OLearning: A Geo-Distributed System for Device-Cloud Collaborative Computing

Min Fang$^{1(*)}$, Zhihui Fu$^{1(*)}$, Xiangmou Qu^{1}, Ruiguang Pei^{1}, Jun Wang$^{1(\boxtimes)}$, and Lan Zhang^{2}

^{1} OPPO Research Institute, Shenzhen, China
^{2} Independent Researcher
`mfang.cs@gmail.com, {luca, lokinko}@oppo.com, {peiruiguang, junwang.lu, zhanglan03}@gmail.com`

Abstract. Device-cloud collaborative learning trains a combined model using the cloud and distributed edge devices without exposing raw data. Each co-computing round involves: 1) devices fetching the latest model weights from the cloud for local training; 2) the cloud aggregating delta weights from devices to update the model. However, edge devices are geo-distribution and may perform co-computing in different time zones, which introduces three new challenges to existing systems: 1) task execution is delayed until reaching the device's available time window, the cloud needs an extended period to collect model weights. 2) devices have varying availability, but current task assignment assumes equal available training time, causing task imbalance. 3) massive model weights may arrive at unpredictable times, the cloud must reserve huge resources for peak loads. To fill the gap, we propose OLearning, a geo-distributed production system. Specifically, OLearning 1) applies a two-layer multi-zone design. The first layer performs inter-zone weight aggregation on various zones from the second layer, while the second layer performs intra-zone weight aggregation on selected devices. 2) adds a reward indicator for each task, allowing devices to decide task scheduling orders locally. 3) deploys a shared-task resource cluster for aggregating millions of delta weights over undetermined arrival times. Comprehensive experiments on the public dataset show the effectiveness and robustness of OLearning.

Keywords: Device-cloud · Co-computing · Geo-distribution

1 Introduction

With the growing demand for personal privacy and the introduction of various rigorous data protection regulations (such as GDPR), cross-device collaborative computing (e.g., federated learning) has become a research hotspot in both industry [1, 2] and academia [3, 4]. This paradigm uses a large-scale edge device

*Both authors contributed equally to this research.

(distributed worldwide) to collaboratively train high-quality models, while devices only share model updates with the cloud rather than raw data. A standard co-computing process involves multiple rounds of device-to-cloud interactions until the target model accuracy is reached. In each round, a device first pulls the latest model weights from the cloud and then performs the assigned task to compute weight updates. After collecting enough weight updates, the cloud aggregates them as the next round model weight [2]. Thus, the cloud usually has two core components: 1) *Selector*, selecting participants for a co-computing round via a given mechanism (e.g., random [1]); 2) *Aggregator*, aggregating weight updates from participants via a given method (e.g., FedAvg [5]) for next round.

Although there are already some open-source frameworks, such as FedML [6] and Flower [7], they all overlook the geo-distribution uniqueness of edge devices in their design, introducing three new challenges as follows.
1) Unacceptable single-round co-computing time. In a traditional single-layer case, devices participating in a co-computing round may be distributed across multiple time zones. Thus, the cloud needs a relatively long time (possibly up to a day) to collect enough weight updates and obtain a new model.
2) Inefficient device selection across multiple time zones. In cross-time-zone scenarios, it is difficult for the cloud to assess the data quantifies of edge devices. Though some participant selection mechanisms are proposed [3,4], they ignore the heterogeneity in devices' available time windows under geo-distribution.
3) Long-term enormous resource allocation in the cloud. Due to the huge differences in device available time windows caused by geographical locations, the cloud resources may be idle most of the time. Meanwhile, the pre-configured resources may not effectively handle sudden spikes from devices.

To address the above challenges, we design and implement a novel device-cloud collaborative computing system, named OLearning. OLearning adopts a two-layer multi-zone architectural design, where devices within each zone can perform multiple rounds of local computing, and the global aggregator periodically aggregates weights across zones. Further, OLearning introduces a decentralized selector, which only announces task rewards based on training contributions across zones and training processes within each zone. Then, devices select the optimal task set to register into the selector by further evaluating their data quality and available time. Lastly, OLearning implements a shared-task computing cluster and addresses inefficiencies in multi-task scenarios by enabling resource sharing. The main contributions of this paper are summarized as follows:

- We propose OLearning, a novel two-layer device-cloud collaborative computing system that first integrates geo-distribution factors into its design.
- We design a decentralized selector mechanism to optimize task assignment based on training contributions and device-specific factors.
- We deploy a shared-task computing cluster to improve resource utilization in the multi-task scenario.

We conduct extensive experiments to evaluate the performance and effectiveness of OLearning. The experiments present that OLearning can outperform the state-of-the-art work, and fully utilize the resources both in devices and cloud.

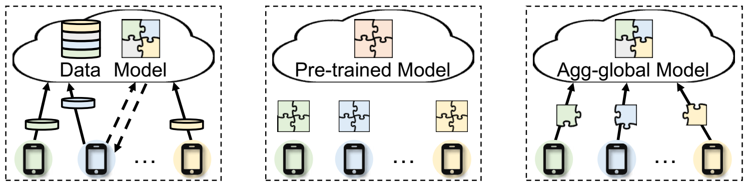
2 Related Work

As awareness of data privacy and security grows, the device-cloud computing paradigm has shifted from a cloud-centric one to a device-centric one, which enhances privacy protection and improves response speed. Furthermore, it is gradually evolving to device-cloud co-computing to integrate multi-party knowledge from enormous edge devices. Fig.1 illustrates their typical workflows.

Cloud-centric computing. A centralized computing paradigm, as shown in Fig.1(a), where devices report data to the cloud, all training and inference are done in the cloud-side, and only inference results are returned to devices. Currently, this one is widely used in multiple scenarios, e.g., the recommendation system in short-form video applications. However, it faces high data privacy risks and cannot be applied in critical applications, e.g., fingerprint recognition.

Device-centric computing. A distributed computing paradigm, as shown in Fig.1(b), where the cloud publishes the pre-trained model to devices, then devices train and infer locally without sharing the raw data, e.g., on-device text-to-image search[3]. Although this one can mitigate privacy leakage risk, the model cannot benefit from other device data due to information isolation.

Device-cloud co-computing. A hybrid computing paradigm, as shown in Fig.1(c), where the cloud issues the global model to devices for local execution (device-centric computing), and only parameters are returned for aggregation (cloud-centric computing), e.g., Apple's automatic speech recognition [2]. Since this one introduces a significant burden in handling device faults and managing cloud computing resources, many co-computing frameworks [1, 2, 6, 7] have emerged. These frameworks mainly optimize two key components: 1) a precise selector to choose participating devices for each training round based on factors such as data quality. 2) an efficient aggregator to aggregate delta weight updates from devices and maximize the utilization of cloud computing resources. Yet, all of them cannot efficiently support geo-distributed scenarios due to ignoring uncertain device availability and cloud computing resource requirements across different time zones. The importance of these considerations is discussed in §3.



(a) Cloud-centric (b) Device-centric (c) Device-cloud

Fig. 1: Device-cloud computing paradigm

[3] https://support.apple.com/en-sg/HT212630

3 Motivation

This section depicts key limitations of existing technologies that motivated ours.
Limitation#1: Lack of attention to geo-distribution in participant selection. Existing participant selection methods [1,3,4] ignore the heterogeneous behavior of enormous edge devices in real-world settings, e.g., different available time windows due to time zones. However, the geo-distribution of participants plays a crucial role in a cross-device co-computing task for the following reasons:

– **Availability:** Due to the uncertainty of geo-distribution, the active time of devices may vary greatly. If the server selects inactive devices as participants, the task may not be responded to promptly, affecting co-computing efficiency.
– **Synchronization:** In a co-computing process, each participant must return results within a specified time. If the location factor is ignored, some devices may be unable to upload local updates in time, resulting in high latency.
– **Fairness:** If the server ignores location factors when selecting participants, devices in certain time zones may be chosen more frequently, while those in other time zones are overlooked. Undoubtedly, this may raise issues of fairness.

Therefore, the geo-distribution factor may have a negative impact on the effectiveness of co-computing and should be considered when picking participants.
Limitation#2: Larger cloud-side resource wastage in multi-tasking. Existing cloud-side aggregation services are strongly correlated with the number of tasks [1, 2], i.e., allocating specific aggregation resources to each task at initialization, resulting in resource requirements growing linearly with the number of tasks. As shown in Fig. 2, multiple persistent aggregators are created and individually assigned to tasks based on available public resources. However, in real-world deployment, the available time windows of edge devices vary significantly due to geographical differences. Thus, task-bound aggregation services are not always busy and the corresponding resources are idle, causing huge resource wastage. Besides, the initialized resource allocation method makes it difficult for existing works to effectively respond to variable traffic peaks from participants.

4 Overview of OLearning

4.1 Architecture

Fig.3 illustrates the architecture of OLearning, including globally distributed edge devices, a centralized *Global Coordinator*, and multiple *Zone Coordinators* organized by geo-location (e.g., time zones).

Global Coordinator integrates three core modules to manage system-wide operations: 1) *Configurator* handles task assignment and device management; 2) *Global Selector* evaluates and prioritizes the contributions of different zones for each task, guiding task distribution; 3) *Global Aggregator* collects updates from all zones to produce the final global model, enabling cross-zone collaboration.

Each *Zone Coordinator* runs independently and contains two key components: 1) *Zone Selector* evaluates tasks training processes at one round and

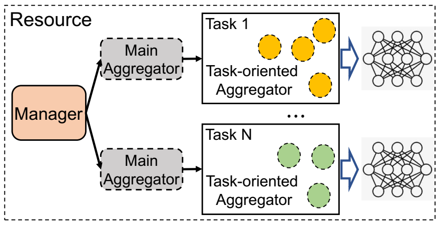


Fig. 2: Traditional aggregator

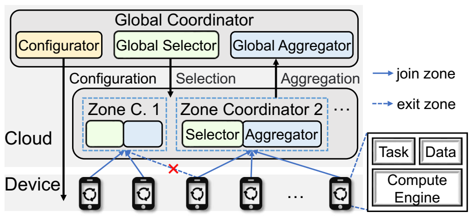


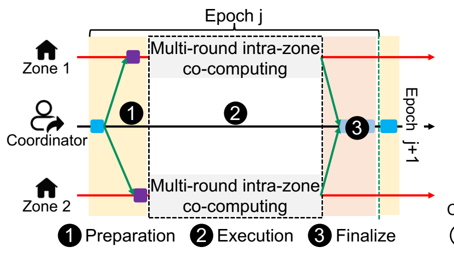
Fig. 3: System architecture of OLearning

assigns it as rewards to devices; 2) *Zone Aggregator* aggregates updates from devices to get the zone-level model, supporting iterative training within the zone.

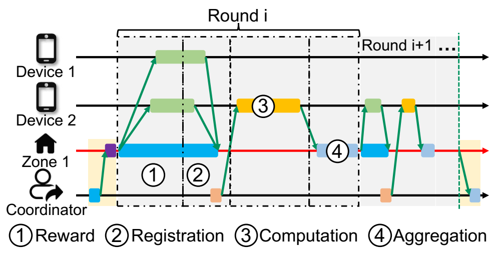
Edge devices, distributed globally in vast numbers with larger data heterogeneity, are dynamically assigned to specific zones based on geo-location. As devices move across zones, they seamlessly transition to maintain continuity.

4.2 Co-computing Process across Zones

OLearning employs a two-layer co-computing design to utilize zone-specific variations in available time windows and data distributions. By integrating inter- and intra-zone collaboration, OLearning enables decentralized decision-making, enhancing scalability, adaptability, and task execution efficiency in real scenarios.
Inter-zone Co-computing Interaction. OLearning adopts an epoch-based inter-zone co-computing flow, where an epoch is a user-defined interval defined by criteria such as time windows or computation rounds in a zone. Shorter epochs allow more frequent global aggregation, enabling finer-grained synchronization and adaptability. The inter-zone process, shown in Fig.4(a), has three key stages:
❶ Preparation: *Global Selector* evaluates the potential contribution of each zone to the task and assigns a reward score accordingly (see §5.1). Then, *Global Coordinator* disseminates task details (e.g., model structure), aggregated weights from the last epoch, and zone-specific reward scores to all participating zones.
❷ Execution: Zones engage in inter-zone collaborative computation for the current epoch, conducting multiple rounds of local computing and coordination.
❸ Finalization: *Global Aggregator* collects updates from all selected zones and performs aggregation to generate the model weights for the next epoch.
Intra-zone Co-computing Interaction. OLearning uses a round-based intra-zone co-computing flow, where tasks are trained iteratively until the epoch stop condition is met. The intra-zone process, shown in Fig.4(b), has four key stages:
① Reward Announcement: *Zone Selector* evaluates task progress and sets reward scores, a lower score means that the task is closer to convergence (see §5.1).
② Registration: Devices receive reward scores from *Global Selector* and *Zone Selector*s and get final task-specific rewards by incorporating local data quality. Then, each device gets an optimal task set under its availability and registers for selected tasks via *Zone Coordinator* (see §5.2). Once a task has received enough device registrations, further attempts will be rejected. To prevent low-priority tasks from starving, *Zone Selector* increases rewards for them in the next round.


Epoch j
Zone 1
Multi-round intra-zone co-computing
Coordinator
❶
❷
❸
Epoch j+1
Zone 2
Multi-round intra-zone co-computing
❶ Preparation ❷ Execution ❸ Finalize

(a) Inter-zone management



(b) Intra-zone co-computing process

Fig. 4: Description of the whole process

③ Computation: Registered devices perform local computing via a predefined task scheduling policy and upload the got update weights to *Zone Aggregator*. ④ Aggregation: *Zone Aggregator* merges local updates from devices to get next-round zone model. This iterative continues until the epoch concludes (see §6).

5 Decentralized Selector

To address uncertainty in data quality and availability for geo-distributed devices, OLearning designs a decentralized selection mechanism. Specifically, a task is first scored independently by *Global Selector*, *Zone Selector*, and device (§5.1). Then, each device selects the optimal task set based on its local status (§5.2).

5.1 Announce Task Reward

The reward of task t depends on: 1) the training profit of $zone_i$ against other zones; 2) the training progress of round r_j in $zone_i$; 3) the data quality of $device_k$. **Global Selector Reward.** *Global Selector* quantifies the relative contribution of $zone_i$ by measuring how far its weights, $\mathbf{W}^{zone_i}$, deviate from the global aggregated weights, $\mathbf{W}^{global}$. This deviation is calculated using the L2 norm of the difference between $\mathbf{W}^{zone_i}$ and $\mathbf{W}^{global}$. To ensure comparability of contributions across zones and tasks, Eqn.1 normalizes the deviation of $zone_i$ by the total deviation of all n zones. $Contrib(zone_i)$ is the proportion of each zone's influence on the task. A smaller $Contrib(zone_i)$ means that the model matches the data in $zone_i$ better, and the profit from further training in $zone_i$ is smaller.

$$Contrib(zone_i) = \frac{|\mathbf{W}^{zone_i} - \mathbf{W}^{global}|_2}{\sum_{i=1}^{n} |\mathbf{W}^{zone_i} - \mathbf{W}^{global}|_2} \tag{1}$$

Zone Selector Reward. *Zone Selector* evaluates the training progress of $zone_i$ at round r_j. Specifically, it uses Eqn.2 to capture reductions in both gradient magnitude and loss value between consecutive iterations. In Eqn.2, $\nabla\mathbf{W}_{r_j}^{zone_i}$ denotes the gradient of the weight vector for $zone_i$ at round r_j, while $Loss_{r_j}^{zone_i}$ is the corresponding loss value. The first term in Eqn.2 calculates the relative reduction in gradient magnitude, providing a measure of convergence stability. The second term quantifies the proportional reduction in loss value, ensuring that only non-negative improvements contribute to the progress metric. To address potential numerical instability, a small constant ϵ is added to the denominators of both terms. A smaller $Progress(r_j)$ means that the training process in $zone_i$ has slowed down, and further training in $zone_i$ may yield diminishing profit.

$$Progress(r_j) = \left(\frac{|\nabla \mathbf{W}^{zone_i}_{r_j-1}|_2 - |\nabla \mathbf{W}^{zone_i}_{r_j}|_2}{|\nabla \mathbf{W}^{zone_i}_{r_j-1}|_2 + \epsilon} \right)^2 \times \max \left(0, \frac{Loss^{zone_i}_{r_j-1} - Loss^{zone_i}_{r_j}}{Loss^{zone_i}_{r_j-1} + \epsilon} \right) \quad (2)$$

Device Reward. Eqn.3, derived from Oort [3], evaluates the data quality of a device by considering the amount of data, denoted as $|Data|$, and the associated loss values. It measures the potential contribution of a device to a training round.

$$Quality(device_k) = |Data| \sqrt{\tfrac{1}{|Data|} \textstyle\sum_{d \in Data} Loss(d)^2} \quad (3)$$

Task Final Reward. Eqn.4 defines the total profit of a $device_k$ within a $zone_i$ during an intra-zone round r_j as the product of the above three factors.

$$Profit(zone_i, r_j, device_k) = Contrib(zone_i) Progress(r_j) Quality(device_k) \quad (4)$$

Further, the reward of task t ($reward_t$) applies a logarithmic transformation to the profit, which compresses large values to maintain numerical stability and reduce the impact of outliers, while preserving the relative ordering of profits.

$$reward_t = \frac{\log(1 + Profit(zone_i, r_j, device_k))}{1 + \log(1 + Profit(zone_i, r_j, device_k))} \quad (5)$$

5.2 Determine Preregistered Task

Given a set of tasks and their rewards, the device calculates the optimal subset of tasks based on its available time. Generally, a device has m different available time slots (the longest continuous available time window), which can be predicted based on the historical operation by an on-device model. To simplify on-device task management, each task must be completed within a single and continuous time slot. Then, we can define a action $reg(t, slot_i) \in \{0, 1\}$ as whether the device executes task t at the available time $slot_i$ ($i = 1, ..., m$). A task t executes at most once across all slots, which can be formulated as the constraint in Eqn.6.

$$reg(t, slot_i) \in \{0, 1\},\ t \in T,\ i = 1, ..., m; \quad \textstyle\sum_{i=1}^{m} reg(t, slot_i) \leqslant 1,\ t \in T \quad (6)$$

Meanwhile, a device can execute at most one task at a time to reduce resource contention. Then, the total time of tasks in a slot must be less than its available time, which can be formulated as the constraint in Eqn.7. The $time_t$ depends on the amount of data on the device and the training speed for the device model.

$$\textstyle\sum_{i=1}^{m} time_t \cdot reg(t, slot_i) \leqslant slot_i,\ i = 1, ..., m \quad (7)$$

Thus, the on-device multi-task scheduling problem is to select a subset of disjoint tasks from T that maximizes the total reward, which can be defined as:

$$\max \textstyle\sum_{t \in T} \sum_{i=1}^{m} reward_t \cdot reg(t, slot_i) \quad (8)$$

Combine constraints in Eqn.6 and Eqn.7 with objective goal Eqn.8, we construct a standard 0-1 Multiple Knapsack Problem (MKP) [8], which is NP-hard. In OLearning, we solve it with a constraint programming solver [9] for small-scale scenarios and a greedy strategy [10] for large-scale scenarios.

6 Elastic Aggregator

In geo-distributed settings, device availability varies with user activity, causing large variations in the number of devices and total computational load across tasks. Thus, it is impossible to predict the required aggregation resources for each task in advance or allocate persistent aggregators accordingly. Given this, OLearning uses an elastic manager to schedule a set of shared-task aggregators, and integrates a multi-actor design to handle concurrent aggregation requests.
Shared-Task Aggregator. OLearning adopts a unified shared-task aggregator design, with task allocation and tracking coordinated by a global manager. This design is supported by a shared pool of computing resources accessible to all aggregators. Within the shared resource pool, each actor can receive task-specific weights. A proxy component routes device requests to actors by consistent hashing [11], leveraging device IDs as hash keys. This way ensures load balancing and reduces the likelihood of actor hotspots. Upon receiving weights, each actor determines the running aggregator based on the manager's instructions. Once an actor finishes the weight aggregation for a task, it transmits intermediate results to a leader actor designated by the manager. The leader actor then performs final aggregation to get the task's output. This ensures that all aggregators share computing resources while balancing weight storage at the actor level. If an actor crashes, the system relies on the task participants to re-upload lost weights.
Real-Time Aggregated Task Scheduling. The elastic aggregator manager periodically gathers metadata from all actors, including the weight arrival rate and remaining unaggregated weights for each task. Using this global view, the manager prioritizes tasks across all actors based on the following criteria: 1) Arrival Rate: Tasks with higher weight arrival rates are given priority to ensure efficient processing of incoming data. 2) Remaining Workload: For tasks with equal arrival rates, priority is assigned based on the number of remaining unaggregated weights to reduce the overall average aggregation completion time.

This scheduling strategy enables the system to efficiently utilize shared computing resources while minimizing task aggregation latency. By dynamically adjusting priorities based on real-time metrics, OLearning achieves a balanced trade-off between resource sharing and aggregation performance.

7 Experiments

In this section, we evaluate the effectiveness of OLearning from multiple dimensions and target to answer the following questions:

- **Q1 (§7.1):** *Does the selection mechanism in OLearning outperform the state-of-the-art work for cross-devices co-computing applications?*

- **Q2 (§7.2):** *How effective is the on-device multi-task scheduling employed in OLearning when facing multi-task scenarios?*
- **Q3 (§7.3):** *How is the resource overhead of shared-task aggregation methods when deploying OLearning on real production environment?*

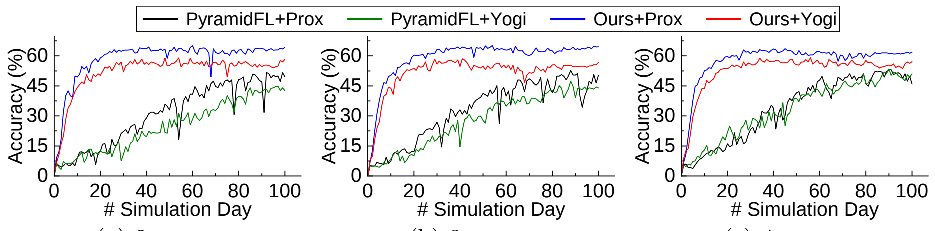
7.1 Performance of Selection Mechanism

All evaluations for this set of experiments are conducted using 4 NVIDIA GeForce RTX 3090 GPUs, each used to simulate one zone. The details are as follows.
Datasets and models. We evaluate OLearning on two public datasets:

- **Google Speech**[4]**:** The dataset for speech recognition, consisting of about $100,000$ audio commands from more than $2,000$ clients. Based on the dataset, we train the ResNet-34 [12] model for a 20-class speech recognition task.
- **HARBox**[5]**:** The 9-axis IMU data about human daily activity recognition, which is collected from 121 users' smartphones via crowdsourcing. We train the lightweight customized DNN model in PyramidFL [4] here.

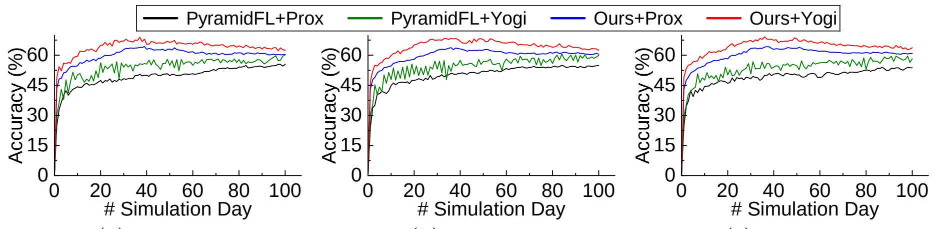
Baselines and optimizers. We compare the participant selection strategy of OLearning with the state-of-the-art PyramidFL [4]. Yogi [13] and Prox [14] are used to assess the generality of the designed method across different optimizers.
Metrics. We use *day-to-accuracy* as the metric, i.e., the total simulation day for a model training task to achieve a target accuracy on the testing set. In both works, we mark the time that *Global Aggregator* collects enough weights as the end of one day. For each experiment, we report the average accuracy over 3 runs.
Parameters. In OLearning, each zone contains $1.3K(K{=}20)$ devices, following PyramidFL. During a single training round within a zone, *Zone Aggregator* selects K participants. After completing 10 rounds in a zone, the aggregated results are uploaded to *Global Aggregator*. Similarly, PyramidFL manages a total of $|Z| \times 1.3K$ devices, where $|Z|$ represents the number of zones. In each training round of PyramidFL, *Global Aggregator* selects $|Z| \times K$ participants. All model training parameters are strictly consistent with the settings in PyramidFL.
Accuracy with varying number of zones. We compare the accuracy change of OLearning with PyramidFL on two datasets over simulation days, as shown in Fig.5 and 6. Results show that OLearning can always reach the final accuracy within smaller simulation days, and then begin to fluctuate slightly. For PyramidFL, 100 simulation days are not enough to converge. Thus, OLearning can greatly reduce the time cost of co-computing by applying an early exit mechanism [15], while ensuring high accuracy and usability of the final model. Multiple tests with varying numbers of zones highlight the strong horizontal scalability of OLearning. Compared to HARBox, OLearning performs significantly better than PyramidFL on Speech dataset. From Fig.5, ResNet-34 performs better on Prox over Yogi with Speech dataset. Yet, regardless of the optimizer used, OLearning can reach a local optimum in about 30 simulation days for varying numbers

[4] https://fedscale.eecs.umich.edu/dataset/google_speech.tar.gz
[5] https://github.com/xmouyang/FL-Datasets-for-HAR



(a) 2 zones (b) 3 zones (c) 4 zones

Fig. 5: Day-to-accuracy for speech recognition on Google Speech dataset



(a) 2 zones (b) 3 zones (c) 4 zones

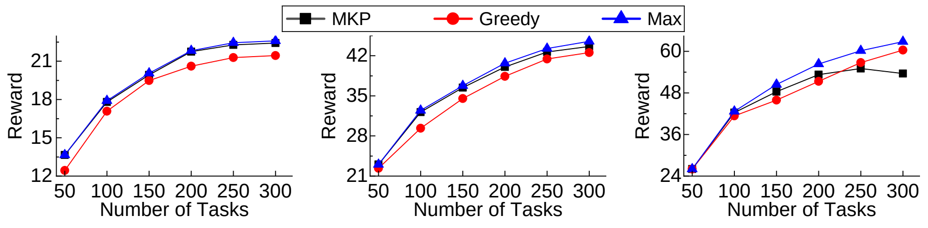
Fig. 6: Day-to-accuracy for human activity recognition on HARBox dataset

of zones. Taking 4 zones in Fig.5(c) as an example, the accuracy improves about 37.82% for Prox and 30.95% for Yogi against PyramidFL. For the customized DNN with HARBox in Fig.6, although Yogi outperforms Prox, OLearning still outperforms PyramidFL and reaches a local optimum at about 30 simulation days. Similarly, in Fig.6(c), OLearning gets about 15.04% improvement for Prox and 16.62% for Yogi. All results reveal the robust speedup of OLearning, which is not impacted by different optimizers under the same model and datasets.

7.2 Robustness of On-device Multi-task Scheduling

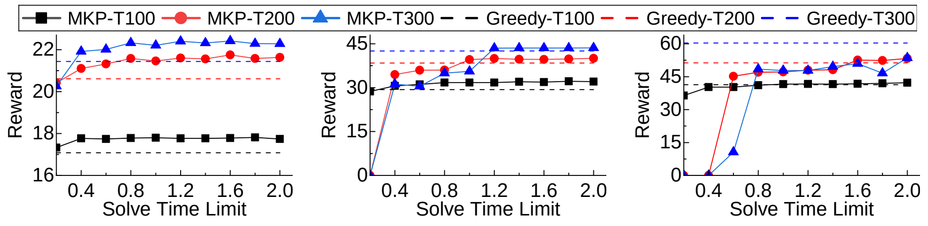
We conduct experiments on a Linux machine (i9-10900K 3.70G 20c/64g/1T) to verify the effectiveness of our multi-task scheduling mechanism in §5.2. There are m available time slots in a device, each slot is randomly from 1-10min, and the reward of a task is randomly from 0.01-1. With varying numbers of tasks and m, we count total reward under two methods, i.e., MKP with Google OR-Tools [9] and reward-maximizing greedy [10]. Fig.7 and 8 give detailed results.

Since the solver may get different optimal solutions under the varying solving time limits, we cap the MKP solving time in Fig.7 at 2s, an acceptable range. The Max in Fig.7 is a globally optimal solution with 60s solve time limit. Results show that when the number of tasks is small (≤ 150), MKP performs similarly to Max regardless of what m is. The difference between MKP and Max is only obvious when m and the number of tasks increases. In contrast, Greedy can reach the same effect as MKP and Max with larger m and smaller tasks (Fig.7(b) and 7(c)), but it performs poorly with smaller m (Fig.7(a)) or larger tasks (Fig.7(b) and 7(c)). Although MKP is subject to the solve time limit, results in Fig.8 show that an optimal solution can be got within 2s. The value 0 in Fig.8 means the



(a) m=10 (b) m=30 (c) m=50

Fig. 7: Total reward under m free time slots against the number of tasks



(a) m=10 (b) m=30 (c) m=50

Fig. 8: Total reward under m free time slots against the solve time limit

solver cannot get an optimal solution under the given time limit, and MKP-T# or Greedy-T# denotes the number of tasks is # for the method. Besides, MKP can achieve an effect comparable to Greedy with a shorter time limit when m is small. However, as the problem complexity rises, such as in Fig.7(c) and 8(c), when m=50 with tasks exceed 200, 2s time limit becomes insufficient for finding an optimal solution. Thus, in real design, one should dynamically choose between MKP or Greedy based on m and the number of tasks to maximize rewards.

7.3 Aggregation In Production

We evaluate the robustness and efficiency of the aggregation module in OLearning within a real production environment. Specifically, we collected statistics on the storage resource occupancy of cloud aggregation services over 24 hours on OPPO's device-cloud co-computing online business platform, as shown in Fig.9. The vertical axis represents the resource usage percentage, calculated with the peak value of 100%. In the traditional task-oriented aggregation, messages reported by devices are cached until certain conditions are met before aggregation begins. This not only leads to idle computing resources during the waiting period, but also causes prolonged over-occupation of storage resources. Conversely, our shared-task aggregation starts computing once receiving messages, smoothing out computational peaks and minimizing time overhead and resource occupancy. Besides, we measure the performance improvement of shared-task aggregation in real-world online industrial scenarios and find that it can achieve an efficiency gain of over 40% to 50%, while requiring fewer computational resources.

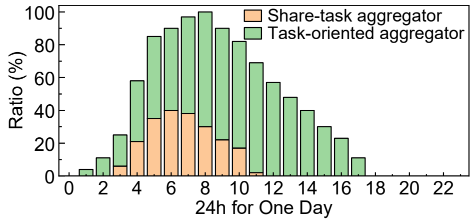


Fig. 9: Storage consumption

8 Conclusion

In this paper, we present OLearning, a novel device-cloud collaborative computing framework leveraging a two-layer multi-zone architecture, decentralized task selection, and shared-task resource optimization. By integrating geo-distribution factors and enhancing resource utilization, our experiments present that OLearning effectively addresses key inefficiencies in multi-task and distributed scenarios.

Acknowledgments. We thank Qiuye Zheng, Lichao Wang, Kun Hou, and Xudong Liu for their great support during the design, development, and deployment of OLearning.

References

1. Bonawitz, K., Eichner, H., et al.: Towards federated learning at scale: System design. Proceedings of machine learning and systems **1**, 374–388 (2019)
2. Paulik, M., Seigel, M., Mason, H., et al.: Federated evaluation and tuning for on-device personalization: System design & applications. arXiv:2102.08503 (2021)
3. Lai, F., Zhu, X., Madhyastha, H.V., Chowdhury, M.: Oort: Efficient federated learning via guided participant selection. In: OSDI. pp. 19–35 (2021)
4. Li, C., Zeng, X., Zhang, M., Cao, Z.: Pyramidfl: A fine-grained client selection framework for efficient federated learning. In: MobiCom. pp. 158–171 (2022)
5. McMahan, B., et al.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. pp. 1273–1282 (2017)
6. Next-Gen Cloud Services for LLMs & Generative AI: (2023), https://fedml.ai/
7. Flower: A Friendly Federated Learning Framework: (2023), https://flower.dev/
8. Martello, S., Toth, P.: A bound and bound algorithm for the zero-one multiple knapsack problem. Discrete Applied Mathematics **3**(4), 275–288 (1981)
9. Google: OR-Tools (2024), https://developers.google.com/optimization/pack
10. Martello, S., Toth, P.: Algorithm 632: A program for the 0–1 multiple knapsack problem. ACM Trans. Math. Softw. **11**(2), 135–140 (Jun 1985)
11. Karger, D., et al.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: STOC. pp. 654–663 (1997)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. pp. 770–778 (2016)
13. Reddi, S., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečnỳ, J., Kumar, S., McMahan, H.B.: Adaptive federated optimization. arXiv:2003.00295 (2020)
14. Li, T., Sahu, A.K., Zaheer, M., et al.: Federated optimization in heterogeneous networks. Proceedings of Machine learning and systems **2**, 429–450 (2020)
15. Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V., Tay, Y., Metzler, D.: Confident adaptive language modeling. NeurIPS **35**, 17456–17472 (2022)