Engineering, National University of Defense Technology. He has authored more than 100 research articles in refereed journals and conference proceedings, such as IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and IEEE INTERNET OF THINGS JOURNAL. His recent research interests include cloud computing, information systems, and complex networks.

Ji Wang received the Ph.D. degree in information systems from the National University of Defense Technology, Changsha, China, in 2019.

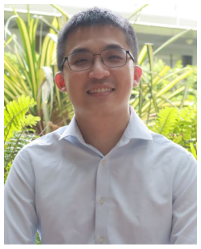
He was a Visiting Ph.D. Student with the University of Illinois at Chicago, Chicago, IL, USA, from March 2017 to September 2018, under the supervision of Prof. Philip S. Yu. He is currently an Associate Professor with the College of Systems Engineering, National University of Defense Technology. He has authored more than 30 research articles in refereed journals and conference proceedings, such as IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, SIGKDD, and AAAI. His research interests include deep learning and edge intelligence.

Jianguo Chen received the Ph.D. degree in computer science and technology from Hunan University, Changsha, China, in 2018.

He is currently an Associate Professor and one of the Hundred Academic Talents with the School of Software Engineering, Sun Yat-sen University (SYSU), Guangzhou, China. He has published more than 70 research papers in international conferences and journals such as IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, and IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS. His major research interests include high-performance artificial intelligence, federated learning, distributed computing, and the application in intelligent transportation and intelligent medicine.

Lingjuan Lyu (Senior Member, IEEE) received the Ph.D. degree from The University of Melbourne, Melbourne, VIC, Australia, in 2018.

She was a Research Fellow (Level B3) with Australian National University, Canberra, ACT, Australia. She is currently a Researcher with Sony AI. Her current research interests include federated learning, trustworthy AI, edge intelligence, and fairness.



Wei Yang Bryan Lim (Member, IEEE) received the Ph.D. degree from Nanyang Technological University (NTU), Singapore, in 2022, under the Alibaba Ph.D. Talent Program and was affiliated with the CityBrain Team, DAMO Academy.

He is currently an Assistant Professor with the College of Computing and Data Science (CCDS), NTU. His research interests include edge intelligence, federated learning, and applied AI.

Dr. Bryan Lim's doctoral efforts earned him accolades such as the "Most Promising Industrial Postgraduate Program Student" Award. He also serves on the Technical Program Committee for FL workshops at flagship conferences (AAAI-FL and IJCAI-FL) and is a Review Board Member for reputable journals such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS.