FedCFC: On-Device Personalized Federated Learning with Closed-Form Continuous-Time Neural Networks

Yimin Dai*
Interdisciplinary Graduate School
Nanyang Technological University
Singapore

Rui Tan
School of Computer Science and Engineering
Nanyang Technological University
Singapore

ABSTRACT

Closed-form continuous-time (CFC) neural networks have superior expressivity in modeling time series data compared with recurrent neural networks. CFC's lower training and inference overheads also make it appealing for microcontroller-based platforms. This paper proposes FedCFC, which advances CFC from the centralized learning setting to the federated learning paradigm. FedCFC features a novel and communication-efficient aggregation strategy to address the problem of class distribution skews across clients' training data. The strategy is designed based on a new empirical property of CFC identified in this paper, i.e., involatility of a subnetwork of CFC with respect to training data's class distribution. Extensive evaluation based on multiple time series datasets shows that FedCFC achieves higher or similar accuracy with 7.6× to 11× reduction in communication overhead, compared with recent federated learning approaches designed to address the class distribution skew problem. Implementations of FedCFC on four microcontroller platforms show its portability to low-end computing devices with 256kB memory and even less.

KEYWORDS

Continuous-time neural network, federated learning, time series processing, edge computing

1 INTRODUCTION

Deep neural networks have been increasingly used to model and process time series data at the Internet-of-Things (IoT) devices. Examples include inertial data processing for activity recognition [15] and navigation [52], radar data processing for object detection [25], as well as physiological signal analysis [37]. However, the feedforward neural networks (FNNs) that have been widely used today do not model the time dimension explicitly. For instance, a neuron in a multilayer perceptron (MLP) models the relationship between the input and output of a biological neuron in the steady state, but does not model its dynamic transient behavior between two steady states. Recurrent neural networks (RNNs) use feedback to develop *network-level dynamics* and thus show better performance in modeling time series data. However, as RNNs assume discrete time steps with a fixed time interval, they require heuristic data preprocessing (e.g., up/down-sampling and interpolation) to address the deviations from the assumption, such as varied sampling rate and irregular sampling.

This paper considers continuous-time neural networks (CT-NNs) [49] that have *neuron-level dynamics* for time series data modeling and processing at the IoT devices. Specifically, a neuron's behavior is modeled by an ordinary differential equation (ODE) with continuous time as an independent and explicit variable. As the ODE contains learnable parameters, the neuron can adjust its internal dynamics during the training of the CT-NN. In addition, as CT-NN also supports network-level feedback, it can learn the dynamics in the time series data through the synergy between the neuron-level and network-level dynamics. Thus, CT-NNs have shown better model expressivity than RNNs [12]. Moreover, due to CT-NNs' continuous-time formulation, they can operate on any timestamped time series data without imposing the requirement of regular sampling with a fixed sampling interval.

Despite the above advantages, CT-NNs have a speed bottleneck that has been hindering its applications. Specifically, as most ODEs have no closed-form solutions, forwarding a CT-NN in general needs to use a numerical ODE solver that incurs compute overhead proportional to the number of steps dividing the time dimension. Encouragingly, a recent study [11] obtains a closed-form approximate solution to the ODE used by a class of CT-NNs called *liquid time constant* (LTC) neural networks [12]. It also designs a *closed-form continuous-time* (CFC) neural network based on three FNN modules to approximate the closed-form solution while preserving explicit consideration of continuous time, in that the CFC network jointly processes the time series data and the associated timestamps. Forwarding a CFC has a compute overhead proportional to the length of the input time series, which is typically one to three orders lower than those of ODE solvers.

CFC is promising for addressing two system challenges faced by the implementations of machine learning algorithms/models at resource-constrained IoT devices. First, its native ability to process any timestamped time series data addresses various system issues, including data intermittency due to hardware/software faults or application nature (e.g., occlusion in object tracking), sampling interval jitters due to uncertain operating system delays [26], and varied sampling rates across different IoT devices. Second, CFC's superior model expressivity implies smaller memory footprint for a certain accuracy level, thereby reducing the resource usage of model training and making on-device learning readily implementable. By combining these two strengths, CFC can process local time series data in real time directly on the IoT devices, which is critical for applications that require immediate responses, such as early detection of anomalies in industry machines' operations.

In this paper, we aim to advance CFC from the centralized learning setting [11] to the federated learning (FL) paradigm that matches the decentralized nature of IoT systems. FL builds a neural network that can model the data distributed on the clients and has a desirable privacy-preserving feature of retaining the raw data at the clients. However, the existing FL approaches incur high computational and communication overheads. A preliminary study [34] shows that FL

*Also with School of Computer Science and Engineering, Nanyang Technological University, Singapore.

can cause two orders of magnitude more carbon emissions than centralized learning. CFC offers the following direct advantages in reducing FL's overheads. First, owing to the lower resource requirements of CFC training, we may push the frontier of FL applications from the high-end edge devices (e.g., smartphones) to low-end IoT devices based on microcontroller units (MCUs). Second, CFC's smaller model size reduces FL's communication overhead. Note that FL can be applied under the continual learning setting to update the IoT devices' models with new data distributed on these devices. If the devices are battery-based (e.g., wearable sensors), the model update computation can be scheduled during the charging cycles.

This paper considers a practical setting that has challenged FL designs, i.e., the clients' data are not independent and identically distributed (non-IID). This can be observed in many IoT sensing applications. For instance, in a human activity recognition application, the distribution of the inertial data samples among the activity classes generally varies with the user. Such class distribution skews can lead to substantial oscillations in the loss trajectory and slow down the convergence when the vanilla FL aggregation strategy FedAvg [27] is adopted. This paper presents a new FL design based on CFC, called *FedCFC*, for the non-IID setting. It is based on a key observation that, one of the three CFC's FNN modules learns task-wide common pattern and is involatile to the training data's class distribution. Therefore, we limit the scope of FL to this FNN module only and leave the other two FNN modules unfederated to capture personal patterns. This design effectively balances the needs of capturing commonality and retaining personality under the non-IID setting. Moreover, we propose a novel geometric aggregation strategy called *fBind*, in which only the loss gradient with respect to the last layer of the selected FNN module is exchanged. This further reduces the communication overhead.

The main contributions of this paper are summarized as follows:

- Our benchmark study identifies the involatility of a key module of CFC with respect to the class distribution of training data. Based on the involatility, we design FedCFC to federate the key module only, with a new geometric aggregation algorithm fBind to efficiently address the non-IID data problem.
- Evaluation shows that the FedCFC with fBind outperforms the FedCFC variants using aggregation strategies of FedAvg [27], FedProx [23], and SCAFFOLD [18], where the latter two were designed for the non-IID setting. On three time series datasets, FedCFC achieves higher or similar accuracy with 7.6× to 14× reduction in communication overhead, compared with LotteryFL [22] and BalanceFL [44].
- Our implementations of FedCFC with communication module on four MCU-based platforms show its less than 256 kB memory footprint and thus its portability to low-end devices. Code is released at https://github.com/hhhddyy/FedCFC

Paper organization: §2 presents background. §3 presents benchmarks for CFC, which provide insights into FedCFC design. §4, §5, and §6 present design, evaluation, and on-device experiments of FedCFC. §7 discusses several issues. §8 concludes this paper.

2 BACKGROUND

This section presents the related work in §2.1 and then the preliminaries about LTC [12] and CFC [11] in §2.2.

2.1 Related Work

2.1.1 On-device training. While on-device training capability is essential to federated learning and continual learning, it is challenging for resource-constrained IoT devices. MDLdroid [53] manages model structures and learning process to achieve on-device training on smartphones with gigabytes dynamic random access memory (DRAM). However, typical MCUs have limited static random access memory (SRAM), in the range of 10^0 kB to 10^3 kB. Therefore, on-device training on MCUs is challenging. A survey [40] reviews the research works by 2022 on various machine learning topics for MCU-class hardware, including feature engineering, model design, and on-device training. It points out that the existing on-device training studies either consider high-end edge devices such as Raspberry Pi or simply limit the training to a few layers to meet the resource constraints. Later, the work [24] achieves on-device centralized training with 256 kB memory. This work will show that, with CFC, on-device FL with 256 kB SRAM can be achieved.

2.1.2 Federated learning. After the introduction of the vanilla FL strategy FedAvg [27], research attempts to address various practical factors affecting FL performance. The first category of research works deals with computation and networking issues of FL. For instance, the work [16] aims at reducing computation and communication overheads of FL with edge computing clients. The work [35] aims at increasing FL's robustness to straggler clients.

The second category of studies addresses the issues of *global class imbalance* and/or *local class skews*. The former refers to the non-uniform class distribution of all the data in the FL system. The latter refers to that the clients' class distributions are distinct, which is often referred to as the non-IID data problem. Local class skews in general imply global class imbalance. The works [6, 48] only address global class imbalance. Approaches addressing the local class skews problem in FL can be classified as *data-centric*, *system-centric*, and *personalized*. Data-centric approaches [5, 14, 44] apply mechanisms like data sharing between clients, data augmentation to enrich local data diversity, or local data self-balancing. Client clustering [33] is a typical system-centric approach, which selects similar clients to form federation. However, data-centric and system-centric approaches often require excessive communication and storage, making their implementations on resource-constrained devices challenging. Personalized FL, as surveyed in [45], addresses local class imbalance, often via either adapting a global model trained by FL to the local dataset (e.g., [8]) or applying specific designs of aggregating heterogeneous local models (e.g., [22]). However, the existing personalized FL approaches are in general not designed for MCU-class hardware. This paper exploits a key property of CFC to design a computation- and communication-efficient personalized FL approach that can work on MCU-class hardware.

The third category of studies addresses the FL clients' data distribution skews, i.e., the clients' local datasets form different domains. A recent work [51] reviews the existing approaches to this issue and applies self-supervised learning to exploit unlabeled local data to better manage the cross-client domain shifts.

2.2 Preliminaries on LTC and CFC Networks

The notation convention in this paper is as follows. Take the letter x as an example. x denotes a scalar; $\mathcal{X}$ denotes a set; $\mathbf{x}$ denotes

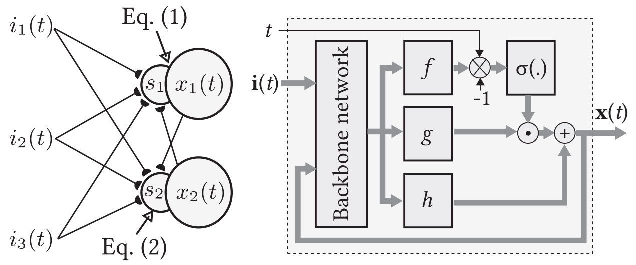


Fig. 1: LTC network (left) and CFC network (right). The illustrated LTC has three presynaptic sources and two neurons. The CFC network has two types of inputs, i.e., a vector $\mathbf{i}(t)$ and a scalar t, and outputs a vector $\mathbf{x}(t)$. Further processing (e.g., by an MLP) can be applied on $\mathbf{x}(t)$ to generate a label.

a column vector; $\mathbf{x}_{[i]}$ denotes the ith element of $\mathbf{x}$; $\mathbf{X}$ denotes a matrix; $\mathbf{X}_{[i]}$ denotes the ith row of $\mathbf{X}$; $\mathbf{X}[i, j]$ denotes the (i, j)th element of matrix $\mathbf{X}$; $\odot$ denotes Hadamard product.

An LTC of N neurons with M inputs is declared by the following system of linear ordinary differential equations (ODEs):

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = -\boldsymbol{\omega} \odot \mathbf{x}(t) + \mathbf{s}(t), \tag{1}$$

$$\mathbf{s}(t) = \phi\left(\mathbf{x}(t), \mathbf{i}(t); \mathcal{W}\right) \odot (\mathbf{a} - \mathbf{x}(t)), \tag{2}$$

where $t \in \mathbb{R}$ is continuous time; $\mathbf{x}(t) \in \mathbb{R}^N$ is the hidden state of the network; $\boldsymbol{\omega} = [1/\tau_1, 1/\tau_2, \ldots, 1/\tau_N]^\top \in \mathbb{R}^N$ with τ_i as a time constant; the ith component of $\mathbf{s}(t) \in \mathbb{R}^N$ represents a non-linear integration of all inputs to the ith neuron; $\mathbf{i}(t) \in \mathbb{R}^M$ is the input signal; $\mathbf{a} \in \mathbb{R}^N$ is constant; $\phi(\mathbf{x}(t), \mathbf{i}(t); \mathcal{W}) \in \mathbb{R}^N$ is a positive, continuous, monotonically increasing and bounded nonlinearity with learnable parameters $\mathcal{W}$. For instance, if the tanh nonlinearity is adopted, the ith component of $\phi(\mathbf{x}(t), \mathbf{i}(t); \mathcal{W})$ can be $\tanh\left(\mathbf{W}_{[i]} \cdot \mathbf{x}(t) + \mathbf{V}_{[i]} \cdot \mathbf{i}(t) + \mathbf{b}_{[i]}\right)$, where the (i, j)th element of the matrix $\mathbf{W} \in \mathbb{R}^{N\times N}$ is the weight of the directional link from the jth neuron to the ith neuron; the (i, k)th element of the matrix $\mathbf{V} \in \mathbb{R}^{N\times M}$ is the weight of the directional link from the kth input to the ith neuron; $\mathbf{b} \in \mathbb{R}^N$ is bias. Thus, $\mathcal{W} = \{\mathbf{W}, \mathbf{V}, \mathbf{b}\}$. When the topology of the directed graph specified by $\mathbf{W}$ contains cycles, the LTC is recurrent. LTC has a basis from biological neural systems. As illustrated by the left part of Fig. 1, for the ith neuron, Eq. (1) characterizes the dynamics of its membrane potential $\mathbf{x}_{[i]}(t)$ given its synaptic current $\mathbf{s}_{[i]}(t)$ [19, 20]; Eq. (2) describes the ith neuron's dendritic integration of its presynaptic sources to generate $\mathbf{s}_{[i]}(t)$ [19, 50]. The model is said to have liquid (i.e., varying) time constant, because the system time constant varies with $\mathbf{x}(t)$ and $\mathbf{i}(t)$. The system time constant is the reciprocal of the coefficient for the term $\mathbf{x}(t)$ in the ODE system specified by Eqs. (1) and (2). The time constant's liquidity improves the neural network's expressivity compared with those with fixed time constants [12].

As discussed in §1, the forwarding and training processes of a CT-NN are slow. A recent work [11] obtains a closed-form approximate solution with known approximation bounds to the scalar version (i.e., $N = 1$) of the ODE system in Eqs. (1) and (2): $x(t) \approx (x(0) - a)\mathrm{e}^{-[\omega+\phi(i(t);\mathcal{W})]t}\phi(-i(t); \mathcal{W}) + a$. The work [11] also describes an algorithm to stack the scalar closed-form approximate solutions for the N dimensions of $\mathbf{x}(t)$ to obtain a vector closed-form approximate solution as follows:

Table 1: Model size and test accuracy on sequential MNIST.

	vRNN	LSTM	GRU	bi-LSTM	**LTC**	**CFC**
Parameters	21k	82k	61k	164k	23k	44k
mFPOs	73	293	220	587	\	159
Accuracy	0.83	0.86	0.85	0.85	0.94	0.94

$$\mathbf{x}(t) \approx \boldsymbol{\beta} \odot \mathrm{e}^{-[\boldsymbol{\omega}+\phi(\mathbf{x}(t),\mathbf{i}(t);\mathcal{W})]t} \odot \phi(-\mathbf{x}(t), -\mathbf{i}(t); \mathcal{W}) + \boldsymbol{\alpha}, \tag{3}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^N$ and $\boldsymbol{\beta} \in \mathbb{R}^N$ are constants. To facilitate the implementation of Eq. (3) using modern deep learning toolboxs (e.g., PyTorch), three FNN modules, i.e., $f(\mathbf{x}(t), \mathbf{i}(t); \mathcal{W}^f)$, $g(\mathbf{x}(t), \mathbf{i}(t); \mathcal{W}^g)$, $h(\mathbf{x}(t), \mathbf{i}(t); \mathcal{W}^h)$, are used to represent the following three terms in Eq. (3), respectively: $\boldsymbol{\omega} + \phi(\mathbf{x}(t), \mathbf{i}(t); \mathcal{W})$, $\phi(-\mathbf{x}(t), -\mathbf{i}(t); \mathcal{W})$, and $\boldsymbol{\alpha}$. In the rest of this paper, these three FNNs are called f, g, and h modules. In addition, the exponential decay in Eq. (3) is replaced by a reversed sigmoid. Thus, the CFC is given by

$$\mathbf{x}(t) \approx \sigma(-f(\mathbf{x}(t), \mathbf{i}(t); \mathcal{W}^f)t) \odot g(\mathbf{x}(t), \mathbf{i}(t); \mathcal{W}^g) + h(\mathbf{x}(t), \mathbf{i}(t); \mathcal{W}_h),$$

where $\sigma(\cdot)$ is a sigmoid satisfying $\sigma(-\infty) = 0$ and $\sigma(0) = 1$. To improve trainability, the three FNN modules f, g, and h share a few feedforward layers called *backbone network*. The right part of Fig. 1 illustrates the structure of CFC.

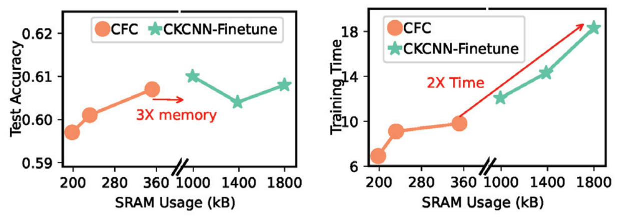
3 CFC BENCHMARKS

In this section, we conduct benchmark experiments to show the following properties of CFC: better model expressivity within 256 kB SRAM capacity, low training overhead on MCUs, robustness to sampling rate variation and sampling irregularity. Moreover, we show a new property of CFC, i.e., its f module is involatile with respect to the class distribution of the training data provided. The results in this section form a basis to design FedCFC.

3.1 Basic Properties of CFC

3.1.1 Model expressivity. First, we provide a benchmark to compare the expressivity of LTC/CFC and RNNs in terms of the model size and achieved accuracy. We consider the following RNNs: vanilla RNN (vRNN), long short-term memory (LSTM) network, gated recurrent unit (GRU) [4], and bi-directional LSTM (bi-LSTM) [10]. We consider the task of recognizing handwritten digits from the sequential MNIST dataset [21], which is derived from the standard MNIST dataset by flattening each 28 × 28 image sample into a 784-length sequence. Thus, each time step in the sequence gives a single pixel's grayscale value. By presenting the pixels one by one in a sequence, a model is trained to recognize the temporal patterns of the ten digits and achieve a classification result at the end of the sequence. This is different from using a convolutional neural network (CNN) to recognize the spatial distribution at one shot. Each of the vRNN, LSTM, GRU, and LTC has a total of 128 hidden neurons. For CFC, each of its f, g, and h modules is an MLP with a single 128-neuron hidden layer.

Table 1 presents the size, inference computation overhead per forward pass measured in mega floating-point operations (mFPOs), and achieved test accuracy of the compared models. Note that LTC's computation overhead is variable during the inference phase,



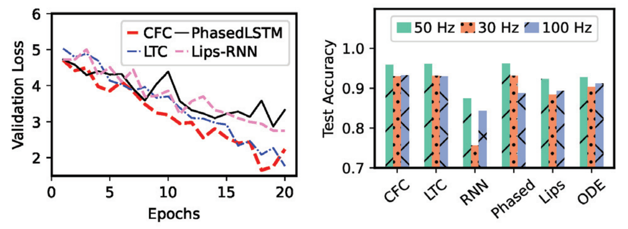
(a) Test accuracy vs. on-device training SRAM usage. **(b) On-device training time vs. SRAM usage.**

Fig. 2: On-device training overhead of CFC and CKCNN on a Cortex-M7 MCU. We train the entire CFC from scratch but only fine-tune the kernel MLP of CKCNN due to memory constraint of the MCU.

depending on the internal operations of the ODE solver. Thus, it is skipped in Table 1. We can see that the RNNs' accuracy saturates at about 85% when the model size increases from vRNN's 21k to bi-LSTM's 164k. LTC and CFC improve the accuracy to 94%. CFC's model size and computation overhead are two to four times lower than the advanced RNNs (i.e., LSTM, GRU, and bi-LSTM). Although LTC's model size is only half of CFC's, LTC's forwarding latency is about 3.6× of CFC's. If we use 32 bits to represent a parameter, the net size of CFC is 88 kB. Typically, the actual SRAM usage of a model implemented with TensorFlow Lite Micro is 2× to 2.5× of the net model size [31]. Thus, CFC can fit into 256 kB SRAM, which is a typical memory size of the MCUs used in tiny IoT devices [24].

3.1.2 On-device training overhead. On-device training capability is essential to continual learning and federated learning. However, on-device training often poses significant challenges for resource-constrained IoT devices. This section provides a benchmark to show the advantages of CFC for implementing on-device training, through the comparison with the continuous kernel CNN (CKCNN) proposed for sequential data processing [38]. We train CFC networks and CKCNNs on the Cortex-M7 core-based STM32H750XB MCU with 2 MB SRAM and a floating point unit (FPU) using the TinyEngine [24]. Each parameter of CFC or CKCNN is represented by a 16-bit floating point number. We use the sequential CIFAR10 dataset, which is derived from the standard CIFAR10 by flattening each image sample into a pixel sequence. We vary the sizes of the f, g, and h modules to generate various CFC networks. Figs. 2a and 2b show the CFC's test accuracy and training times versus the SRAM usage during the training. Training the entire CKCNN is infeasible on the MCU due to limited SRAM. As a workaround, we measure the overhead of CKCNN-based class-incremental learning. Specifically, we pre-train the CKCNN on a workstation with data in eight out of the ten classes of CIFAR10 and then fine-tune its kernel MLP module on the MCU with data in the remaining two classes. We vary the number of the MLP parameters fine-tuned to obtain the different data points shown in Fig. 2a and 2b for CKCNN. We can see that CFC achieves similar accuracy but with 3× and 2× reduction in SRAM usage and time in on-device training.

3.1.3 Dealing with varied sampling rates. We consider the case where the sampling rates of the training and test time series data



(a) Validation loss curves of models trained with 50 Hz data and tested with 30 Hz data. **(b) Accuracy of models trained with 50 Hz data and tested with 50 Hz, 30 Hz, and 100 Hz data.**

Fig. 3: Impact of sampling rate variation.

are different. We use the UCI Human Activity Recognition (HAR) dataset [1] capturing six activities with smartphone's inertial measurement unit (IMU) sampled at 50 Hz. We compare CFC, LTC, vRNN, Lipschitz RNN [12], phased LSTM [30], and another CT-NN called ODE-Nets [3]. Lipschitz RNN and phased LSTM were proposed to improve robustness against input perturbations and mitigate the impact of time irregularity, respectively. We train these models with the original training data sampled at 50 Hz. In addition to the original 50 Hz test data, we create 30 Hz and 100 Hz test data by averaging-based downsampling and cubic spline interpolation-based upsampling, respectively. Each input sample $\mathbf{i}(t)$ consists of readings in 2.54 seconds. Fig. 3a shows the validation loss curves of the models trained with 50 Hz data and tested with 30 Hz data. The absence of sharp fluctuations for CFC and LTC suggests that their training processes are stable. Fig. 3b shows the accuracy of the models tested with the 50 Hz, 30 Hz, and 100 Hz data. The CT-NNs (i.e., CFC, LTC, and ODE-Nets) show superior robustness against the sampling rate variations, due to their native capability of processing the timestamps derived from the sampling rate. Moreover, CFC and LTC outperform ODE-Nets.

3.1.4 Processing irregularly sampled data. We consider the case where the sensor measurements in a trace are collected with irregular intervals. We use the UCI Air Quality (AQ) dataset [47] for air pollution detection, electrocardiogram from PhysioNet dataset [17] for heart attack detection, Bosch CNC Machining (CNC) dataset [46] for machine anomaly detection. The data traces in these three datasets are sampled with irregular intervals. The sampling in UCI AQ and PhysioNet is event-triggered. The sampling intervals in CNC are influenced by network latency. The sampling intervals for these three datasets are at the scales of hours, minutes, and seconds, respectively. CT-NNs (i.e., CFC, LTC, and ODE-Nets) take the timestamps as a part of the input.

Table 2 shows the accuracy of the compared models on the three datasets. CFC and LTC achieve better modeling accuracy. If we use the average accuracy achieved by the three RNNs as the baseline, CFC outperforms RNNs by 15%, 11%, and 3% on the three datasets with hours-, minutes-, and seconds-level sampling intervals. Thus, CT-NNs show greater advantage in addressing longer irregular sampling intervals. This is because RNNs highly rely on the correlation between consecutive samples, but such correlation tends to decrease with the sampling interval.

Table 2: Model accuracy on irregularly sampled data.

	CFC	LTC	vRNN	Phased LSTM	Lips RNN	ODE Nets
UCI AQ	**0.794**	0.787	0.687	0.694	0.697	0.764
PhysioNet	0.827	**0.842**	0.723	0.771	0.734	0.822
CNC	**0.897**	0.893	0.852	0.884	0.881	0.842

3.2 A New Property: f Module Involatility

This section presents a new empirical property of CFC, i.e., f module's involatility with respect to the class distribution of the training data. In what follows, we first define a measure of volatility. Then, we present the benchmark results.

3.2.1 Definitions. Denote an L-layer MLP trained with dataset $\mathcal{D}$ by $\pi(\cdot;\mathcal{W}_{\mathcal{D}})$, where $\mathcal{W}_{\mathcal{D}} = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L\}$ and $\mathbf{W}_i \in \mathbb{R}^{n_{i-1}\times n_i}$ is the weight matrix of the connection from the $(i-1)$th layer with n_{i-1} neurons to the ith layer with n_i neurons. Define $\mathbf{\Pi}_{\mathcal{D}} = \prod_{l=1}^{L}\mathbf{W}_l$. Let $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_P$ denote P partitions of $\mathcal{D}$, i.e., $\bigcup_{p=1}^{P}\mathcal{D}_p = \mathcal{D}$, where $\mathcal{D}_p \cap \mathcal{D}_q$ is unnecessarily empty. Let $\Psi(\mathbf{\Pi}_{\mathcal{D}_p})$ denote the subspace spanned by $\mathbf{\Pi}_{\mathcal{D}_p}$. We define *principal angle* to measure the angular alignment between two subspaces.

Definition 1 (Principal angle). The angle between the subspaces $\Psi(\mathbf{\Pi}_{\mathcal{D}_p})$ and $\Psi(\mathbf{\Pi}_{\mathcal{D}_q})$ is denoted by $\Theta(\mathbf{\Pi}_{\mathcal{D}_p}, \mathbf{\Pi}_{\mathcal{D}_q}) = [\theta_1, \theta_2, \ldots, \theta_k]$, where k is the minimum between the dimensions of the two subspaces; $0 \le \theta_1 \le \theta_2 \le \cdots \le \theta_k \le 90°$. The θ_i is recursively defined from $i=1$ to $i=k$ by $\theta_i = \arccos\left(\frac{|\langle \mathbf{u}_i^*, \mathbf{v}_i^*\rangle|}{\|\mathbf{u}_i^*\|\|\mathbf{v}_i^*\|}\right)$ and $\mathbf{u}_i^*, \mathbf{v}_i^* = \operatorname{argmin}_{\mathbf{u}_i,\mathbf{v}_i} \arccos\left(\frac{|\langle \mathbf{u}_i, \mathbf{v}_i\rangle|}{\|\mathbf{u}_i\|\|\mathbf{v}_i\|}\right)$, where $\mathbf{u}_1$ and $\mathbf{v}_1$ are any column vectors of $\mathbf{\Pi}_{\mathcal{D}_p}$ and $\mathbf{\Pi}_{\mathcal{D}_q}$, respectively; for $i \in [2,k]$, $\mathbf{u}_i \in \Psi(\mathbf{\Pi}_{\mathcal{D}_p})$ and $\mathbf{v}_i \in \Psi(\mathbf{\Pi}_{\mathcal{D}_q})$ satisfy $\mathbf{u}_i \perp \mathbf{u}_j^*$, $\mathbf{v}_i \perp \mathbf{v}_j^*$, $\forall j \in \{1, \ldots, i-1\}$. The angle $\max \Theta(\mathbf{\Pi}_{\mathcal{D}_p}, \mathbf{\Pi}_{\mathcal{D}_q}) = \theta_k$ is called *principal angle*.

Definition 2 ((ϵ,θ)-volatility). An MLP π is (ϵ,θ)-volatile on dataset partitions $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_P\}$ if (1) $\sup_{\forall p,q\in[1,P]} \|\pi(\mathbf{1};\mathcal{W}_{\mathcal{D}_p}) - \pi(\mathbf{1};\mathcal{W}_{\mathcal{D}_q})\|_\infty \le \epsilon$ and (2) $\sup_{\forall p,q\in[1,P]} \max\Theta(\mathbf{\Pi}_{\mathcal{D}_p}, \mathbf{\Pi}_{\mathcal{D}_q}) \le \theta$.

The (ϵ,θ)-volatility measures the sensitivity of an MLP with respect to the training data provided. Assume two MLPs, π_A and π_B, are (ϵ_A,θ_A)-volatile and (ϵ_B,θ_B)-volatile, respectively, on the same training dataset. If $\epsilon_A \ge \epsilon_B$ and $\theta_A \ge \theta_B$, we say π_A is more volatile than π_B with respect to the training data provided.

3.2.2 Involatility of f module. We compare the volatility of CFC's three FNN modules, f, g, and h, as well as other neural networks.

First, we check Condition (1) of Definition 2. It is introduced in [29] to assess the stability of MLP during training. Now, we show that the analytical counterparts of the f and g modules satisfy Condition (1) with the same ϵ level, where the analytical counterpart of the h module cannot find a certain ϵ to ensure Condition (1). The analytical counterpart of f in Eq. (3) is $\boldsymbol{\omega} + \phi(\mathbf{x}(t), \mathbf{i}(t); \mathcal{W})$. Thus, when trained on two dataset partitions $\mathcal{D}_p$ and $\mathcal{D}_q$, we have $\|f(\cdot;\mathcal{W}_{\mathcal{D}_p}) - f(\cdot;\mathcal{W}_{\mathcal{D}_q})\|_\infty = \|\phi(\mathbf{x}(t), \cdot;\mathcal{W}_{\mathcal{D}_p}) - \phi(\mathbf{x}(t), \cdot;\mathcal{W}_{\mathcal{D}_q})\|_\infty \le \phi_{\max} - \phi_{\min}$, where $\phi_{\max}$ and $\phi_{\min}$ are the maximum and minimum values of the bounded nonlinearity $\phi(\cdot)$. Thus, as long as $\epsilon \ge \phi_{\max} - \phi_{\min}$, Condition (1) is satisfied for f. The same holds for g with an analytical counterpart of $\phi(-\mathbf{x}(t), -\mathbf{i}(t); \mathcal{W})$ in Eq. (3). Differently, the analytical counterpart of h in Eq. (3) is $\boldsymbol{\alpha}$, which is a constant vector that best fits the given training data. There is no known upper bound for $\|\boldsymbol{\alpha}_{\mathcal{D}_p} - \boldsymbol{\alpha}_{\mathcal{D}_q}\|_\infty$.

Then, we conduct benchmark experiments to compare the principal angles of the f, g, and h modules concerned in Condition (2) of Definition 2. In particular, we focus on the volatility with respect to the class distribution of training data. Specifically, the sets of classes contained in the partitions $\mathcal{D}_1, \ldots, \mathcal{D}_P$ are distinct. We use the following three datasets: UCI-HAR [1], PPG-DaLiA [36], and URBANSOUND8K [41]. UCI-HAR has been introduced in §3.1.3. PPG-DaLiA includes photoplethysmography (PPG) data captured by sensors strapped on chest. URBANSOUND8K includes acoustic traces collected from different locations in urban areas. We use the traces to infer their collection locations. From a dataset with K classes, we create C_2^K partitions, where each partition contains data in two classes. Then, we measure the principal angle between two instances of a certain FNN module trained on any two partitions. This generates $C_2^{C_2^K}$ principal angles. Fig. 4 shows the distribution of such $C_2^{C_2^K}$ principal angles of each of f, g, and h. We can see that the f module has the smallest principal angles in any dataset. This suggests that f is less volatile than g and h. Empirically, the f modules with sigmoid activation for the three datasets are $(1, 33°)$-, $(1, 38°)$-, and $(1, 42°)$-volatile, respectively.

We also compare the volatility of CFC's f module and other neural networks including CKCNN, vRNN, and GRU. As the (ϵ,θ)-volatility is defined for MLP only and thus inapplicable to these

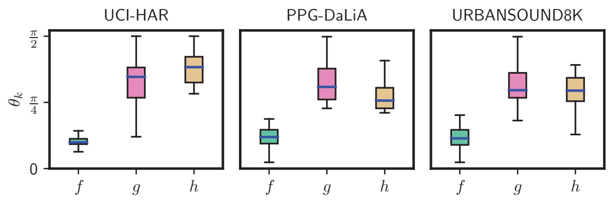


Fig. 4: Principal angle between any two instances of a certain FNN module trained on datasets with class skews.

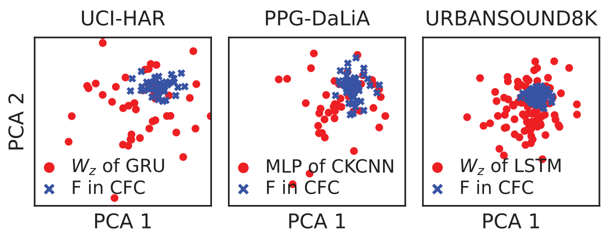


Fig. 5: PCA on f module's and other models' weights.

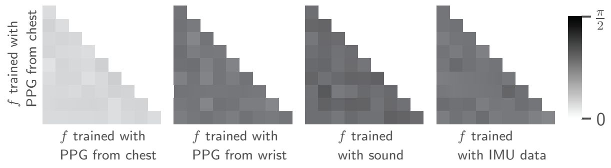


Fig. 6: Principal angle between f instances trained with (a) same dataset, (b) different datasets in same modality, (c-d) different datasets in different modalities.

other neural networks, we employ the more general principal component analysis (PCA) on the model weights. Fig. 5 shows the scatter plot of the C_2^K instances of a certain neural network in the space spanned by the first two PCA components. The higher concentration degree of the f instances suggests its less volatility compared with other neural networks.

Lastly, we investigate whether the f module remains involatile across datasets. Definition 1 can be extended to the cross-dataset case by drawing $\mathcal{D}_p$ and $\mathcal{D}_q$ from different datasets' partitions. We use four datasets: chest PPG and wrist PPG from PPG-DaLiA, URBANSOUND8K, and UCI-HAR. Each dataset is divided into nine partitions. Fig. 6(a) shows the principal angles between the f instances trained from any two chest PPG partitions, which serves as a baseline. Figs. 6(b)-(d) shows the cross-dataset results. We can see that the f module is no longer involatile across datasets. Note that the chest PPG and wrist PPG datasets are for the same downstream application of heartbeat rate monitoring, but with domain shifts due to different sensor placements. In the rest of this paper, we focus on the situation of non-IID classes without domain shifts.

4 FEDERATED LEARNING WITH CFC

4.1 Motivation

The basic properties of CFC shown in §3.1 suggest CFC's advantages on single IoT devices. A natural question next is whether we can build an efficient FL system based on CFC for an IoT network. FL well matches the distributed nature of IoT networks and possesses a key advantage of retaining the training data at IoT devices, which has a privacy-preserving implication. A well-designed FL system can learn more versatile models, as it uses wider training data. When the training data at each client is limited, FL can increase the model accuracy substantially compared with the case where the clients train their models independently.

However, the *local class skews* (i.e., non-IID data) problem [45] has challenged efficient FL designs. The problem is formally stated as follows. For a set of clients C, let $\mathcal{D}_c = \{(\mathbf{X}_c, Y_c)\}$ denote client c's local dataset, where $c \in C$, $\mathbf{X}_c$ is a data sample and Y_c is the corresponding label. The $\mathcal{D}_c$ can be viewed as a partition of the global dataset $\mathcal{D} = \bigcup_{c \in C} \mathcal{D}_c$. In this paper, we assume that the conditional probability distribution $P(\mathbf{X}_c | Y_c = y)$ is identical across all clients. That is, the clients' local datasets have no domain shifts. The local class skews problem refers to that the prior probability distribution $P(Y_c)$ varies with c. It impedes FL's capability to let each local model converge to the desired global optimal model that minimizes the loss function on $\mathcal{D}$. This is because the losses computed by the clients with their local datasets are distinct and therefore the local model updates can be hardly harmonized. If the vanilla FL strategy FedAvg is applied, oscillations in the global model updates can be observed. Note that this paper only focuses on the local class skews problem. Addressing domain shifts among multiple datasets, though important as well, is often studied separately [26, 51]. Real IoT systems usually face both problems. Efficient solutions to the local class skews problem form a basis to achieve the ultimate goal of addressing the two problems together.

While the existing studies have proposed various FL strategies to deal with the local class skews as discussed in §2.1, the CFC f module's involatility with respect to $P(Y_c)$ offers an opportunity to design a simple yet effective FL strategy. Specifically, the f module captures the task-wide pattern common across the non-IID dataset partitions, while the g, h modules capture the partition-specific patterns. This separation inspires the basic idea of the proposed FedCFC, i.e., we federate the f modules learned by the clients, while leaving other parts of CFC unfederated. This configuration offers two advantages. First, the federated learning for f can improve its versatility while not facing significant oscillations supposedly, since the local updates to f are likely harmonized owing to f's involatility. Second, leaving the g and h unfederated allows the local models to be personalized. Note that applying existing non-IID FL strategies on g and h might bring some further accuracy improvement. However, they also introduce more computation and communication overheads, as shown in §5.

4.2 Approach Overview

A straw man solution is to apply FedAvg on the f modules, which is called FedCFC-fAvg in the rest of this paper. In each communication round of FedCFC-fAvg, the server disseminates the global f module. Then, each client sends back the update. Although the local f modules across the clients with non-IID data are nearly identical after the convergence of FedCFC-fAvg, they traverse different trajectories during the FL process. In this paper, we propose a more efficient FL strategy called FedCFC-fBind, where "fBind" refers to the strategy that explicitly binds the evolution trajectories of the clients' f modules. Its efficiency is from two aspects. First, it explicitly preserves the involatility of f throughout the FL process. This preservation aims at harmonizing the local updates to f, speeding up the convergence. Second, different from FedCFC-fAvg that exchanges the f module between the server and client, FedCFC-fBind only exchanges the gradient of the last layer of the f module, reducing the communication overhead. In what follows, we formalize the objective of FedCFC-fBind and then provide an overview of its workflow.

Let $\mathcal{W}_c^f$, $\mathcal{W}_c^g$, and $\mathcal{W}_c^h$ denote the parameters of the f, g, and h modules of client c's CFC. Denote $\mathcal{W}_c = \{\mathcal{W}_c^f, \mathcal{W}_c^g, \mathcal{W}_c^h\}$. Let $L(\mathcal{D}_c, \mathcal{W}_c)$ denote the cross-entropy loss at client c. Each round of FedCFC-fBind aims at addressing the following problem:

$$\min_{\mathcal{W}_c, \forall c \in C} \sum_{\forall c \in C} L(\mathcal{D}_c, \mathcal{W}_c) \text{ s.t. } \max \Theta(\Pi^f_{\mathcal{D}_p}, \Pi^f_{\mathcal{D}_q}) \leq \theta, \forall p, q \in C, \quad (4)$$

where $\Theta(\Pi^f_{\mathcal{D}_p}, \Pi^f_{\mathcal{D}_q})$ is the principal angle between the f modules of any two clients p and q. The constraint in Eq. (4) preserves the involatility of the f module. Although the θ in Eq. (4) can adopt a setting according to the prior knowledge about the f module's empirical (ϵ, θ)-volatility, it will not be needed in a Lagrangian relaxation of Eq. (4) solved by FedCFC-fBind, which will be presented in §4.3 shortly. In each communication round of FedCFC-fBind, the clients and server perform the following actions.

Client step: A client applies stochastic gradient descent (SGD) to solve an unconstrained optimization problem relaxed from Eq. (4). The relaxation uses regularization based on a common update direction (CUD) and a common update magnitude (CUM) received from the server regarding the f. This is called *bound local updater*. Then, the client reports the local update direction (LUD) to the server.

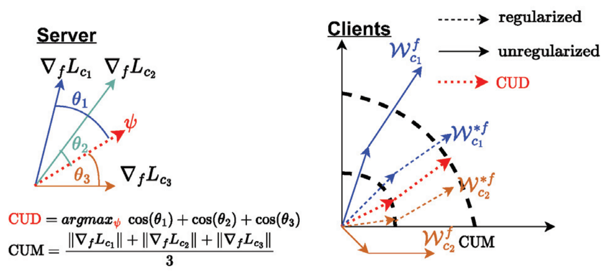


Fig. 7: Illustrations of the server and client steps. ***Server*: Given gradients from three clients, the server sets CUD to be the direction ψ that maximizes the sum of cosine similarities with the three client gradients, and CUM to be the average magnitude of the three gradients. *Clients*: The drawing illustrates the trajectories of two clients in two communication rounds. The regularization penalizes direction deviation from CUD and magnitude overstepping from CUM.**

Server step: Based on the LUDs received from all the clients, the server applies an optimization-based geometric aggregation algorithm called *bound updates aggregator* to generate the next CUD and CUM, and disseminates them to the clients.

Through iterating the above steps, the clients' f modules keep aligned throughout the FL process. The next subsections present the details of the client's and server's steps.

4.3 Client's Bound Local Updater

Client c applies SGD to update its f, aiming to minimize the overall loss $L_c = l_t + \lambda_d R_d + \lambda_m R_m + \lambda_a R_a$, where l_t is the loss of the task (e.g., cross-entropy loss of classification task); R_d, R_m, and R_a are three regularization terms presented below; the lambdas are weights and hyperparameters. The above unconstrained optimization encompasses the Lagrangian relaxation to Eq. (4). At the end of the local update, client c transmits the gradient of L_c with respect to the weights in the last layer of f module, denoted by $\nabla_f L_c$, as the LUD to the server.

The *directional* regularization term R_d is the cosine distance between $\nabla_f l_t$ and CUD, i.e., $R_d = 1 - \frac{\nabla_f l_t \cdot \text{CUD}}{\|\nabla_f l_t\|_2 \|\text{CUD}\|_2}$, where $\nabla_f l_t$ is the gradient of l_t with respect to the weights in the last layer of module f and the CUD is disseminated by the server in the last round. Thus, R_d encourages the directional alignment between the local gradient update and the CUD suggested by the server.

The *magnitudinous* regularization term R_m is given by $R_m = \max(0, \|\nabla_f l_t\|_2 - \text{CUM})$, where the CUM is disseminated by the server in the last round. Thus, R_m encourages that the magnitude of the local gradient update remains within CUM.

The R_d and R_m together bind all clients' local gradient updates. However, they may lead to widespread inactive neurons in the f module. This can impede the convergence of the FL process. The *active* regularization term R_a is applied to increase the activity of neurons. It is given by $R_a = \Upsilon(\mathcal{W}^f_c) - \upsilon$, where υ is a desired activity threshold and $\Upsilon(\mathcal{W}^f_c)$ is the mean activation of the neurons in the module f. Specifically, $\Upsilon(\mathcal{W}) = \min_{i=1}^{L} \max_{j=1}^{n_i} o_{ij}$, where L is the number of layers of the MLP f, n_i is the width of the ith layer, and o_{ij} represents the output of the jth neuron of the ith layer. Note that υ is a hyperparameter.

The right part of Fig. 7 illustrates the local updates at two clients in two communication rounds, where the two dotted red arrows are two CUDs and the two dashed circles are two CUMs received from the server. Solid arrow and dash arrow represent the unregularized and regularized local updates.

4.4 Server's Bound Updates Aggregator

The aggregator running at the server determines CUM and CUD based on the LUDs received from all clients, i.e., $\{\nabla_f L_c | c \in C\}$. The aggregator aims to harmonize the clients' local updates. The CUM is determined by CUM $= \frac{1}{|C|}\sum_{c\in C}\|\nabla_f L_c\|_2$. We aim to find a CUD to maximize the overall cosine similarity between the CUD and the LUDs, i.e., CUD $= \text{argmax}_{\boldsymbol{\psi}} \sum_{c\in C} \frac{\nabla_f L_c \cdot \boldsymbol{\psi}}{\|\nabla_f L_c\|_2 \|\boldsymbol{\psi}\|_2}$. This is illustrated by the left part of Fig. 7. As it is non-convex optimization, gradient-based methods can be easily stuck at local optimums.

We approach the above problem in a Hilbert space. Specifically, we minimize the distance between the CUD and the centroid of the LUDs in the Hilbert space, which is convex. As a result, the key is to find the mapping function from the gradient space (denoted by $\mathbb{G}$) to the Hilbert space (denoted by $\mathbb{H}$). In what follows, we present a corollary, which is an application of the Moore Aronszajn Theorem [2] to our context and will be used to develop the solution to the original CUD determination problem.

Corollary 1. *For any mapping $\Gamma : \mathbb{G} \to \mathbb{H}$, there exists a kernel function $\kappa(\cdot,\cdot)$ defined on domain $\mathbb{G}\times\mathbb{G}$ such that the Hilbert-space distance* $\left\| \frac{1}{|C|}\sum_{\forall c\in C}\Gamma\left(\nabla_f L_c\right) - \Gamma(\boldsymbol{\psi}) \right\|^2_{\mathbb{H}} = \text{tr}(\mathbf{AB})$, *where* $\mathbf{A} = \begin{bmatrix} \mathrm{A}_1 \in \mathbb{R}^{|C|\times|C|} & \mathrm{A}_2 \in \mathbb{R}^{|C|\times 1} \\ \mathrm{A}_3 \in \mathbb{R}^{1\times|C|} & \kappa(\boldsymbol{\psi},\boldsymbol{\psi}) \end{bmatrix}$, $\mathbf{B} \in \mathbb{R}^{(|C|+1)\times(|C|+1)}$, $\mathrm{A}_1[i,j] = \kappa(\nabla_f L_i, \nabla_f L_j)$, $\mathrm{A}_2[i,1] = \kappa(\nabla_f L_i, \boldsymbol{\psi})$, $\mathrm{A}_3[1,j] = \kappa(\boldsymbol{\psi}, \nabla_f L_j)$,

$$\mathbf{B}[i,j] = \begin{cases} \frac{1}{|C|^2} & \text{if } i \le |C|, j \le |C|; \\ 1 & \text{if } i > |C|, j > |C|; \\ -\frac{2}{|C|} & \text{otherwise.} \end{cases}$$

Proof. Let $\langle\cdot,\cdot\rangle_{\mathbb{H}}$ denote the inner product operation defining $\mathbb{H}$. The Moore Aronszajn theorem [2] states that $\forall(\mathbf{x},\mathbf{x}') \in \mathbb{G}\times\mathbb{G}$, $\forall\Gamma$, $\exists\kappa$, such that $\kappa(\mathbf{x},\mathbf{x}') = \langle\Gamma(\mathbf{x}),\Gamma(\mathbf{x}')\rangle_{\mathbb{H}}$. Therefore, we have $\left\|\frac{1}{|C|}\sum_{\in C}\Gamma(\nabla_f L_c) - \Gamma(\boldsymbol{\psi})\right\|^2_{\mathbb{H}} = \frac{1}{|C|^2}\sum_{\forall p,q\in C}\langle\Gamma(\nabla_f L_p),\Gamma(\nabla_f L_q)\rangle + \langle\Gamma(\boldsymbol{\psi}),\Gamma(\boldsymbol{\psi})\rangle - \frac{2}{|C|}\sum_{\forall c\in C}\langle\Gamma(\nabla_f L_c),\Gamma(\boldsymbol{\psi})\rangle \overset{(*)}{=} \frac{1}{|C|^2}\sum_{\forall p,q\in C}\kappa(\nabla_f L_p,\nabla_f L_q) + \kappa(\boldsymbol{\psi},\boldsymbol{\psi}) - \frac{2}{|C|}\sum_{\forall c\in C}\kappa(\nabla_f L_c,\boldsymbol{\psi}) \overset{(\dagger)}{=} \text{tr}(\mathbf{AB})$, where the step marked by (*) follows the Moore Aronszajn theorem. □

Now, we discuss how Corollary 1 is related to the original objective of maximizing the overall cosine similarity between the CUD and the LUDs. Cosine similarity is a type of kernel function. If $\kappa(\cdot,\cdot)$ in Corollary 1 satisfies that $\kappa(\mathbf{x},\mathbf{x}')$ is a constant (which holds for the cosine similarity kernel $\kappa(\mathbf{x},\mathbf{x}') = \frac{\mathbf{x}\cdot\mathbf{x}'}{\|\mathbf{x}\|\|\mathbf{x}'\|}$), from the step marked by (†), we have $\text{argmax}_{\boldsymbol{\psi}}\sum_{\forall c\in C}\kappa(\nabla_f L_c,\boldsymbol{\psi}) = \text{argmin}_{\boldsymbol{\psi}}\text{tr}(\mathbf{AB})$, where the left-hand side of the above equation is a more general form of the original objective. However, $\text{argmin}_{\boldsymbol{\psi}}\text{tr}(\mathbf{AB})$ for any

```
Algorithm 1: FedCFC-fBind
Data: Local datasets $\mathcal{D}_c, \forall c \in C$
Result: Local CFC models $\mathcal{W}_c = \{\mathcal{W}_c^f, \mathcal{W}_c^g, \mathcal{W}_c^h\}, \forall c \in C$
Configuration: Learning rate $\eta$; regularization weights $\lambda_d$,
  $\lambda_m$, $\lambda_a$; neuron activation threshold $\upsilon$
while not converged do
    server: disseminates CUD and CUM to clients
    for each client c in parallel do
        Compute task loss $l_t$ and gradients $\nabla_f l_t$, $\nabla_g l_t$, $\nabla_h l_t$
        $\mathcal{W}_c^g \leftarrow \mathcal{W}_c^g - \eta \dot{\nabla}_g l_t, \quad \mathcal{W}_c^h \leftarrow \mathcal{W}_c^h - \eta \dot{\nabla}_h l_t$
        $R_d \leftarrow 1 - \frac{\nabla_f l_t \cdot \mathrm{CUD}}{\|\nabla_f l_t\|_2 \|\mathrm{CUD}\|_2}, \quad R_a \leftarrow \Upsilon(\mathcal{W}_c^f) - \upsilon$
        $R_m \leftarrow \max(0, \|\nabla_f l_t\|_2 - \mathrm{CUM})$
        $L_c \leftarrow l_t + \lambda_d R_d + \lambda_m R_m + \lambda_a R_a$ and compute $\nabla_f L_c$
        $\mathcal{W}_c^f \leftarrow \mathcal{W}_c^f - \eta \dot{\nabla}_f L_c$
        Upload $\nabla_f L_c$ to server
    server: CUM $\leftarrow \frac{1}{|C|} \sum_{c \in C} \|\nabla_f L_c\|_2$ and compute CUD
```

given $\kappa(\cdot,\cdot)$ (e.g., cosine similarity kernel) is still a non-convex optimization problem. Moreover, searching for an appropriate kernel function is conducted within a large, often infinite-dimensional space. We follow the relax-then-check strategy presented below to avoid the direct search for the kernel function. The main idea is that, instead of finding the kernel itself, we focus on directly determining the outcome this kernel would produce.

Relax: We ignore the internal structure of $\mathbf{A}$ and solve the following semi-definite programming problem which is a sub-type of linear programming: $\{\mathbf{X}^*\} = \operatorname{argmin}_{\mathbf{X} \in \mathbb{R}^{(|C|+1)\times(|C|+1)}} \operatorname{tr}(\mathbf{XB})$. The solution $\{\mathbf{X}^*\}$ can be an infinite set.

Check: We check a sufficient number of $\mathbf{X}^*$ within the latency requirement imposed on the aggregator. For each $\mathbf{X}^*$, we apply the kernel PCA method [42] to determine the $\boldsymbol{\psi}^*$ by $\boldsymbol{\psi}^*_{[i]} = \sum_{j=1}^{|C|+1} \mathbf{v}_{i[j]} \mathbf{X}^*[|C|+1, j]$, where $\mathbf{v}_i$ is the eigenvector of $\mathbf{X}^*$ corresponding to the ith largest eigenvalue. The $\boldsymbol{\psi}^*$ that best fits $\mathbf{A}$ with its internal structure defined by the cosine similarity kernel is yielded as the CUD.

4.5 Entire FedCFC-fBind Process

Algorithm 1 shows the entire FedCFC-fBind process, including the clients' local updates for f, g, h, and the server's aggregation.

5 PERFORMANCE EVALUATION

5.1 Evaluation Methodology and Settings

Our evaluation uses the following three datasets:

- **HASC-PAC2016** [13] includes 19,172 data samples collected by IMU on wrist for human activity recognition. Each trace lasts for 300 seconds and corresponds to one of 6 activities. We divide a trace into 3-second input samples.
- **NTU RGB+D** [43] includes 3-dimensional skeleton data of 25 joints and contains 60 classes of human activities. Each input sample consists of 3,600 readings for 25 joints.
- **AMIGOS** [28] includes electroencephalogram and electrocardiogram data from 40 subjects for emotion recognition. A sliding window with 1-second window size and 0.2-second overlapping is used to divide the trace into input samples.

Our evaluation recreates three practical challenges faced by real-world deployments of FL. The first challenge is non-IID data. We consider two scenarios of class heterogeneity and the more general local class skews. Specifically, in the first scenario, each client's dataset partition only covers a small subset of the classes; in the second scenario, each client's dataset partition covers all classes but the distribution of data among the classes is distinct from each other. The second challenge is hardware heterogeneity among the clients, which results in the model precision heterogeneity. The third challenge is that the FL system encounters new classes after deployment. For each recreated situation, we compare the following FedCFC variants and two existing non-IID FL approaches:

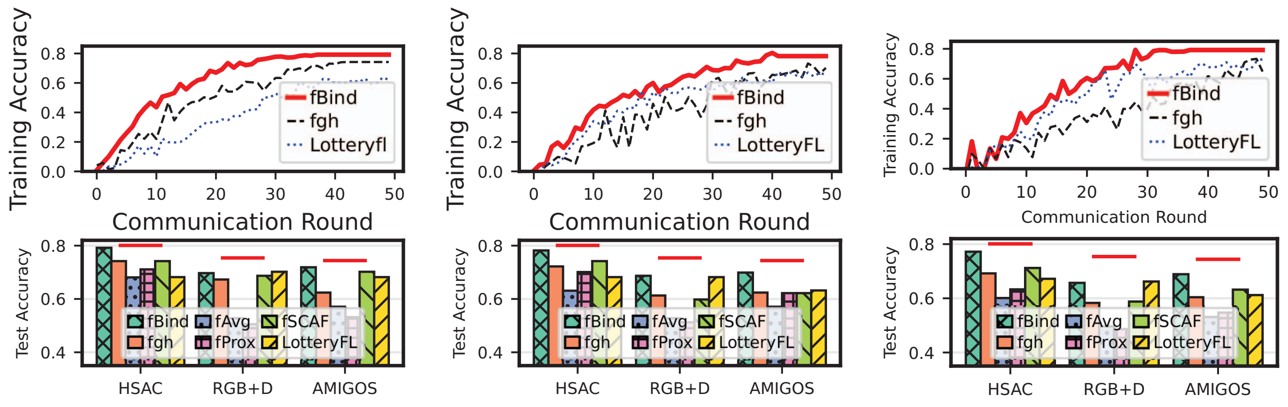
- **FedCFC-fBind** is our main proposal described in §4.
- **FedCFC-fAvg** adopts FedAvg [27] to aggregate f modules.
- **FedCFC-fSCAF** adopts SCAFFOLD [18] to aggregate local f modules. SCAFFOLD corrects local updates using variance reduction to align local models with the global model, thereby mitigating the effects of non-IID data.
- **FedCFC-fProx** uses FedProx [23] to aggregate f modules.
- **FedCFC-fgh** uses fBind to federate local f modules and SCAFFOLD to federate the local g and h modules.
- **LotteryFL** [22] is a personalized FL approach that only federates a subnetwork. It is designed to address non-IID data.
- **BalanceFL** [44] is an FL approach that applies a local self-balancing technique to deal with the non-IID data issue.

All the above FL approaches are implemented with PyTorch. For all FedCFC variants, the CFC consists of a single-layer backbone and three two-layer MLPs as the f, g, and h modules. Note that the on-device FedCFC to be presented in §6 is implemented without using PyTorch. For both implementations, the batch size is set to one, for consistency and also facilitating the on-device training process. The hyperparameters of our FedCFC-fBind implementation are as follows: $\lambda_d = 0.6$, $\lambda_m = 0.2$, $\lambda_a = 0.2$, $\upsilon = 1.59$, $\eta = 0.01$. In §5.6, we conduct a sensitivity analysis on the three regularization coefficients λ_d, λ_m, and λ_a.

5.2 Evaluation with Class Heterogeneity

In this set of experiments, the dataset partition owned by each client only covers a subset of the classes of the entire dataset. When assigning the data samples to a total of 12 clients, we control the degree of class heterogeneity among the clients, denoted by γ. Specifically, $\gamma = |\mathcal{D}_p \cap \mathcal{D}_q|/|\mathcal{D}_p|$. We adopt three γ settings: 1) $\gamma = 1$ means the IID case, in which each client has data in all classes; 2) $\gamma = 0.5$ means a moderate non-IID case, in which any two clients' dataset partitions have 50% overlap in terms of classes; 3) $\gamma = 0$ means the most severe non-IID case, in which the clients' dataset partitions are disjoint in terms of classes. BalanceFL is skipped in this subsection, because it is inapplicable in the $\gamma = 0.5$ and $\gamma = 0$ cases. In each round of the experiment, we vary the classes each client obtains while fixing the degree of class heterogeneity γ.

The three columns of Fig. 8 show the results under the three γ settings. The top row shows the training accuracy versus the



(a) $\gamma = 1$ **(IID)** **(b)** $\gamma = 0.5$ **(c)** $\gamma = 0$ **(disjoint)**

Fig. 8: Performance of FL approaches under three degrees of class heterogeneity γ. Top row: training accuracy vs. communication rounds; Bottom row: test accuracy after FL converges. For FedCFC variants, the "FedCFC-" prefix is omitted in legends.

Table 3: Convergence speed and communication overhead under three settings for the degree of class heterogeneity γ.

Approach	Dataset	γ = 1 (IID)		γ=0.5		γ=0 (disjoint)	
		Comm. rounds	Comm. vol. (kB)	Comm. rounds	Comm. vol. (kB)	Comm. rounds	Comm. vol. (kB)
FedCFC-fBind	HSAC	37	92.5	41	100.2	37	93.7
	NTU RGB+D	47	137.7	42	103.6	49	162.1
	AMIGOS	31	79.1	34	87.8	37	92.2
FedCFC-fgh	HSAC	42	3992.5	71	7000.7	87	9772.1
	NTU RGB+D	64	6332.1	89	10023.2	91	13622.1
	AMIGOS	41	3779.1	62	6017.4	57	5729.2
FedCFC-fAvg	HSAC	47	4517.1	59	8788.2	99+	19379.7
	NTU RGB+D	57	8351.1	77	11311.7	99+	24321.7
	AMIGOS	39	6147.3	45	7322.8	67	9847.2
FedCFC-fProx	HSAC	37	3217.1	52	5478.3	59	8301.7
	NTU RGB+D	47	6451.1	64	7101.7	70	10201.6
	AMIGOS	35	3108.3	40	6828.2	59	6922.2
FedCFC-fSCAFFOLD	HSAC	39	3318.7	49	3988.2	57	5123.2
	NTU RGB+D	51	4122.7	54	4796.4	69	6125.0
	AMIGOS	33	2788.4	36	3072.8	48	4946.4
LotteryFL	HSAC	57	1111.5	61	1188.3	81	1569.7
	NTU RGB+D	72	1287.3	70	1301.7	91	1688.0
	AMIGOS	49	984.9	46	912.6	76	1463.2

number of communication rounds during the FL process. We only plot the results of three approaches for clarity of the figure. The smoothness of the trajectories is negatively affected by the class heterogeneity among the clients. Nevertheless, the trajectory of FedCFC-fBind is smoother than others. The bottom row shows the test accuracy after FL converges. Here, we assume that the convergence is achieved when all clients' validation accuracy improvement is less than 0.05% of the maximum validation accuracy for the first time. In Fig. 8, the horizontal red lines represent the test accuracy when CFC is trained in the centralized manner. It shows that when the data distribution is IID, the difference between the accuracy of centralized training and the accuracy of FedCFC-fBind is within 6%. In addition, FedCFC-fBind achieves higher test accuracy on the three datasets, compared with all FedCFC variants and LotteryFL. FedCFC-fBind is also more robust to the class heterogeneity. For instance, on the AMIGOS dataset, FedCFC-fBind has a test accuracy drop of 3.04% when γ decreases from 1 to 0. In contrast, LotteryFL has a drop of 9.22%. Additionally, we increase the number of client to 50 and conduct the same experiment on the NTU RGB-D dataset. When γ is 0, 0.5, and 1, the average test accuracy of FedCFC-fBind only shows a drop of 3.9%, 3.19%, and 2.70%, respectively. Among all compared approaches, FedCFC-fAvg gives the lowest test accuracy, because FedAvg is a vanilla FL strategy without non-IID considerations. This also suggests that, specific designs in the FL strategy are needed to exploit the f's involatility. Compared with FedCFC-fBind, FedCFC-fgh does not show advantages although it additionally applies SCAFFOLD to federate g and h. This is due to the inherent difficulty in federating volatile models.

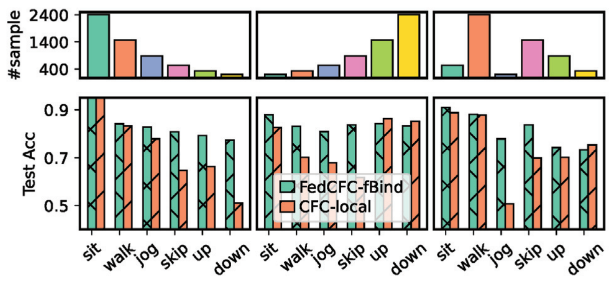


Fig. 9: Top: class distribution on three clients. Bottom: Corresponding class-specific test accuracy.

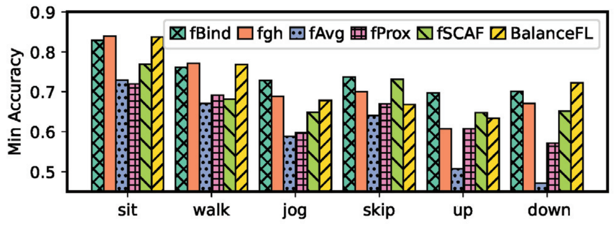


Fig. 10: Lowest class-specific test accuracy among ten clients.

Table 3 shows the number of communication rounds to achieve convergence and the associated per-client communication volume. We can see that FedCFC-fBind converges faster. In summary, FedCFC-fBind speeds up the convergence by 16.4%, 34.8%, and 45.5% with respect to the average of other three FedCFC variants on all datasets, when $\gamma = 1$, $\gamma = 0.5$, and $\gamma = 0$, respectively. The speed-up ratios with respect to LotteryFL are 35.5%, 33.4%, and 50.8%. As FedCFC-fBind only transmits f's last layer gradient, it is communication-efficient. FedCFC-fBind reduces per-client communication volume by 51×, 72×, and 96× with respect to the average of the other three FedCFC variants on all datasets, when $\gamma = 1$, $\gamma = 0.5$, and $\gamma = 0$, respectively. The reduction ratios with respect to LotteryFL are 11×, 12×, and 14×. The lower communication overhead of FedCFC-fBind makes it suitable for battery-based on-device FL because wireless transceiver is power-intensive.

5.3 Evaluation with Local Class Skews

We construct a subset of the HSAC-PAC2016 dataset using the data division algorithm in BalanceFL. The constructed dataset has global class imbalance characterized by a long-tail distribution. In addition, a total of 10 clients have distinct class distributions. The top row of Fig. 9 shows the class distributions of three clients. The bottom row of Fig. 9 shows the class-specific test accuracy of each client's CFC model trained by FedCFC-fBind or by the client solely with its local data without FL (referred to as CFC-local). The class-specific test accuracy of FedCFC-fBind is more consistent across the classes, compared with CFC-local. This is because, without FL, the model overfits to the classes with abundant samples.

Fig. 10 shows the lowest class-specific test accuracy among the 10 clients. Thus, it highlights the worst-case performance of each approach. FedCFC-fBind outperforms other approaches on 3 out of 6 classes. On the remaining classes, FedCFC-fBind is close to the best performing approach. Over all 6 classes, FedCFC-fBind

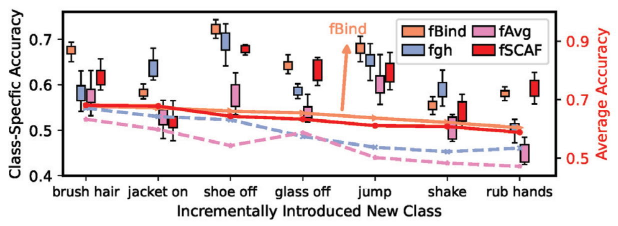


Fig. 11: Class-specific accuracy on a new class (depicted by error bar) and average accuracy on the original 53 classes (depicted by curve) versus the FL process depicted by x-axis with each tick representing a newly introduced class.

achieves 1.3% and 7.1% higher accuracy on average and on maximum, compared with BalanceFL. The per-client communication volumes of FedCFC-fBind, -fgh, -fAvg, -fSCAF, and BalanceFL in kB are 114.5, 2877.4, 3855.3, 3022.1, and 877.4, respectively. Thus, FedCFC-fBind reduces communication overhead by 7.6×, compared with BalanceFL.

5.4 Evaluation of Class-Incremental FL

We consider a scenario of class-incremental FL which learns new classes over time. Specifically, after a pre-trained CFC is deployed, the model is incrementally fine-tuned when training samples in new classes are obtained. For instance, users may wish to add their own types of exercise for the smartwatch to identify. Our experiments use the NTU RGB+D dataset with 60 classes. We use data in 53 classes for pre-training. Training samples in the remaining 7 classes are used during the FL process. We use two methods to test the accuracy of the local models when the training on a new class completes. Fig. 11 shows the results, where the x-axis represents the FL progress with the tics representing incrementally introduced new classes. In the first test method, we use test data from the same new class to check whether the local models capture the new class. The class-specific accuracy results are the error bars in Fig. 11 using the left y-axis, showing the distribution over the clients. In the second method, we use test data from the original 53 classes to check whether the local models experience forgetting. The accuracy results averaged over all the 53 classes and all clients are the curves in Fig. 11 using the right y-axis. We can see that FedCFC can learn new classes and fBind outperforms other aggregators. Some forgetting effect can be observed. Specifically, the accuracy drops for the original 53 classes are 7.1% for fBind and 14.3%, 17.2%, 10.1% for other aggregators. Future work can investigate training new columns of f, g, and h to prevent the forgetting, as inspired by the progressive neural networks [39].

5.5 Evaluation with Precision Heterogeneity

IoT devices, especially constrained by limited power supply or computing resources, may have to adopt data representation formats with lower precision to manage resource utilization. Such data precision heterogeneity imposes a challenge to on-device FL involving diverse IoT devices. This is because the precision disparities can lead to inconsistent loss landscape. We investigate the impact of the data precision heterogeneity on various FedCFC variants.

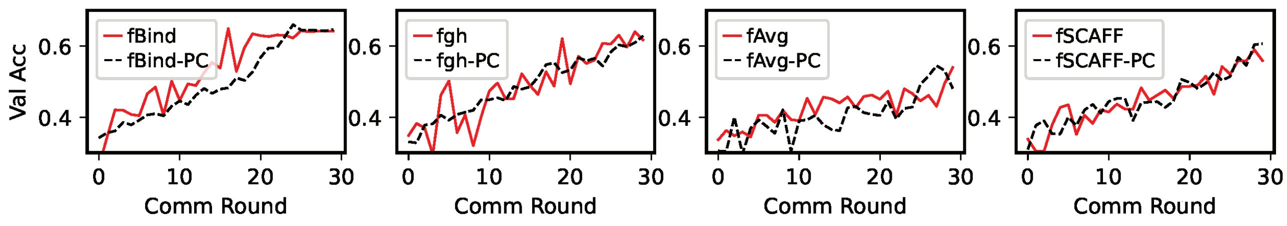


Fig. 12: Training accuracy vs. communication rounds with data precision heterogeneity. Suffix "-PC" denotes precision cluster.

We recreate a scenario of using FL to fine-tune the models that have been deployed on IoT devices. We train a base CFC model represented with float32 data type on a workstation using 10% of the NTU RGB+D dataset. Then, we quantize the model to float16 and int8 data types. We deploy each of the float32, float16, and int8 models to four clients. To deal with data precision heterogeneity, we introduce a mechanism called *precision cluster*. It organizes the clients into clusters based on their data precision. Only the clients with the same data precision are included into the same FL process. Therefore, when precision cluster is applied, there are three FL groups in our experiments.

Fig. 12 shows the training accuracy versus the number of communication rounds during the FL process. When precision cluster is not applied, FedCFC-fBind converges to the highest training accuracy, compared with the other three FedCFC variants. However, its trajectory is bumpier than that in Fig. 8(a). This suggests that the data precision heterogeneity negatively affects FedCFC-fBind. When precision cluster is applied, the trajectory labeled with the "-PC" suffix is the average result of the three FL groups. We can see that the precision cluster mechanism smooths the trajectory. Moreover, it does not reduce the accuracy after the FL converges. Therefore, the precision cluster is a simple yet effective approach to deal with data precision heterogeneity.

5.6 Sensitivity on Regularization Coefficients

We vary the three regularization coefficients, λ_d, λ_m, and λ_a and analyze their impacts on both the accuracy and computational cost. Specifically, we vary one of them individually from 0 to 0.8, while keeping the other two at 0.1. The experiment is conducted using the HSAC dataset with 12 clients. Computation time is measured on an Intel Core i7-11800H CPU. From Fig. 14(a), a non-zero regularization coefficient results in a model accuracy increase of around 5%. In addition, as shown in Fig. 14(b), the computation time per epoch remains largely unaffected with variations within approximately 10 to 16 milliseconds when a regularization coefficient changes.

6 ON-DEVICE EXPERIMENTS

6.1 Implementations and Deployments

We implement FedCFC-fBind on the following four MCU-based platforms shown in Fig. 13: (1) Arduino Nano BLE Sense, (2) Arduino Nano ESP32, (3) Adafruit M4 Express, (4) STM32F303. As these platforms have different resource constraints, we choose proper architectures for CFC's three FNN modules for each platform to make sure the CFC model can fit in, while maintaining the dimension of hidden state **x** the same. We pre-train the CFC with a subset of the used dataset. Then, we perform int8 quantization using STM32 CUBE.AI for STM32F303 and TensorFlow Lite Micro for the remaining platforms. TensorFlow Lite Micro does not support on-device training because it does not implement automatic differentiation. To realize on-device training, we implement the CFC model and model weight update algorithm for the client in the C language. On the server side, we employ the Splitting Conic Solver (SCS) [32] of the CVXPY library to solve the semidefinite programming problem and compute the Common Update Direction (CUD). The theoretical upper bound of the complexity of solving semidefinite programming problems with SCS [32] is $O((|C|+1)^M)$, where $|C|$ is the number of clients and M is a constant related to SCS solver's internal mechanism. Therefore, the computation complexity on the server side is polynomial with respect to the number of clients $|C|$.

When deploying the FedCFC-fBind implementation on a platform, we store CFC in a heap. Although accessing heap is relatively slow, using heap prevents memory overflow problems because the program can directly modify heap content without saving intermediate values. After assigning memory space for loading the program and CFC, we allocate one third of the remaining memory for storing intermediate variables for the gradient computation and model update algorithms, and two thirds for all other supporting functions such as wireless or UART communications. Table 4 summarizes the MCU specification (clock rate and SRAM capacity), the number of layers of CFC's three modules, and CFC model size.

6.2 Experiment Results

The first set of experiments are conducted on a network of 10 client nodes (7 Nano Sense nodes, and one node from each of the other three platforms) and a laptop computer as the server node. The communications of Nano Sense, Nano ESP32, and the remaining two platforms with the laptop computer are via BLE, Wi-Fi, and UART, respectively. We use the UCI-HAR dataset and 40% of it for pre-training CFC. We measure the test accuracy of the initial pre-trained and quantized CFC, the average test accuracy of the clients' CFC models at the end of the FL process, training time per communication round, and the average SRAM usage and utilization. Each input sample consists of 125 readings over 2.54 seconds. The results are presented in Table 4. The accuracy achieved by FedCFC-fBind on the four platforms ranges from 79% to 87% and is correlated with the model size and SRAM usage. Note that the highest accuracy achieved on the UCI-HAR dataset in literature is 90.6% [7]. The accuracy gaps of 3.6% to 11.6% are due to the mandatory compromises of using int8 quantization and smaller models to fit into small SRAMs. Note that the quantization can lead to a discrepancy between the direction of the model weight update and the

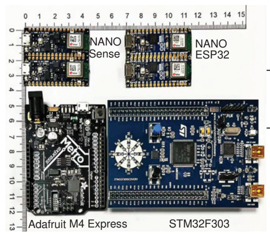
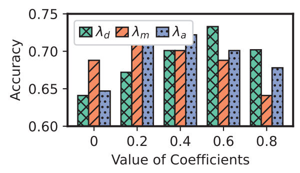


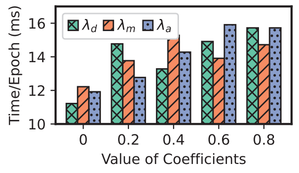
Fig. 13: Platforms.

Table 4: MCU specification and on-device training resource usage profile of FedCFC-fBind.

Platform	MCU rate (MHz)	SRAM (kB)	f,g,h #layer	CFC size (kB)	Init acc	End acc	Training time (ms)	Avg SRAM usage (kB)	Avg SRAM utilization
Nano Sense	64	256	2, 2, 2	6.7	0.73	0.87	373.13	149.76	59%
Nano ESP32	240	320	2, 2, 2	6.8	0.72	0.85	202.47	167.29	52%
M4 Express	120	192	2, 1, 1	4.6	0.72	0.84	311.37	139.88	73%
STM32F303	72	80	1, 1, 1	3.7	0.71	0.79	287.92	78.67	98%

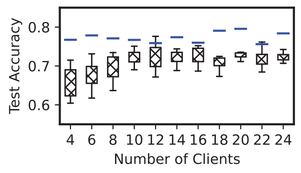


(a) Impact of regularization coefficients λ_d, λ_m, and λ_a on test accuracy.

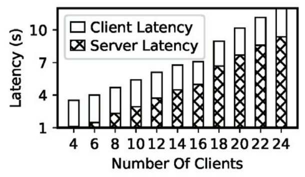


(b) Impact of regularization coefficients λ_d, λ_m, and λ_a on computation time per epoch.

Fig. 14: Impact of regularization coefficients λ_d, λ_m, and λ_a.



(a) Test accuracy of FedCFC-fBind with increasing number of clients.



(b) Client and server latency of FedCFC-fBind with increasing number of clients.

Fig. 15: Evaluation of the scalability of the FedCFC-fBind federated learning system in terms of accuracy and latency.

actual gradient, compromising optimality of training. Overall, our profiling experiments show that FedCFC-fBind has the adaptability and flexibility for deployments on diverse MCU-based platforms with 256 kB SRAM and even less.

The second set of experiments evaluate the impact of the number of clients (i.e., $|C|$) on learning performance. We use up to 24 Nano Sense nodes and the HASC-PAC2016 dataset. We use 20% of the training data for pre-training and allocate 10% of the remaining training data, randomly drawn, to each node for on-device FL. Each input sample has 240 readings. An error bar in Fig. 15(a) shows the distribution of the clients' test accuracy. The test accuracy shows an increasing trend when $|C|$ increases from 4 to 12, because the FL system accesses more data. Then, the system maintains accuracy when $|C|$ increases from 12 to 24. The blue lines in the figure represent the training accuracy obtained on the CPU of a workstation computer using centralized learning. We can see that the accuracy obainted on MCU has about 5% drop. Fig. 15(b) shows the breakdown of the time needed by the entire FL process into client part and server part. We can see a linear trend of the total training time versus $|C|$. The client's computation time is nearly constant. Differently, the server's computation time increases with $|C|$. This is because the time complexity of the aggregation algorithm presented in §4.4 increases with $|C|$. Parallelized implementation of the aggregator might suppress this increase. We also measure the STM32F303 MCU's energy consumption profile. In its idle state, it draws a current of 14.3 mA. During the training phase, the MCU reaches a maximum current of 25.7 mA. In the testing phase, the average current is about 22.4 mA. Operating on a 100 mAh battery, the projected lifetime for back-to-back inference activities on the STM32F303 MCU is about 268 minutes.

Lastly, we conduct the class-incremental FL experiment described in §5.4 on a network of 10 Nano Sense nodes. Compared with the simulation results presented in Fig. 11, the on-device implementation has about 5% class-specific accuracy drop on new classes and 1% average accuracy drop on the original 53 classes, which are primarily attributed to the model quantization.

7 DISCUSSION

From Table 4, in our current design, some devices underutilize their SRAM resources, while others nearly use up their SRAM. A future work of designing an *adaptive learning* feature within the CFC framework is interesting. This feature dynamically adjusts the computation complexity and resource usage of the learning process, tailoring to the specific capability of a device. Relationship between CFC's structure and model accuracy can be explored for optimizing both the efficiency and accuracy of CFC implementations.

Building *quantization awareness* in CFC is also interesting for future work. Quantization-aware training involves simulating the lower precision arithmetic during the training process, allowing the model to adapt to the quantization-induced perturbations. Different MCUs require different quantization approaches [9]. As a result, quantization-aware training can balance trade-off between model complexity and the inherent limitations of edge devices.

Deploying sensors in real-world environments and conducting on-device model training may face a number of practical challenges. For example, the unpredictable nature of network delays and sensor failures can result in varying quantities of data collected by different sensors. Furthermore, the datasets collected by sensors may have differences in the data distribution, in addition to the differences in the label distribution. This may further challenge the convergence of FL.

8 CONCLUSION

This paper demonstrates the appealing features of CFC neural networks for resource-constrained IoT devices in processing time series data. The proposed FedCFC advances CFC from the centralized learning setting to the federated learning paradigm. The design of

FedCFC is based on an empirical property of CFC identified in this paper, i.e., its f sub-network's involatility with respect to training data's class distribution. This paper develops a novel geometric aggregation strategy called fBind to deal with the clients' local class skews problem. Extensive evaluation and on-device experiments show superior performance of FedCFC-fBind and its portability to low-end devices with 256 kB memory and even less.

ACKNOWLEDGMENTS

This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (RG88/22).

REFERENCES

[1] 2012. Human Activity Recognition Using Smartphones. UCI Machine Learning Repository. https://doi.org/10.24432/C54S4K
[2] Nachman Aronszajn. 1950. Theory of reproducing kernels. *Transactions of the American mathematical society* 68, 3 (1950), 337–404.
[3] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018. Neural ordinary differential equations. *NeurIPS* (2018).
[4] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS Workshop on Deep Learning* (2014).
[5] Liam Collins, Hamed Hassani, Aryan Mokhtari, and S. Shakkottai. 2021. Exploiting shared representations for personalized federated learning. *PMLR* (2021).
[6] Moming Duan, Duo Liu, Xianzhang Chen, Yujuan Tan, Jinting Ren, Lei Qiao, and Liang Liang. 2019. Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications. *ICCD* (2019).
[7] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, Xiaoli Li, and Cuntai Guan. 2021. Time-series representation learning via temporal and contextual contrasting. *arXiv* (2021).
[8] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. 2020. Personalized federated learning: A meta-learning approach. *arXiv* (2020).
[9] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. *Low-Power Computer Vision* (2022).
[10] A. Graves, N. Jaitly, and A. Mohamed. 2013. Hybrid speech recognition with deep bidirectional LSTM. In *Autom. Speech Recognition & Understanding Workshop*.
[11] Ramin Hasani, Mathias Lechner, Alexander Amini, Lucas Liebenwein, Aaron Ray, Max Tschaikowski, Gerald Teschl, and Daniela Rus. 2022. Closed-form continuous-time neural networks. *Nature Machine Intelligence* 4, 11 (2022).
[12] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. 2021. Liquid time-constant networks. *AAAI* (2021).
[13] Haruyuki Ichino, Katsuhiko Kaji, Ken Sakurada, Kei Hiroi, and Nobuo Kawaguchi. 2016. HASC-PAC2016: large scale human pedestrian activity corpus and its baseline recognition. *UbiComp* (2016).
[14] Eunjeong Jeong, Seungeun Oh, Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. 2018. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *arXiv* (2018).
[15] Jeya Vikranth Jeyakumar, Liangzhen Lai, Naveen Suda, and Mani Srivastava. 2019. SenseHAR: A Robust Virtual Activity Sensor for Smartphones and Wearables. *SenSys* (2019).
[16] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. 2022. Model pruning enables efficient federated learning on edge devices. *IEEE Trans. Neural Netw. Learning Syst.* (2022).
[17] Alistair Johnson, Tom Pollard, Lu Shen, Li-wei Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Celi, and Roger Mark. 2016. MIMIC-III, a freely accessible critical care database. *Scientific Data* 3 (05 2016).
[18] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. 2020. Scaffold: Stochastic controlled averaging for federated learning. *ICML* (2020).
[19] C. Koch and I. Segev. 1998. *Methods in neuronal modeling.* MIT press.
[20] LM Lapicque. 1907. Recherches quantitatives sur l'excitation electrique des nerfs. *J Physiol Paris* 9 (1907), 620–635.
[21] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. 2015. A simple way to initialize recurrent networks of rectified linear units. *arXiv* (2015).
[22] Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. 2021. Lotteryfl: Empower edge intelligence with personalized and communication-efficient federated learning. *Symp. Edge Comput.* (2021).
[23] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, A. Talwalkar, and V. Smith. 2020. Federated optimization in heterogeneous networks. *MLSys* (2020).
[24] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. 2022. On-device training under 256kb memory. *NeurIPS* 35 (2022), 22941–22954.
[25] Hansi Liu, Abrar Alali, Mohamed Ibrahim, Bryan Bo Cao, Nicholas Meegan, Hongyu Li, Marco Gruteser, Shubham Jain, Kristin Dana, Ashwin Ashok, Bin Cheng, and Hongsheng Lu. 2022. Vi-Fi: Associating Moving Subjects across Vision and Wireless Sensors. *IPSN* (2022).
[26] Akhil Mathur, Tianlin Zhang, Sourav Bhattacharya, Petar Velickovic, Leonid Joffe, Nicholas D Lane, Fahim Kawsar, and Pietro Lió. 2018. Using deep data augmentation training to address software and hardware heterogeneities in wearable and smartphone sensing devices. *IPSN* (2018).
[27] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. 1273–1282.
[28] Juan Abdon Miranda-Correa, Mojtaba Khomami Abadi, Nicu Sebe, and Ioannis Patras. 2018. Amigos: A dataset for affect, personality and mood research on individuals and groups. *IEEE Trans. Affective Comput.* 12, 2 (2018), 479–493.
[29] Vaishnavh Nagarajan and J Zico Kolter. 2019. Generalization in Deep Networks: The Role of Distance from Initialization. *arXiv* (2019).
[30] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. 2016. Phased lstm: Accelerating recurrent network training for long or event-based sequences. *NeurIPS* (2016).
[31] Pierre-Emmanuel Novac, Ghouthi Boukli Hacene, Alain Pegatoquet, Benoit Miramond, and Vincent Gripon. 2021. Quantization and deployment of deep neural networks on microcontrollers. *Sensors* 21, 9 (2021), 2984.
[32] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. 2016. Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding. *Journal of Optimization Theory and Applications* (2016).
[33] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. 2021. ClusterFL: A Similarity-Aware Federated Learning System for Human Activity Recognition. *MobiCom* (2021).
[34] Xinchi Qiu, Titouan Parcollet, Javier Fernandez-Marques, Pedro PB de Gusmao, Yan Gao, Daniel J Beutel, Taner Topal, Akhil Mathur, and Nicholas D Lane. 2023. A first look into the carbon footprint of federated learning. *J. Mach. Learn. Res.* 24 (2023), 129–1.
[35] Amirhossein Reisizadeh, Isidoros Tziotis, Hamed Hassani, Aryan Mokhtari, and Ramtin Pedarsani. 2022. Straggler-resilient federated learning: Leveraging the interplay between statistical accuracy and system heterogeneity. *IEEE J. Sel. Areas Inf. Theory* 3, 2 (2022), 197–205.
[36] Attila Reiss, Ina Indlekofer, Philip Schmidt, and Kristof Van Laerhoven. 2019. Deep PPG: Large-scale heart rate estimation with convolutional neural networks. *Sensors* 19, 14 (2019), 3079.
[37] Beanbonyka Rim, Nak-Jun Sung, Sedong Min, and Min Hong. 2020. Deep learning in physiological signal data: A survey. *Sensors* 20, 4 (2020), 969.
[38] David W Romero, Anna Kuzina, Erik J Bekkers, Jakub Mikolaj Tomczak, and Mark Hoogendoorn. 2021. CKConv: Continuous Kernel Convolution For Sequential Data. *ICLR* (2021).
[39] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv* (2016).
[40] SS Saha, SS Sandha, and M Srivastava. 2022. Machine Learning for Microcontroller-Class Hardware: A Review. *IEEE Sensors Journal* 22, 22 (2022).
[41] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. 2014. A dataset and taxonomy for urban sound research. *ACM MM* (2014).
[42] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. 1998. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation* 10, 5 (1998), 1299–1319.
[43] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. 2016. NTU RGB+ D: A Large Scale Dataset for 3D Human Activity Analysis. *CVPR* (2016).
[44] Xian Shuai, Yulin Shen, Siyang Jiang, Zhihe Zhao, Zhenyu Yan, and Guoliang Xing. 2022. BalanceFL: Addressing class imbalance in long-tail federated learning. *IPSN* (2022).
[45] Alysa Ziying Tan, Han Yu, Lizhen Cui, and Qiang Yang. 2022. Towards personalized federated learning. *IEEE Trans. Neural Netw. Learning Syst.* (2022).
[46] Mohamed-Ali Tnani, Michael Feil, and Klaus Diepold. 2022. Smart Data Collection System for Brownfield CNC Milling Machines: A New Benchmark Dataset for Data-Driven Machine Monitoring. *Procedia CIRP* 107 (2022), 131–136.
[47] Saverio Vito. 2016. UC Irvine Machine Learning Repository: Air Quality. https://doi.org/10.24432/C59K5F.
[48] Lixu Wang, Shichao Xu, Xiao Wang, and Qi Zhu. 2021. Addressing class imbalance in federated learning. *AAAI* (2021).
[49] Xin Wang and Edward K Blum. 1992. Discrete-time versus continuous-time models of neural networks. *J. Computer and System sciences* 45, 1 (1992), 1–19.
[50] Stephen R Wicks, Chris J Roehrig, and Catharine H Rankin. 1996. A dynamic network simulation of the nematode tap withdrawal circuit: predictions concerning synaptic function using behavioral criteria. *J. Neuroscience* 16, 12 (1996).
[51] Huatao Xu, Pengfei Zhou, Rui Tan, and Mo Li. 2023. Practically Adopting Human Activity Recognition. *MobiCom* (2023).
[52] Hang Yan, Qi Shan, and Yasutaka Furukawa. 2018. RIDI: Robust IMU double integration. *ECCV* (2018).
[53] Yu Zhang, Tao Gu, and Xi Zhang. 2020. a ChainSGD-reduce Approach to Mobile Deep Learning for Personal Mobile Sensing. *IPSN* (2020).