VENN: RESOURCE MANAGEMENT FOR COLLABORATIVE LEARNING JOBS

Jiachen Liu [1] **Fan Lai** [2] **Ding Ding** [1] **Yiwen Zhang** [1] **Mosharaf Chowdhury** [1]

ABSTRACT

In recent years, collaborative learning (CL) has emerged as a promising approach for machine learning (ML) and data science across distributed edge devices. As the deployment of CL jobs increases, they inevitably contend for limited resources. However, efficient resource scheduling in this context is challenging because of the *ephemeral nature and resource heterogeneity of devices*, coupled with the *overlapping resource requirements of diverse CL jobs*. Existing resource managers often assign devices to CL jobs randomly for simplicity and scalability, but this approach compromises job efficiency.

In this paper, we present Venn, a CL resource manager that efficiently schedules ephemeral, heterogeneous devices among multiple CL jobs to reduce the average job completion time (JCT). Venn formulates the *Intersection Resource Scheduling (IRS)* problem to identify complex resource contention among multiple CL jobs. It then proposes a contention-aware scheduling heuristic to minimize the average scheduling delay. Furthermore, it proposes a resource-aware device-to-job matching heuristic to optimize response collection time by mitigating stragglers. Our evaluation shows that, compared to the state-of-the-art CL resource managers, Venn improves the average JCT by up to 1.88×. The code is available at https://github.com/SymbioticLab/Venn.

1 INTRODUCTION

Collaborative learning (CL) enables distributed edge devices to perform collaborative machine learning (ML) without moving raw data into the cloud (Bonawitz et al., 2019; Ramage & Mazzocchi, 2020). CL has been adopted by many large corporations including Apple, Meta, Google, and LinkedIn to protect user data privacy while improving user experience. For example, Google adopts CL for a wide range of applications such as speech recognition (Warden, 2018), healthcare study (Sadilek et al., 2021), next-word prediction (Hard et al., 2019; Xu et al., 2023), emoji prediction (Ramaswamy et al., 2019), and query suggestion on keyboard (Yang et al., 2018). Each CL training job in practice often requires 1000∼10000 device participants in each training round and takes 4∼8 days to finish (Yang et al., 2018). As the number of CL applications continues to grow, efficient edge resource management has become the key to fast and resource-efficient CL.

Unlike cloud resources, CL resources are not accessible all the time; they only become available for training when they are connected to WiFi and charging (Bonawitz et al., 2019). Moreover, they are highly heterogeneous in terms of hardware capacity, software versions, and training data availability (§2.1). Even when running on the same client population, different CL jobs often compete for different subsets of devices, based on their specific model characteristics and training objectives. The resource heterogeneity and job's resource requirement diversity lead to complex multi-resource contentions among ongoing CL jobs, where eligible resources for each job are not only limited but also may *overlap, contain, or be within* those of one or more other jobs.

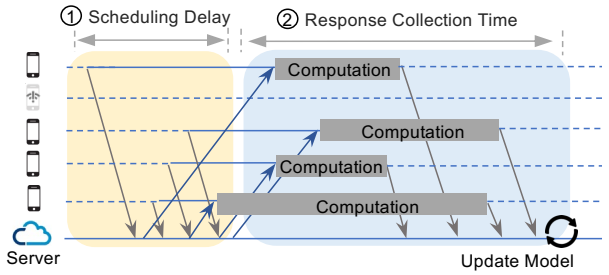


Figure 1. Composition of the completion time of one round of a CL job.

However, existing CL efforts often overlook resource contention among coexisting CL jobs, assuming that sufficient resources are always available (Bonawitz et al., 2019). For example, many recent works (Lai et al., 2021a; Abdelmoniem et al., 2023; Lai et al., 2021b; Balakrishnan et al., 2022) primarily focus on optimizing the response collection time of individual CL jobs. In practice, however, multiple CL jobs may run concurrently and compete for a subset of devices, leading to resource contention (Bonawitz et al., 2019). As a result, existing works fail to account for schedul-

[1] Computer Science and Engineering, University of Michigan, Michigan, USA [2] Computer Science and Engineering, University of Illinois at Urbana-Champaign, Illinois, USA. Correspondence to: Jiachen Liu <amberljc@umich.edu>.

Proceedings of the 8th MLSys Conference, Santa Clara, CA, USA, 2025.

ing delay (§2.2)—the time needed to acquire all necessary resources for a CL job—as a key factor influencing job completion time (JCT) (Figure 1).

With the proliferation of CL in production, large corporations such as Apple (Paulik et al., 2021), Meta (Huba et al., 2022), and Google (Bonawitz et al., 2019) have developed their own CL infrastructures to manage multiple jobs at scale. However, despite their low-level differences, these CL resource managers can largely be characterized as *random device-to-job matching* in various forms. We observe that such random matching can result in higher scheduling delays and longer response collection times as an increasing number of CL jobs compete for a shared pool of devices.

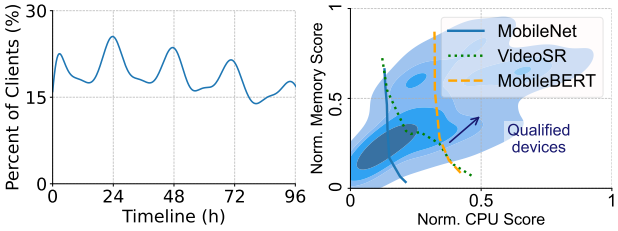
In this paper, we present Venn, a CL resource manager designed to minimize the average JCT of multiple CL jobs competing for a large pool of edge resources. First, Venn models the complex resource contention among CL jobs as an instance of the *Intersection Resource Scheduling (IRS)* problem (§4.2), where a job's resource demands may overlap with, contain, or fall within those of other jobs. Venn then introduces a contention-aware scheduling heuristic that prioritizes jobs requiring scarce resources and lower overall demand to minimize the average scheduling delay. Second, Venn employs a resource-aware device-to-job matching heuristic to mitigate the effect of stragglers on response collection time (§4.3). Together, these techniques jointly optimize the average JCT of multiple CL jobs operating under dynamically changing and uncertain resource conditions.

We have implemented and evaluated Venn across various CL workloads derived from real-world scenarios (§5). Compared to state-of-the-art CL resource allocation solutions (Bonawitz et al., 2019; Huba et al., 2022; Paulik et al., 2021), Venn improves the average JCT by up to $1.88\times$.

Overall, we make the following contributions in this paper:

1. We introduce Venn, a CL resource manager designed to enable efficient sharing of heterogeneous devices across a large number of CL jobs.
2. To minimize the average JCT of CL jobs, we propose a job scheduling and client matching joint solution to optimize both the scheduling delay and response collection time. Our approach achieves a logarithmic-linear time complexity and is supported by theoretical analysis.
3. We have implemented and evaluated Venn, along with its scheduling and matching algorithms, demonstrating improvements in the average JCT compared to the state-of-the-art across various real-world CL workloads.

2 Background and Motivation



(a) Diurnal device availability. (b) Device hardware heterogeneity.

Figure 2. CL resources exhibit both high variance in availability and capacity.

2.1 Collaborative Learning

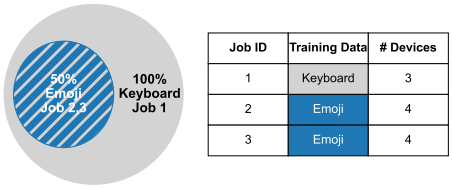
Collaborative learning (CL) has been widely adopted in industry to train ML models on diverse datasets stored on edge devices without transferring raw data to the cloud. Example applications include healthcare study, speech recognition, next-word prediction, etc. However, CL introduces unique resource management challenges due to the distinct characteristics of edge devices and the jobs that rely on them.

Single CL Job's Resource Management: CL resources often refer to edge devices with limited capacities, including smartphones, laptops, and Internet of Things (IoT) devices. A major challenge of resource management in CL stems from the unique characteristics of CL resources in terms of availability and heterogeneity.

Dynamic availability. Unlike cloud resources (e.g., GPUs) that are accessible on demand, edge devices in CL are only available for training under specific conditions, such as when they are charging and connected to WiFi. We analyze a real-world client availability trace (Lai et al., 2021a), which encompass 180 million trace items of devices behavior over a week. Figure 2a shows that the number of available devices (charging and connected to WiFi) changes over time with diurnal pattern. This fluctuation creates inherent scheduling delays, as CL jobs must wait for enough devices to become available before training can begin.

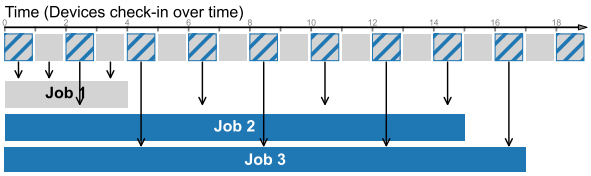
Device heterogeneity. Edge devices in CL vary widely in hardware capabilities, including memory and CPU capacity, as well as in data availability and software version, leading to inconsistent response times. Figure 2b showcases this heterogeneity, focusing on variations in memory and CPU capacities among edge devices, based on data from AI Benchmark (Ignatov et al., 2019). It also annotates the minimum hardware requirements needed to execute three popular on-device ML models within a reasonable time. Additionally, CL jobs often target specific device subsets based on factors like data availability, software version, or hardware capacity. These subsets can *overlap, nest, or be mutually exclusive*, leading to complex resource contention patterns where jobs compete for the same devices.

Together, dynamic availability and device heterogeneity

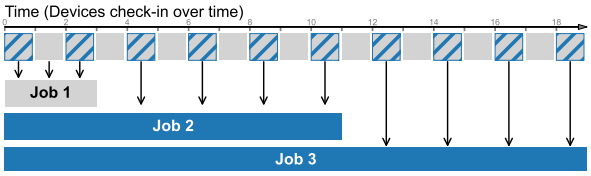


Job ID	Training Data	# Devices
1	Keyboard	3
2	Emoji	4
3	Emoji	4

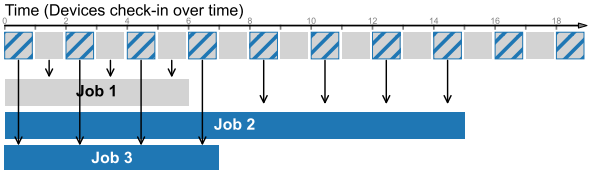
(a) Setup.



(b) Random Matching ($J\bar{C}T$=12).



(c) Shortest remaining service first (SRSF) ($J\bar{C}T$=11).



(d) Optimal ($J\bar{C}T$=9.3).

Figure 3. Toy example showing three resource schedules across multiple CL jobs. Job demands and resource eligibility are shown in the top row. Devices check in at a constant rate. Eligible devices only for Emoji jobs are marked with blue; all devices are eligible for the Keyboard job. The label of each client indicates its job assignment. Random Matching and SRSF inefficiently allocate scarce Emoji-eligible devices to Job 1, which already has sufficient Keyboard-eligible resources. In contrast, the optimal schedule allocates these scarce resources to Job 2 followed by Job 3, minimizing the average JCT.

slow down CL training, prompting solutions like client selection (Lai et al., 2021b; Abdelmoniem et al., 2023; Balakrishnan et al., 2022), quantization (Reisizadeh et al., 2020), and model aggregation (He et al., 2023) to optimize single-job performance.

2.2 Multiple CL Jobs' Resource Management

To orchestrate multiple CL jobs, several CL resource managers have been proposed with three primary designs.

1. Apple's CL resource management (Paulik et al., 2021) is driven by clients, where each client independently samples from a list of CL jobs they are able to execute.
2. Meta's CL resource manager (Huba et al., 2022) is centralized, where it randomly matches each client with one eligible CL job.

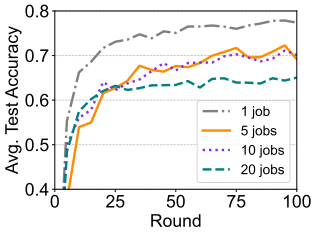


Figure 4. Impact of resource contention.

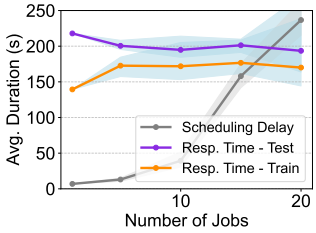


Figure 5. JCT breakdown in a single round.

3. Google's CL resource manager (Bonawitz et al., 2019) is driven by jobs, where each job independently samples from available clients.

Despite the seeming variety in their designs, existing resource managers all rely on random device-to-job matching. This approach works when devices are abundant but fails to address resource contention when demand exceeds supply, resulting in longer scheduling delays and inefficient resource use.

2.3 Limitation and Opportunities

Limitations of the state-of-the-art. To highlight the shortcomings of existing methods, consider the toy example in Figure 3, which compares three scheduling strategies for three CL jobs—"Emoji Prediction Job 1," "Emoji Job 2," and a "Keyboard Prediction Job"—competing for devices with varying eligibility. We assume all jobs arrive simultaneously and clients with different eligibilities become online constantly over time. Both random matching and SRSF waste scarce resources (i.e., Emoji clients) on the Keyboard job, which has ample device options, while the optimal schedule prioritizes efficiency, reducing JCT to 9.3 time units. Note that the contention patterns in real-world scenarios are often more complex and larger in scale than the one presented in this example.

Impact of resource contention. Single-job FL optimizations, such as client selection (Lai et al., 2021b; Abdelmoniem et al., 2023; Balakrishnan et al., 2022), often assume that CL jobs have access to sufficient number of online devices at any time. However, in practice, there could be multiple CL jobs running at the same time and competing for the same set of devices, leading to resource contention (Bonawitz et al., 2019). This can significantly impact the performance of CL jobs, such as accuracy and end-to-end training time.

We analyze the impact of resource contention on CL jobs' performance in Figure 4. In this experiment, the resource pool is evenly partitioned and managed by each job, who aims to train a ResNet-18 model with 100 clients per round on the FEMNIST dataset (Cohen et al., 2017). We vary the number of concurrently running jobs to observe its impact.

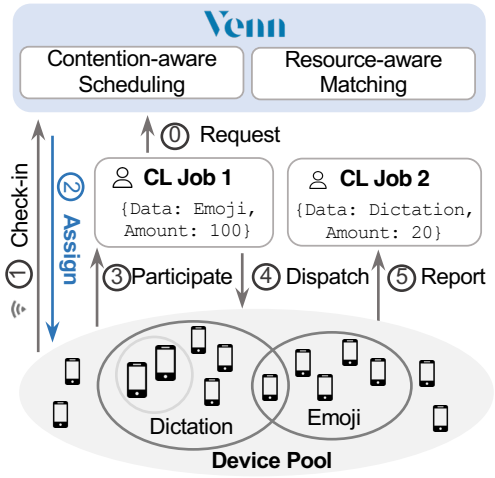


Figure 6. Venn system overview.

As more jobs share the same device pool, the available device choices for each job become increasingly constrained, leading to a noticeable degradation in the round-to-accuracy performance. Hence, evenly partitioning resource pool for each CL job is no better than a shared device pool in terms of participant diversity.

Breakdown of request completion time. While most existing CL efforts, such as quantization (Reisizadeh et al., 2020) and client selection (He et al., 2023), predominantly focus on model convergence rate or *response collection time*——i.e., the time needed to collect a sufficient number of responses——they often overlook a critical component: *scheduling delay*, as depicted in Figure 1.

Figure 5 breaks down JCT for a single training round under varying contention levels, using the same setup as Figure 4. Utilizing the same experimental setup as in Figure 4, we quantify both the average scheduling delay and response collection time with random device-to-job matching during one round of training and testing. The shaded regions cover the duration for each individual job. As our results in Figure 5 indicate, scheduling delay can significantly impact overall JCT, especially when resource supply falls short of demand.

3 Venn Overview

Venn serves as a standalone CL resource manager that operates at a layer above all CL jobs, and it is responsible for allocating each checked-in resource to individual jobs. Figure 6 illustrates Venn's workflow and its role within the lifecycle of an CL job.

In each execution round, a job submits resource requests to Venn, specifying its device requirements (e.g., minimum CPU capacity) and resource demands (the number of devices needed per round) (⓪). Meanwhile, edge devices continuously check in with Venn as they become available over time (①). Based on the real-time resource demand and supply, Venn generates a resource allocation plan to assign one CL job to each checked-in device until the job's needs are met (②). Upon receiving the task assignment from Venn, each device adheres to the allocation plan and participates in the corresponding job (③). The device then retrieves the computation plan from the job and performs on-device training or inference (Lv et al., 2022; CoreML) (③ and ④). Finally, devices either submit their training results to the job or disconnect if their availability changes (e.g., battery runs low) (⑤).

Steps ③ to ⑤ follow standard CL protocols between jobs and devices, while Venn focuses on optimizing the job-to-device assignment in Step ②. This allocation step is critical for managing resource contention and reducing scheduling delays. A detailed breakdown of responsibilities is provided in Appendix A.

4 Resource Scheduling in Venn

In this section, we outline the resource scheduling algorithm in Venn, starting with defining the scheduling problem in CL (§4.1). Next, we present our scheduling algorithm in two parts: determining the job scheduling order to minimize scheduling delay (§4.2) and matching devices to jobs to optimize response collection time (§4.3). Finally, we describe enhancements for real-world deployment (§4.4).

4.1 Problem Statement

Given a collection of CL jobs—along with their device requirements and resource demands—and a set of heterogeneous devices that arrive and depart over time, Venn should efficiently assign devices to CL jobs in order to reduce the average job completion time (JCT). The scheduling problem can be mathematically modeled as a multi-commodity flow (MCF) problem with integer constraints, where each CL job is modeled as a distinct commodity and each device serves as an intermediate vertex between the source and sink of its corresponding eligible CL job. Then the goal of this integer MCF problem is to minimize the average JCT of jobs, which is known to be NP-hard (Even et al., 1975). Even for its linear approximation, the time complexity is exacerbated by the planetary scale of devices involved and diverse device requirements from jobs, making existing solutions computationally infeasible.

Problem definition. Let $\mathbb{J} = J_1, J_2, \ldots, J_m$ be the set of jobs, with resource demands $\mathbb{D} = D_1, D_2, \ldots, D_m$, where D_i is the number of devices required by job J_i. Let $\mathbb{S} = S_1 \cup S_2 \cup \cdots \cup S_n$ be the set of available devices, where S_k is the subset of devices eligible for jobs with specific

requirements (e.g., hardware capacity), and $f(J_i) = S_k$ maps job J_i to its eligible device subset. The goal is to match each checked-in device $s \in S_k$ with one job J_i, where $f(J_i) = S_k, \forall s \in \mathbb{S}$, in order to minimize average JCT, which consists of scheduling delay and response collection time.

Tradeoff between scheduling delay and response collection. Minimizing scheduling delay often conflicts with minimizing response collection time. Assigning devices quickly reduces delay but may involve slower devices, increasing response time, since the response collection time is usually determined by the final responding device among the target number of participants. Waiting for faster devices, however, reduces response time at the cost of higher delay.

At its core, Venn aims to find a sweet spot in the trade-off in order to optimize average JCT. Specifically, we decouple the CL resource allocation problem as following two questions:

1. *How should jobs be ordered to minimize scheduling delay? (§4.2)*
2. *How should devices be matched to jobs to minimize response collection time while optimizing JCT? (§4.3)*

4.2 Intersection Resource Scheduling (IRS)

We now tackle the first question, minimizing the scheduling delay. Directly matching jobs to devices can be computationally expensive, especially when dealing with the immense scale of devices and jobs. The challenges posed by this problem are not solely due to its scale; they are also compounded by the diverse resource requirements of CL jobs. Their various requirements introduce intricate resource contention patterns, further complicating the scheduling.

To this end, Venn introduces the Intersection Resource Scheduling (IRS) problem to account for this resource contention. Basically, each CL job $J_i \in \mathbb{J}$ may compete for a subset of devices $S_k \in \mathbb{S}$, denoted as $f(J_i) = S_k$, where these resource subsets can exhibit relationships that are *inclusive, overlapping, or nested*. We created an integer linear programming (ILP) formulation (Appendix B) to optimally allocate resources to minimize scheduling delay and propose a heuristic to tackle the problem.

To tackle the scale of devices and jobs, Venn aims to determine a job scheduling order, where each checked-in device is assigned to the first eligible job in the order, rather than scattering resources across multiple jobs. Such a fixed job order can minimize the scheduling delay while reducing computational complexity.

With the objective of determining a job scheduling order, Venn first groups jobs $\mathbb{J}$ into *Resource-Homogeneous Job Groups* $\mathbb{G} = \{G_1, G_2, ..., G_n\}$ by their resource require-

Algorithm 1 Intersection Resource Scheduling

```
1:  Input: Job Groups 𝔾, Devices 𝕊
    Function Venn-Sched Job Groups 𝔾, Devices 𝕊
    ▷ Sort within job group (§4.2.1)
2:  for G_j in G do
3:      Sort J_i by D_i in ascending order, ∀J_i ∈ G_j
4:  end for
    ▷ Generate initial allocation S'_j for each group G_j
5:  S = ∪_{j=1}^{n} S_j
6:  Sort G_j by |S_j| in ascending order, ∀G_j ∈ 𝔾
7:  for G_j in G do
8:      S'_j = S ∩ S_j, S = S \ S'_j
9:  end for
    ▷ Allocate resource S'_j for each group G_j
10: Sort G_j by |S_j| in descending order, ∀G_j ∈ 𝔾
11: for G_j in G do
12:     if |S'_j| > 0 then
13:         for G_k ∈ 𝔾 : |S_k| < |S_j|, S_k ∩ S_j ≠ ∅ do
14:             m'_j, m'_k = get-queue-len()
15:             if m'_j/|S'_j| > m'_k/|S_k| then
16:                 S'_j = S'_j ∪ (S_j ∩ S_k)
17:                 S'_k = S'_k − S'_j
18:             else
19:                 break
20:             end if
21:         end for
22:     end if
23: end for
24: Return {G_j[0], S'_j}, ∀j ∈ [1, n]
```

ments, where each job group $G_j = \{J_i | f(J_i) = S_j, \forall J_i \in \mathbb{J}\}_{i=1}^{m_j}$ contains all jobs with the same resource requirement. Venn addresses the problem using a two-step approach, with each step occurring at a different level of scheduling granularity, as outlined in Algorithm 1: (i) Ordering **within a job group** to optimize local resource scheduling (§4.2.1). (ii) Merging orders **across job groups** to ensure global scheduling efficiency (§4.2.2). Venn invokes Algorithm 1 on job's request arrival and completion. By breaking down the overall problem into two steps, we further reduce the problem's complexity without affecting the scheduling efficiency. We provide theoretical insights to show the effectiveness of this problem decomposition in Appendix C.

4.2.1 Intra-Job Group Scheduling

Within each job group that shares the same device specifications, Venn prioritizes jobs based on their remaining resource demand (Algorithm 1 line 3). This ordering strategy aims to minimize the intra-group scheduling delay. Prioritizing jobs with smaller remaining resource demands has been shown to be effective in similar scheduling problems (Garey et al., 1976). We choose this locally optimal scheduling strategy with the observation that it aligns well with the

goal of achieving a globally optimal scheduling order. By default, the remaining resource demand refers to the needs of a single request within one round. However, it can also encompass the total remaining demand for all upcoming rounds, provided such data is available.

4.2.2 *Inter-Job Group Scheduling*

Addressing the scheduling problem across multiple job groups introduces an additional layer of complexity due to the intricate patterns of resource contention. Traditional scheduling algorithms, such as Random Matching and Shortest Remaining Service First (SRSF), are not designed to account for resource contention across job groups, leading to poor average JCT.

An effective scheduler should recognize and adapt to the resource contention patterns among jobs. Venn prioritizes groups with scarce resources (fewer eligible devices) to prevent delays from resource-rich groups. Additionally, when a particular resource type is in high demand, the scheduler should judiciously allocate intersected resources to the job group with a longer queue, as this group contributes more significantly to the average scheduling delay.

To achieve this, Venn allocates the intersected resources across different job groups based on two factors related to average scheduling delay:

1. *Amount of eligible resources allocated*: the job group with smaller amount of eligible resources may have a longer scheduling delay under the same condition.
2. *Queue length*: the job group with a longer queue length contributes more to the average scheduling delay as more jobs are waiting for the same type of resources.

Algorithm 1 outlines the steps Venn takes to allocate the current resource across job groups. First, Venn initializes resource allocation among job groups by starting to allocate resources to the job group with the most scarce resources (line 6). This results in an initial allocation plan with no resource sharing across job groups (lines 5–9), setting the stage for subsequent cross-group allocations.

To determine how to allocate intersected resources across job groups, Venn greedily evaluates whether a job group with more abundant resources should acquire intersected resources from groups demanding more scarce resources with the objective of minimizing the average scheduling delay. This evaluation starts with the job group possessing the most abundant resources (line 10). If the resources allocated to this group remain unclaimed by other groups (line 12), Venn will decide how much resources from subsequent job groups should be allocated to it. Specifically, Venn prioritizes job groups with longer queue lengths and fewer allocated resources, guided by a ratio that balances the number of affected jobs against the amount of allocated resources (line 15). If this ratio $\frac{m'_j}{|S'_j|}$ is larger than the one for the target resource group $\frac{m'_k}{|S'_k|}$, Venn reallocates resources accordingly (lines 16–17). Otherwise, the algorithm ceases to allocate additional resources to the current job group from remaining groups (line 19). The reason is that if this job group needs more resources, it should first take the resources from job groups with relatively abundant resources.

Function `get-queue-len()` (line 14) would return the number of jobs m' whose JCT would be delayed by potential group prioritization. For example, the affected queue length m' may contain jobs from other job groups that have been deprioritized previously. An easier way to approximate m' is to use the group queue length. If intersected resources have been decided to be allocated to job group G_j over G_k, Venn accumulates and updates their current resource allocation S'_j, S'_k and queuing length m'_j (line 16–17).

The **time complexity** of Algorithm 1 is $\max(O(m \log m), O(n^2))$, where m is the number of ongoing jobs and n is the number of job groups. We illustrate the **theoretical insight** behind the scheduling algorithm in Appendix D to illustrate the effectiveness of Venn's approach.

4.3 Device Matching

Now we focus on minimizing the response collection time, a significant contributor to the overall JCT, particularly when resource contention is low. Existing cluster-level device-to-job matching solutions, either stick to a certain job order such as FIFO and SRSF, or match devices without a strategic algorithm such as random match, where none of them optimizes the job response collection time.

Response collection time is usually determined by the last successfully responding devices. Hence, it can be reduced by allocating devices with similar higher capacity to the CL job. Meanwhile, these high-end devices have lower probability to fail due to their quick task execution.

However, as mentioned in Section 4.1, there is a trade-off between scheduling delay and response collection time. Intuitively, with limited device influx, priority should be given to minimizing the scheduling delay, which dominates the average JCT. On the other hand, with sufficient device influx to fulfill a job request within a short period, we should consider minimizing the response collection time while obeying the scheduling order given by Section 4.2.

To this end, we propose a resource-aware tier-based device-to-job matching solution to reduce the response collection time for each job as illustrated in Algorithm 2.

The matching algorithm is activated only for jobs that are currently served, as scheduled by Algorithm 1. For each

Algorithm 2 Device Matching

1: **Input:** Jobs J_i, Devices $S'_j \in Venn-SCHED(\mathbb{G}, \mathbb{S})$
2: $S = \{S^1, S^2, ..., S^V\}$ ▷ Evenly partition devices
3: $g_v = \frac{t^v}{t^0}, \forall v \in [1, V]$ ▷ Response time speed-up
4: **Function:** Venn-Match(Job J_i, Resource S'_j)
5: $c_i = \frac{t_{response}}{t_{schedule}}$ ▷ Assign tiers in a rotating manner ▷ Decide whether to trigger tier-based matching
6: $u = \texttt{randint}(0, V)$ ▷
7: **if** $V + g_u \times c_i < c_i + 1$ **then**
8: Update $S'_j = S'_j \cap S^u$ ▷ Assign tier u devices to J_i
9: **end if**
10: **Return:** $\{J_i, S'_j\}$

such job, Venn partitions the eligible devices into V tiers based on their hardware capabilities, where V denotes the granularity of this partitioning. If a job has been served before, Venn adaptively sets the tier partition thresholds based on the hardware capacity distribution of the devices that participated in earlier rounds. Otherwise, Venn forgoes tier-based matching and profiles the devices allocated to the job's current request to inform future device tier partitioning.

For each served job request, Venn randomly selects a device tier, denoted as S^u, to the job (Algorithm 2 line 5). This randomized tier assignment aims to expose each CL job to a diverse set of devices, rather than confining them to high-end devices. Given that the response collection time is determined by the slowest responding participant, tier-based assignment does not adversely affect this metric.

As illustrated in Figure 7, while tier-based matching may increase the scheduling delay by a factor of $V \geq 1$, it can concurrently reduce the response collection time by a factor of $g \leq 1$. The algorithm proceeds to perform such tier-based device-to-job matching for the job J_i only if its JCT can be reduced, i.e., $1 + c_i > V + c_i g_u$ (line 7). If the condition holds, Venn allocates device tier S_u to the job, effectively updating the set to $S'_j \cap S^u$. Meanwhile the leftover device tiers would be allocated to subsequent jobs in the job group, maximizing the utilization of available resources.

To determine the response time speed-up factor g for tier-based matching, we note that the device response time distribution adheres to a log-normal distribution (Wang et al., 2023). We use the 95th percentile as the statistical tail latency to account for the overall response collection time, thereby excluding failures and stragglers. Venn profiles and estimates the response collection time for each device tier $v \in [1, .., V]$ and subsequently computes the speed-up factor $g_v = \frac{t^v}{t^0}$ relative to a non-tiered scenario (line 3).

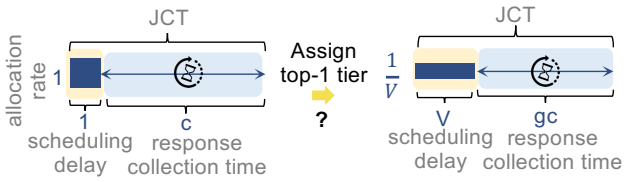


Figure 7. Visualized tier-based device-to-job matching condition.

4.4 Enhancements

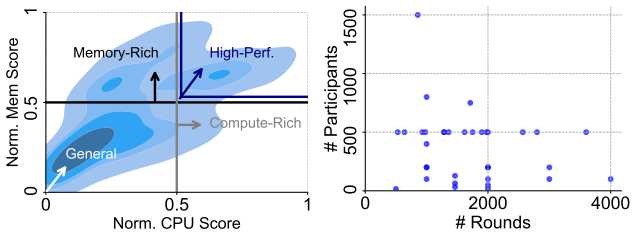
Dynamic resource supply. As shown in Figure 2a, the total available CL resources change significantly over time. To address this, Venn continuously records each device eligibility through a time-series database. This database is then queried for resource eligibility distribution from the past time window. However, relying solely on momentary eligible resource rates for input into the scheduling algorithm is inaccurate. This is primarily due to the varying resource arrival patterns, as demonstrated in Figure 2a. Since CL jobs span multiple days with diurnal resource patterns, Venn averages eligibility over 24 hours for robust scheduling. As a result, the scheduler can become both farsighted and robust, effectively accommodating the dynamic nature of resource availability.

Starvation prevention. Our heuristic can lead to larger CL jobs being starved due to the preference given to smaller jobs. This is not acceptable especially when the jobs are initiated by different CL developers who require performance guarantees. Venn grants fairness to jobs to avoid such starvation. Specifically, our goal is to guarantee that the scheduling latency of a job J_i is no worse than fair sharing, which is defined as $T_i = M * sd_i$, where M is the number of simultaneous CL jobs and sd_i represents the JCT without contention. Then, we adjust each job demand to be $d'_i = d_i \times (\frac{t_i}{T_i})^\epsilon$ to ensure fairness within a job group, and adjust each group queue length $q'_j = q_j \times (\frac{\sum_{J_i \in G_j} T_i}{\sum_{J_i \in G_j} t_i})^\epsilon$ to ensure fairness across job groups. t_i is the time usage of job J_i at the moment and $\epsilon \in [0, \infty)$ is a fairness control knob. When $\epsilon = 0$, the algorithm is identical to the one in Section 4.2. As $\epsilon \to \infty$, the fairness multiplier dominates the scheduling, resulting in maximum fairness. We show that Venn improves JCT over its counterparts with our starvation design (§5.5).

5 Evaluation

In this section, we evaluate the effectiveness of Venn through event-driven simulation and testbed experiments. Our key takeaways are:

- Venn reduces average job completion time (JCT) by up to 1.88× compared to state-of-the-art baselines, with-



(a) Device eligibility trace. (b) CL job demand trace.

Figure 8. Device and job trace used in experiments. (a) Devices are stratified into four regions to explore different overlap patterns. (b) The diverse workloads in experiments are derived from the job demand trace based on demand characteristics.

out compromising model accuracy, across diverse real-world CL workloads (§5.2).

- Venn outperforms its counterparts by intelligently scheduling jobs and matching devices to jobs, leveraging different design components (§5.3).
- Venn's benefits are robust under a wide range of CL workloads and environment setups (§5.5).

5.1 Experiment Setup

Testbed. To rigorously evaluate Venn, we employ a two-pronged approach. First, we have developed a high-fidelity simulator that replays client and job traces, effectively emulating the dynamics of the scheduling environment for large-scale evaluations. Second, we deploy real CL systems to execute actual CL jobs at a smaller scale of devices.

Resources. To faithfully emulate heterogeneous device runtimes, networking, and availability, we use device traces from FedScale (Lai et al., 2021a) (Figure 2) and AI Benchmark (Ignatov et al., 2019) (Figure 8a). Each unique device trace is limited to one CL job per day for realism.

CL jobs. We evaluate Venn over multiple synchronous CL jobs (Yang et al., 2018), where each successful training round requires a minimum of 80% target participants to report back within a deadline, which is set to be 5min - 15min depending on the round demand. Our approach also extends to asynchronous CL jobs, as scheduling decisions depend solely on remaining resource demand, not request submission timing. Jobs are drawn from diverse real-world CL applications (Xu et al., 2023; Yang et al., 2018; Hard et al., 2019; Ramaswamy et al., 2019; Warden, 2018; Sadilek et al., 2021), whose resource demand is depicted in Figure 8b. In the real CL experiment, each job aims to train a ResNet-18 (He et al., 2016) and MobileNet-V2 (Sandler et al., 2018) on FEMNIST dataset.

Workloads. Our evaluation includes five workload scenarios that sample differently from the job trace in Figure 8b to rigorously evaluate Venn's performance. *Even:* Sampled from all jobs, which is the default trace. *Small:* Uniformly sampled only from jobs with below-average total demand. *Large:* Uniformly sampled only from jobs with above-average total demand. *Low:* Uniformly sampled only from jobs with below-average demand per round. *High:* Uniformly sampled only from jobs with above-average demand per round. Default simulation and real-world workloads contain 50 and 20 jobs, respectively. Jobs arrive via a Poisson process with a 30-min average inter-arrival.

Device requirements are stratified into four categories based on the CPU and memory capacities (Figure 8a) to create various resource contention patterns where the eligible resources for each job may overlap, contain, or be within the eligible resources of other jobs. By default, each job is randomly mapped to one category among *General* resources, *Compute-Rich* resources, *Memory-Rich* resources, *High-Performance* resources.

	FIFO	SRSF	Venn
Even	1.38×	1.69×	1.87×
Small	1.48×	1.68×	1.78×
Large	1.64×	1.57×	1.72×
Low	1.55×	1.66×	1.88×
High	1.42×	1.41×	1.63×

Table 1. Summary of improvements on average JCT over random matching on different CL workloads.

Baselines. We compare Venn against FIFO, SRSF, and an optimized random matching. The random matching algorithm typically assigns devices to eligible jobs randomly but is optimized here to schedule jobs in a randomized order, reducing round abortions under contention and establishing a stronger baseline. Note that we only run Venn with the starvation prevention strategy in Section 5.5.

Metrics. Our primary performance metrics include the average job completion time (JCT). Note that while Venn does not explicitly optimize for CL job accuracy, it does not adversely affect it either.

5.2 End-to-End Performance

Venn achieves better average JCT Improvement. We assess the performance of different scheduling algorithms by evaluating its performance over different workloads. We report the average JCT speed-up for each scheduling algorithm compared to the random scheduling in Table 1. We observe that our scheduling algorithm consistently provides stable improvements in the average JCT across various workloads, which underscores the robustness of Venn.

Venn achieves faster convergence without affecting accuracy. We report the final model test accuracy of CL jobs under different schedules with the help of our CL system at a smaller experiment scale. As shown in Figure 9, we ob-

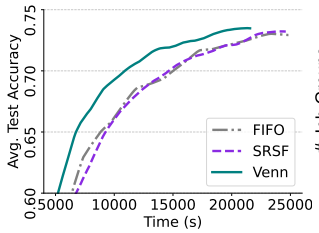


Figure 9. Venn does not affect the average test accuracy.

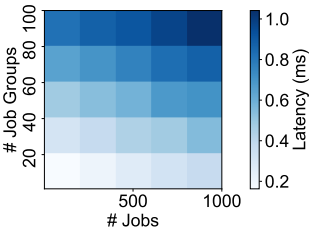
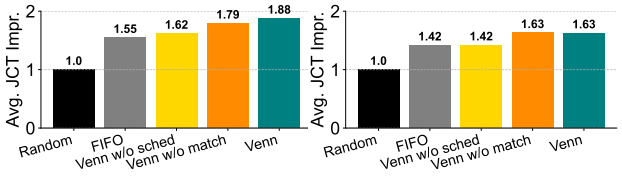


Figure 10. Venn introduces negligible overhead at scale.



(a) Low workload. (b) High workload.

Figure 11. Average JCT improvement breakdown.

	25th	50th	75th
Even	11.5×	7.2×	5.6×
Small	6.8×	5.2×	4.3×
Large	3.7×	2.9×	2.7×
Low	11.6×	7.5×	4.7×
High	5.1×	3.3×	3.1×

Table 2. Breakdown of average JCT improvement across jobs with lowest 25%, 50%, and 75% of total demands. Venn benefits more on smaller jobs.

	General.	Compute.	Memory.	High-perf.
Even	1.5×	7.2×	5.3×	3.9×
Small	0.9×	6.0×	2.8×	2.6×
Large	0.9×	3.7×	1.8×	2.6×
Low	0.8×	3.4×	2.1×	8.7×
High	0.8×	2.2×	2.2×	5.6×

Table 3. Breakdown of average JCT improvement across jobs that ask for General resources, Compute-rich resources, Memory-rich resources and High-performance resources. Venn benefits more on jobs that ask for scarcer resources.

serve that Venn does not affect the final model test accuracy but speeds up the overall convergence process.

Venn has negligible overhead. We emulated a large number of CL jobs and groups to evaluate the scheduler's scalability. Our results in Figure 10 demonstrate that the latency incurred by one-time triggering for scheduling and matching remains low, even with a substantial increase in job and group numbers. This is due to its low time complexity of $\max(O(m \log m), O(n^2))$, where m is the number of jobs and n is the number of device groups.

	FIFO	SRSF	Venn
General	1.46×	1.78×	1.94×
Compute-heavy	1.73×	2.08×	2.23×
Memory-heavy	1.68×	2.05×	2.27×
Resource-heavy	1.65×	1.90×	2.01×

Table 4. Average JCT improvement on four biased workloads.

5.3 Performance Breakdown

We present a performance breakdown of Venn, which consists of two parts: a job scheduling algorithm that determines the job order to minimize the scheduling delay, and a device-to-job matching algorithm to reduce the response collection time. We evaluate the performance of Venn with only the scheduling algorithm (Venn w/o matching), Venn with only matching algorithm and FIFO (Venn w/o scheduling), and Venn with both algorithms (Venn). We show the improvement of the average JCT over the default random scheduling for each component. As shown in Figure 11, the tier-based device-to-job matching algorithm primarily benefits low workload where the resource contention is small, which is aligned with our original design intention. The reason is that when resource supply is sufficient, the response collection time would dominate the JCT, which can be optimized by our matching algorithm.

To analyze the impact of Venn on different types of jobs, we break down jobs by their total demands and device requirements (Figure 8a), and then analyze the average JCT improvement for each type. Table 2 and Table 3 quantifies how Venn improves average JCT across varying total demands (25th, 50th, 75th percentiles) and eligibility types (General, Compute-rich, Memory rich, High-performance). Notably, jobs with smaller total demands and scarcer resources benefit the most from Venn.

5.4 Case Study on Biased Workload

This section delves into an in-depth analysis of Venn's adaptability and performance across four distinct workloads, each characterized by a specific bias in job resource requirements. These workloads include General, Compute-Heavy, Memory-Heavy, and Resource-Heavy categories. For example, the Compute-Heavy workload is structured such that half of its jobs are predominantly geared towards compute-intensive resources, with the rest evenly distributed across the other three resource types. This setup introduces varied queue lengths in different job groups, providing a robust testbed for evaluating Venn's capability in effectively managing these variations.

The design of these workloads aims to scrutinize Venn's proficiency in navigating diverse resource requirement distributions, while maintaining uniformity in job demands as illustrated in Figure 8b. The outcomes of these experiments are systematically presented in Table 4, offering insights into

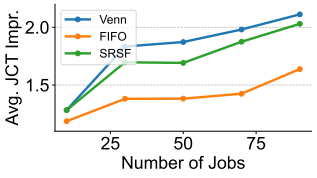


Figure 12. Venn outperforms FIFO and SRSF across different numbers of jobs.

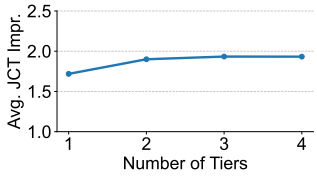
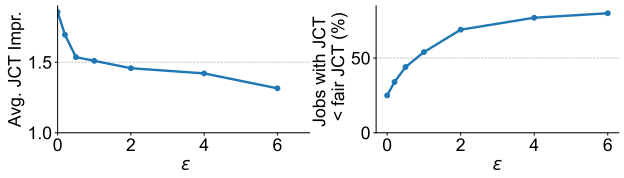


Figure 13. Venn's improvement across different numbers of tiers.



(a) Venn's improvement over different ϵ.

(b) Ratio of jobs meet the fair-share JCT.

Figure 14. Fairness knob.

the algorithm's performance under each workload scenario.

5.5 Ablation Study

Impact of number of jobs. We evaluate Venn with different numbers of CL jobs arriving over time. As the number of jobs increases, resource contention becomes more pronounced, highlighting the importance of efficient scheduling under such conditions. We present the average JCT speed-up with different numbers of jobs in even workload to demonstrate the effectiveness of our algorithm. As shown in Figure 12, Venn consistently provides improvement across various numbers of jobs, with its benefits becoming more pronounced as the number of jobs increases.

Impact of number of tiers. We evaluate the matching algorithm's performance across varying numbers of client tiers, ranging from a single tier to multiple. Figure 13 shows that increased tier granularity enhances device-to-job matching and improves performance. However, the gains plateau beyond a certain point, as finer tiers increase scheduling delay without yielding further reductions in response collection time. Thus, optimizing the number of tiers is crucial for balancing scheduling efficiency and JCT improvement.

Impact of fairness knob. We incorporated a fairness knob (ϵ) to strike a balance between performance and fairness. We tune the value of ϵ and report the average JCT speed-up against these values in Figure 14a. The results demonstrate that, as ϵ increases, the JCT speed-up tends to decrease. As shown in Figure 14b, the percentage of jobs that meet the fair-share JCT increases with the ϵ, where ϵ =2 gives 69% of jobs receive their fair-share JCT. This observation highlights the trade-off between performance and fairness within our CL resource scheduling algorithm, which can be fine-tuned by adjusting the value of ϵ.

6 Related Work

Cluster resource managers. There are many cluster resource managers that schedule resources with constraints (Thinakaran et al., 2017; Ghodsi et al., 2013; Narayanan et al., 2020). Existing ML cluster resource managers mainly focus on managing the stable data-center resources like GPU and CPU (Xiao et al., 2018; Mohan et al., 2022) in order to improve JCT, utilization and fairness (Peng et al., 2018; Chaudhary et al., 2020). Some research delves into GPU-specific optimizations (Gu et al., 2019; Yu et al., 2021; Hwang et al., 2021), while others co-design resource managers with the specific characteristics of ML workloads (Qiao et al., 2021; You et al., 2023). However, they are mostly designed for data center and fail to capture the level of availability and heterogeneity of CL resources, nor do they consider both scheduling delay and response collection time of CL jobs.

CL client selectors. Several recent works have studied client selection at the single CL job level. In addition to enforce device requirements including software version, hardware capacity and data quality, they further cherry-pick clients based on their state, system and statistical utility (Lai et al., 2021b; Abdelmoniem et al., 2023; Balakrishnan et al., 2022; Jee Cho et al., 2022; Chai et al., 2020; He et al., 2023) to speed up the training. However, they solely focus on response collection time (Liu et al., 2022a;b) and overlook the time required to acquire adequate resources. Additionally, optimizing individual CL job performance is insufficient as the deployment scale of CL applications continues to grow.

CL resource managers. Large companies including Apple, Meta and Google have proposed their CL infrastructures; however, CL resource management is not their primary focus. These resource managers simply adopt random device-to-job matching in various forms, resulting in suboptimal scheduling delays and response collection times.

7 Conclusion

In this paper, we present Venn, our CL resource manager designed to efficiently share large-scale heterogeneous device resources among multiple CL jobs with diverse requirements. Venn features a contention-aware job scheduling algorithm and a resource-aware device-to-job matching algorithm, aiming to minimize the average job completion time (JCT) for CL workloads. Our evaluation using a range of real-world CL workloads demonstrates that Venn improves average JCT by up to $1.88\times$.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for valuable feedback. This work was supported in part by NSF grant CNS-2106184 and a grant from Cisco.

REFERENCES

Abdelmoniem, A. M., Sahu, A. N., Canini, M., and Fahmy, S. A. Resource-efficient federated learning. *EuroSys*, 2023.

Balakrishnan, R., Li, T., Zhou, T., Himayat, N., Smith, V., and Bilmes, J. Diverse client selection for federated learning via submodular maximization. In *ICLR*, 2022.

Bonawitz, K., Eichner, H., and et al. Towards federated learning at scale: System design. In *MLSys*, 2019.

Bonawitz, K. A., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. Practical secure aggregation for federated learning on user-held data. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

Chai, Z., Ali, A., Zawad, S., Truex, S., Anwar, A., Baracaldo, N., Zhou, Y., Ludwig, H., Yan, F., and Cheng, Y. Tifl: A tier-based federated learning system. In *Proceedings of the 29th international symposium on high-performance parallel and distributed computing*, 2020.

Chaudhary, S., Ramjee, R., Sivathanu, M., Kwatra, N., and Viswanatha, S. Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning. In *EuroSys*, pp. 1–16, 2020.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. EMNIST: an extension of MNIST to handwritten letters. In *arxiv.org/abs/1702.05373*, 2017.

CoreML. Apple core ml. https://developer.apple.com/documentation/coreml.

Even, S., Itai, A., and Shamir, A. On the complexity of time table and multi-commodity flow problems. In *16th annual symposium on foundations of computer science (sfcs 1975)*, pp. 184–193. IEEE, 1975.

Garey, M. R., Johnson, D. S., and Sethi, R. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1976.

Geyer, R. C., Klein, T., and Nabi, M. Differentially private federated learning: A client level perspective. 2017.

Ghodsi, A., Zaharia, M., Shenker, S., and Stoica, I. Choosy: Max-min fair sharing for datacenter jobs with constraints. In *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013.

Gu, J., Chowdhury, M., Shin, K. G., Zhu, Y., Jeon, M., Qian, J., Liu, H., and Guo, C. Tiresias: A GPU cluster manager for distributed deep learning. In *NSDI*, pp. 485–500, Boston, MA, February 2019. USENIX Association. ISBN 978-1-931971-49-2. URL https://www.usenix.org/conference/nsdi19/presentation/gu.

Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., and Ramage, D. Federated learning for mobile keyboard prediction, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

He, S., Yan, Q., Wu, F., Wang, L., Lécuyer, M., and Beschastnikh, I. Gluefl: Reconciling client sampling and model masking for bandwidth efficient federated learning. *MLSys*, 2023.

Huba, D., Nguyen, J., Malik, K., Zhu, R., Rabbat, M., Yousefpour, A., Wu, C.-J., Zhan, H., Ustinov, P., Srinivas, H., et al. Papaya: Practical, private, and scalable federated learning. *MLSys*, 2022.

Hwang, C., Kim, T., Kim, S., Shin, J., and Park, K. Elastic resource sharing for distributed deep learning. In *NSDI*, 2021.

Ignatov, A., Timofte, R., Kulik, A., Yang, S., Wang, K., Baum, F., Wu, M., Xu, L., and Van Gool, L. Ai benchmark: All about deep learning on smartphones in 2019. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 3617–3635. IEEE, 2019.

Jee Cho, Y., Wang, J., and Joshi, G. Towards understanding biased client selection in federated learning. In *AISTATS*, 2022.

Lai, F., Dai, Y., Zhu, X., and Chowdhury, M. Fedscale: Benchmarking model and system performance of federated learning. *CoRR*, abs/2105.11367, 2021a. URL https://arxiv.org/abs/2105.11367.

Lai, F., Zhu, X., Madhyastha, H. V., and Chowdhury, M. Oort: Efficient federated learning via guided participant selection. In *OSDI 21*, July 2021b.

Liu, J., Jia, J., Ma, B., Zhou, C., Zhou, J., Zhou, Y., Dai, H., and Dou, D. Multi-job intelligent scheduling with cross-device federated learning. *IEEE Transactions on Parallel and Distributed Systems*, 34(2):535–551, 2022a.

Liu, J., Lai, F., Dai, Y., Akella, A., Madhyastha, H., and Chowdhury, M. Auxo: Heterogeneity-mitigating federated learning via scalable client clustering. *arXiv preprint arXiv:2210.16656*, 2022b.

Lv, C., Niu, C., Gu, R., Jiang, X., Wang, Z., Liu, B., Wu, Z., Yao, Q., Huang, C., Huang, P., et al. Walle: An {End-to-End},{General-Purpose}, and {Large-Scale} production system for {Device-Cloud} collaborative machine learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 249–265, 2022.

Mohan, J., Phanishayee, A., Kulkarni, J., and Chidambaram, V. Looking beyond {GPUs} for {DNN} scheduling on {Multi-Tenant} clusters. In *OSDI*, 2022.

Narayanan, D., Santhanam, K., Kazhamiaka, F., Phanishayee, A., and Zaharia, M. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pp. 481–498, 2020.

Paulik, M., Seigel, M., Mason, H., Telaar, D., Kluivers, J., van Dalen, R., Lau, C. W., Carlson, L., Granqvist, F., Vandevelde, C., et al. Federated evaluation and tuning for on-device personalization: System design & applications. *arXiv preprint arXiv:2102.08503*, 2021.

Peng, Y., Bao, Y., Chen, Y., Wu, C., and Guo, C. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *EuroSys*, pp. 1–14, 2018.

Qiao, A., Choe, S. K., Subramanya, S. J., Neiswanger, W., Ho, Q., Zhang, H., Ganger, G. R., and Xing, E. P. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *OSDI*, 2021.

Ramage, D. and Mazzocchi, S. Federated analytics: Collaborative data science without data collection. https://blog.research.google/2020/05/federated-analytics-collaborative-data.html, 2020.

Ramaswamy, S., Mathews, R., Rao, K., and Beaufays, F. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329*, 2019.

Reisizadeh, A., Mokhtari, A., Hassani, H., Jadbabaie, A., and Pedarsani, R. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *AISTATS*, 2020.

Sadilek, A., Liu, L., Nguyen, D., Kamruzzaman, M., Serghiou, S., Rader, B., Ingerman, A., Mellem, S., Kairouz, P., Nsoesie, E. O., et al. Privacy-first health research with federated learning. *NPJ digital medicine*, 4 (1):132, 2021.

Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Thinakaran, P., Gunasekaran, J. R., Sharma, B., Kandemir, M. T., and Das, C. R. Phoenix: A constraint-aware scheduler for heterogeneous datacenters. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017.

Wang, E., Kannan, A., Liang, Y., Chen, B., and Chowdhury, M. Flint: A platform for federated learning integration. *MLSys*, 2023.

Warden, P. Speech commands: A dataset for limited-vocabulary speech recognition. In *arxiv.org/abs/1804.03209*, 2018.

Xiao, W., Bhardwaj, R., Ramjee, R., Sivathanu, M., Kwatra, N., Han, Z., Patel, P., Peng, X., Zhao, H., Zhang, Q., et al. Gandiva: Introspective cluster scheduling for deep learning. In *OSDI*, pp. 595–610, 2018.

Xu, Z., Zhang, Y., Andrew, G., Choquette-Choo, C. A., Kairouz, P., McMahan, H. B., Rosenstock, J., and Zhang, Y. Federated learning of gboard language models with differential privacy. *arXiv preprint arXiv:2305.18465*, 2023.

Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D., and Beaufays, F. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

You, J., Chung, J.-W., and Chowdhury, M. Zeus: Understanding and optimizing gpu energy consumption of dnn training. *NSDI*, 2023.

Yu, P., Liu, J., and Chowdhury, M. Fluid: Resource-aware hyperparameter tuning engine. *MLSys*, 2021.

A Detailed Venn Responsibility

Venn delegates responsibilities such as device selection, device fault tolerance, and privacy protection to individual CL jobs. Device failures are both inevitable and difficult to predict in CL. Rather than imposing a one-size-fits-all solution, Venn empowers CL jobs to take the reins on fault tolerance based on their specific workloads and objectives. Therefore, Venn offloads handling device fault tolerance to CL jobs, who can better detect and react to device failures (e.g., deciding the amount of overcommit (Bonawitz et al., 2019)). Similarly, Venn offers CL jobs the freedom to design their own device selectors (Lai et al., 2021b), where they can incorporate customized resource criteria into their requests. Venn also does not interfere with job-specific privacy solutions such as secure aggregation (Bonawitz et al., 2016; Huba et al., 2022) or differential privacy (Geyer et al., 2017; Xu et al., 2023).

B ILP Formulation of IRS

We now formulate the IRS that allocates resources to jobs under the constraints with the objective of minimizing the average scheduling delay. Assume we have devices $\mathbb{S} = \{s_1, s_2, ..., s_q\}$ continuously arriving at times $\{t_i, t_2, ..., t_q\}$. There are m jobs $\mathbb{J} = \{J_1, J_2, ..., J_m\}$ with their resource demands $\mathbb{D} = \{D_1, D_2, ..., D_m\}$. Let e_{ij} be a binary variable in the eligibility matrix, which is set to 1 if device i is eligible to job j, and 0 otherwise. Let x_{ij} be a binary variable of resource allocation, which is 1 if device i is assigned to job j, and 0 otherwise.

We have to follow these constraints during the resource allocation:

$$\sum_{j=1}^{m} x_{ij} \leq 1, \forall i \in [1, q]$$

$$\sum_{j=1}^{m} x_{ij} \times e_{ij} \leq 1, \forall i \in [1, q]$$

$$\sum_{i=0}^{q} x_{ij} = D_j, \forall j \in [1, m]$$

Therefore, the scheduling delay of each job is determined by the time it acquires the last needed device, i.e., $T_j = \max_i(x_{ij} \times t_i)$ under these constraints. The overall objective can then be expressed as:

$$\min \frac{\sum_{j=1}^{m} T_j}{m}$$

C Theoretical Insight to the Heuristic of IRS

Lemma 1. *Given a diverse set of CL jobs with one round request, if jobs are scheduled optimally in terms of the average JCT, first within each job group and then across job groups, the resulting average JCT is optimal.*

Proof. Let us assume there is an optimal scheduling algorithm that optimizes the average JCT within each group, and there is an optimal scheduling algorithm which decides how to merge the job order across job groups to minimize the average JCT. Since the second step is assumed to provide optimal average JCT based on the previous within group job order, we only need to prove the global optimal schedule follows the order generated by the within job group step.

Venn employs smallest remaining job demand first algorithm within each job group. Since prioritizing jobs with smaller remaining resource demands has been shown to be effective in similar scheduling problems (Garey et al., 1976), we skip the proof that the scheduling algorithm within job group gives the local optimal average JCT for each group.

We prove the rest by contradiction. Assume that there exists an optimal schedule S that does not follow the order given by each job group. In this assumed optimal schedule S, let us say there are two jobs J_A and J_B in the same group such that J_A comes after J_B, but J_A has fewer resource requirements than J_B. Let us swap J_A and J_B to create a new schedule S'. Since J_A has fewer resource demand, the average JCT of S' will be less than that in S. This contradicts our original assumption that S is an optimal schedule, as we've found a schedule S' with a lower average JCT. Therefore, the assumption is false, and the order given by each job group (sorted by resource demands) must be part of the optimal schedule. If we have an optimal scheduling across job groups, the overall average JCT will be optimal.

□

D Effectiveness of Venn Scheduling Heuristic

To illustrate the effectiveness of Venn's approach, we start with proving Lemma 2, which considers a simplified case involving only two job groups with arbitrary resource contention patterns. Through mathematical proof, we can demonstrate that our algorithm achieves the optimal solution under this setting.

Lemma 2. *Given two job groups with arbitrary resource contention patterns, the scheduling plan generated by Venn as in Algorithm 1 is capable of minimizing the average scheduling delay, if a future resource allocation plan is set.*

To better prove the Lemma, we introduce a new representation of the scheduling problem in a more scalable way. Firstly, as depicted in Figure 15a, we represent the two job groups by two distinct sets of squares, where the area of each square corresponds to the size of the request demand

for that job.

Secondly, to visualize the temporal dynamics of resource allocation, we refer to Figure 15c. For the sake of this example, let's assume a constant inflow of 100 devices per time unit. Within this set, 'x' devices possess memory $\geq$ 2GB, while all 100 devices have memory $\geq$ to 1GB. The y-axis is partitioned into two segments: the 0 to 'x' range signifies devices with memory exceeding 2GB, and the 'x' to 100 range represents devices with memory ranging between 1GB and 2GB.

Resource allocation over time is illustrated using rectangles, each indicating the job request to which devices are assigned. For instance, in the right subfigure of Figure 15c, devices in the 0 to 'x' memory range are allocated to job group B at time 0, while those in the 'x' to 100 range are allocated to job group A. This representation allows us to dynamically track resource allocation across different jobs over time.

Proof. As shown in Figure 15a, there are m_A requests that ask for devices with 1GB memory and m_B jobs that request devices with 2GB memory, resulting in two job groups A and B. The devices constantly check-in and execute one CL task, where 100% devices with memory size $\geq$ 1GB and x% of the devices have memory size $\geq$ 2GB. Note that, the proof is not limited to the contention pattern draw in Figure 15a, it can be generalized to job group with intersected resource contention and give the same conclusion.

Based on Algorithm 1, the first step is to sort these jobs within each job group by job size in ascending order (Figure 15b). In the second step, we generate an initial resource allocation for each job group by focusing on the job group with the scarcest resources. This results in an initial allocation plan that avoids resource sharing across job groups, setting the stage for subsequent cross-group allocations.

Based on the group-level initial allocation plan (left subfigure in Figure 15c), we need to determine the job order across groups, that is, to decide whether to prioritize jobs from Group A over Group B (right subfigure in Figure 15c) at current time in order to achieve a smaller average scheduling delay. In this case, we focus on determining the order of the first job with size l in Group A and calculate the queuing delay difference (Δt) if we prioritize the first job from Group A over Group B.

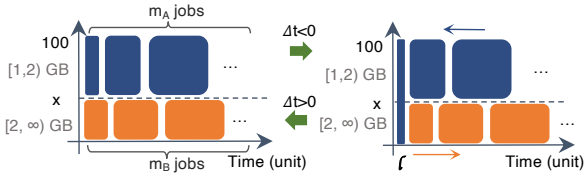
$$\Delta t = l * m'_B - (\frac{l}{1-x} - l) * m'_A$$

where m'_A, m'_B represents the number of remaining jobs whose queuing delay may be affected by this prioritization. Since the future resource allocation is set by the previous initial allocation or assumed to be given, m'_A, m'_B are feasible to get. We prioritize the first job from Group A only if $\Delta t < 0$, which gives $\frac{m'_A}{1-x} > \frac{m'_B}{x}$, otherwise we stick with the original plan. $\Delta t < 0$ is actually the prototype of the scheduling decision as in Algorithm 1 line 15.

□

(a) Resource supply and demand. There are 100% devices with memory size $\geq$ 1GB and x% of the devices have memory size $\geq$ 2GB. There are two job groups where Group A with m_A jobs requests for devices with memory size $\geq$ 1 GB and Group B with m_B jobs requests for devices with memory size $\geq$ 2 GB.

(b) Step 1: Sort within Job Group.



(c) Step 2: Schedule across Job Groups.

Figure 15. Venn scheduling algorithm.

By leveraging the conclusion of Lemma 2, Venn can further generalize to the scenario with more than two job groups with arbitrary resource contention patterns. Specifically, Venn greedily compares each pair of job groups (G_j, G_k) following the order. For each pair, Venn applies the logic proven in Lemma 2 to minimize the average scheduling delay between G_j and G_k.