Eco-FL: Adaptive Federated Learning with Efficient Edge Collaborative Pipeline Training

Shengyuan Ye, Liekang Zeng, Qiong Wu, Ke Luo, Qingze Fang, Xu Chen*
School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
{yeshy8,zenglk3,wuqiong23,luok7,fangqz}@mail2.sysu.edu.cn,chenxu35@mail.sysu.edu.cn

ABSTRACT

Federated Learning (FL) has been a promising paradigm in distributed machine learning that enables in-situ model training and global model aggregation. While it can well preserve private data for end users, to apply it efficiently on IoT devices yet suffer from their inherent variants: their available computing resources are typically constrained, heterogeneous, and changing dynamically. Existing works deploy FL on IoT devices by pruning a sparse model or adopting a tiny counterpart, which alleviates the workload but may have negative impacts on model accuracy. To address these issues, we propose Eco-FL, a novel **E**dge **Co**llaborative pipeline based **F**ederated **L**earning framework. On the client side, each IoT device collaborates with trusted available devices in proximity to perform pipeline training, enabling local training acceleration with efficient augmented resource orchestration. On the server side, Eco-FL adopts a novel grouping-based hierarchical architecture that combines synchronous intra-group aggregation and asynchronous inter-group aggregation, where a heterogeneity-aware dynamic grouping strategy that jointly considers response latency and data distribution is developed. To tackle the resource fluctuation during the runtime, Eco-FL further applies an adaptive scheduling policy to judiciously adjust workload allocation and client grouping at different levels. Extensive experimental results using both prototype and simulation show that, compared to state-of-the-art methods, Eco-FL can upgrade the training accuracy by up to 26.3%, reduce the local training time by up to 61.5%, and improve the local training throughput by up to 2.6×.

CCS CONCEPTS

• **Computing methodologies** → **Parallel computing methodologies**; • **Computer systems organization** → **Distributed architectures**.

KEYWORDS

Federated learning, edge intelligence, pipeline parallelism, parallel processing

*Xu Chen is the corresponding author.

ICPP '22, August 29-September 1, 2022, Bordeaux, France

ACM ISBN 978-1-4503-9733-9/22/08…$15.00
https://doi.org/10.1145/3545008.3545015

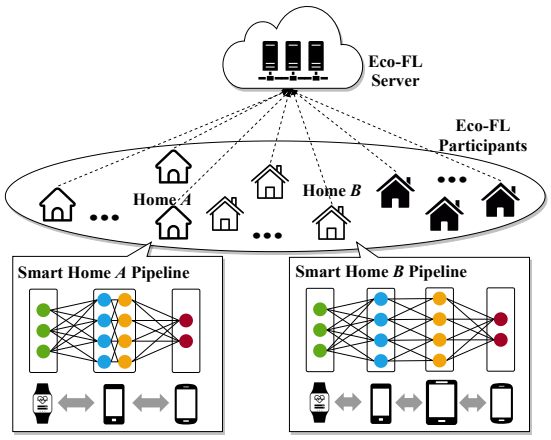


Figure 1: An illustration of Eco-FL in smart home scenario, in which each smart home as a participating client leverages an edge collaborative pipeline over in-home trusted devices to accelerate local model training in FL.

ACM Reference Format:
Shengyuan Ye, Liekang Zeng, Qiong Wu, Ke Luo, Qingze Fang, Xu Chen. 2022. Eco-FL: Adaptive Federated Learning with Efficient Edge Collaborative Pipeline Training. In *51st International Conference on Parallel Processing (ICPP '22), August 29-September 1, 2022, Bordeaux, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3545008.3545015

1 INTRODUCTION

Federated learning (FL) is an emerging learning paradigm that is able to exploit the distributed computing resources from multiple clients for training a mutual Machine Learning (ML) model. Specifically, in FL, each client provisions a local model counterpart and uses its own data to learn model parameters, while a central server aggregates updated parameters to train the global model and periodically synchronizes with clients. By employing a wealth of clients while physically isolating their private data and the central server, FL provides a scalable mechanism for training ML models without violating data de-sensitive protocols and is thus widely adopted in a wide range of privacy-concerned edge intelligence applications such as smart home and smart factory [19, 31].

Despite the advantages, applying FL in edge scenarios still suffers from the inherent shortage of edge computing resources. In particular, a significant yet inevitable issue is how to accelerate deep neural network (DNN) training on resource-constrained IoT devices, given its extremely computation-intensive and resource-demanding nature. While traditional federated learning systems

typically require participants to train a complete model locally, devices with restricted computing capability can fall short in their training speed, and may even fail to maintain a conventional training given their limited memory capacity. For instance, Andrew et al. [8] point out that participating in FL for keyboard prediction imposes a minimum available memory space of 2 GB. To address that, existing literature [11, 28] have utilized model compression techniques such as pruning and quantization, to reduce the computing and memory overhead for IoT devices. These methods, however, are all modifying the network architectures or model parameters, which may defect the model test accuracy as well as FL's training convergence. Alternatively, we observe that in typical edge scenarios like smart home, there are usually a number of trusted devices (e.g., owned by family members) in physical proximity that can share their idle resources with the participating client in FL. This therefore motivates us to take advantage of computing resources across edge devices to accelerate the FL's client training, namely *edge collaboration*.

To achieve that, we propose to leverage the pipeline parallelism to orchestrate the edge collaboration for model training acceleration. As illustrated in Fig. 1, for each smart home, we can split a DNN model in layer granularity, dispatch layer-wise training workloads to in-home domestic devices, and launch a distributed computing pipeline across them. While edge collaboration can mitigate the resource restriction, applying pipeline training over multiple edge devices is non-trivial, given the following challenges. 1) *Heterogeneity:* In practical deployment, the available trusted devices that a participating client can collaborate with usually vary, inducing issues of *system heterogeneity*. Some state-of-the-art works adopt grouping-based architecture [1, 2], which clusters clients with similar response latency into groups and performs semi-synchronous model aggregations. However, these grouping methods ignore the data distribution of each client, which may result in severe performance degradation due to unbalanced and non-independent and identically (non-IID) data distribution across groups, namely *data heterogeneity*. 2) *Dynamics*: IoT devices in edge deployments like smart home usually accomplish high variation in available resources, heavily depending on the usage habits and purposes of their users [13, 30]. Worse still, edge collaborative training may involve a great magnitude of IoT devices into the whole FL system, which further exacerbates the resources dynamics. Since the maximum performance of pipeline training is limited by the slowest device, fluctuation of available resources for IoT devices will severely affect the throughput of collaborative training as well as the response latency of the FL participating clients. This can largely disable the existing static optimization methods for FL and further affect the convergence performance of the global model.

To address these challenges, we propose **Eco-FL**, a novel **E**dge **Co**llaborative pipeline based **F**ederated **L**earning framework. Eco-FL envisions an edge environment where a client can have multiple trusted devices in proximity and they are willing to cooperate to accelerate local model training. This can be relevant in many edge scenarios such as smart home and smart factory, wherein the devices are managed by the same owner or possessed by trusted users. With such dependable resources, Eco-FL conquers the above problems on three levels. First, to orchestrate heterogeneous assisted devices in maximal resource utilization to facilitate local model training, Eco-FL maintains a pipeline by partitioning DNN models into successive segments and dispatching them among devices. In particular, Eco-FL designs a heterogeneity-aware workload partitioning mechanism with memory capacity constraints into consideration, running prior to deployment. Second, Eco-FL adopts a grouping-based hierarchical model aggregation mechanism, along with a novel grouping strategy that simultaneously accounts for clients' response latency and data distribution divergence. Last but not least, Eco-FL develops adaptive schemes to accommodate dynamics during the runtime. Specifically, a workload migration strategy is devised to alleviate the pipeline bottleneck under resource fluctuation, effectively retaining the training throughput even with external load spikes. A dynamic regrouping method is further employed to avoid the occurrence of stragglers in faster groups, maintaining a high aggregation frequency and training performance. The main contributions of the paper are summarized as follows.

- We devise a novel edge collaborative pipeline parallelism as a key mechanism to achieve edge resource pooling over trusted devices in proximity for local FL model training acceleration. To accommodate the heterogeneous and resource-constrained nature of edge devices, a resource-efficient pipeline strategy is developed, as well as a heterogeneity-aware workload partitioning and scheduling mechanism for efficient pipeline training.
- We propose Eco-FL, a hierarchical FL system framework that builds upon the edge collaborative pipeline model training. To tackle system heterogeneity and data heterogeneity exacerbated by edge collaboration, a novel grouping-based hierarchical aggregation mechanism is designed that jointly considers both the response latency and data distribution divergence.
- We feature adaptive scheduling in both FL server and client sides to tackle system dynamics inherent in edge scenarios, which allows flexible workload migration across edge devices for pipeline efficiency and dynamic re-grouping for FL training speed-up.
- We implement Eco-FL, and conduct extensive evaluations in both realistic testbeds and large-scale simulations. Experimental results show that Eco-FL can upgrade the training accuracy by up to 26.3%, reduce the local training time by up to 61.5%, and improve the local training throughput by up to 2.6× against state-of-the-art baselines.

2 BACKGROUND AND RELATED WORK

Federated learning. Federated Learning (FL) is an emerging distributed learning paradigm that can train a global, shared model without accessing clients' private data. Formally, the FL algorithm optimizes the objective function $F(w) = \sum_{c=1}^{N} \frac{|\mathcal{D}_c|}{|\mathcal{D}|} F_c(w)$, where $F_c(w) = \frac{1}{|\mathcal{D}_c|} \sum_{i \in \mathcal{D}_c} f_i(x_i, y_i; w)$ is the local loss function of client c and f_i is the loss function for data sample $\{x_i, y_i\}$. N is the total number of clients, $\mathcal{D}_c$ and $\mathcal{D}$ are the dataset of client c and the global datase, respectively. Despite the advantages in privacy preservation, FL's efficient deployment in IoT environments is yet up

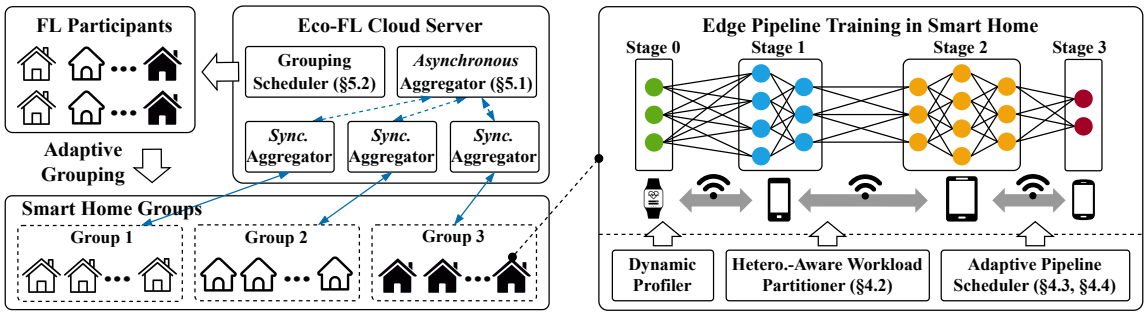


Figure 2: Overview of Eco-FL framework. Eco-FL adopts a hierarchical architecture in which each participant group (i.e., a cluster of smart homes) first performs fast synchronous aggregation intra-group, and then elastic asynchronous aggregations among the groups are performed to obtain the global model. At each smart home, an edge collaborative pipeline is constructed to orchestrate available trusted edge devices for local training acceleration.

against critical challenges incurred by system and data heterogeneity [12]. In particular, system heterogeneity in computing resources (such as memory capacity, hardware configurations, etc.) can exacerbate challenges such as the straggler problem, which is negative to the performance of synchronous updating. Data heterogeneity stems from the diverse distributions of IoT devices' data, where the non-IID characteristics can seriously harm the convergence of model training [21, 24, 25]. State-of-the-art methods usually adopt a hierarchical architecture [1–3] to conquer one of the heterogeneity issues, but a joint consideration in both system and data aspects still lacks exploration. Instead, Eco-FL proposes an adaptive client grouping strategy to alleviate the problem.

DNN training on edge devices. Training DNN models on edge devices can fully reserve data in situ, and is therefore employed in many privacy-sensitive applications. However, its computation-intensive and resource-demanding nature brings significant challenges to resource-constrained IoT devices. To address this, existing arts [11, 28] usually target a tiny model through model pruning and compression, which can reduce the workload yet largely decline the model accuracy undesirably. Meanwhile, the increased number of IoT devices has led to another scheme that parallelizes training across multiple devices [7, 29]. For instance, Hao et al. [7] adopts data parallelism training across embedded devices cluster, which requires each device to accommodate a replica of the entire model and frequently synchronizes their weights. Given the limited bandwidth between edge devices, such parallelism may induce considerable transmission overhead and slow down the complete training, which motivates us to leverage other collaborative mechanisms.

Pipeline parallelism. Pipeline parallelism is a training strategy that trains DNN model in a pipelined manner across multiple agents. By overlapping the communication and computation, it can achieve higher training throughput than traditional data and model parallelisms. In particular, PipeDream [20] proposes an asynchronous pipeline with a *one-forward-one-backward* strategy (1F1B-Async), where a forward pass is executed followed by a backward pass. While such an asynchronous strategy minimizes devices' idle time occupancy, it requires maintaining multiple historical versions of the model to avoid weights mismatching in forward and backward passes, which is unaffordable for memory-limited IoT devices. Gpipe [9] employs a synchronous pipeline with *backward-after-forward* strategy (BAF-Sync). Its pipeline accumulates the gradient of each micro-batch and updates the model after finishing all backward passes. This synchronous strategy does not require multi-version backups, but is still memory-consuming since the activations produced by forwarding tasks have to be kept for all micro-batches until backward tasks begin. Eco-FL instead tames this problem by designing a memory-efficient pipeline strategy with *one-forward-one-backward synchronous* training (1F1B-Sync), which is much more resource-friendly for the edge computing scenarios.

3 ECO-FL FRAMEWORK OVERVIEW

Fig. 2 depicts an overview of the proposed Eco-FL framework in smart home scenario. In principle, Eco-FL adopts a grouping-based hierarchical FL architecture that conducts fast model aggregation (§5.1) from FL participants with similar capability in the same group and next carries out elastic asynchronous model aggregations from the heterogeneous groups at the Eco-FL server. Particularly, the participants, *e.g.* smart homes in the figure, are divided into groups according to their response latency per training round as well as the distribution of their local data. The rationale behind that is to take system heterogeneity and data heterogeneity into joint consideration, such that the overall training convergence is ameliorated. This is accomplished by the grouping scheduler (§5.2), which is capable of generating grouping decisions and dynamically adapts during the runtime.

As illustrated in Fig. 2, to accelerate the local model training at each smart home, we exploit trusted and idle edge devices within the home, and leverage a resource-efficient paradigm of pipeline parallelism to orchestrate their execution flow (§4.1). Specifically, the DNN model is partitioned into a series of successive layer segments, which are dispatched to multiple devices to constitute the pipeline. To bridge the pipeline with the overall FL flow, we select one of the devices in the collaboration to be the **portal node** that is responsible for scheduling the pipeline, collecting updated models, and communicating with the Eco-FL server. Within the pipeline, a workload partitioner (§4.2) is employed to align model layers to diverse devices' capabilities by considering computation and communication heterogeneity, aiming at maximizing the training

throughput. Besides, a pipeline scheduler is further designed that manages pipeline configurations such as devices' order and input batch size (§4.3), such that overall resource utilization is maximized. Runtime dynamics, *e.g.* resource fluctuation due to the external load, is also accounted for by applying dynamic profiling and workload migration (§4.4). We note that different from traditional FL where a single device acts as a participant, Eco-FL targets at a novel scenario wherein each participant is a whole smart-home with multiple trusted and collaborative devices. In this case, data are transmitted locally only within one home instead of multiple homes, and hence privacy protection of Eco-FL is still guaranteed. With the above modules, Eco-FL focuses on the following design goals:

- A grouping-based hierarchical FL aggregation mechanism that can simultaneously tackle system heterogeneity and data heterogeneity for fast training convergence and upgraded test accuracy.
- An efficient collaborative pipeline for local model training that is able to accommodate the heterogeneous device resource constraints in the edge computing environment.
- An adaptive scheduling design that is capable of coping with dynamic edge resource variations, by re-organizing the training pipeline and participants' grouping.

We next explain the design details that attain the goals, in terms of the local training pipeline modules (§4) and FL aggregation modules (§5).

4 COLLABORATIVE EDGE TRAINING VIA PIPELINE PARALLELISM

4.1 Resource-Efficient Pipeline Strategy

To accelerate the local model training in Eco-FL, a participant client can leverage multiple available and trusted devices in proximity (e.g., smartphones and laptops at home) to perform collaborative model training. To exploit available resources from multiple devices, existing arts [7, 10, 29] usually apply data parallelism and model parallelism. While these traditional wisdom act well for a cluster of powerful machines with high-speed links (e.g., GPU/TPU accelerators in a datacenter), it can fall short in edge computing environments, where edge devices are typically loosely coupled with wireless connections [31]. The large volume of intermediate data exchange in data and model parallelism can thus severely bottleneck the entire training flow within devices' synergy. As we will show later in §6.3, the transmission overhead can occupy 66.29% in data parallelism and even perform worse in throughput metrics than single device training. This motivates us to leverage pipeline parallelism (as illustrated in Fig. 2), which segments DNN models in layer granularity and dispatches workload among devices in a successive manner. By judiciously partitioning layers and overlapping communication with computation, pipeline parallelism can effectively hide the transmission overhead and boost training throughput.

Nonetheless, applying pipeline parallelism over edge devices is non-trivial, given their inherent resource-constrained nature, e.g., limited in processors' capability and memory capacity [31]. For pipeline execution across these devices, we highlight memory usage as a crucial knob, which stringently restricts the size of training batches and the concurrent resident activations, and thus bounds the system throughput. This therefore requires a tailored memory-efficient optimization to match edge devices' available resources. To tackle such an issue, we adopt a novel synchronous pipeline strategy that works in a *one-forward-one-backward* (**1F1B-Sync**) fashion . Specifically, as shown in Fig. 3, 1F1B-Sync splits a mini-batch into several **micro-batches**, and injects them concurrently into the pipeline for maximizing the degree of parallelism. We name the training process of one mini-batch as a **sync-round** and the synchronous model update as **pipeline flush**. DNN model are partitioned across the IoT devices with each device undertaking the *forward pass* (**FP**) and *backward pass* (**BP**) for only a subset of the global model. Meanwhile, 1F1B-Sync employs the early backward schedule to release the memory produced by forward tasks early for reuse, which helps memory-constrained devices accommodate more micro-batches to hit a high degree of parallelism. In contrast to PipeDream [20] and Gpipe [9] discussed in §2, 1F1B-Sync does not store historical model versions, and employs the early backward schedule to release the memory produced by forward tasks for space reuse. This enables 1F1B-Sync to be memory-efficient, and accords with the requirement of edge collaboration.

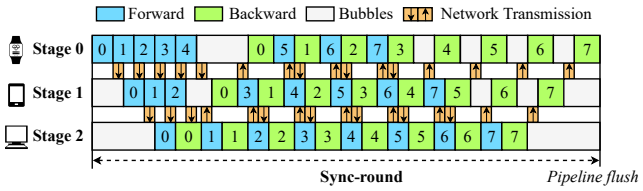


Figure 3: An instance of edge collaborative pipeline with three devices (i.e., stages).

4.2 Heterogeneity-Aware Workload Partitioning

To perform pipeline training, we need to divide the model layers into multiple segments such that each device executes a model segment as a pipeline **stage**. The global throughput of the pipeline is determined by the execution time of the slowest stage (**lagger**). The lagger will starve other faster stages and lead to an idle **bubble** (i.e., idle waiting time), resulting in resource under-utilization. Meanwhile, we need to consider communication delay inter-stage because although communication can be overlapped with computation, excessive communication delay can also degrade pipeline performance.

To partition the model into separate balanced stages, we leverage the idea of dynamic programming [20] and develop an optimized and tailored algorithm to support heterogeneous IoT environments by evenly distributing the DNN workload based on the computation capacity of heterogeneous devices. Specifically, our algorithm consists of two phases, namely profiling and workload partitioning.

Profiling. Pipeline profiler will monitor the total computation time across the FP and BP for layer l on d-th devices, denoted as T_l^d. While profiling the computation time, the profiler will also record the output activations, input gradients and weight parameters of

layer l in bytes, respectively denoted as a_l, ∂_l and w_l. We use B_n to indicate the bandwidth between device n and device $n+1$.

Workload partitioning. We denoted $D_n = \{d_0, d_1, ...d_{n-1}\}$ as the set of first n devices in pipeline. $A(i \rightarrow j, D_n)$ denote the time taken by the lagger in the optimal sub-pipeline between layer i to j with D_n devices. $T(i \rightarrow j, n)$ denotes the time taken by n-th device spanning layers i through j.

The goal of our algorithm is to find $A(0 \rightarrow L, |\mathbf{D}|)$, where L is the total number of layers in the model and $\mathbf{D}$ is the set of all IoT devices involved in pipeline parallelism. To solve this partitioning problem, we can break the optimal pipeline into sub-pipelines and optimize them recursively with the dynamic programming algorithm. The formula of the dynamic programming algorithm can be written as:

$$A(0 \rightarrow j, D_n) = \min_{1 \leqslant s < j} \max \begin{cases} A(0 \rightarrow s, D_{n-1}), \\ (a_s + \partial_s)/B_{n-2}, \\ T(s+1 \rightarrow j, n-1), \end{cases} \quad (1)$$

where $T(i \rightarrow j, n) = \sum_{l=i}^{j} T_l^n$. Ideally, the dynamic programming algorithm will output a partitioning point of the DNN model balancing both training workload on each stage and inter-stage communication delay.

4.3 Pipeline Orchestration

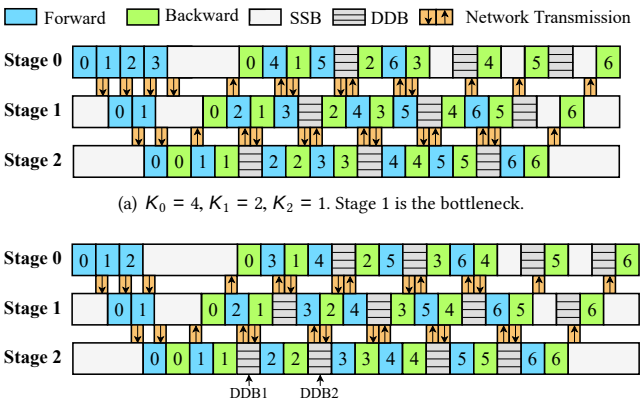
We next elaborate on the scheduling optimization for our pipeline. To simplify the difficulty of scheduling and modeling, following [6, 9, 20], we assume that: (1) our workload partitioning algorithm outputs a balanced pipeline, which means the execution time of micro-batches in each stage are equal. (2) Inter-stage communication delay of activations/gradients can be well overlapped by the FP/BP execution time of micro-batches, which is reasonable because we will not choose pipeline parallelism to train the DNN models with huge inter-stage activations.

Pipeline bubble analysis. To maximize the throughput of the pipeline, we want to minimize the idle bubble of each stage. We divide the idle bubbles in the synchronous pipeline into two types, as shown in Fig. 4. We name the first type of bubble as *Synchronous Static Bubble* (**SSB**). This type of bubble is caused by the periodic pipeline flush, which is inevitable in synchronous strategy. We name the second type of bubble as *Data Dependency Bubble* (**DDB**), which is caused by the data dependency of micro-batches in pipeline training.

Micro-batch scheduling to minimize bubbles. We denote S as the total number of stage, where $S > 1$. Let $T_{t,f}^s$ and $T_{t,b}^s$ denote the total time taken by stage s for FP and BP, respectively, where $s \in \{0, 1, ..., S-1\}$. $T_{c,f}^s$ and $T_{c,b}^s$ denote the communication time between stage s and $s+1$ in FP and BP, respectively. As shown in Fig. 3, we can observe that computation and communication in a sync-round form a trapezoid. Therefore, SSB in each stage are same and can be calculated by counting the bubble before the first FP and after the last BP in the last stage:

$$SSB = \sum_{s=0}^{S-2} (T_{t,f}^s + T_{t,b}^s + T_{c,f}^s + T_{c,b}^s). \quad (2)$$

We can reduce the proportion of the time SSB occupies by increasing the number of micro-batches injected in the pipeline concurrently (**M**) in a sync-round.



(a) $K_0 = 4, K_1 = 2, K_2 = 1$. Stage 1 is the bottleneck.

(b) $K_0 = 3, K_1 = 2, K_2 = 1$. Stage 0 is the bottleneck.

Figure 4: Examples of pipeline with different K_s.

DDB is originally caused by the data dependency of the tasks in the pipeline. Due to the limited memory available in IoT devices, we may not be able to execute all FP before executing BP as Gpipe does. Adopt 1F1B-Sync can reduce peak memory consumption but have the potential to cause more DDB and degrade the performance of the pipeline. Therefore, we need to schedule the pipeline to minimize memory usage while maintaining a high throughput. Actually, as shown in 3, if we well control the number of forward task residing concurrently in each stage, we can avoid the occurrence of DDB while releasing memory pressure of each stage.

To find a set $\{P_0, P_1, ..., P_{S-1}\}$ that can minimize both DDB and the peak memory usage of each stage. We purpose an iterative approach to calculate the P_s from the last stage:

$$P_{s-1} \geq P_s + \frac{T_{t,f}^s + T_{t,b}^s + T_{c,f}^s + T_{c,b}^s}{T_{t,f}^{s+1} + T_{t,b}^{s+1}}, \quad (3)$$

when $s = S-1$, $P_s = 1$. We derive this inequality based on ideal model of pipeline in Fig. 3. To eliminate DDB in stage s, we need to make sure that before stage $s-1$ finish and send the P_{s-1}-th FP to stage s, stage s is not starved for waiting task. Because we iteratively calculate from the last stage, there will be no DDB in sub-pipeline after stage s when we calculate P_{s-1}. We denote set $\{P_0, P_1, ..., P_{S-1}\}$ satisfying (3) as the optimal set of number of forward tasks residing concurrently on each stage in pipeline. According to (3), for some workloads where inter-stage communication delays can be ignored compared with forward and backward execution time, we calculate with $P_s = S - s$ for stage s. In IoT environment, limited bandwidth between IoT devices makes inter-stage communication delays non-negligible, thus we calculate with $P_s = 2(S-s) - 1$ for stage s.

Schedule devices' order in pipeline. To maximize the throughput of the pipeline, we want to pick up a devices' order resulting in the least sync-round time. If there are no DDB in the pipeline, we can estimate the time of a sync-round by the sum of M micro-batches total execution time and SSB. However, in practice, IoT devices with limited available memory even may not be able to accommodate enough forward tasks to avoid the occurrence of

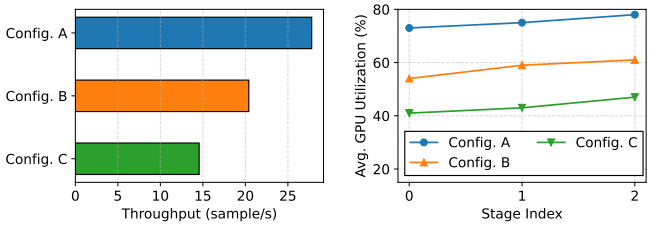


Figure 5: Pipeline performance of different configurations. Configuraton *A* means the devices' order of ⟨TX2, Nano, Nano⟩ with a micro-batch size (mbs) as 16, Configuration *B* is ⟨Nano, TX2, Nano⟩ with mbs = 8, and Configuration *C* is ⟨Nano, TX2, Nano⟩ with mbs = 16.

DDB. We denote $K_S = min(P_S, Q_S)$ as the actual number of forward tasks residing in stage S in pipeline run-time, where Q_S is the maximum number of FP that can be accommodated in the available memory of stage S. If any stage S can not accommodate P_S forward task, DDB will occur, as shown in Fig. 4.

As we know, the throughput of the pipeline is greatly dependent on the narrowest stage, namely **bottleneck**. As shown in Fig. 4(b), stage 0 is more of a bottleneck in the pipeline than stage 1 because of that even if stage 1 can hold three forward tasks concurrently (DDB 1 will be eliminated), stage 2 still need to wait for time DDB1+DDB2, since forward task 3 has not yet arrived. In Fig. 4(a), when $K_0 = 4$, stage 1 is more of the bottleneck of this pipeline than stage 1. We can empirically observe that the occurrence of DDB is periodic and related to the width of the bottleneck. Therefore, it is obvious that the proportion of DDB can not be reduced by injecting more micro-batches concurrently into a sync-round like SSB and these recurring DDB will generate periodically unavoidable idle bubbles in each stage, which can extremely degrade the performance of the global pipeline.

A smaller micro-batch size may allow each stage to accommodate more forward tasks. However, too tiny micro-batch size will result in the under-utilization of computational resources and may not well overlap the communication delay. We conduct an experiment on EfficientNet with a three-stage pipeline consisting of one TX2 and two Nano to inspect the pipline performance under different devices' orders, as shown in Fig. 5 (experimental settings are elaborated in §6.1). EfficientNet is composed of convolutional layers and large activations mostly concentrated in the front of networks. If we use Nano with smaller memory to undertake stage 0, we can only choose a smaller micro-batch size (Config. B) or a smaller K_0 while keeping the same batch size (Config. C). We observe that both Config. B and Config. C lead to a low GPU utilization and degrade the throughput of the pipeline.

To avoid the occurrence of DDB while maximize the size of micro-batch, we will start the search from a relatively large micro-batch size. If all devices' order can not satisfy $K_S = P_S$, we will appropriately reduce the micro-batch size until there exists a device order that can accommodate enough forward tasks in each stage.

4.4 Adaptive Pipeline Re-scheduling

Existing distributed on-device training usually statically partitions training workload among the training workers [6, 9, 20]. However, this static partitioning approach is not suitable for the IoT environment because of two reasons. First, IoT devices usually have high variation in available computing capability and memory resources. Second, the maximum throughput of the pipeline is greatly determined by the lagger, and fluctuation in the execution time of any stage will seriously affect the overall throughput of the pipeline.

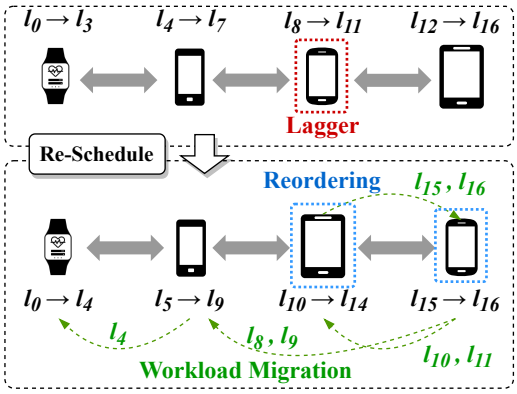


Figure 6: Towards the lagger in the pipeline, the scheduler boosts the pipeline performance by migrating workload and reordering devices.

To solve the above problems, we adopt an adaptive workload migration strategy. Each training worker will periodically report to the portal node the execution time of FP and BP. If the portal node detects a large deviation between the current and historical execution time of any device, it will adaptively re-schedule the pipeline and generate a new configuration including devices' order and workload partitioning point. All devices will do workload migration concurrently according to the new scheduling configuration, as shown in Fig. 6. After the workload migration is completed, the portal node will notify the head node in the pipeline to restart the training.

5 GROUPING-BASED HIERARCHICAL FL AGGREGATIONS

5.1 Hierarchical Federated Learning Mechanism

After obtaining the local models by edge collaborative pipeline training from the FL participating clients (e.g., smart homes), the remaining key issue is how to efficiently aggregate the local models to derive a global model in a fast manner. Nevertheless, the available computing capability that each client can collaborate with may vary dramatically in practice, which could induce severe system heterogeneity issues. Existing synchronous FL solutions [17, 21] that synchronously aggregate clients' model updates can achieve high training accuracy, but the slowest client can significantly prolong the training time when stragglers occur. Alternatively, asynchronous FL approaches [27] can effectively alleviate the straggler issue, but it only aggregates the updated model from one client at a time, introducing biases while sacrificing the global model accuracy and convergence speed. To combine the best of both mechanisms, we propose to adopt a hierarchical scheme with clients separated into

multiple groups. Specifically, as described in the left part of Fig. 2, the aggregation is performed in two levels: first a synchronous aggregation is applied to aggregate model updates from the clients within the same group, and next an asynchronous aggregation is made for global model aggregation among different groups.

Specifically, we denote that the set of all clients are classified into $|\mathcal{G}|$ groups: $\mathcal{G} = \{\partial_0, \partial_1, ..., \partial_{|\mathcal{G}|-1}\}$ and we denote the set of clients in group ∂ as C^∂. In each synchronous round, a client $c \in C^\partial$ will perform local training using the pipeline strategy above with its collaborative devices, and then synchronizes with sync-aggregator after every e steps of local updates. We denote $w_c^\partial(t_c)$ as the local model of client c after t_c-th update and $w^\partial(t_\partial)$ as the group model after t_∂-th update. At the beginning of each synchronous round, both t_c and t_∂ are initialized with 0 and $w^\partial(t_\partial)$ is equivalent to the current global model. $\mathcal{D}_c^\partial$ and $\mathcal{D}^\partial$ are denoted as dataset of client c and aggregated dataset under group ∂, respectively. The intra-group synchronous model aggregation process is as follow:

$$w_c^\partial(t_c) = \begin{cases} w_c^\partial(t_c-1) - \eta \nabla h_c^\partial\ w_c^\partial(t_c-1), & t_c \bmod e \neq 0, \\ \sum_{c \in C^A} \frac{|\mathcal{D}_c^A| [w_c^A(t_c-1) - \eta \nabla h_c^A(w_c^A(t_c-1))]}{|\mathcal{D}^A|}, & t_c \bmod e = 0, \end{cases}$$

where $h_c^\partial\ w_c^\partial(t_c) = F_c^\partial\ w_c^\partial(t_c) + \frac{\mu}{2}\ w^\partial\ t_\partial - w_c^\partial(t_c)$. Following FedProx[21], we add a proximal term to the local loss function F_c^∂ to alleviate the data heterogeneity issue during the intra-group synchronous process. For the asynchronous model aggregation process from the groups, we utilize FedAsync [27]: In the k-th global update, async-aggregator receives a new group model w_{new}^∂ from an arbitrary group ∂, and updates the global model by weight averaging: $w(k) = (1-\alpha)w(k-1) + \alpha w_{new}^\partial$, where $\alpha \in (0,1)$ is the mixing hyperparameter which is related to the staleness of group models and can be fine-tuned globally.

5.2 Adaptive Client Grouping

The divergence of grouping strategy and the data distribution of clients will have a significant impact on the global model convergence and accuracy [3, 18]. Based on the proposed hierarchical architecture, Eco-FL adopts a heterogeneity-aware client grouping strategy considering both response latency and data distribution to strike a balance between system and data heterogeneity. Due to the high variation of the response latency of participating clients, the Eco-FL server will continuously collect the response latency of all clients and dynamically re-schedule group members to maintain a decent throughput of the global FL system. Particularly, our adaptive grouping strategy consists of three phases: profiling, initial grouping, and dynamic re-grouping.

Profiling. Eco-FL server will profile the response latency and data distribution of each client, denoted as $\{L_0, L_1, ...L_{N-1}\}$ and $\{\pi_0, \pi_1, ..., \pi_{N-1}\}$, respectively. Response latency of a client is composed of the computing latency for local training and the communication latency with the Eco-FL server. The data distribution of a client records the proportion of different labels on the local dataset.

Initial grouping. After profiling the response latency and data distribution of each client, the Eco-FL server will start to group clients. The principle of grouping is to let the response latency of the members in the group be as close as possible while having an associated data distribution as close as possible to the IID distribution π_{iid}. We define a cost function (4) to strike a balance between response latency and data distribution:

$$\text{COST}_n^\partial = |L^\partial - L_n| + \lambda \text{JS}\ \pi_n^\partial, \pi_{\text{iid}}\ , \tag{4}$$

where L^∂ is the central response latency of group ∂. We use Jensen-Shannon (JS) divergence [16] based on Kullback-Leibler (KL) divergence to quantify the dissimilarity between IID distribution and union data distribution of group ∂ and client n. JS divergence has the advantage of symmetry and normalized values between $[0, 1]$ over KL divergence.

At the beginning of grouping, we use K-means algorithm [15] to cluster clients based on their response latency and denote L^∂ as the average value of each group ∂. We note that the clustering algorithm adopted and the number of groups to cluster can be properly adjusted according to the actual distribution of clients in Eco-FL. Then, the Eco-FL server will pick a client with the smallest cost for each group in turn until nobody in the client pool can be associated with any group. Every time we associate a new client with a group, we will update the current data distribution of the group and remove the client from the client pool. To avoid severe straggler issues, we can set a response latency threshold RT^∂ for each group ∂. If $|L^\partial - L_n| > RT^\partial$, we will not associate the client into group ∂, even if it has the smallest cost. Finally, the unassociated clients in the client pool will be dropped out temporarily, until their response latency returns to the threshold range of any group.

Dynamic re-grouping. As aforementioned, the response latency of each client can be varying greatly occasionally due to the significant changes in its collaborative device resources or the network conditions, which forces us to dynamically re-schedule group members accordingly during run-time to maintain a stable system performance. Therefore, the Eco-FL server will continuously collect and monitor the response latency of each client in run-time. If any

Algorithm 1: Adaptive Grouping Process

```
Process Eco-FLServer():
    Collect and monitor response latency of each client;
    if Client n in group ∂ satisfy |L^∂ − L_n| > RT^∂ then
        Regroup(n);
    end
Function Regroup(n):
    MinCost ← +∞, t ← −1;
    for ∂ ∈ {0, 1, ..., |G| − 1} do
        if COST_n^∂ < MinCost and |L^∂ − L_n| ≤ RT^∂ then
            t ← ∂;
            MinCost ← COST_n^∂;
        end
    end
    if t ≠ −1 then
        Move client n to group t;
    else
        Drop out client n until its response latency L_n meets
          the threshold range of any group;
    end
```

client n in any group ∂ satisfy $|L^{\partial} - L_n| > RT^{\partial}$, Eco-FL server will calculate the cost between the client and all groups based on (4). Then, this client will be recombined to the new group t with smallest cost if $|L^t - L_n| \leq RT^t$. If we can not find an appropriate group to accommodate the client, it will be temporarily dropped out as we mentioned before. We summarize our adaptive grouping process in Algorithm 1.

6 EVALUATION

6.1 Experimental Setting

We implement and evaluate our proposed Eco-FL system using both a realistic prototype and large-scale simulation. Specifically, we build a testbed with representative edge devices in Table 1, and use the recorded parameters in realistic experiments to assist the numerical FL simulation with hundreds of clients. We next elaborate on the detailed setup of them.

Experimental settings for FL training. We deploy the Eco-FL server and client worker on a virtual machine instance (48 vCPUs and 64GB memory) and each client gets assigned 2 CPU cores. We evaluate 300 clients in FL and for each training round, we will select at most 20 clients concurrently. The same DNN models are employed as in FedAVG [17] on CIFAR-10 [4], Fashion-MNIST [26] and MNIST [5] datasets. The hyperparameters follow the configuration: local epoch = 3, batch size = 10, learning rate = 0.0001, proximal term parameter in local subproblem $\mu = 0.05$. To simulate the system heterogeneity in a real IoT environment and the different computation resources that each client can utilize, we add different response delays to each client. We first sample the original response delay for each client from a normal distribution. Then, each client is randomly assigned a **collaborative degree** from $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ and the actual delay of each client is the product of the original delay and collaborative degree. We use K-means methods to associate them into 5 response latency groups (**RLG**). In a hierarchical FL system, there are two levels of non-IIDness. To simulate data heterogeneity in an IoT environment, we generate a non-IID data distribution for each client where the samples in each client are only assigned from two random classes. To verify the robustness of our grouping method, we further consider the following two non-IID cases for MNIST:

- **RLG-IID**: Assign clients to each RLG with all 10 kinds of classes and the data distribution of each RLG is IID distribution.
- **RLG-NIID**: Assign clients to each RLG with 3 classes of labels. This can be relevant in many realistic scenarios, e.g., businessmen of certain areas usually possess devices with higher computing capability and have similar behavioral characteristics [14].

To simulate the random variation of available computing resources in edge collaboration environments, we consider a **dynamic setting** where each client in Eco-FL periodically changes the collaborative degree in a determined probability.

Experimental settings for pipeline experiments. We evaluate our edge collaborative pipeline on realistic edge devices, where Table 1 summarizes the hardware configurations of them. Jetson Nano and Jetson TX2 collaboratively perform parallelism training. By adjusting the power mode of Nano and TX2, we can effectively simulate four kinds of heterogeneous devices with combinations of different computation capacities and available memory. For convenience, we use Nano-L and Nano-H to represent two power modes of Nano while TX2-Q and TX2-N indicate two power modes of TX2, respectively. Our evaluation is performed under a 100Mbps network environment to emulate a typical network condition in prevalent IoT environments [7]. Two widely-adopted DNN models on mobile IoT devices are employed:

Table 1: Specifications of the used edge devices.

Hardware	Power Mode	GPU Max Frequency	Memory	Network Bandwidth
Jetson Nano	5W (L) 10W (H)	640MHz 921.6MHz	4GB	100Mbps
Jetson TX2	Max-Q Max-N	850MHz 1.3GHz	8GB	100Mbps

- **EfficientNet [23]:** A convolutional neural network architecture adopts a compound scaling method that can be properly scaled up on both width and depth for better accuracy. In our experiment, we will adjust compound coefficient parameters (**B**) according to our requirements.
- **MobileNetV2 [22]:** A convolutional neural network architecture that is specifically tailored for IoT and resource-constrained environments. It has a tunable hyperparameter named width multiplier (**W**) to achieve accuracy/performance trade-offs.

Baselines. We compare Eco-FL with both the traditional baseline method and grouping-based hier-architecture method:

- **FedAvg [17]:** A baseline FL method with synchronous aggregation strategy. At each round, it randomly selects a certain ratio of clients for local model training and global model aggregation.
- **FedAsync [27]:** A baseline asynchronous federated learning algorithm that updates the global model without waiting for straggling clients.
- **FedAT [2]:** A hierarchical FL scheme that associates client into different tiers based on their response latency and performs synchronous intra-tier training and asynchronous inter-tier training.
- **Astraea [3]:** A hierarchical FL scheme that alleviates local data imbalance by grouping clients into clusters with balanced data. Since Astraea adopt a synchronous process on both global and group level, for a fair comparison, we only compare with its client grouping method.

6.2 Federated Learning Performance

Training Performance. Fig. 7 illustrates the performance of different training methods on CIFAR-10 and Fashion-MNIST datasets, all under the same dynamic settings. We observe that Eco-FL significantly outperforms the FedAvg and FedAsync baselines with faster convergence and higher achieved accuracy, which attributes to the hierarchical architecture of Eco-FL and the heterogeneity-aware adaptive client grouping strategy. Fig. 7 also witness that both Eco-FL without dynamic grouping strategy (w/o DG) and FedAT will

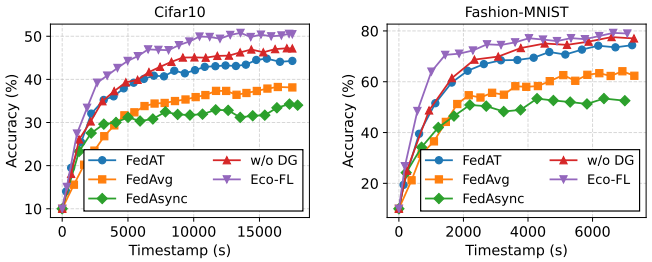


Figure 7: Training performance of Eco-FL and other FL approaches with different datasets.

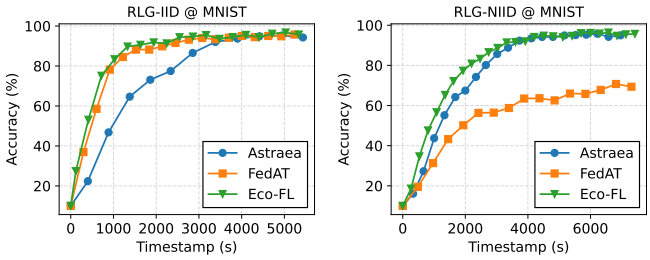


Figure 8: Training performance of Eco-FL and other grouping-based approaches under RLG-IID and RLG-NIID settings.

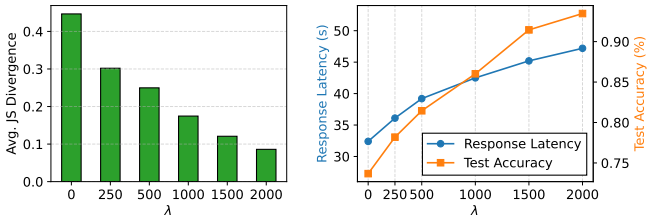


Figure 9: JS divergence, average response latency and test accuracy of global model under different λ.

suffer from straggler issues and cause distinct performance degradation under the dynamic setting. And the dynamic issue will be further magnified in a real scenario when more clients are involved in each training round. In contrast, Eco-FL with the adaptive scheduler can still maintain a decent performance under the dynamic setting because it will monitor clients' run-time response latency and dynamically adjust clients to appropriate groups. Regardless of the scheduler, Eco-FL without DG still outperforms FedAT by applying our heterogeneity-aware grouping strategy.

Effectiveness of grouping. Fig. 8 examines the effectiveness of Eco-FL's grouping strategy with other state-of-the-art grouping-based FL methods, by experiments on both RLG-IID and RLG-NIID settings with MNIST. On RLG-IID setting, both Eco-FL and FedAT can achieve a fair training performance since the data distributions across groups are relatively uniform and the synchronous process of faster groups will not be dragged down by straggler issues. Conversely, Astraea's grouping strategy ignores the response latency, which may combine the fast and slow clients into the same group and induce straggler problems. On RLG-NIID setting, FedAT groups clients only considering response latency, which leads to an extremely unbalanced data distribution across each group. Although FedAT assigns a higher weight to slower groups when updating the global model to relieve the biased toward the faster groups, it still shows poor convergence performance under severe imbalanced data distribution. Both Eco-FL and Astraea grouping methods achieve a comparable decent convergence performance because they both consider the data heterogeneous issue between each group. Specifically, Eco-FL can upgrade the training accuracy by up to 26.3% compared to FedAT. Moreover, for clients with similar data distribution, our method will preferentially select the client with response latency closer to the group center, which brings an faster convergence than Astraea. In essence, both response latency based grouping (e.g., FedAT) and data distribution based grouping (e.g., Astraea) can be considered as extreme cases of our proposed method when $\lambda = 0$ and $\lambda = +\infty$, respectively.

Sensitivity of λ. We further analyze the effect of parameter λ on the average response latency of each group and the training accuracy of the global model on RLG-NIID setting. As depicted in Fig. 9, with the value of λ increases, the average JS divergence across all groups decreases while the test accuracy of the global model increases. However, when λ increases, some stragglers may be involved in faster groups, leading to a higher average group response latency and slower convergence rate of the global model. Our heterogeneity-aware grouping methods can effectively strike a balance between model accuracy and convergence rate by tuning parameter λ according to different requirements in practice.

6.3 Pipeline Training Performance

Performance analysis. We analyze the performance of the proposed pipeline mechanism with realistic testbeds. We compare Eco-FL's pipeline parallelism with the data-parallel training (DP) method and single device training on EfficientNet and MobileNet with two-stage and three-stage pipelines. We use Nano-L collaborating with a Nano-H to form a 2-stage pipeline and two Nano-H collaborate with a TX2-Q to form a 3-stage pipeline. For a fair comparison, all methods share the same global mini-batch size for a certain DNN model. We use the largest micro-batch size to hit the peak throughput of each training method. For DP, we evenly distribute the workload to heterogeneous devices based on their training speed to minimize the overhead of each synchronous training round. Fig. 10 shows the training curves and Fig. 11 shows average training time per epoch of all methods. The results demonstrate the superiority of Eco-FL's pipeline design with faster training convergence and smaller epoch time. The low-speed links (100Mbps) between IoT devices translate DP's frequent gradient synchronization to prolonged idle waiting time, while pipeline parallelism can hide the transmission overhead by overlapping the computation and communication processes. For DP on MobileNet-W3, the time used for gradient synchronous communication is even longer than the training time in an epoch, which leads to severe under-utilization of devices and the throughput of DP is even far slower than training on a single TX-Q. On the contrary, our Eco-FL pipeline efficiently collaborates the computation power of all IoT devices and reaches the target accuracy 2.6× faster than DP.

Pipeline strategy. Table 2 shows the performance comparisons of our 1F1B-Sync with Gpipe's BAF-Sync pipeline planning. We

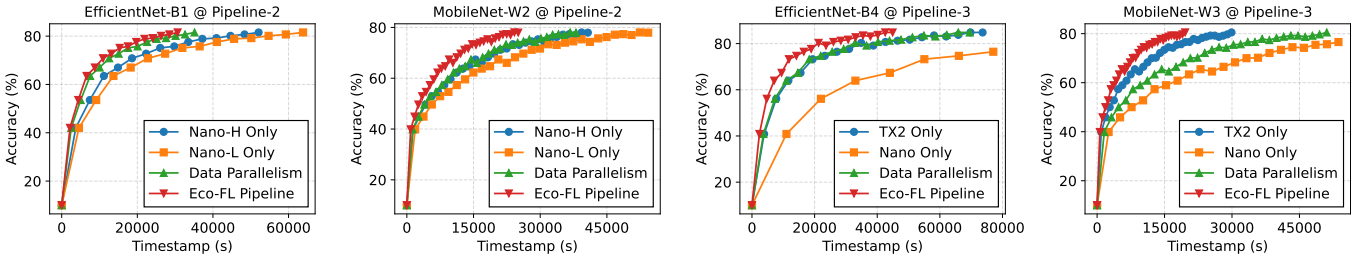


Figure 10: Training performance of EfficientNet and MobileNet with 2-stage and 3-stage pipeline.

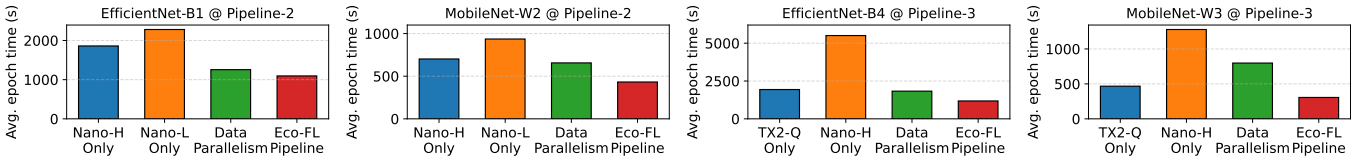


Figure 11: Average epoch time of EfficientNet and MobileNet with different training methods.

Table 2: Performance comparion with Gpipe.

Config.	# of Micro Batch	Avg. Peak GPU Memory (GB)		Avg. GPU Utilization	
		Stage 0	Stage 1	Stage 0	Stage 1
Gpipe	6	2.70	1.20	62.5%	67.8%
(mbs = 8)	8	-	OOM	-	-
Ours	8	1.70	0.92	69.5%	74.2%
(mbs = 8)	16	1.70	0.92	74.6%	78.4%
Ours	8	2.70	0.96	77.9%	79.5%
(mbs = 16)	16	2.70	0.96	83.6%	84.4%
Ours	8	4.60	1.00	86.0%	85.0%
(mbs = 32)	16	4.60	1.00	90.4%	89.5%

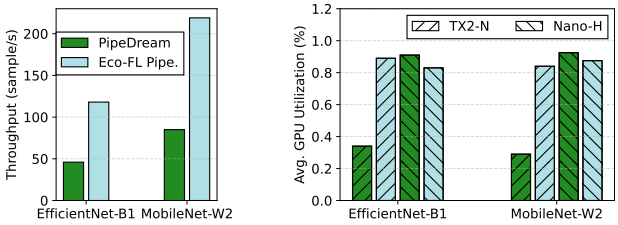


Figure 12: Performance comparison with PipeDream.

focus on the average GPU utilization and peak GPU memory usage on EfficientNet-B6 with a 2-stage pipeline consisting of a TX2-N and a Nano-H. When micro-batch size (**mbs**) fix 8, Gpipe can accommodate up to 6 micro-batch concurrently in pipeline, which lead to low GPU utilization 62.5% and 67.8% on stage 0 and 1, respectively. 1F1B-Sync planning adopts an early backward schedule to release the memory occupied by activations for reuse, which can inject more micro-batches concurrently into the pipeline in a sync-round. Our method with $M = 16$ can achieve GPU utilization 74.6% and 78.4% while consuming 63.0% and 76.7% averaged peak GPU memory compared to Gpipe on stage 0 and 1 respectively. Moreover, thanks to our memory-efficient 1F1B strategy, we can further increase the micro-batch size to improve GPU utilization of devices while keeping enough micro-batches in the pipeline to maintain a high throughput. When mbs = 32 and $M = 16$, we can achieve GPU utilization 90.4% and 89.5% on stage 0 and 1 respectively without OOM problem occurrence.

Heterogeneity-aware workload partitioning. We compare our heterogeneous-aware workload partitioning algorithm with PipeDream on EfficientNet-B1 and MobileNet-W2 with a 2-stage pipeline consisting of a TX2-N and a Nano-H. Fig. 12 shows the comparison of throughput and average GPU utilization on each stage between ours and the PipeDream method. PipeDream's workload partitioning algorithm is originally designed for homogeneous devices and the workload will be evenly divided into different stages. We can find that the device with stronger computing capability (TX2-N) has extremely low GPU utilization and lead to a low throughput of pipeline. Our partitioning method considers the different computation capacities of heterogeneous devices involved in the pipeline and will try to achieve a balance partition across all stages. GPU utilization of both two stage stays at a high level and results in a high throughput of the global pipeline.

Dynamic pipeline re-scheduling. We evaluate our adaptive pipeline re-schedule module on Efficient-B4 with a 3-stage pipeline consisting of a TX2-Q and two Nano-H. As shown in Fig. 13, we applied an external GPU workload to device 2 at the 100-th timestamp. Without pipeline re-scheduling and workload migration, after applying external workload, the training speed of device 2 will significantly slow down and become the lagger in the pipeline, which will seriously degrade the parallelism efficiency of the global system. With our adaptive pipeline scheduler, device 2 will migrate part of model layers to device 1 and 3 to rebalance the workload across each stage. By doing so, the pipeline can be promoted to a closer throughput level comparing to that before the external load spike. In our experiment, pipeline re-scheduling can be finished in several seconds. In practical deployment, we can further reduce the

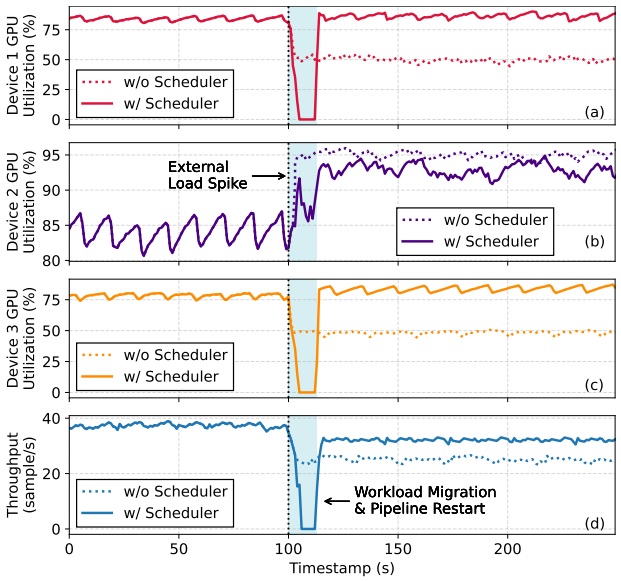


Figure 13: When Device 2 experiences an external load spike, Eco-FL's scheduler can migrate workload across devices and restart the pipeline, and therefore effectively boost the resource utilization and the system throughput.

overhead of pipeline re-boosting by scheduling workload with a relatively coarser granularity (e.g., residual blocks).

7 CONCLUSION

This paper proposes Eco-FL, a hierarchical FL system with pipeline parallelism to collaborate edge devices for accelerating clients' model training. Eco-FL adopts a memory-efficient pipeline strategy with adaptive workload scheduling to maximize the resource utilization in edge collaboration, and a grouping mechanism to address the heterogeneity issues in FL. Extensive evaluations demonstrate Eco-FL's effectiveness and efficiency upon existing baselines.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful feedback. We also thank Shenghao Fu and Weijun Ma for their support in implementation and the fruitful discussion. This work was supported in part by the National Science Foundation of China (No. U20A20159, No. 61972432); Guangdong Basic and Applied Basic Research Foundation (No. 2021B151520008); the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (No. 2017ZT07X355); the Pearl River Talent Recruitment Program (No. 2017GC010465).

REFERENCES

[1] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. 2020. Tifl: A tier-based federated learning system. In *HPDC*. 125–136.

[2] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. 2021. FedAT: a high-performance and communication-efficient federated learning system with asynchronous tiers. In *SC*. 1–16.

[3] Moming Duan, Duo Liu, Xianzhang Chen, Renping Liu, Yujuan Tan, and Liang Liang. 2021. Self-balancing federated learning with global imbalanced data in mobile systems. *TPDS* 32, 1 (2021), 59–71.

[4] Krizhevsky et. al. 2009. CIFAR-10. https://www.cs.toronto.edu/~kriz/cifar.html.

[5] LeCun et al. 1998. MNIST. http://yann.lecun.com/exdb/mnist/.

[6] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. 2021. DAPPLE: A pipelined data parallel approach for training large models. In *PPoPP*. 431–445.

[7] Pengzhan Hao and Yifan Zhang. 2021. EDDL: A Distributed Deep Learning System for Resource-limited Edge Computing Environment. In *SEC*.

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

[9] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*. 103–112.

[10] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond Data and Model Parallelism for Deep Neural Networks. *MLSys* 1 (2019), 1–13.

[11] 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. *TNNLS* (2022).

[12] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient federated learning via guided participant selection. In *OSDI*. 19–35.

[13] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. 2019. Edge AI: On-demand accelerating deep neural network inference via edge computing. *TWC* 19, 1 (2019), 447–457.

[14] Lumin Liu, Jun Zhang, SH Song, and Khaled B Letaief. 2020. Client-edge-cloud hierarchical federated learning. In *ICC*. IEEE, 1–6.

[15] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[16] AP Majtey, PW Lamberti, and DP Prato. 2005. Jensen-Shannon divergence as a measure of distinguishability between mixed quantum states. *Physical Review A* 72, 5 (2005), 052310.

[17] 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*. PMLR, 1273–1282.

[18] Naram Mhaisen, Alaa Awad Abdellatif, Amr Mohamed, Aiman Erbad, and Mohsen Guizani. 2021. Optimal user-edge assignment in hierarchical federated learning based on statistical properties and network topology constraints. *TNSE* 9, 1 (2021), 55–66.

[19] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *MobiSys*. 94–108.

[20] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *SOSP*. 1–15.

[21] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. 2018. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127* 3 (2018), 3.

[22] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*. 4510–4520.

[23] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*. PMLR, 6105–6114.

[24] Qiong Wu, Xu Chen, Zhi Zhou, and Junshan Zhang. 2020. Fedhome: Cloud-edge based personalized federated learning for in-home health monitoring. *TMC* (2020).

[25] Qiong Wu, Kaiwen He, and Xu Chen. 2020. Personalized federated learning for intelligent IoT applications: A cloud-edge based framework. *OJ-CS* 1 (2020), 35–44.

[26] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[27] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934* (2019).

[28] Zirui Xu, Zhao Yang, Jinjun Xiong, Janlei Yang, and Xiang Chen. 2019. Elfish: Resource-aware federated learning on heterogeneous edge devices. *Ratio* 2, r1 (2019), r2.

[29] Liekang Zeng, Xu Chen, Zhi Zhou, Lei Yang, and Junshan Zhang. 2020. Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices. *ToN* 29, 2 (2020), 595–608.

[30] Liekang Zeng, En Li, Zhi Zhou, and Xu Chen. 2019. Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial Internet of Things. *IEEE Network* 33, 5 (2019), 96–103.

[31] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *PIEEE* 107, 8 (2019), 1738–1762.