User-Centric Machine Learning Systems

by

Jiachen Liu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2025

Doctoral Committee:

Associate Professor Mosharaf Chowdhury, Chair
Associate Professor Ang Chen
Assistant Professor Fan Lai, University of Illinois Urbana-Champaign
Associate Professor Raed Al Kontar

To my family

Jiachen Liu
amberljc@umich.edu
ORCID iD: 0000-0003-2317-1956

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Professor Mosharaf Chowdhury, for accompanying me on this journey. Thank you for your patience, your wisdom, and for believing in me even when the path wasn't clear. I also feel incredibly fortunate to have received support from so many intelligent and inspiring individuals during my five-year PhD journey at the University of Michigan.

Looking back on these five years, I wouldn't trade my time in the SymbioticLab for anything. It gave me a chance to shut out the noise and dive deep into the world of machine learning (ML) systems. But more than that, it gave me a chance to grow into myself. I joined the lab not as a typical academic, but Mosharaf was so patient and supportive, giving me so many opportunities to grow and find my footing. Mosharaf always held a very high standard for research, not just for getting papers accepted, but for the integrity of the work and tackling problems that genuinely matter in the real world. He constantly encouraged me to push beyond 'good enough' towards excellence. Along the way, one thing I appreciate most about Mosharaf is his honesty, where he always gave me direct feedback. It helped me to iterate and improve faster, not only in my research, but also in how I led projects, how I explained my ideas, and how I stood up and presented my work.

I was incredibly fortunate not only to have Mosharaf's guidance but also to collaborate with a group of exceptionally talented peers. Each collaborator brought different strengths to the table, complementing my own and igniting fresh perspectives and inspiration within our work. Fan Lai, who held my hand and taught me how to do research in my earlier years, had this incredible grasp of ML systems; he just could see the path through intricate research challenges and knew how to navigate them with such clarity. Jae-won Chung consistently maintained an exceptionally high standard for research, which motivated me to strive for perfection and ensure our work was bullet-proof. Coupled with his impressive execution skills, we efficiently completed many exciting and impactful projects together. Patrick Kon possesses a remarkable ability to think deeply about truly meaningful problems. His insights consistently inspired me to broaden my thinking and delve deeper into the core of our research questions. It has been a genuine joy and privilege to explore the frontiers of technology alongside these remarkable individuals.

I would also like to express my sincere gratitude to my committee members and other faculty who have shaped my academic experience: *Ang Chen, Fan Lai, Raed Al Kontar, Matei Zaharia, Aditya Akella, Harsha Madhyastha, Myungjin Lee, Mi Zhang, Lei Cao, Samuel Madden, Rada Mihalcea, David Jurgens, Lu Wang and Xinyu Wang.* I also extend my heartfelt thanks to my other wonderful collaborators: *Yiwen Zhang, Peifeng Yu, Insu Jang, Shiqi He, Jeff Ma, Ruofan Wu, Kevin Xue, Runyu Lu, Yiming Qiu, Zhenning Yang, Yibo Huang, Yinwei Dai, Jie You, Juncheng Gu, Hasan Al Maruf, Sanjay S. Singapuram, and Xiangfeng Zhu.* To my dear friends — *Hao Shen, Weiwei Chu, Jiecao Yu, Muqun Li, Qinqing Zheng, Yunjie Pan, Riyaaz Shaik, Noyan Tokgozoglu, Shibo Chen, Jiadi Zhang, Yujuan Fu, Haoyun Zhu, Rui Liu, Naihao Deng, Jinyang Li, Xinyu Ma, Yueming Li, Magdalena Calvillo, Luke Zhu, Hongjun Liu, Shuowei Jin, Zhiheng Yin, Yuxin Chen, Bohao Zhang and Ziyou Wu.* Finally, to my mentees — *Zhiyu Wu, Eric Ding, Qiuyi Ding, Haotian Zhang, Xinyi Zhu, Jingjia Peng, Yuxuan Zhu, and Yuxuan Xia.* It was a privilege to guide you, and I wish you all incredibly bright futures.

In the middle of my PhD journey, I am particularly fortunate to have witnessed the emergence of Generative AI, which I decided to pivot my research direction toward. I am grateful that Mosharaf was willing to take that risk and support me. This turned out to be a crucial and rewarding decision. During this period, I had the opportunity to work on a large-scale ML model training solution to support Llama training on up to 100k GPU clusters at Meta. Furthermore, my work on improving the quality of experience for LLM serving systems has contributed to changing how LLM service providers design their systems and deliver responses to users. Thanks to the academic freedom Mosharaf fostered, I also worked on a series of AI agent works, building a co-scientist AI agent to help automate research experimentation with enhanced rigor. This was an exciting and promising endeavor, as AI Agents will truly help researchers accelerate their scientific discovery. Many years from now, as GenAI services become increasingly sophisticated and seamlessly integrated into our lives, I will remember that Rome wasn't built in a day. It is being constructed, brick by brick, by our generation of researchers and engineers. I feel incredibly excited to be part of this GenAI wave and eagerly anticipate where it will lead.

These five years have been more than just a professional pursuit; they've been a profound journey of self-discovery and finding the meaning of my life. I am so lucky to have had such a great advisor, Mosharaf, who guided me onto the right path and was willing to trust me. I have felt both the profound pain and exhilarating joy of research. It was through repeated struggles that I found my passion, metaphorically breaking myself into pieces and rebuilding to discover the direction I excel in and the career I truly love. I hope to use my expertise in ML systems to make a positive impact on the world. This journey has taught me

that education isn't just about becoming a 'smart person', but about becoming a 'complete person'. I learned how to persevere through challenges, collaborate effectively, communicate complex ideas, and critically assess my own work and the world around me.

Nevertheless, these five years have been more than just a professional pursuit; they've been a profound journey of self-discovery, finding the meaning of my life. I have felt both the profound pain and exhilarating joy of research. It was through repeated struggles that I discovered the direction I excel in and the career I truly love. This journey has taught me that education isn't just about becoming a 'smart person', but about becoming a 'complete person'. I learned how to persevere through challenges, collaborate effectively, communicate complex ideas, and critically assess my own work and the world around me. I hope to use my expertise in ML systems to make a positive impact on the world. Looking back, the me from five years ago would never have predicted the person I am today, and I hope the next five years are just as unpredictable.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

ABSTRACT

Over the past five years, artificial intelligence (AI) has evolved from a specialized technology confined to large corporations and research labs into a pervasive tool integrated into everyday life. While AI extends its reach beyond niche domains to individual users across diverse contexts, the widespread adoption has given rise to new needs for machine learning (ML) systems to balance user-centric experiences—such as real-time responsiveness, accessibility, and personalization—with system efficiency, including operational cost and resource utilization. However, designing such systems is complex due to diverse AI workloads—spanning conversational services, collaborative learning (CL), and large-scale training—as well as heterogeneous computing resources, ranging from cloud data centers to resource-constrained edge devices.

My research aims to address these challenges and achieves these dual objectives through a set of design principles centered on server-client co-designed resource schedulers. These principles emphasize:

- **User-Centricity**: Our system design centers around the end user's needs, delivering user-friendly, affordable, and personalized AI services while also preserving user privacy.
- **Workload-Aware ML System Design**: The system is tailored to the unique demands of various AI workloads and computing environments, optimizing their specialized scheduling objectives.
- **Server-Client Collaboration**: Our system design leverages a server-client co-design paradigm, enabling collaborative efforts between both sides to make more informative scheduling decisions so as to enhance both user experience and server-side efficiency.

Our contributions are threefold. First, we propose Andes to address the critical need for real-time responsiveness in LLM-backed conversational AI services by introducing the concept of Quality-of-Experience (QoE). It proposes a co-designed solution including a server-side token-level request scheduling algorithm that dynamically prioritizes token generation based on user-centric QoE metrics, and a client-side token buffer to smooth the token streaming experience. This approach significantly enhances user experience during peak demand and achieves substantial GPU resource savings.

Second, we propose Auxo to deliver better personalized AI services in CL, addressing statistical data heterogeneity and resource constraints of end users. It introduces a scalable client-clustering mechanism that groups users into cohorts with lower intra-cohort heterogeneity, minimizing the impact of data heterogeneity on model performance. Furthermore, Auxo incorporates a cohort affinity mechanism, allowing clients to join preferred clusters while maintaining user privacy. This approach improves the personalized model performance and training efficiency in real-world CL scenarios.

Third, with the increasing demand for CL jobs, we propose Venn to efficiently share heterogeneous edge resources in multi-job CL environments. It designs a resource scheduler that proactively resolves complex resource contention to accelerate job completion times. It features a job offer abstraction that enables client resources to identify eligible jobs based on their local capabilities without exposing sensitive information. This significantly reduces job completion times and improves resource efficiency for CL jobs.

In conclusion, this thesis contributes to the field of machine learning systems by addressing critical challenges in making AI more user-centric and efficient. Guided by the principles of user-centricity, workload-aware design, and server-client co-design, this dissertation offers practical solutions to the complexities of diverse AI workloads and heterogeneous computing resources. As the field progresses, these insights will guide the development of a more connected, efficient, and human-centered AI ecosystem.

CHAPTER 1

Introduction

1.1 AI Extending Its Reach to Everyone

Five years ago, artificial intelligence (AI) was largely limited to niche domains like image recognition and recommendation systems, primarily utilized by large corporations or research labs [107, 14]. Throughout my doctoral research, we have observed AI's rapid evolution into a technology that now seamlessly integrates into individuals' everyday life and professional workplace [46]. This shift, propelled by advancements in generative AI models like ChatGPT—which boasted over 400 million weekly active users by early 2025 [82, 117]—has extended AI's reach beyond exclusive, resource-rich corporations to individual users across diverse contexts [43].

Today, AI supports a wide range of tasks—assisting essay writing for students, accelerating coding tasks for developers [74], helping generate images based on text for designers [134]—delivering benefits to a broad population. As AI continues to embed itself into every corner of the world, its presence in daily life and diverse sectors will become as commonplace as essential utilities like water, electricity, and the internet. This trend of democratizing AI reveals critical needs in machine learning (ML) system design to balance user-centric experiences [145, 140] with server-side efficiency:

1. **User-Centric Experience**: To ensure AI benefits and satisfies everyone, ML systems must be designed to deliver AI services that are user-friendly, affordable, and personalized, while preserving users' privacy [3].

2. **System Efficiency**: The growing number of AI use cases and users increases pressure on computational infrastructure and operational costs [104]. This efficiency is vital to delivering AI to a broader audience, because it enables providers to scale various AI services without escalating expenses and maintain performance under resource constraints. To ensure the efficient delivery of AI services to this expanding demand, ML

systems should be designed to maximize computational efficiency, minimize operational costs, and sustain reliable service even under peak usage conditions.

1.2 System Challenges Towards Pervasive AI

However, as AI become more pervasive, they encounter substantial system-level challenges that must be addressed to pave the way for truly democratized and efficient AI. These challenges stem from the inherent diversity and heterogeneity of the new AI landscape.

1. **Diversity of AI Workloads:** The first primary challenge arises from the diversity of AI workloads [65], which span applications such as conversational AI services, experimental model training, on-device collaborative learning, and agentic AI. These workloads differ in goals and resource requirements: ranging from maximizing Quality-of-Experience for real-time user interactions [98], minimizing training job makespan for rapid experimentation [173], improving the model accuracy for collaborative learning applications, to reducing average job completion time for multi-collaborative learning job efficiency [97].

2. **Heterogeneity of Resources:** Another substantial challenge arises from the heterogeneous nature of computation resources. As AI becomes integral to the average person's daily tasks, these pervasive AI workloads increasingly rely on diverse hardware resources, ranging from powerful GPU clusters in cloud datacenters to resource-constrained edge devices like smartphones and personal computers [65]. Such resource heterogeneity manifests as significant disparities in computational capacity, available memory, network latency, and device availability, requiring dedicated scheduling approaches. Furthermore, end-user data, as another valuable resource to ML model performance, varies considerably in terms of distribution and quality [57, 96], adding another layer of complexity.

The workload diversity and resource heterogeneity require specialized system design along with dedicated resource management strategies to achieve better system efficiency.

1.3 Efficient and User-Centric ML Systems

In this dissertation, we argue that rather than simply squeezing efficiency, we aim to develop ML systems through sophisticated resource management with the objectives of enhancing

both end-user experience and server-side efficiency. Our solutions are informed by a comprehensive understanding of AI workloads—encompassing job-specific requirements and the intrinsic characteristics of available resources. Moreover, our solutions leverage a server-client co-design paradigm, enabling collaborative efforts from both sides to achieve these dual objectives. This approach is exemplified across three key contributions, each addressing distinct challenges while collectively advancing the broader vision of building ML systems towards pervasive AI.

Enhancing Real-Time User Experiences in Conversational AI Services: Conversational AI services powered by large language models (LLMs) have rapidly emerged as a primary and intuitive interface for individuals to interact with AI, through text or audio interfaces, acting as a key catalyst for AI's integration into everyday life. As an important step towards realizing pervasive AI, these conversational AI services must maintain real-time responsiveness and smooth token generation to sustain user engagement and satisfaction. However, existing LLM serving systems predominantly focus on traditional system-centric metrics such as throughput or latency, often neglecting direct user experience. This oversight leads to unpredictable token delivery delays and user dissatisfaction, particularly during bursty request arrivals at peak times. Moreover, to mitigate such delays, service providers often need to overprovision resources, inflating server-side costs and limiting affordability and equitable access for a broader user base.

To address this, we are the first to introduce the concept of Quality-of-Experience (QoE) explicitly tailored to AI conversational services [98]. Our solution features a token-level scheduling algorithm on the server side that dynamically prioritizes token generation based on user-centric QoE metrics, pinpointing the unique bottlenecks in LLM serving workflows. Additionally, we co-design a token buffer on the client side to smooth token streaming, further enhancing perceived responsiveness. Our extensive evaluations demonstrate significant QoE improvements, including up to a 4.7× enhancement in user experience during bursty request periods, alongside substantial GPU resource savings of up to 61%, ensuring efficient and equitable access to conversational AI.

Empowering Diverse End Users via Cohort-Based Collaborative Learning: Personalized AI models are vital for serving a global, diverse population. However, the inherent heterogeneity of user data across edge resources pose significant challenges to delivering effective personalization. Centralized ML model training approaches are impractical due to privacy concerns and high data-transfer costs, making collaborative learning (CL) paradigms, such as federated learning, a promising alternative for training personalized models without

compromising user data. To overcome the data heterogeneity challenges, some CL solutions try to find similar user groups and train the model; however, they often make unrealistic assumptions of full device availability and unlimited computational resources, which have limited practical applicability.

We propose a cohort-based CL framework, Auxo [96], that addresses these challenges through a scalable client-clustering mechanism that adaptively groups statistically similar clients into cohorts, while explicitly accounting for real-world edge resource characteristics like limited availability and constrained computational resources. Moreover, we co-design a cohort affinity mechanism, which enables clients to join their preferred cohorts, which empowers the stateless coordinator to efficiently match clients to appropriate groups while preserving privacy through minimal data disclosure. With reduced intra-cohort data heterogeneity, our approach accelerates convergence and improves the performance of personalized models. This solution advances pervasive AI by making CL models more personalized and performant on diverse populations. Our results show significant improvements, including 2.1%-8.2% increases in final accuracy, up to 2.2× faster convergence, and reductions in accuracy bias (4.8%-53.8%), thereby making personalized CL practical and equitable for diverse user populations.

Optimizing Resource Sharing in Multi-Job Collaborative Learning Environments: With AI democratization driving an increasing number of CL jobs, the demand for efficiently sharing limited resources—such as edge devices—is becoming increasingly critical. However, running multiple CL training or inference jobs simultaneously often results in complex resource contention, where jobs compete for the same set of edge resources based on their specific requirements, directly delaying job completion times and degrading resource efficiency. Current resource management approaches typically treat each CL job in isolation, assuming edge resources are abundant. Moreoever, for resource management approaches that account for multiple CL, they fail to capture the complex contention patterns due to the diverse resource requirements among CL jobs—such as varying objectives, data availability, hardware, and software needs. This complex resource contention pattern can be visualized as overlapping circles in a Venn diagram, where the eligible resources for each job are limited and may intersect, be contained within, or encompass those of other jobs. Consequently, this leads to suboptimal resource utilization and prolonged job completion times.

We introduce an efficient resource scheduler, Venn [97], that coordinates across multiple concurrent CL jobs, systematically capturing these contention patterns and strategically allocating resources to minimize the average scheduling delay and response collection time. On the end-user side, we propose a job offer abstraction, whereby the scheduler presents

available jobs along with their requirements, enabling users' devices to identify eligible jobs based on their local resource information, therefore addressing edge resource uncertainty and preserving privacy through controlled information disclosure. By proactively resolving resource contention, our approach significantly reduces average job completion time by up to 2.25× compared to state-of-the-art baselines, effectively supporting larger-scale concurrent CL scenarios. Venn advances pervasive AI by enabling more efficient and scalable resource sharing, allowing a broader range of users and organizations to participate in CL with better resource efficiency.

1.4 Dissertation Plan

The remainder of this dissertation is organized as follows:

- **Chapter 1** provides an overview of AI's expanding reach trends and key system challenges that need to be addressed.
- **Chapter 2** introduces Andes, a Quality-of-Experience (QoE) aware LLM serving system. We define a novel QoE metric for AI conversational services, along with a token-level scheduling algorithm that optimizes user experience while reducing GPU resource usage.
- **Chapter 3** presents Auxo, a cohort-based collaborative learning system that enables personalized AI model training across diverse edge devices. We detail our adaptive client clustering mechanism and cohort affinity approach that addresses real-world challenges of data heterogeneity and resource constraints.
- **Chapter 4** describes Venn, a resource scheduler for multi-job collaborative learning environments. We introduce our job offer abstraction and contention-aware scheduling algorithm that efficiently manages shared resources across concurrent collaborative learning jobs.
- **Chapter 5** concludes the dissertation by summarizing my perspectives on AI trends and lessons learned for ML system design, discussing the impact of the presented research, and outlining promising directions for future work.

CHAPTER 2

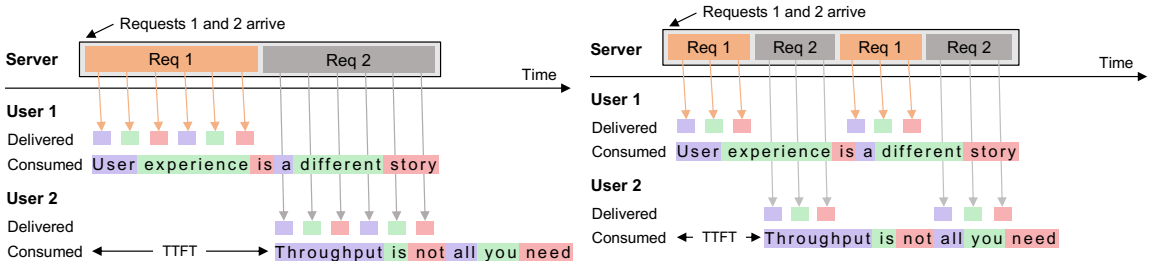
Quality of Experience in Conversational AI Services

Large language models (LLMs) are at the core of *conversational AI services*, such as chatbots, which provide live user interaction by incrementally streaming text or audio. While such services are *user-centric* by nature, we find that existing LLM serving systems are primarily optimized for *server-centric* metrics like token generation throughput, failing to deliver a good user experience.

The goal of this paper is to design an LLM serving system for conversational AI services that can satisfy more concurrent user requests without additional GPU resources. First, in order to quantify user experience, we introduce a mathematical definition of Quality-of-Experience (QoE) that considers each user's end-to-end interaction timeline. Based on this, we propose Andes, an LLM serving system that combines a server-side token-level preemptive scheduler that dynamically prioritizes requests based on their expected QoE gain and GPU resource usage, and a client-side *token pacer* that delivers tokens to users at a smooth, digestible pace. Compared to state-of-the-art LLM serving systems, Andes can serve up to 1.74× more requests under bursts with the same GPU resource while achieving comparable or higher QoE.

2.1 Introduction

Large language models (LLMs) [152, 27, 176, 40, 70, 12, 8, 149] have revolutionized many user-facing online applications. Particularly, conversational AI has emerged as a dominant use case, driving over 60% of LLM-backed applications [52], including chatbots, virtual assistants, language translation, and customer support. The meteoric rise of ChatGPT [115], now with over 400 million weekly active users [62], underscores the massive scale and demand for such services.



(a) Existing systems significantly degraded User 2's TTFT.

(b) QoE-aware systems schedule token generations over time. User 2's TTFT is improved without affecting User 1's token consumption.

Figure 2.1: Timelines for token generation, delivery, and user consumption for (a) existing and (b) QoE-aware LLM serving systems, both with capacity 1. Users consume tokens at their reading speed.

Conversational AI services provide *interactive* conversations between the user and autoregressive LLMs that generate text tokens[1] one by one. Generated tokens are incrementally streamed to and consumed by users – as written text or synthesized speech – to provide a conversational experience. This interactivity makes conversational AI services inherently *user-centric*, prioritizing user experience as a key optimization goal. Particularly, user experience for conversational AI hinges on mainly (1) the timely delivery of initial tokens, and (2) consistent delivery of subsequent tokens at a smooth and digestible pace (§2.2.1).

Existing LLM serving systems [172, 83, 183, 11, 160], however, often prioritize throughput over user experience, leading to suboptimal initial and subsequent token delivery (§2.2.2). Particularly during periods of request and/or prompt token load surge, simplistic first-come-first-served (FCFS) scheduling policies adopted by existing systems cause head-of-line blocking. This inflates the user's initial wait time or time-to-first-token (TTFT), as shown in Figure 2.1a.

However, opportunity lies in the observation that existing systems generate tokens at a pace that significantly exceeds the typical user's reading or listening speed, which does not further improve user experience because the token consumption speed of users is capped at their reading speed (for text) or listening speed (for speech). This allows the server to control the generation and delivery of each token in each request to ensure a smooth and digestible pace, which not only enhances user experience but also creates opportunities to serve more users simultaneously with the same amount of hardware resource (§2.2.3). As shown in Figure 2.1b, we can redistribute computation resources across requests over time

[1]LLMs process and generate text in units of *tokens*. For instance, the word "streaming" may be broken down into two tokens: "stream" and "ing."

at token granularity to improve TTFT without affecting the token consumption timeline of other users.

Existing LLM serving systems have so far failed to recognize this opportunity because there is a fundamental *misalignment* between user experience and optimization metrics used by existing systems. Server-centric metrics (e.g., token generation throughput [172, 83]) or simple statistics derived from a subset of token delivery timestamps (e.g., average/P90/P99 time-per-output-token or TTFT [183, 11, 160]) fail to fully capture user experience during the whole interactive session. To accurately reflect user experience, it is vital to consider the user's *end-to-end* interaction timeline.

To capture this timeline effectively, we propose a mathematical definition of Quality-of-Experience (QoE) that can be used as an optimization objective for conversational AI services (§2.3.1). This is challenging in itself, as it needs to capture the user's end-to-end interaction timeline and reflect our intuitions of good and bad user experiences. For instance, long pauses before and during token delivery should degrade the right amount of QoE, while generating tokens faster than the user's consumption speed should not improve QoE.

To align system objectives with QoE, we design Andes, an LLM serving system for conversational AI services. Andes co-designs the LLM inference server and the user-side client (§2.3.2). On the server side, Andes adopts a preemptive request scheduler that operates at the token granularity to optimize QoE (§2.4). Designing such a scheduler is challenging:

(a) **Diverse and unpredictable resource demand.** Requests have varying prompt lengths, response lengths, and QoE parameters (e.g., user reading speed). This dynamism and diversity preclude the adoption of simple one-size-fits-all scheduling policies.

(b) **Interdependent user experience objectives.** On the one hand, we want to minimize the initial waiting time for users by serving more requests in parallel that maximizes GPU memory utilization. On the other hand, serving with a larger batch size may slow down the generation of individual tokens, potentially failing to meet user's ideal token delivery speed. With request input lengths and QoE parameters varying widely over time, this is critical but challenging to control.

(c) **Token-level preemption overhead.** While token-level request preemption is promising for QoE optimization (Figure 2.1), such fine-grained scheduling introduces additional overhead that may degrade system throughput. This may in turn degrade the QoE of every request.

To handle these challenges, Andes continuously monitors the attained QoE and compute/memory usage of each request to dynamically prioritize requests that are at risk of degrading their QoE. Moreover, Andes modulates the frequency of decision-making and incorporates the overhead of request preemption and restart in its scheduling decision.

Age group	Reading speed
18-24 (28.0%)	236 WPM
25-44 (51.9%)	200 WPM
45-54 (11.2%)	192 WPM
55-64 (5.6%)	185 WPM
65+ (3.3%)	175 WPM

Table 2.1: Reading speed by age group [28].

Language	Speaking speed
English (79.3%)	150 WPM
Chinese (7.0%)	158 WPM
Korean (6.9%)	150 WPM
French (3.6%)	195 WPM
Spanish (3.2%)	218 WPM

Table 2.2: Speaking speed by language [119, 19].

Andes's server implements push-based streaming, which transmits tokens to the client as soon as they are generated. This is because the server is typically resource-constrained, and generated tokens are delivered exclusively to the user who submitted the request. To provide a smooth streaming experience at a consistent pace, Andes's server works with the client-side *token pacer*, which temporarily buffers tokens generated by the server and delivers them to the user precisely at the user's consumption speed (§2.5).

We evaluate Andes with LLMs with various sizes (3.8B to 70B) and architectures (Dense and Mixture-of-Experts, Multi-Head Attention and Grouped-Query Attention) on three different request datasets with varying input and output sequence lengths (§4.5). We also evaluate Andes in the context of both *text-based* and *audio-based* conversational AI services. Compared with state-of-the-art LLM serving systems (vLLM [83] and Sarathi-Serve [11]), given the same amount of GPU resource, Andes serves up to 1.74× more requests under bursts while maintaining the same high QoE, reduces average TTFT by up to 4.8×, and lowers silence time ratio in text-to-speech (TTS) playback by up to 3.3× for audio-based conversational AI services.

Overall, we make the following contributions:

- We identify a misalignment between the user experience of conversational AI services and optimization metrics pursued by state-of-the-art LLM serving systems.
- We propose a formal definition of QoE for conversational AI services that captures their user experience.
- We design and implement Andes, an LLM serving system that co-designs the server (token-level preemptive request scheduler) and the client (token pacer).
- We evaluate Andes on diverse workloads and show that it significantly improves QoE and serving capacity.

2.2 Background and Motivation

In this section, we first introduce conversational AI services and their user experience (§2.2.1). We then discuss the limitations of existing LLM serving systems (§2.2.2) and opportunities for improving user experience (§2.2.3).

2.2.1 Conversational AI Services and User Experience

Conversational AI services including chatbots, virtual assistants, and real-time translation aim to provide a smooth conversational experience by incrementally streaming text to users, either in visual text or audible synthesized speech, instead of having them wait for tens of seconds before the whole response is generated.

The user's interaction timeline with such conversational AI services can be divided into (1) the *initial waiting phase* and (2) the *token consumption phase*. During the former phase, time-to-first-token (TTFT) matters; the first token should be delivered to users before they lose patience. The target TTFT should depend on the service, be it constant or proportional to the prompt length. During the latter phase, the server's token delivery speed (TDS) should match the users' consumption speed (Tables 2.1 and 2.2). More specifically, a slower TDS hurts user experience as users will perceive the service as slow or lagging, but a TDS faster than the user's consumption speed does not improve user experience.

2.2.2 Existing Systems Provide Poor User Experience

Existing LLM serving systems [172, 83, 183, 11] fail to deliver good user experience for conversational AI services. To gain deeper insight into how existing LLM serving systems behave with real-world conversational AI services, we replay a one-hour slice of BurstGPT [157], a real-world LLM serving request trace, with vLLM [83] serving Phi-3.5-MoE 16×3.8B [8] on 8 × A100 GPUs. Figure 2.2 (first row) shows vLLM's GPU memory utilization and the number of running and waiting requests over time. Above all, we can observe large and frequent load surges caused by spikes in both request rate (second row) and the number of tokens in requests (third row). During surge periods, vLLM's first-come-first-served (FCFS) scheduling policy – adopted by most existing LLM serving systems as well – causes significant head-of-line blocking and queuing delay. This leads to an average TTFT of 10.4 seconds, which is likely beyond the patience limit of most users [13]. Conversely, vLLM's average TDS of 11.2 tokens/s far exceeds typical user consumption speeds (Tables 2.1 and 2.2). Delivering tokens faster than users can consume them does not improve user experience, as user's consumption timeline remains unchanged.

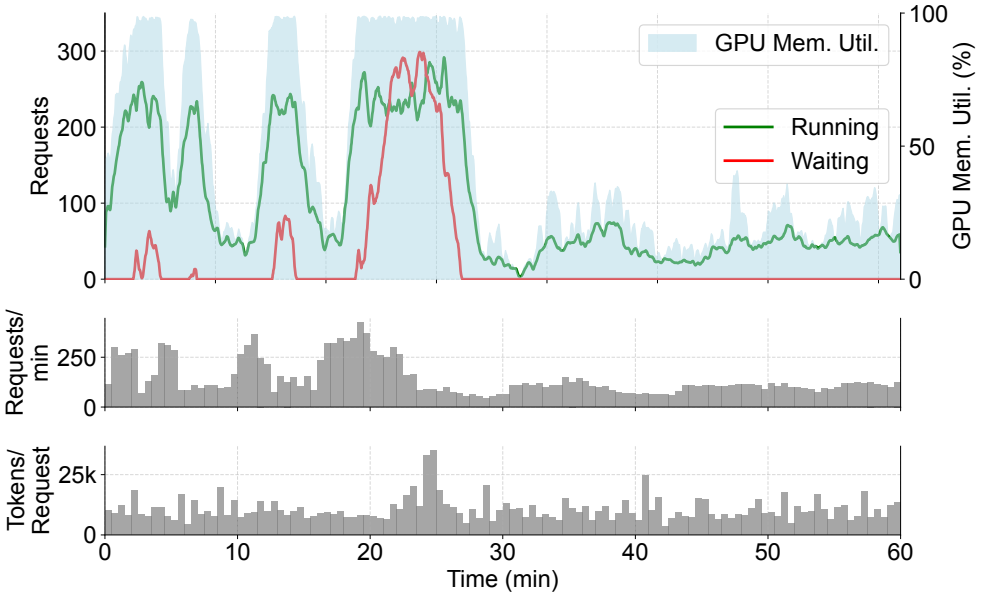


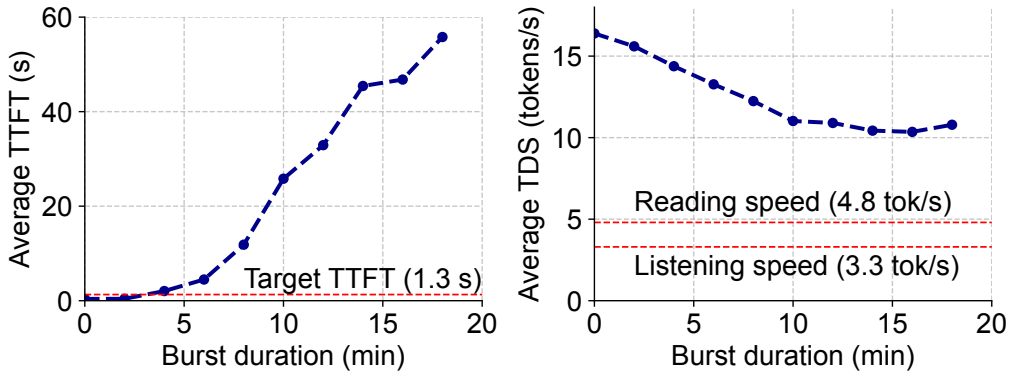
Figure 2.2: vLLM serving requests from BurstGPT. During load surges, vLLM builds up a very long queue due to head-of-line blocking. Still, under normal load, GPU memory is underutilized.

To understand this further under varying degrees of load surges, we create synthetic 20-minute request traces that contain a load surge period with varying durations of bursts (See Section 2.6.3 for more details). Figures 2.3a and 2.3b show the average TTFT and average TDS of vLLM serving Phi-3.5-MoE 16×3.8B, respectively. Consistent with the BurstGPT trace, we observe: (1) TTFT is significantly inflated due to head-of-line blocking, even under moderate load surges, and (2) TDS consistently exceeds any reasonable user token consumption speed, even under severe load surges.

2.2.3 Opportunities for Improving User Experience

In essence, Section 2.2.2 reveals a *misallocation* of computational resources over time – instead of continuing to allocate compute to requests that have generated enough tokens for their users to consume, it makes more sense to redistribute compute to requests that have not generated enough (or, *any*) tokens for their users. This motivates a preemptive scheduling approach that can reduce TTFT inflation caused by head-of-line blocking. Because we know each user's speed – and thus timeline – of token consumption, we can preempt requests that have generated sufficient tokens and keep preemption transparent to user experience.

How much is the potential gain under ideal circumstances? Opportunity fundamentally comes from the gap between the server's excessive token generation speed and the user's



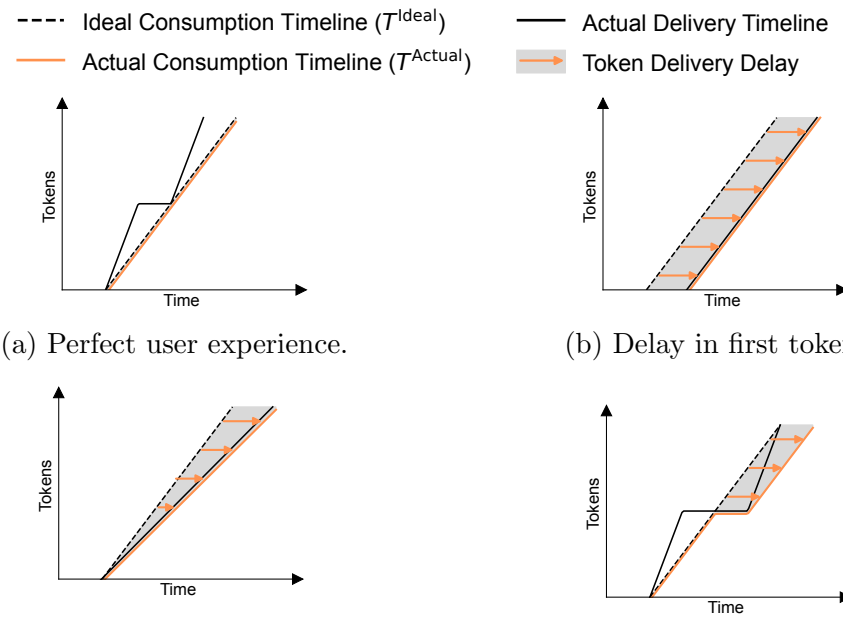
(a) Average TTFT (b) Average TDS

Figure 2.3: vLLM's average TTFT and TDS while varying the duration of the load surge in our 20-minute trace. TTFT target is set to 1.3 s, as recommended by Google for web page loading [13]. User reading/listening speeds are 4.8 and 3.3 tokens/s, derived from Tables 2.1 and 2.2. TTFT is inflated significantly, whereas TDS remains excessively fast compared to any token consumption speed.

token consumption speed. Instead of serving the same set of requests to completion and over-generating tokens for them, an ideal preemptive scheduler could switch to different requests back and forth while ensuring that each request generates tokens at a rate that matches the user's consumption speed. For instance, for the workload in Figure 2.3, the server generates 11 tokens/s under moderate load surges (with burst duration 10 minutes), while the user can only consume 4.8 tokens/s. Therefore, each request just needs to be served for 1 second every $11/4.8 = 2.3$ seconds, allowing the server to serve 2.3× more requests concurrently. This is an ideal upper limit estimation assuming no scheduling, prefill, preemption, and resumption overhead, but we show in Section 4.5 that Andes can still realize a significant portion of the ideal gains.

2.3 Andes Overview

Andes is an LLM serving system for conversational AI services that enhances user experience by co-designing the server and the client. We first present a definition of Quality-of-Experience (QoE) for conversational AI (§2.3.1), which acts as Andes's optimization objective. Thereafter, we provide an overview of Andes's architecture and request lifecycle (§2.3.2).



(a) Perfect user experience.

(b) Delay in first token.

(c) Delay in all subsequent tokens.

(d) Long pause in the middle.

Figure 2.4: User experience examples. Users expect tokens to be delivered along the fixed Ideal Consumption Timeline, with its slope being their reading/listening speed. (a) Without any Token Delivery Delays, users' Actual Consumption Timeline aligns with their Ideal Consumption Timeline. (b, c, d) However, when tokens are delayed, users perceive the delay and user experience degrades.

2.3.1 Quality-of-Experience in Conversational AI

Existing LLM systems are inefficient in resource usage for conversational AI services (§4.2.3). This is because widely used system optimization metrics fail to fully capture user experience by only considering a limited portion of the user's *token consumption timeline*. Server-centric metrics like token generation throughput ignore the token delivery timeline of individual requests. TTFT only captures the user's consumption of the first token, and average/P90/P99 time-per-output-token (TPOT) can miss outlier TPOT inflations that lead to user-perceived pauses during interaction.

As such, we need a unified QoE metric that fully captures user experience in conversational AI services. Figure 2.4 shows four foundational cases that guide the design of QoE. The Ideal Consumption Timeline represents the user's ideal experience, with low TTFT and every subsequent token being delivered precisely at the user's token consumption speed.

1. Figure 2.4a (*Perfect experience*): The server delivered every token no later than the user's expectation, allowing users to consume them following the ideal timeline. Delivering tokens earlier than the ideal timeline does not improve user experience, as

users cannot consume them faster.

2. Figure 2.4b (*Long initial delay*): The request stayed in the server's queue for a long time due to head-of-line blocking. As a result, the user experienced a long initial wait time, followed by subsequent tokens delivered at the user's reading speed.
3. Figure 2.4c (*Slow token delivery speed*): When the server processes a larger batch of requests, its token generation latency increases, potentially failing to meet the user's ideal token consumption speed. In this case, the first token was delivered on time, but the user experienced delays in every subsequent token.
4. Figure 2.4d (*Pause during token delivery*): The server preempted the request to avoid OOM errors, pausing token delivery in the middle. When the Actual Delivery Timeline crosses the Ideal Consumption Timeline, the user runs out of tokens to read and experiences a pause. This case is particularly insidious, as TTFT or average TPOT does not degrade; both the first and last tokens were delivered on time, hiding the long pause in the middle.

As can be seen above, we can gauge user experience (and the degradation thereof) based on how much the user's Actual Consumption Timeline (T^{Actual}) deviated from the Ideal Consumption Timeline (T^{Ideal}). Thus, we define:

$$S_{\text{delay}} = \sum_{i=1}^{n} \left(T_i^{\text{Actual}} - T_i^{\text{Ideal}} \right), \tag{2.1}$$

where T_i^{Actual} and T_i^{Ideal} respectively denote token i's actual and ideal consumption timestamp, and n is the total number of tokens consumed by the user. S_{delay} measures how much the Actual Consumption Timeline deviated from the Ideal Consumption Timeline (gray shaded area in Figure 2.4), and properly reflects the delay of earlier tokens creating cascading delays in later tokens. However, as more tokens are generated for the request, user experience degradation from earlier delays ought to be diluted. Thus, we introduce a normalizer:

$$S_{\text{whole}} = \sum_{i=1}^{n} \left(T_n^{\text{Actual}} - T_i^{\text{Ideal}} \right). \tag{2.2}$$

S_{whole} covers S_{delay} and the area below the Actual Consumption Timeline, always being larger than S_{delay}. With this, our QoE definition is:

$$\text{QoE} = 1 - \frac{S_{\text{delay}}}{S_{\text{whole}}}. \tag{2.3}$$

With no token delay at all, S_{delay} will be zero, leading to a perfect QoE of 1. On the other

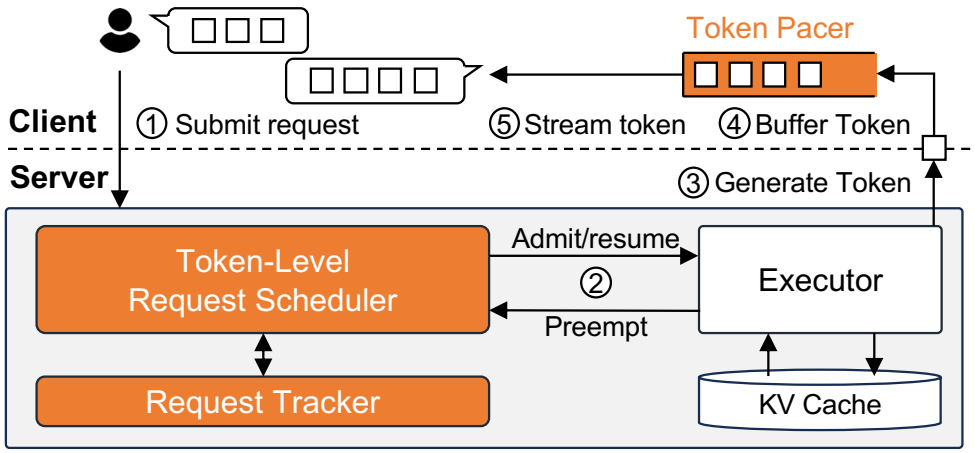


Figure 2.5: High-level architecture of Andes. Components designed by Andes are colored in orange.

hand, when tokens are delayed, S_{delay} will take up a larger proportion inside S_{whole}, reducing the value of QoE. Finally, when no tokens arrive, S_{delay} will equal S_{whole}, leading to the worst possible QoE of 0.

QoE can be computed on requests in any state, be it queued, running, or finished. The optimization objective of Andes is to maximize QoE across all requests.

2.3.2 Andes Architecture

System Components. Andes co-designs the LLM inference server and the client. Figure 2.5 shows the high-level architecture of Andes. On the server side, the **Request Tracker** maintains the control state of each request, including its QoE parameters (TTFT target and user token consumption speed), prompt and partial response, timestamps of each generated token, and resource usage. Based on this information, the **Token-Level Request Scheduler** (§2.4) makes runtime decisions on which requests to admit/resume or preempt. Such decisions are carried out by the data plane (**Executor** and **KV Cache**), which is also responsible for running LLM inference and generating tokens. The server implements push-based streaming, and the client-side **Token Pacer** (§2.5) smooths out the pace of incoming tokens by temporarily buffering tokens and delivering them to the user at their consumption speed. The server is fully aware of the Token Pacer, and will generate and push just enough tokens so that the Token Pacer does not run out of tokens to deliver.

Request Lifecycle. First, a request is **1** submitted through the application-integrated client, and the client also informs the Token Pacer of the user's QoE parameters.[2] The request is then enqueued into the server, and the Request Tracker initializes and continuously tracks the request's status. During its lifetime, the request can either be waiting (enqueued or preempted) or running. **2** When the Token-Level Request Scheduler decides to preempt a running request, it will transition into the waiting state after saving its intermediate state. On the other hand, a waiting request begins running when the scheduler decides to either newly admit or resume it, restoring its intermediate state if any. When a request is running, **3** the Executor generates tokens and **4** pushes them immediately to its corresponding user's Token Pacer. Regardless of the request's state in the server, the Token Pacer **5** drains buffered tokens and delivers tokens to users following their Ideal Consumption Timeline.

2.4 QoE-Aware Token-Level Scheduler

We can now dive into how Andes optimizes the QoE of conversational AI services by performing token-level preemptive request scheduling. We start by formulating the scheduling problem assuming that there is no preemption overhead (§2.4.1), as the overhead itself depends on the scheduling decisions dictated by the scheduling policy. We then propose an efficient scheduling algorithm for the problem (§2.4.2) and refine the solution to incorporate preemption overhead (§2.4.3).

2.4.1 Problem Formulation

We first discuss the objective and constraints of Andes, and then put them together to formulate the scheduling problem.

Scheduling Setup and Objective. Andes's scheduler operates in an online setting where user requests arrive over time with diverse input lengths and QoE parameters. Its objective is to maximize the *average QoE* across all requests.[3]

Like any other online serving system, it is very difficult, if not impossible, to perfectly plan execution into the future because the arrival time, input and output lengths, and QoE parameters of each request are not known in advance. Instead, among ongoing (waiting and

[2]QoE parameters may be derived from service-level objectives (SLOs) defining target TTFT and TDS for the application, personalized user settings based on individual reading or listening speeds (e.g., from Tables 2.1 and 2.2), or pricing tiers (e.g., more expensive tokens guarantee higher delivery speed).

[3]Alternative objectives can also be used. See Appendix A.1.

running) requests, Andes decides which requests to serve at the beginning of each scheduling quantum. Based on this decision, requests are admitted/resumed and preempted as needed.

Andes decides whether or not to serve a request based on the *QoE gain* it is expected to bring when it is served compared to when it is not served, calculated as:

$$Q_{\text{serve},i} - Q_{\text{wait},i} \tag{2.4}$$

where $Q_{\text{serve},i}$ and $Q_{\text{wait},i}$ are request i's QoE when it is served and not served, respectively. As we are uncertain about whether a request will be continued to be served or preempted in the future, we estimate the QoE gain of a request in the upcoming time frame of length Δt. We will shortly go into more depth into how this estimation is done, and evaluate the impact of Δt in Section 4.5.5.

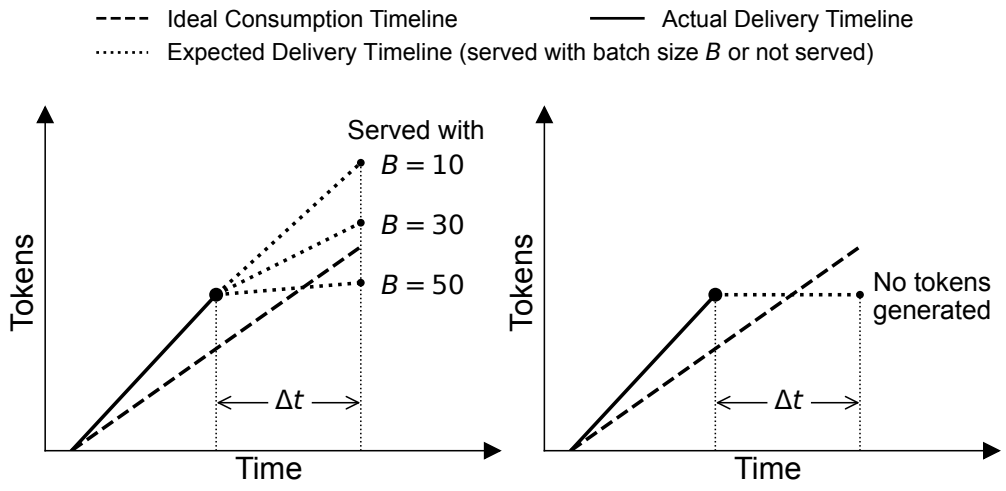
Resource Constraints. LLM serving systems are bottlenecked primarily by two GPU resources: memory and compute. These impose constraints on which requests can be concurrently served by the system.

First, each token in a request's context (input and output tokens) consumes one entry in the LLM serving system's KV cache [27]. As GPU memory is limited, there is a limit on the number of KV cache entries the GPU can hold with model weights and intermediate tensors. The total KV cache size of requests that are served must not cross this upper limit.

In addition to memory constraints, Andes must also consider compute constraints, which affects the computation latency of token generation by the executor; a larger batch size generally increases computation latency. [4] Thus, while a large batch size B serves more requests concurrently, it will also increase the latency to generate one token from the perspective of each request. This may lead to individual requests generating tokens too slowly and degrading QoE. On the other hand, a smaller batch size would lead to faster token generation, but the server is serving fewer requests, potentially degrading the QoE of those that are left waiting.

As such, determining the right batch size B is critical in maintaining the right token generation speed for requests, which in turn affects the QoE of serving each request. Therefore, in estimating the QoE gain of each request in the scheduling objective, Andes takes batch size B into account. Figure 2.6 provides an example of this. When batch size is small ($B = 10$), tokens are generated quickly and the request maintains perfect QoE. However, as B increases ($B = 30$ and $B = 50$), token generation slows down due to higher computation

[4] More precisely, token generation latency depends on the batch size and total number of tokens, but batch size and total number of tokens are nearly correlated (Appendix A.2), allowing us to eliminate the number of tokens.



(a) $Q_{\text{serve},i}(B)$ (b) $Q_{\text{wait},i}$

Figure 2.6: Visualization of $Q_{\text{serve},i}(B)$ and $Q_{\text{wait},i}$. The former depends on batch size B whereas the latter is a constant. With batch size 50, request i no longer has perfect QoE.

load, and perfect QoE will be broken in the case of $B = 50$. On the other hand, when the request is not served and waiting, it does not generate any tokens and therefore batch size does not affect its QoE.

Problem Formulation. Putting these together, we have the following scheduling problem:

$$\begin{aligned} \max_{x} \quad & \sum_{i=1}^{N} (Q_{\text{serve},i}(B) - Q_{\text{wait},i}) \cdot x_i \\ \text{s.t.} \quad & x_i \in \{0,1\},\ i \in 1,\ldots,N \\ & \sum_{i=1}^{N} x_i = B \\ & \sum_{i=1}^{N} l_i x_i \leq M \end{aligned} \tag{2.5}$$

where N is the total number of ongoing requests, and for request i, l_i is its context length, and x_i equals 1 if the request will be served and 0 if not. The objective has been updated so that QoE gain properly depends on batch size B. The second constraint enforces that batch size should be exactly B, and the third ensures that the total context length in the batch does not exceed the GPU's memory. Notice that batch size B is treated as a given; the optimization problem in Equation 2.5 has to be solved for $\forall B \in [1, N]$ and the x that leads to the largest optimum across $\forall B$ is the optimal solution.

Given the problem formulation, we observe that it resembles that of the classic knapsack problem [76]. The goal is to select items (requests) to put in a knapsack (inference server) so that total item value (QoE gain) is maximized and total weight (context length) does not exceed the knapsack's capacity (memory capacity). Yet, the fact that the item value of each item (QoE gain) depends on how many items end up in the knapsack (batch size) makes it a harder variant.

Problem Hardness. The problem in Equation 2.5 is weakly NP-Hard [76]. 3D dynamic programming (DP) can solve the problem optimally in pseudo-polynomial time $O(MN^2)$ (See Appendix A.3), which is likely too slow as the number of requests N and the maximum number of tokens that can fit in memory M are easily in the order of hundreds and thousands, respectively. Furthermore, Equation 2.5 has to be solved for $\forall B \in [1, N]$, which is clearly intractable.

2.4.2 Priority-Based QoE-Aware Scheduling

It is computationally prohibitive to solve the problem in Equation 2.5 as is, particularly when the scheduler is normally invoked before every token generation iteration. Therefore, we propose an approximate but efficient solution.

Walkthrough. We propose a *priority*-based greedy heuristic that approximates the solution to the knapsack problem. The algorithm assigns a priority to each request based on its potential QoE gain relative to its resource usage. Intuitively, requests that require less GPU resource but still bring large QoE gain should be favored, allowing the system maximize average QoE under resource constraints.

We illustrate the scheduling process with a toy example to demonstrate the system's behavior as in Figure 2.7, along with the token delivery process of one request R_1 (bottom row). At $t = 0$, three requests R_1, R_2, and R_3 with different GPU memory demands – represented by the number of tokens in the prompt – arrive and start running. Even at $t = 0$, request R_1 is assigned a lower priority than R_2 and R_3 because it has a longer context length and thus consumes more GPU resources. All three requests generate one token per iteration, as also reflected by request R_1's token timeline. At $t = 3$, two new requests R_4 and R_5 arrive. However, the executor is fully occupied, so the scheduler needs to decide which requests to preempt and which requests to admit. At the moment, R_1 has the lowest priority because (1) it has accumulated enough tokens for the user to consume, leading to a lower potential QoE gain, and (2) it consumes the most amount of GPU memory. Therefore, the scheduler preempts R_1 temporarily and admits R_4 and R_5. Even after R_1 gets preempted,

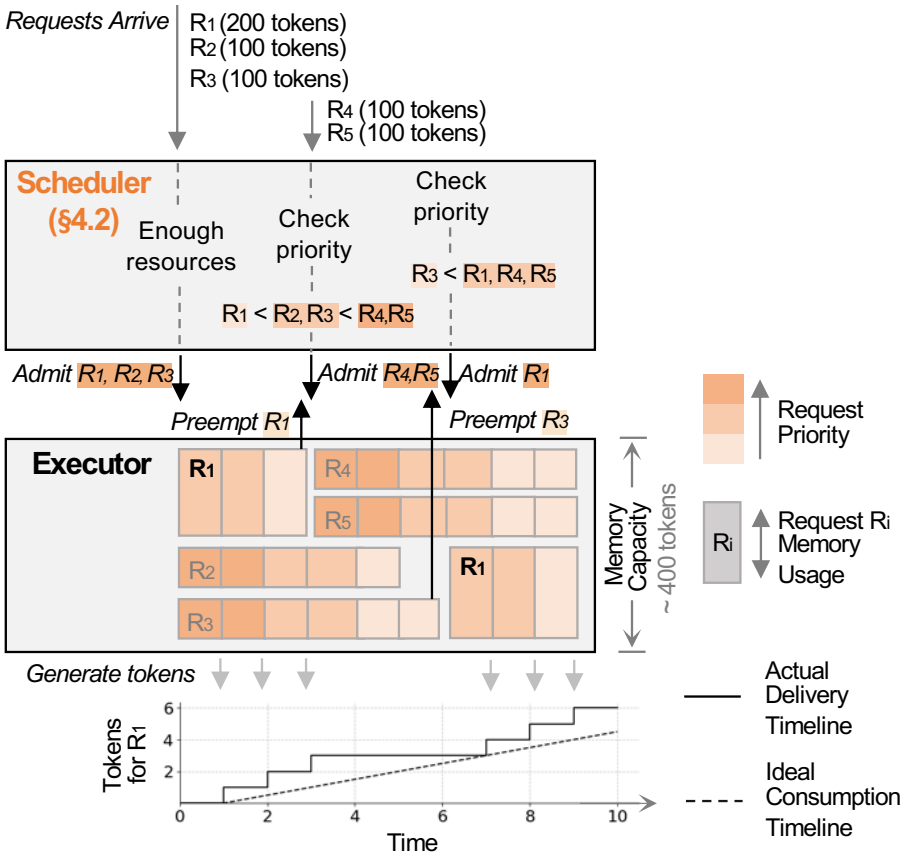


Figure 2.7: Andes's token-level preemptive request scheduler in action. The priority of each request changes over time, as indicated by colors. Scheduling decisions are made based on this dynamic priority and the amount of available GPU resources.

its Actual Delivery Timeline curve is still above the Ideal Consumption Timeline, meaning the user will not experience any pause. At $t = 6$, the Actual Delivery Timeline of R_1 is about to drop below the Ideal Consumption Timeline, so the scheduler preempts R_3 with the lowest priority and resumes R_1. This allows R_1 to resume token generation, ensuring the user continues to receive tokens without experiencing any pause.

Priority-Based Greedy Packing. Based on this intuition, we define request i's priority as:

$$\frac{Q_{\text{serve},i}(B) - Q_{\text{wait},i}}{l_i}. \tag{2.6}$$

This priority function meets our design goals:

- Requests with higher QoE gain and lower GPU resource usage will be prioritized.
- Requests with long context length (l_i) are preempted first, freeing ample GPU memory

Algorithm 1 Priority-based greedy packing for Equation 2.5

Inputs: Number of requests N, memory capacity M, request context length array $l[N]$, request QoE gain array $q[N]$, and target batch size B.
Output: Scheduling decision array $x[N]$.

for all $i \in [1, N]$ **do**
    $p[i] \leftarrow q[i]/l[i]$ ▷ Set priority of request i
$M_{\text{current}} \leftarrow N_{\text{current}} \leftarrow 0$
Initialize solution array $x[N]$ with all zeros
for all $i \in [1, N]$ in descending order of $p[i]$ **do**
    if $M_{\text{current}} + l[i] \leq M$ **and** $N_{\text{current}} + 1 \leq B$ **then**
        $x[i] \leftarrow 1$ ▷ Serve request i
        $M_{\text{current}} \leftarrow M_{\text{current}} + l[i]$
        $N_{\text{current}} \leftarrow N_{\text{current}} + 1$
    else
        break
return x

to potentially bring in *more than one* waiting requests.[5] This reduces the number of preemptions required to alleviate head-of-line blocking.

- As a request receives service, its context length (l_i) will increase, automatically deprioritizing itself. On the other hand, the QoE gain of requests will increase the longer they wait, automatically prioritizing itself. Both aspects contribute to preventing starvation.

The scheduling algorithm executed at every scheduling quantum for each batch size B is given in Algorithm 1. In essence, the algorithm sorts requests in descending order of priority and decides to serve the request until the memory capacity or the batch size is reached. The greedy packing algorithm provides an efficient time complexity of $O(N \log N)$. On top of this, we apply two optimizations that reduce the number of times the scheduling algorithm must be invoked.

Selective Triggering. As QoE is only affected when the system is under load surge, it is not necessary to solve the knapsack problem otherwise. Therefore, we can selectively trigger Algorithm 1 only when we detect that the system is under resource pressure (memory capacity or compute). For the former, Andes monitors the GPU KV cache occupancy and triggers the solver only when occupancy exceeds a high watermark (e.g., 90%). For the latter, Andes monitors token generation latency and triggers the solver when it begins to exceed what is required to satisfy the fastest token delivery speed requirement among in-flight

[5]The overhead of preemption depends on how much memory was freed, not the number of requests. Therefore, for the same amount of memory freed from preemption, it's better to free a smaller number of requests.

requests.

Batch Size Pruning. In order to further reduce invocations, we reduce the search space of batch size B from $[1, N]$ to $[B_{\min}, B_{\max}]$. First, there is no point in exploring very large batch sizes that cannot be realized. Thus, $B_{\max}$ is determined by adding to the batch requests with the shortest context lengths until the total number of tokens in the batch reaches M, at which point batch size is the largest possible. On the other hand, very small batch sizes that can generate tokens faster than the user token consumption speed of *every* request are also suboptimal. This is because generating tokens that fast does not increase the QoE of served requests, but on the other hand will serve fewer requests, potentially degrading the QoE of requests that are left waiting. Thus, $B_{\min}$ is set as the largest batch size that generates tokens faster than the most stringent user consumption speed across all requests.

We show in Section 4.5.5 that this solution can achieve average QoE close to the 3D DP algorithm.

2.4.3 Incorporating Preemption Overhead

We introduce an overhead-aware refiner that enhances the priority-based scheduler's decisions by accounting for preemption overhead's impact on overall QoE. This is important because preemptions are not free; they introduce either extra computation or memory movement overhead at the time of preemption and resumption [142, 83, 9, 183]. This overhead can last hundreds of milliseconds to even seconds, and interrupts the whole token generation process. Frequent preemptions can accumulate overhead, delaying token generation and degrading QoE for all requests. Conversely, restricting preemptions too much may prevent the system from serving urgent requests, missing chances to improve overall QoE.

Walkthrough. The overall intuition is that if ongoing requests have enough *slack* before they begin degrading QoE, Andes can afford to preempt more requests to maximize overall QoE. In Figure 2.8, the priority-based scheduler (§2.4.2) decides to admit $\{R_8, R_9, R_{10}\}$ and preempt $\{R_1, R_2, R_3\}$, and passes the decision to the refiner. The refiner estimates the overhead for each potential admission and preemption and assesses its impact on the QoE of all ongoing requests. Starting with the highest priority request to admit, R_{10}, the refiner identifies the minimal set of lowest priority requests to preempt, such as R_1, to free sufficient resources for R_{10}. With this, Andes estimates the total latency overhead of admitting R_{10} and preempting R_1. In this case, Andes finds that the QoE gain of admitting R_{10} is higher than the total QoE loss of ongoing requests caused by the admission and preemption overhead. Thus, R_{10} is added to the final admission list and R_1 to the final preemption list. This process

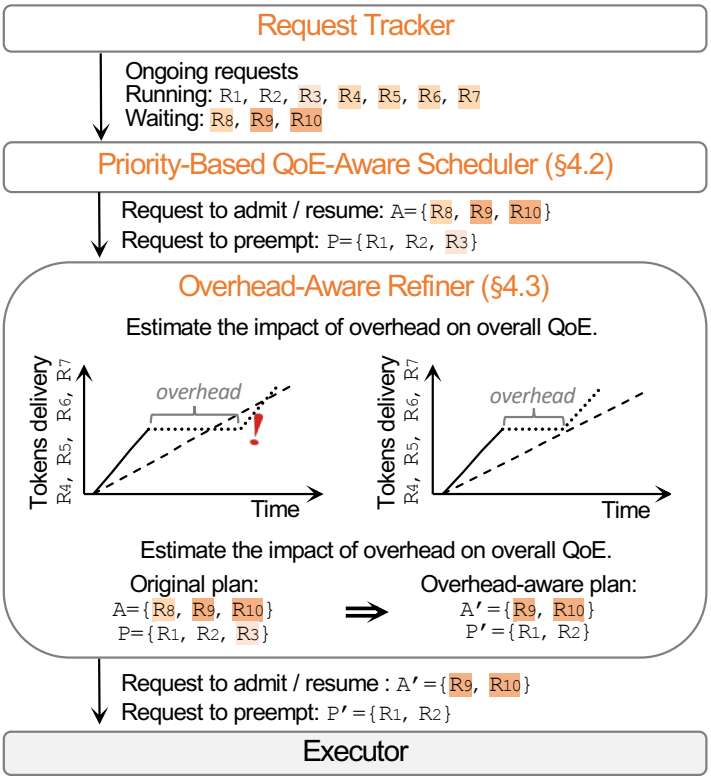


Figure 2.8: Andes refines the decision made by the priority-based scheduler so that it does not degrade more QoE than it improves. This example does not continue from Figure 2.7.

stops when Andes finds that the system can no longer tolerate additional overhead without degrading QoE. Ultimately, the refiner admits only $\{R_9, R_{10}\}$ and preempts $\{R_1, R_2\}$.

Overhead Estimation. As seen earlier, Andes needs to estimate the overhead of request admission/resumption and preemption. Andes supports any request preemption mechanism, with recomputation and swapping being the most common in existing systems. The former drops the KV cache of the request upon preemption, incurring no overhead, and then recomputes them on resumption, incurring overhead equivalent to one prefill. The latter moves the request's KV cache between GPU and CPU memory, which incurs copying overhead. As overhead is predictable [183], so Andes offline-profiles the latency overhead of available preemption mechanisms under various context lengths and selects the faster one, and uses that data to estimate overhead for requests.

Balancing QoE Gain and Overhead. Preemption overhead delays token generation for all ongoing requests, but the priority-based scheduler's decision does not account for

this. Thus, excessive preemption delays may cause a net QoE degradation if the decision is implemented naively. Thus, for each potential admission or resumption, the refiner calculates the QoE loss resulting from the associated preemption overhead and only proceeds if the QoE gain exceeds this loss. The QoE gain from admitting or resuming a request is already computed by the priority-aware scheduler (§2.4.1). On the other hand, the QoE loss of an ongoing request i can be computed identically with how $Q_{\text{wait},i}$ was estimated, where Δt is set to be the preemption overhead. The total QoE loss of all ongoing requests are then added up. If the net QoE change is positive, the refiner retains that set of admission/resumption and preemption decisions, and moves on to the next set of requests in priority-order. Otherwise, it means that the system can no longer tolerate admitting/resuming requests without degrading overall QoE, so the refiner cancels the rest of the admission/resumption and preemption decisions from the priority-based scheduler.

We show in Section 4.5.5 that the overhead-aware refiner is critical in maintaining high QoE across requests.

2.5 Implementation

Andes mainly consists of the server-side QoE-aware token-level preemptive scheduler and the client-side token pacer.

Request Scheduler. Andes's scheduling algorithm can work with any LLM serving system that supports continuous batching and at least one preemption mechanism (swapping or recomputation). As a reference, we implemented Andes's scheduler as an alternative scheduling policy in vLLM [83]. The scheduler only manages requests coming into the vLLM instance it is integrated with, assuming that cluster-level load balancing and fault tolerance are done separately.

Token Pacer. The server implements push-based streaming because (1) tokens generated for a single user are only streamed to that user exactly once, and (2) the server is typically resource constrained – particularly so if CPU memory is being used for storing the KV cache of preempted requests – and therefore prefers to deallocate request state as quickly as possible. As such, the client-side *token pacer* receives tokens as soon as they are generated, even if they were generated at a pace that exceeds the user's consumption speed. The token pacer temporarily buffers excess tokens and yields them smoothly along the user's Ideal Consumption Timeline.

We give a concrete example of how the token pacer works with the server-side request scheduler in the Appendix A.4.

2.6 Evaluation

We evaluate Andes on a variety of models, hardware configurations, and request datasets with varying input and output length characteristics (Tables 2.3 and 2.4) and find the following:

- Andes shows end-to-end improvements on a 10-hour real-world trace, improving average QoE from 0.9 to 0.99 and reducing average TTFT from 11.2s to 2.4s (§2.6.2).
- On synthetic bursty traces, Andes enhances average QoE by up to 4.7× over vLLM and boosts serving capacity by up to 1.74× while maintaining high QoE (§2.6.3).
- Andes provides superior user experience compared to vLLM for audio-based conversational AI services (§2.6.4).
- Andes's keeps preemption overhead under control and outperforms baselines under different solvers, QoE gain estimation time horizons, and request distributions (§4.5.5).

2.6.1 Experiment Setup

Model	**Architecture**	**Memory**	**Hardware**
Phi-3-mini 3.8B [8]	Dense, MHA	7 GB	A100×4
Command R 32B [35]	Dense, GQA	61 GB	A100×8
Phi-3.5-MoE 16×3.8B [8]	MoE, GQA	80 GB	A100×8
Llama 3.1 70B [40]	Dense, GQA	132 GB	A100×8

Table 2.3: Models and hardware configurations.

Dataset	**Input Length**		**Output Length**	
	Mean	Std.	Mean	Std.
Multi-Round ShareGPT [150]	3171	7943	385	300
ArXiv Summarization [33]	17855	11401	605	153
Coding Challenges [60]	675	1552	5423	21293

Table 2.4: Request dataset statistics.

Models, Hardware, and Requests. We evaluate Andes on models with diverse architectures (Dense vs. Mixture-of-Experts, Multi-Head Attention vs. Grouped-Query Attention) and request input/output datasets with varying sequence lengths. We leverage both

real-world request traces from BurstGPT [157] and synthetic bursty traces designed to conduct controlled experiments as detailed in Section 2.6.2 and Section 2.6.3 respectively. We deployed models with tensor parallelism on NVIDIA A100 SXM4 40GB GPUs in one AWS p4d.24xlarge instance. See Tables 2.3 and 2.4 for full details.

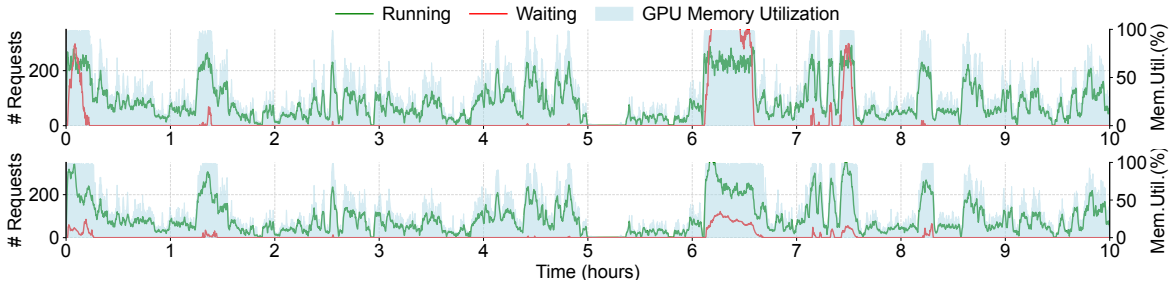
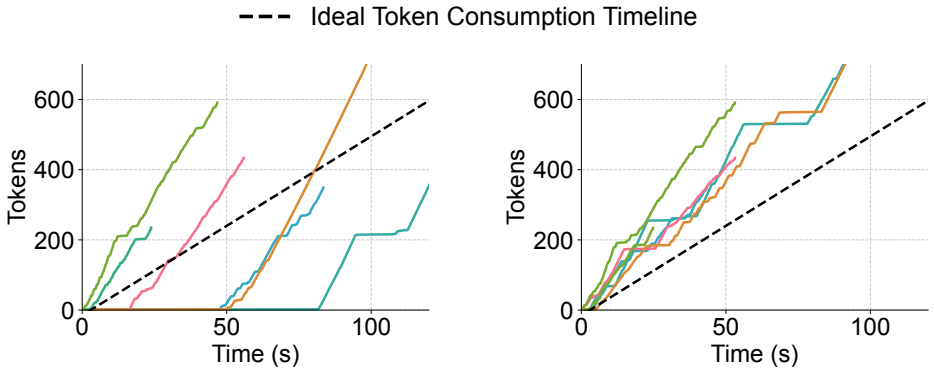


Figure 2.9: System status while replaying BurstGPT. Compared to vLLM (first row), Andes (second row) significantly reduces the waiting queue length and improves overall QoE.



(a) vLLM (b) Andes

Figure 2.10: The Token Generation Timeline of select requests, with request submission time shifted to zero. Requests served by Andes, as opposed to vLLM, are well above the Ideal Consumption Timeline.

QoE Parameters. In real-world scenarios, QoE parameters (target TTFT and token consumption speed) of users will vary widely. For each request, we set the target TTFT to be max(input length // 5000, 1) seconds, where 5000 tokens/second is the prefill throughput of our hardware setup. This reflects the expectation of an average non-technical user who expect a response time that is proportional to the input length. On the other hand, we use the human reading speed distribution in Table 2.1 to assign user token reading speeds to requests. In practice, this should be tailored to the application's requirements. For instance,

API price tiering can also be implemented by providing a higher token delivery speed for higher-priced tokens, allowing users to select the tier suitable for their downstream application.

Baselines. We compare with vLLM [83] (v0.6.1) and Sarathi-Serve [11] (i.e., chunked prefill), all of which adopts the FCFS scheduling policy. Additionally, we compare with the Least QoE Slack First (LQSF) scheduling algorithm, which prioritizes requests most at risk of QoE degradation based on the QoE gain we defined. vLLM server configurations across all approaches are kept identical (Appendix A.5 lists all).

Metrics. We report the following key metrics:

- **Average QoE**: The QoE value of requests when they finish, averaged across all requests.
- **Serving capacity**: The highest burst request rate the system can serve while keeping the average QoE across all requests above a specific threshold.

2.6.2 End-to-End Improvements on Real-World Traces

We replay a 10-hour segment of the BurstGPT [157] request arrival trace, captured from real-world LLM services, using the Multi-Round ShareGPT dataset and Phi-3.5-MoE 16×3.8B.

QoE Improvements. Over the whole 10-hour trace, Andes improves average QoE from 0.9 to 0.99 and reduces average TTFT from 11.2s to 2.4s. Improved QoE without additional resources allows Andes to handle more concurrent requests while maintaining high QoE or reduce GPU usage for the same QoE, yielding cost savings. We provide a more detailed breakdown analysis of Andes for the burst that occurs during the one-hour window (between hour 7 and 8) in Appendix A.6.

Queue Length Reduction and Serving Capacity Gain. To examine the system's real-time behavior, we visualize the status of Andes and vLLM during this 10-hour period in Figure 2.9. During high-load periods (0-1h and 7-8h), Andes can reduce the peak waiting queue length by a significant 85% through token-level preemptive request scheduling. Furthermore, unlike vLLM, a large portion of the waiting requests in Andes are those that have been preempted by the scheduler after generating sufficient tokens for their users, explaining the high average QoE achieved by Andes. Practically, this means Andes can satisfy these waiting requests without overprovisioning the GPUs to host more serving instances, effectively reducing the cost or increasing the serving capacity compared to vLLM.

Token Generation Timeline Improvements. Figure 2.10 presents a comparison of token generation timelines for a selected set of requests from the trace, under both vLLM and Andes. With QoE-aware, token-level preemptive scheduling, Andes consistently delivers each token ahead of the ideal consumption point, ensuring smooth user experience. In contrast, vLLM suffers from head-of-line blocking, leading to noticeable degradation in QoE for many requests. With the help of token pacer, each request's token delivery timeline aligns with the user ideal consumption timeline.

2.6.3 Improvements Under Controlled Burstiness

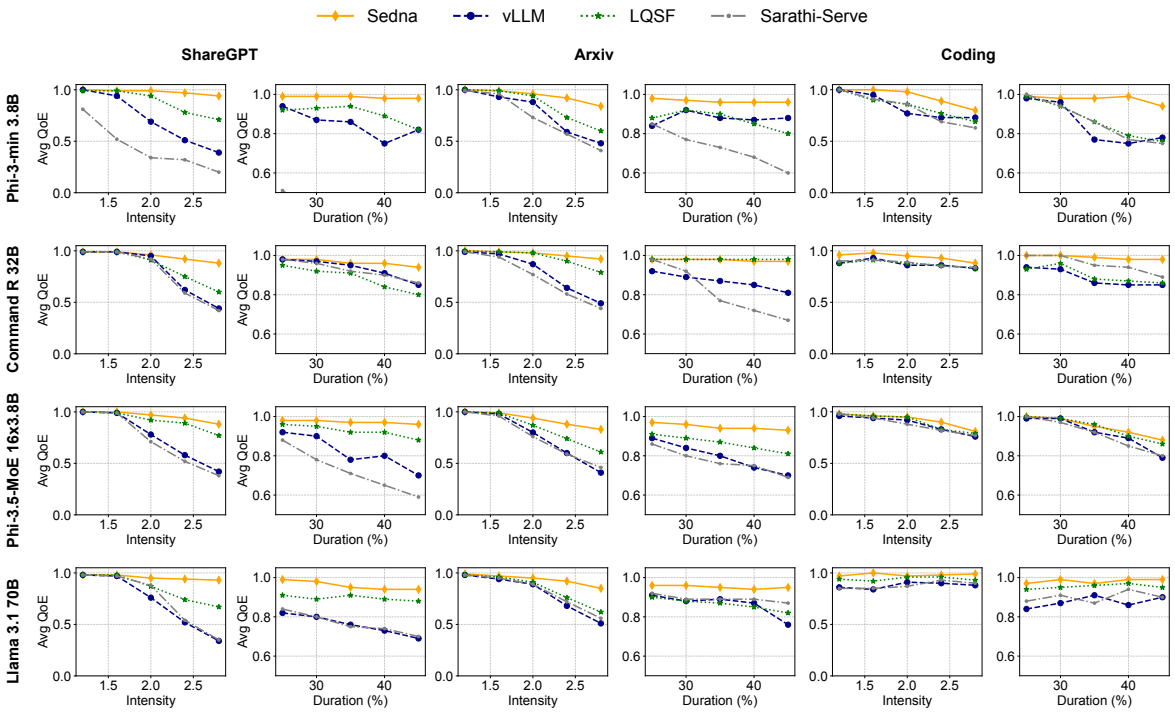


Figure 2.11: Andes improves average QoE under different request burstiness.

We have seen that Andes can effectively handle requests from real-world traces. To quantify its QoE improvement and serving capacity gains over baselines, we evaluate Andes on synthetic traces containing request bursts that resemble those in BurstGPT. BurstGPT exhibits an average of three burst periods per hour, each lasting ∼7 minutes and showing 2× higher request rates than the average. Following this, we introduce the *Cyclic Burst Load Pattern*, mimicking BurstGPT's burst frequency and intensity. Bursts are defined by two parameters: *intensity* (ratio of burst request rate to average) and *duration* (fraction of time spent in burst). Request arrivals in both burst and non-burst phases follow a Poisson process, alternating cyclically—details in AppendixA.7.

Target QoE	Phi-3-mini 3.8B	Command R 32B	Phi-3.5-MoE 16×3.8B	Llama 3.1 70B
0.95	1.74×	1.05×	1.35×	1.22×
0.96	1.73×	1.05×	1.29×	1.15×
0.97	1.71×	1.04×	1.22×	1.08×
0.98	1.65×	1.02×	1.15×	1.33×
0.99	1.58×	1.00×	1.08×	1.17×

Table 2.5: Andes improves the serving capacity on Multi-Round ShareGPT dataset for different target QoE compared to vLLM.

Following statistics from BurstGPT, we set the default burst intensity to 2 and the burst duration to 35%. In our experiments, we adjust either burst intensity or burst duration while leaving the other at its default value. The average request rate of the whole cycle is set to be the serving system's throughput without any burstiness, preventing the system's queue from growing indefinitely over time. This also allows us to evaluate systems on one cycle of the trace, as most requests will have been handled by the end of one cycle.

Improvements Under Varying Burst Intensity. We report the average QoE under different burst intensities in Figure 2.11. As burst intensity increases, Andes continues to maintain high average QoE, achieving up to 4.7× QoE improvement. Thus, Andes sustains higher burst intensity than vLLM while achieving the same QoE. We report the serving capacity gain to achieve different target average QoE in Table 2.5, Andes can improve the serving capacity up to 1.74× more burst intensity on Multi-Round ShareGPT dataset. More results on serving capacity gain can be found in Appendix A.8.

Improvements Under Varying Burst Duration. We report average QoE under varying burst durations in Figure 2.11. Andes consistently achieves up to 3.5× higher QoE than vLLM across all models and request datasets. Conversely, First-Come-First-Serve (FCFS) baselines suffer from head-of-line blocking during bursts, reducing QoE. Sarathi-Serve's chunked prefill increases Time-to-First-Token by disrupting the prefill stage. Least QoE Slack First (LQSF) scheduling slightly improves QoE by prioritizing at-risk requests but underperforms Andes by neglecting resource demands, causing resource-intensive requests to starve others.

How Much Potential Gain was Realized? As derived in Section 2.2.3, Andes can serve 2.3× more requests under ideal circumstances for the case of Phi-3.5-MoE 16×3.8B with the Multi-Round ShareGPT request dataset. In practice, as shown in Figure 2.11, Andes achieves

a 1.4× improvement, realizing approximately 60% of the ideal gain under this setup. This gap is due to scheduling overhead, request prefill, and request preempt/resume overhead, which limit the full utilization of the system's available slack.

2.6.4 Audio-Based Conversational AI

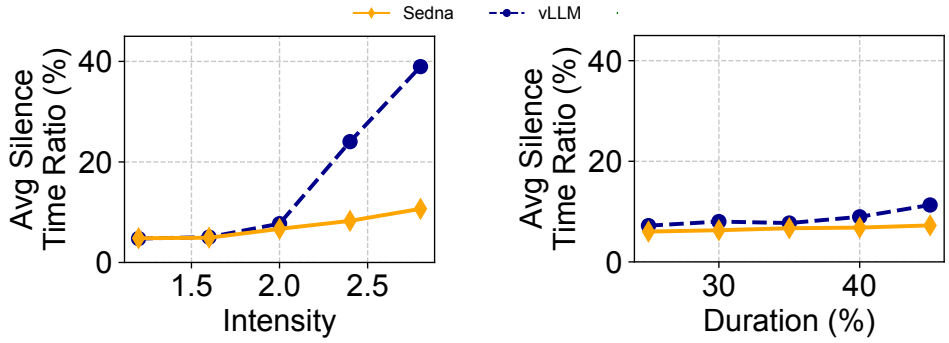


Figure 2.12: In an audio-based conversational AI service, Andes provides superior silence time ratio compared to vLLM.

We evaluate Andes in the context of an audio-based conversational AI service with two components: An LLM serving system that generates responses (vLLM vs. Andes), and a text-to-speech (TTS) engine (GPT-4o mini [116]) that converts text chunks into speech and plays them back to the user. We use the speaking speed of English (Table 2.1) as the target token delivery speed for all requests.

Figure 2.12 reports *silence time ratio*, the percentage of time during end-to-end user interaction when the user heard nothing, averaged across Multi-Round ShareGPT requests served by Llama 3.1 70B. This metric, inspired by Buffering Ratio [38] in video streaming, includes both join time and intermediate pauses. At low burst intensities, vLLM and Andes yield similar silence time ratios. As burst intensity rises, vLLM's silence time ratio reaches 40%, while Andes lowers it by up to 3.3× with consistent token delivery.

2.6.5 Sensitivity Analysis and Ablation Studies

We evaluate Andes's robustness across various settings and configurations. We report results using the Llama 3.1 70B model on the Multi-Round ShareGPT with a default cyclic burst load pattern, observing consistent trends across setups.

The Overhead-Aware Refiner is Indispensable. We assess the overhead-aware refiner's effectiveness (§2.4.3) via an ablation study comparing Andes with and without it.

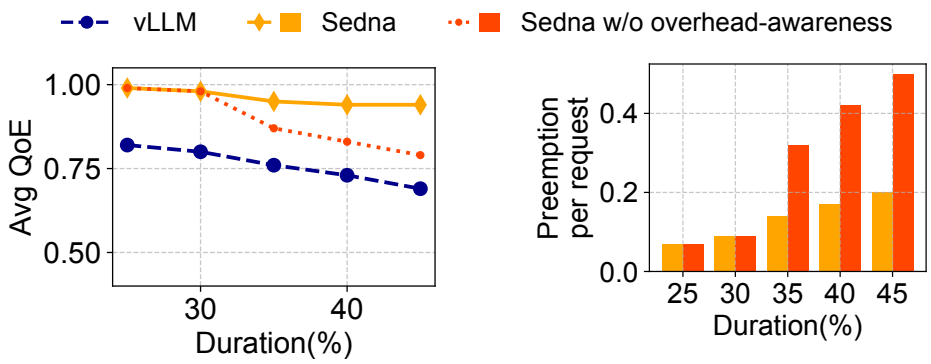


Figure 2.13: Overhead-aware refiner is critical in optimizing QoE.

Figure 2.13 illustrates average QoE and preemptions per request across burst durations. Without the overhead-aware refiner, Andes experiences a high number of preemptions as burst duration increases, leading to delays in token generation for ongoing requests and significantly degrading QoE. In contrast, Andes with the overhead-aware refiner balances QoE improvement and scheduling costs, consistently achieving higher QoE across various burstiness conditions.

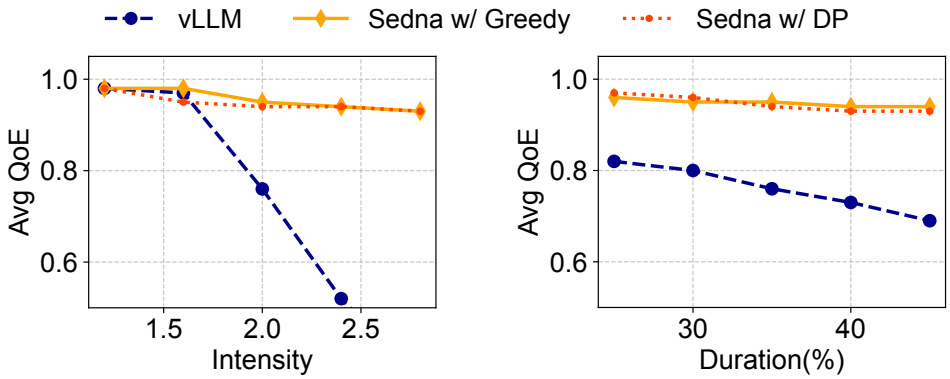


Figure 2.14: Andes's greedy solver yields competitive performance compared to the optimal DP solver.

Different Knapsack Solvers. We compare Andes's greedy knapsack solver with the optimal-but-slow 3D Dynamic Programming solver. Figure 2.14 shows that the greedy solver achieves slightly better average QoE under longer burst durations or higher burst intensities. Andes outperforms the exact 3D DP solver because its greedy solver is more suitable for real-time decision-making, being $\sim 20\times$ faster while still delivering high-quality approximate solutions.

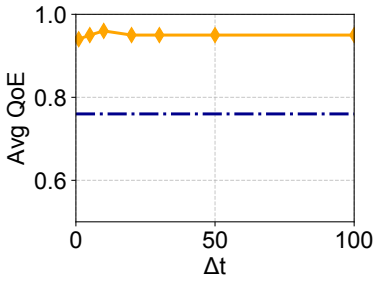


Figure 2.15: Varying Δt.

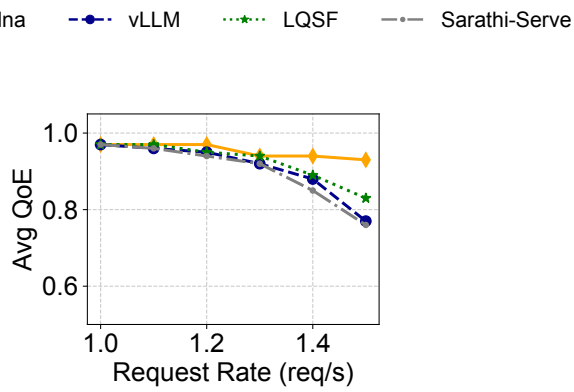


Figure 2.16: Poisson arrival.

QoE Gain Estimation Time Horizon. We evaluate how varying Δt, the time horizon over which a request's QoE gain is estimated, influences average QoE. Figure 2.15 shows that the average QoE remains roughly consistent for various Δt values and significantly outperforms the baselines. Optimal Δt varies by model and request distribution, requiring pre-deployment tuning.

Poisson Arrival Pattern. For our main results, we use the real-world BurstGPT trace and synthetic traces that resemble BurstGPT's load surges. For completeness, we present results from running Andes and baselines on a Poisson arrival trace. Figure 2.16 shows the average QoE of systems under varying Poisson arrival rates over a duration of 20 minutes. Andes still consistently delivers higher average QoE compared to baselines, particularly under high request rates.

2.7 Related Work

LLM Inference. LLMs have numerous applications including chatbots, code generation, and agents. Andes focuses on online LLM inference serving for conversational AI services, a dominant use case of LLM serving [62].

On the systems-side, Orca [172] introduced iteration-level batching to enhance the throughput of LLM inference, followed by vLLM [83] developing PagedAttention to optimize memory usage. DistServe [183], Sarathi-Serve [11], and LoongServe [160] optimize prefill and decode computations, but keeps FCFS scheduling. Llumnix [146] proposes cluster-wide load balancing and request live migration, VTC [143] a non-preemptive fair request scheduler, and CacheGen [101] a streaming and reusing technique for KV cache entries across LLM server

instances. Andes is the first to identify the insufficiency of existing optimization metrics for user experience in conversational AI, and to optimize QoE using a preemptive scheduling approach. Our QoE definition can be extended and adapted to suit other application scenarios as well.

Video Streaming and QoE. QoE in conversational AI services has some parallels with QoE in video streaming [38, 16, 100, 72, 178] but encounters unique challenges. While video streaming is primarily influenced by network conditions [18], LLM serving is mainly constrained by GPU compute and memory [154]. Additionally, factors important to video streaming QoE include startup time, average bitrate, and buffering time ratio [71]. Some have parallels in conversational AI (e.g., startup time), whereas others do not. We proposed a QoE metric tailored to conversational AI and designed a serving system that optimizes it.

2.8 Conclusion

In this work, we identify that user experience has thus far been overlooked in optimizing systems for conversational AI services. This motivates us to define a QoE metric for conversational AI and build Andes, an LLM serving system that optimizes for it. Andes delivers significantly higher QoE compared to existing systems, which also translates to being able to serve more concurrent requests while maintaining the same level of QoE. We hope that Andes will encourage the community to dive deeper into understanding and optimizing user experience for conversational AI services.

CHAPTER 3

Empowering Diverse End Users via Cohort-Based Collaborative Learning

Collaborative learning (CL) is an emerging machine learning (ML) paradigm that enables heterogeneous edge devices to collaboratively train ML models without revealing their raw data to a logically centralized server. However, beyond the heterogeneous device capacity, CL participants often exhibit differences in their data distributions, which are not independent and identically distributed (Non-IID). Many existing works present point solutions to address issues like slow convergence, low final accuracy, and bias in CL, all stemming from client heterogeneity.

In this chapter, we explore an additional layer of complexity to mitigate such heterogeneity by grouping clients with statistically similar data distributions (*cohorts*). We propose Auxo to gradually identify such cohorts in large-scale, low-availability, and resource-constrained CL populations. Auxo then adaptively determines how to train cohort-specific models in order to achieve better model performance and ensure resource efficiency. Our extensive evaluations show that, by identifying cohorts with smaller heterogeneity and performing efficient cohort-based training, Auxo boosts various existing CL solutions in terms of final accuracy (2.1%–8.2%), convergence time (up to 2.2×), and model bias (4.8% - 53.8%).

3.1 Introduction

Collaborative learning (CL) enables distributed clients to collaboratively train an ML model without centralizing their local data to the cloud. It circumvents the systematic privacy risk and cost of data transfers in centrally collecting user data. Hence, CL is increasingly being adopted by many popular applications, such as Google's Gboard [5], Apple's Siri [121], NVIDIA's medical platform [92], Meta's Ads recommendation [66], and WeBank risk prediction [108].

Collaborative Learning (CL) typically involves a substantial number of clients, ranging from hundreds to millions, and the training process can span days or even weeks [170]. Given the limited availability and resource constraints of client devices, only a fraction of clients contribute to each round of training in practice. Therefore, it is essential to reduce the training time while accommodating these practical constraints. However, CL encounters unique challenges stemming from statistical heterogeneity among user data, which contributes significantly to extended training time and suboptimal model performance [93, 180, 148, 102]. Several studies that try to mitigate the effect of statistical heterogeneity, such as FedYoGi [127], q-FedAvg [91], FTFA [32], have shown that their convergence speed depends on the degree of heterogeneity, both theoretically and empirically (§3.2.2).

We explore the possibility of mitigating this issue at its core by grouping clients with similar data distributions, known as *cohorts* [175] (§3.2.3). If a population has K cohorts, training K separate models – one for each cohort with lower statistical heterogeneity – can boost the performance of many existing CL algorithms that are complementary to ours and focus on convergence [127, 90, 87], fairness optimization [91], communication efficiency [6, 133, 73], etc.

Although recent works attempted to identify cohorts and train separate models for them [26, 50, 102, 169], they are not applicable to real-world CL deployments. This is because unlike easy-to-deploy solutions such as FedAvg and FedYoGi [127, 109], clustering clients at scale and in the wild poses unique challenges (§3.2.4). Existing solutions often ignore the scale and sparsity of the device participation. They also ignore the constraints on availability and capacity of end-user devices, which calls for low-overhead algorithms.

We propose **Auxo** to enable 1) scalable cohort identification to reduce intra-cohort heterogeneity in large-scale and limited-availability CL scenarios; and 2) efficient cohort-based training to facilitate most CL optimizations, such as faster training completion and better model accuracy, without additional resource requirements. Auxo addresses the following challenges toward practical CL deployment (§4.4). First, unlike existing clustering strategies which require exhaustive passes through all clients [137], on-demand device availability [26], or additional on-device training for every participant [50, 39], Auxo introduces a more flexible client clustering solution. It allows sporadic client availability, respects client resource constraints, and maintains client privacy. Auxo can progressively identify cohorts and scalably cluster clients based on their gradients in spite of the absence of anchored gradients for straightforward comparison. Second, unlike expensive and ad-hoc hyper-parameter tuning stages used in existing solutions, Auxo progressively generates the appropriate number of cohorts and identifies suitable timings to create them. Thus, Auxo maximizes the use of

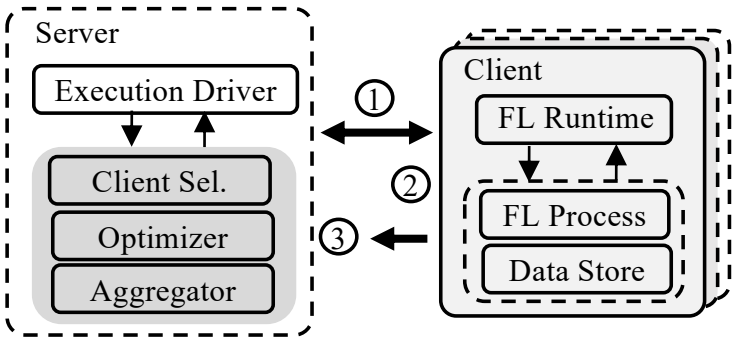


Figure 3.1: Traditional CL overview. The server first selects from available clients and sends out model weights. Clients train the updated model on their local dataset. After training is finished, clients report their model gradient to the server.

limited client resources to enhance training speed and model performance.[1] Finally, we design a scalable system to support efficient cohort clustering and training at scale while being robust to uncertainties (e.g., failure tolerance and unfavorable settings) at scale (§3.5).

We have implemented (§3.6) and evaluated (§4.5) Auxo on a wide variety of real-world CL datasets, tasks, and algorithms at scale. Compared to existing solutions, Auxo improves the performance for various CL algorithms, such as better model accuracy (2.1%-8.2%) and convergence speed (up to 2.2×) and smaller bias of model accuracy (4.8% - 53.8%).

Overall, we make the following contributions in this paper:

1. We propose a systematic clustering mechanism to identify cohorts for the practical large-scale, low-availability and resource-constrained CL setting.

2. We identify a sweet spot for jointly optimizing model convergence and training cost, and provide analytical insights to ensure good model performance.

3. We implement and evaluate Auxo at scale, showing large improvements in final accuracy, convergence time, and model fairness over the state-of-the-art. Auxo is open-source and available on GitHub.[2]

3.2 Background and Motivation

We start with a brief introduction of collaborative learning (§3.2.1), followed by the challenges it faces in real-world settings (§3.2.2) . Next, we describe some opportunities to improve CL

[1]We refer to the number of participants that contribute to a round of CL training as training resource throughout this paper.

[2]https://github.com/SymbioticLab/FedScale/tree/master/examples/auxo

that motivates our work (§3.2.3). Finally, we explain the limitations of related works that motivate our algorithm and system design (§3.2.4).

3.2.1 Collaborative Learning

A typical cross-device CL system consists of two primary components (Figure 3.1): A logically centralized cloud *server* that maintains a single global model and many distributed *clients* with private local data. The overall lifecycle of an CL training round can be divided into three broad stages.

1. **Selection stage:** Clients check in with the server continuously to announce their availability for CL computation. The server selects a number of *participants* for that round based on its client selection strategy.

2. **Execution stage:** The selected participants download the current model from the server and perform server-specified computation on their local data.

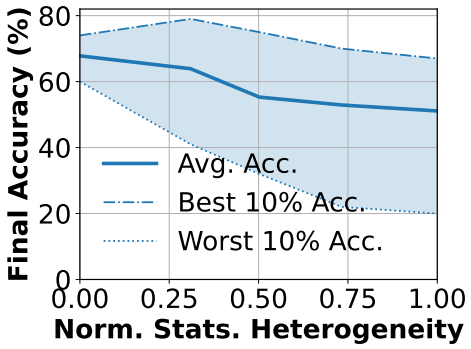
3. **Aggregation stage:** Participants that successfully complete the execution stage send model updates back to the server. The server aggregates the updates to finalize an updated model for the next round.

3.2.2 Heterogeneity Challenges in CL

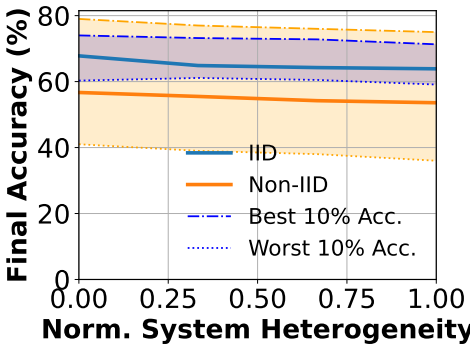
Unlike centralized ML, CL faces unique challenges in terms of statistical and system heterogeneity. The former refers to the varying data volumes and difference of data distribution across clients, which hinders model convergence; the latter refers to variations in system characteristics among participants' devices, which results in large differences in training performance. Increasing heterogeneity in either dimension leads to poor performance.

Impact of statistical heterogeneity. Under large statistical heterogeneity across clients, poor model accuracy, training time and fairness are often exacerbated, because the model is deployed on individual clients but is often trained over all the clients. Existing works that address statistical heterogeneity in CL assume bounded heterogeneity to simplify the problem complexity [93, 180, 127, 90]. However, we notice this does not hold in practical CL settings, which leads to great performance degradation under larger statistical heterogeneity.[3] Indeed, our analysis of FedYoGi [127] (a state-of-the-art CL algorithm) on OpenImage [1] (an CL image dataset), in Figure 3.2a shows that the model accuracy and its fairness across clients

[3]In this experiment, we measure the statistical heterogeneity among a set of clients using the popular L2 distance on their data distributions [87].



(a) Statistical heterogeneity.



(b) System heterogeneity.

Figure 3.2: The impact of heterogeneity on final accuracy.

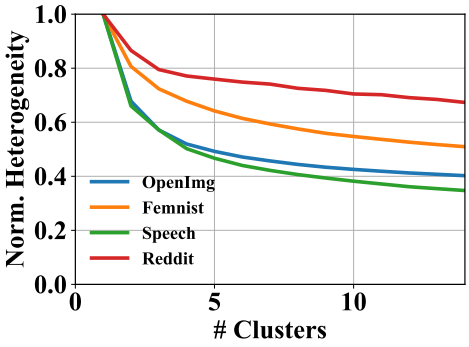


Figure 3.3: Intra-cluster heterogeneity in real datasets.

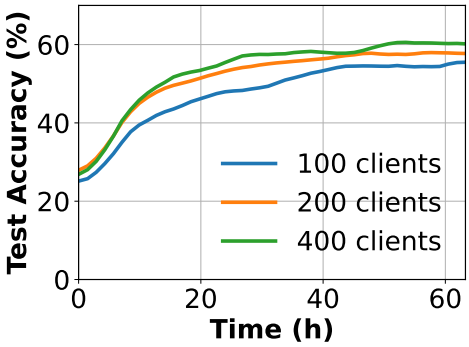


Figure 3.4: Diminishing return when adding more participants.

worsens with increasing statistical heterogeneity. To achieve the same model performance under larger heterogeneity, more communication and/or computation costs are needed. This is true for personalization algorithms as well [148].

Impact of system heterogeneity. Heterogeneity of system-level characteristics raise challenges such as fault tolerance and straggler mitigation [90, 66]. Over-commitment [23], which discards updates from slowest-responding participants, is commonly used to reduce the impact of stragglers, but it may lead to participation bias against slow devices. Figure 3.2b shows the final accuracy of the OpenImage task under different degrees of system heterogeneity (variance of system speed). For each experiment, we control the round duration and the number of successful participants to be the same; as a result, participation bias exacerbates with increasing system heterogeneity. Since participation bias may enhance statistical heterogeneity in another form, the final accuracy decreases with increasing system heterogeneity (albeit at a slower rate than statistical heterogeneity).

	CCL	CL+HC	FlexCCL	IFCA	Auxo
Partial part.	×	✓	✓	✓	✓
Low avail.	×	×	✓	✓	✓
Res. constraint	×	×	×	×	✓
Training perf.	×	×	×	×	✓

Table 3.1: Comparing Auxo with existing Clustered CL.

3.2.3 Opportunities

The opportunity for improving CL training performance, therefore, lies in decreasing heterogeneity especially the statistical heterogeneity based on the observation of the previous subsection. By identifying statistically homogeneous groups and performing CL within each group, we may be able to boost model performance of most CL algorithms that are suffered by the heterogeneity.

Despite large statistical heterogeneity across the entire CL client population, there exist groups of statistically similar clients in most large populations. Figure 3.3 shows that for four representative CL workloads [85] in the real world. We use K-means clustering (with increasing values of K) on clients' data distribution by their L2-distance metric. As the number of clusters increases from one (i.e., traditional CL with one global model) to larger values, we observe a small number of statistically similar groups emerge for most datasets.

However, training K models to converge may need more training resources compared to training one model. As shown in Figure 3.4, increasing training resources has diminishing returns on the model convergence, which presents the primary opportunity leveraged in this work: *instead of letting all available clients contribute to a single global model, it may be more beneficial to partition them into several cohorts, each with smaller heterogeneity.*

3.2.4 Limitations of Existing Clustered CL

Recent efforts in the ML community have (theoretically) explored to create smaller groups of statistically similar clients. Yet, existing clustered CL algorithms often fall short across multiple dimensions in practical deployments, which motivates us to design systems support for efficient cohort identification and training. We empirically show the superior performance of Auxo over them too (§3.7.2).

Scalability. CL in practice often involves millions of clients, and only a small fraction (∼5% [23, 85]) are available to participate in during a time window. Such low availability and partial participation limit the available information for clustering algorithms. This,

unfortunately, is ignored by CCL [137], multi-center [102] and CL+HC [26], making their deployment impractical as they require a complete pass over the entire population to identify clusters. Furthermore, clients usually have limited on-board resources, but IFCA [50], FlexCCL [39], ICCL [169], k-FED [37] and CL+HC require extra computation for *every* client to assign them to a cluster. This imposes a significant computational and communication burden on already resource-constrained devices and diverts resources away from the primary task of model training. For example, IFCA initiates multiple global models and broadcasts all models for each participant to choose from in each round; and FlexCCL and CL+HC require pre-training for every client to identify their clusters.

Efficiency. In addition to the challenge of identifying statistically similar groups at scale, how to leverage those similar groups to improve model performance introduces new tradeoffs in deciding the right number of cohorts and time to partition. Given a fixed amount of resources, generating more cohorts results in smaller heterogeneity; but it divides up the fixed training resource and unique training data per cohort, which hurts model convergence and generalizability. Moreover, partitioning clients too early can lead to model bias as the model is not generalized well by training on various clients, while partitioning too late can result in model variance over high heterogeneity. Unfortunately, most existing clustered CL algorithms are unaware of these tradeoffs, and rely on ad-hoc hyper-parameter tuning, which is prohibitively expensive as CL training can take many days and consume a large amount of resources.

In conclusion, as detailed in Table 3.1, an effective client clustering solution in Collaborative Learning (CL) should take into account the following realistic constraints:

1. Partial participation: The algorithm should accommodate CL training that involves only a fraction of total participants in each round.

2. Low availability: The algorithm should respect clients' sporadic availability, without necessitating participation from any clients at a specified time.

3. Resource constraints: The algorithm should avoid demanding additional on-device computation for performing clustering.

4. Training performance: The algorithm should optimize model performance—focusing on convergence and generalizability—within the constraints of a fixed training resource. This includes consideration of how clustering the CL population might positively impact performance despite reduced heterogeneity.

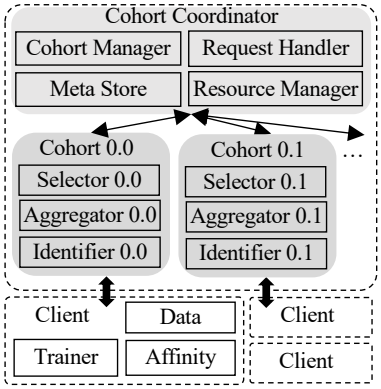


Figure 3.5: Auxo architecture. Auxo server guides Auxo clients to train on their best-fit cohorts.

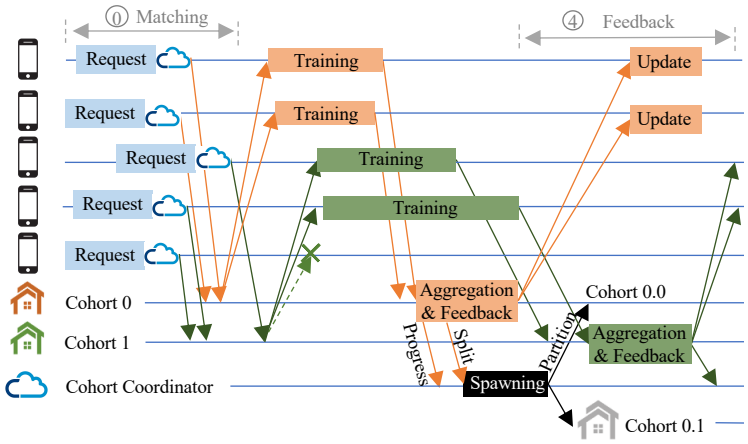


Figure 3.6: Auxo lifecycle. Clients check in with their affinity requests, and participate training within matched cohorts. Cohorts are gradually identified based on participants response.

3.3 Auxo Overview

Auxo progressively reduces the intra-group heterogeneity and improves the model performance through cohort identification and cohort-based training toward practical CL. In this section, we introduce the cohort abstraction, provide an overview of how Auxo manages cohorts in a distributed fashion and fits into the CL life cycle.

3.3.1 Cohort Abstraction

Instead of training only one global model, Auxo trains a model separately for each group of clients that shares similar statistical data characteristics. We refer to each of these groups, which can perform independent CL training over more homogeneous clients than the overall population, as a *cohort* $C_m (m \in [1, M])$ with two associated properties:

1. A cohort should hold a specialized model that targets on it data distribution with smaller heterogeneity.

2. A cohort should have enough members $|C_m|$ to form a meaningful group and deliver the benefit of partition.

Traditional (i.e., cohort-agnostic) CL training has a single cohort with unbounded heterogeneity among the members.

3.3.2 Auxo Architecture

Auxo server consists of two primary components (Figure 3.5):

1. A logically centralized **cohort coordinator** performs three main functions. First, it manages existing cohorts for fault tolerance. Second, it matches clients to their best-fit cohorts. Finally, it monitors the progress of cohort training and identification in order to decide cohort partition when it observes an opportunity for better model convergence .

2. A set of **cohorts** each performs independent CL training. Each cohort contains traditional CL components such as aggregator and client selector. On top of traditional CL training activities, each cohort continuously identifies its internal composition, reports its progress to the coordinator and waits for the partition instruction from the coordinator.

CL Lifecycle in Auxo As shown in Figure 3.6, following the traditional CL stages in Section 3.2.1, Auxo adds a matching stage **0** and a feedback stage **4** before and after the traditional round.

0 Matching stage: When checking in, clients using Auxo optionally include an affinity request (a hint about their cohort preference) to the cohort coordinator. If it took part in the training of one or more cohorts in the past, its preference is dependent on previous feedback. Otherwise, it has no preference. The cohort coordinator forwards the affinity request to the corresponding cohort based on its search algorithm and client's request.

1 -3 Traditional CL stages: Each cohort starts a traditional CL training round independently after continuously receiving its client requests from the cohort coordinator. These traditional stages include client selection, client training, server aggregation, and so on.

4 Feedback stage: After the traditional CL round finishes, each cohort updates the affinity feedback for its current participants based on the Auxo clustering algorithm (§4.4). Then, each participant receives an affinity feedback – w.r.t. the cohort it trained with — and updates the corresponding affinity record for submitting requests in a future round of CL training. During this stage, each cohort also reports its training and identification progress to the cohort coordinator.

Resource management: Auxo jointly maximizes model convergence and resource efficiency in two ways. First, its scalable cohort identification algorithm does not require extra on-device computation and uses the same amount of resources as traditional CL algorithms(§4.4.1- §3.4.3). Second, it carefully chooses the number of cohorts and time to partition to theoretically guarantee better model convergence and generalizability despite each cohort having less training resources than the previous global model (§3.4.4).

Threat model and robustness. Like state-of-the-art production CL systems [23, 66, 155], Auxo considers an *honest-but-curious* centralized server for aggregation, which can infer any information without interfering with the CL training. Auxo also assumes that most clients are honest (correct), and only a small fraction can act maliciously under the control of a bad actor.We elaborate on how Auxo can provide robustness under this threat model in Section 3.5.2.

3.4 Auxo Clustering

In this section, we present the core clustering algorithm used in Auxo to identify cohorts (§4.4.1- §3.4.3). Then, we introduce the systems techniques to enable cohort-based training under realistic constraints (§3.4.4).

3.4.1 Problem Formulation and Overview

Auxo aims to accurately cluster clients by their statistical heterogeneity into appropriate cohorts under the following real-world CL constraints:

1. *Scalability:* The participants $\mathcal{P}^r$ in each round are only a small fraction of all clients (N), i.e., $|\mathcal{P}^r| \ll N$. How to identify cohorts and cluster clients at scale under such low client availability?

2. *Resource Efficiency:* How to conduct the clustering process without incurring overhead on devices, such as extra model training and client participation that do not contribute to model training?

3. *Information Deficiency:* The information available to today's CL central server is limited to such as gradients and training loss. How to cluster clients without requesting additional information from clients?

Problem Formulation: The *input* to the server is a list of participants along with their gradients collected over training rounds based on these two constraints. Intuitively,

the gradient of client relies on its local dataset x_i and the received model weights (unique for the round r and cohort m), and this gradient is multi-dimensional, embedding more information than its counterparts (e.g., training loss). As such, we can formulate the *input* of the clustering algorithm in each round r as $\{\{g_m^r(x_i)\}_{i\in\mathcal{P}_m^r}\}_{m\in[1,M_r]}$, where $g_m^r(x_i)$ is the gradient of participant i , $\mathcal{P}_m^r$ is the participants list, and M_r is the number of cohorts.

The *output* is the cohort membership $\{S_i \in [1, M]\}$ for each client $i \in [1, N]$. Following the objective of traditional clustering algorithms [105], Auxo also aims to minimize the average intra-cohort heterogeneity (J) defined as:

$$J = \sum_{m=1}^{M} \frac{1}{2|\{x|S_x = m\}|} \sum_{S_i,S_j=m} ||x_i - x_j||^2. \tag{3.1}$$

Intuitively, we can model it as a clustering problem $\{x_1, ..., x_N\} \rightarrow \{S_1, ..., S_N\}$, whereas doing so encounters new challenges.

1. How to derive client data similarity without direct access to data and without iterating all but part of the clustering objects every round.

2. How to assign new incoming clients to the best-fit cohort without prior information after Auxo generates more than one cohorts.

Following this problem definition and challenge, Algorithm 2 illustrates the overview of Auxo clustering mechanism, which consists of an online cluster algorithm to cluster clients at scale (§3.4.2) and the cohort selection for individual CL clients (§3.4.3). Note that, Auxo's clustering algorithm can operate in the background, imposing no additional overhead on the training process.

3.4.2 Online Clustering

Auxo resorts to the similarity of clients' gradients to capture their statistical similarity. Our design is inspired by the recent advances in ML theory [137, 141], which show that the data heterogeneity can attribute to the gradient divergence [90] and a smaller heterogeneity would have smaller gradient divergence for the *same* initial model weight. Here, we measure such gradient divergence using the widely-used cosine similarity [168] among the input batch of gradients $g_m^r(x_i), i \in \mathcal{P}_m^r$ to investigate client similarity.[4] Compared to other counterparts such as L-2 distance which does not take into account the direction of the gradients, cosine similarity better quantifies how similarly their needed model changes are directed.

[4]Cosine similarity measures the similarity between two vectors of an inner product space [168].

Algorithm 2 Auxo Clustering Algorithm

Input: Participants list $\mathcal{P}$, Exploration factor ϵ
Output: Client-cohort membership list $S_{\mathcal{D}}$

$M \leftarrow 1$ ▷ Initialize the number of cohorts
$S_{\mathcal{D}} \leftarrow 0$ ▷ Initialize client-cohort membership
$R_{\mathcal{D},M} \leftarrow 0$ ▷ Initialize client-cohort reward
$L_{\mathcal{D},M} \leftarrow \text{N/A}$ ▷ Initialize client-cohort cluster id
for each round $r = 1, 2, \ldots$ **do**
    $\mathcal{P}_m^r \leftarrow \{i | S_i = m, i \in \mathcal{P}^r\}$
    for each cohort $m = 1, \ldots, M$ in parallel **do**
        $R_{\mathcal{P}_m^r} \leftarrow \textsc{ClientClustering}(\mathcal{P}_m^r)$
        $S_{\mathcal{P}_m^r} \leftarrow \textsc{CohortSelection}(R_{\mathcal{P}_m^r}, \epsilon, r)$
return $S_{\mathcal{D}}$
function $\textsc{ClientClustering}(\mathcal{P}_m^r)$
    if $r = 1$ **then**
        $L_{\mathcal{P}_m^r,m} \leftarrow \textsc{Kmeans}(g_m^r(x_{\mathcal{P}_m^r}), K)$
    else
        $\mathcal{P}_k \leftarrow \{i | L_{i,m} = k, i \in \mathcal{P}_m^r\}, \forall k \in [0, K)$
        $C_k \leftarrow \overline{\{g_m^r(x_{\mathcal{P}_j})\}}, \forall k \in [0, K)$
        $L_{\mathcal{P}_m^r,m} \leftarrow \arg\min_k \|g_m^r(x_{\mathcal{P}_m^r}) - C_k\|_2$
    if $\textsc{PartitionCriteria}(m)$ **then**
        $M \leftarrow M + K - 1$
        $R_{\mathcal{D},m+k} \leftarrow R_{\mathcal{D},m} + 0.1 \cdot \mathbb{1}(L_{\mathcal{D},m} = k), \forall k \in [0, K)$
    if $M > 1$ **then**
        $R_{\mathcal{P}_m^r,m} \leftarrow \textsc{ExploitReward}(R_{\mathcal{P}_m^r,m}, x_{\mathcal{P}_m^r})$
        $R_{\mathcal{P}_m^r,m'} \leftarrow \textsc{ExploreReward}(R_{\mathcal{P}_m^r}, m'), \forall m \neq m'$
    return $R_{\mathcal{P}_m^r}$
function $\textsc{CohortSelection}(R_{\mathcal{P}_m^r}, \epsilon, r)$
    for client i in $\mathcal{P}_m^r$ **do**
        if random(0,1) $> \epsilon^r$ **then**
            $S_i \leftarrow \text{random}(0, M)$
        else
            $S_i \leftarrow \arg\max R_i$
    return $S_{\mathcal{P}_m^r}$

However, the sporadic participation of clients in each training round limits the data available for clustering algorithms to a subset of the entire client population at any given time. Traditional clustering algorithms, such as K-means and KNN, require a complete pass of the population, rendering them inapplicable here. Mini-batch clustering algorithms [139], on the other hand, operate on small batches of the population each round, maintain a running centroid for cluster assignments. Nonetheless, this strategy cannot directly be applied in our case because we only know the gradients $g_m^r(x_{\mathcal{P}m^r})$ and not the raw data $x_{\mathcal{P}}$. Further, since the gradient $g_m^r(\cdot)$ depends on the initial model of round r and client data - both unknown and different across rounds and cohorts. These complexities preclude us from maintaining absolute cluster centroids over successive rounds in a straightforward manner, making naive mini-batch clustering infeasible.

Algorithm 2 outlines how Auxo starts with one cohort for the entire CL population, and then adaptively identifies cohorts based on gradients of mini-batch clients. After using K-means to initialize the cluster prototype (Line 13), in each round, Auxo collects the training feedback from the clients and assigns clients to their closest clusters (Line 15). Meanwhile, Auxo incrementally refines cluster centers based on the gradients of newly assigned clients in each round (Line 16). With repeated cluster updating and clients assignment, Auxo can effectively identify the clusters at scale (Line 17). Each new cohort starts with the parent cohort model weights with the same architecture, performs conventional CL steps separately, and converges to different model weights. Once discernible clusters emerge and certain partition criteria are fulfilled (e.g., enough participants left for model convergence after partition), Auxo decides to spawn cohorts based on these pre-identified clusters (Line 18) and train cohort models separately within their corresponding client groups. At runtime, Auxo adaptively decides the right time and the right number of cohorts to partition to find the sweet spot of model performance and the resource consumption of training multiple cohorts (§ 3.4.4).

3.4.3 Cohort Selection

Although clustering captures the membership of already-identified clients, doing so for a new client is unkown a priori, since we neither have access to client data nor have absolute cohort centers that can inform a new client to choose the closest cohort. This challenge is further amplified by the large training population, wherein more CL clients participate in model training for the first time than not.

To address this, Auxo adopts an *exploration-exploitation* strategy to efficiently identify the cohort membership for new participants. This allows us to first randomly assign a new

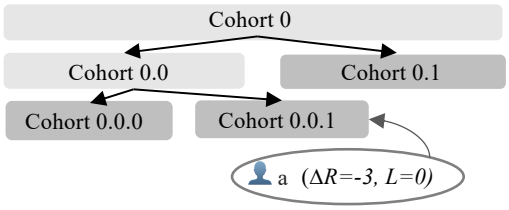
client to a cohort. After getting the feedback on how well the client fits in that cohort, Auxo attempts to identify a more suitable cohort for it the next time it participates again.

Auxo uses reward-based decaying ϵ-greedy selection [147] to help the client find the best-fit cohort (Line 9). With an aim to maximize the expected reward for each client, there is a $1-\epsilon$ probability of selecting a cohort with a maximum reward and a ϵ probability of selecting cohorts randomly, where $\epsilon \in [0, 1]$ is the exploration factor that decays over time to account for the latest information. Intuitively, smaller gradient divergence compared to the members within the explored cohort means a better fit and gives a higher reward. Hence, Auxo calculates the relative divergence between the client gradients and the explored cohort center. This is done by first estimating the cohort center via averaging the client gradients within the cohort $\mathcal{P}^r_{m,Known}$, to be $D = ||g^r_m(x_{\mathcal{P}^r_m}) - \overline{g^r_m(x_{\mathcal{P}^r_{m,Known}})}||_2$, where $\overline{g^r_m(x_{\mathcal{P}^r_{m,Known}})}$ represents the estimated cluster centers for cohort m. Next, we take the popular approach to identify outlier clients [10]. Specifically, we consider clients as outliers if their distance to the cohort center exceeds the threshold, which is calculated as the sum of the mean and the standard deviation of D, denoted as $avg(D) + std(D)$. If the client gradient distance to the cohort center is larger than this threshold, this client is not considered as the cohort member. As such, the instant reward becomes $\Delta R = 1 - \frac{1}{avg(D)+std(D)}D$, where the client with a negative ΔR would be considered as an outlier of the cohort. Then, Auxo updates the reward between each client and its explored cohort with a decay factor γ as $R_{\mathcal{P}^r_m,m} = \gamma * \Delta R + (1-\gamma) * R_{\mathcal{P}^r_m,m}$, γ=0.2 by default in popular exploration-exploitation designs.

Efficient cohort exploration. During exploration, there may exist multiple cohorts for a client to try out with. To improve the searching efficiency and save device training resources, during both training and deployment, Auxo enables a new client to perform a binary search to find the most appropriate cohort by predicting the rewards for other unexplored cohorts m' through function `ExploreReward()` (Line 27): $R_{\mathcal{P}^r_m,m'} \mathrel{+}= \frac{R_{\mathcal{P}^r_m}}{d(m,m')+1}, \forall m \neq m'$.

The intuition behind the cohort search is that the client may perform similar to or receive similar rewards from the cohorts that are closer/similar to the previously explored ones, and vice versa. To find out the cohort similarity, we first define the distance (d) between two cohorts to be the distance to their lowest common ancestral cohorts based on the hierarchical cluster relationship among cohorts. Given an explored cohort m and the reward ΔR_m for a participant, Auxo calculates the distance d and updates the rewards for unexplored cohorts to be inversely proportional to their distance. For example, if a client receives a negative reward for the chosen cohort, then he is more likely to explore another furthest cohort with higher reward given by `ExploreReward()` next time.

Taking Figure 4.3 as an example, a new client a explores $Cohort_{0.0.1}$ and receives the



	Cohort 0.0.0	Cohort 0.0.1	Cohort 0.1
d	1	0	2
ΔR_m	- 3/(1+1) = -1.5	- 3/(0+1) = -3	- 3/(2+1) = -1

Figure 3.7: Reward update based on the hierarchical structure among all cohorts.

corresponding feedback rewards -3. Then, with the intuition that the client may have similar performance on a closer cohort, Auxo calculates the distance between $Cohort_{0.0.1}$ and other cohorts. As shown in Table 4.3, Auxo updates the rewards to be inversely proportional to the cohort distance $d_{m'}$: $\Delta R_{m'} = \frac{\Delta R}{d_{m'}+1}$ Since $Cohort_{0.0.1}$ and $Cohort_{0.1}$ have a larger distance between them, $Cohort_{0.1}$ ends up with a relatively higher reward and has higher probability to be explored by client a in the future.

3.4.4 Cohort-Based Training

While clustering reduces heterogeneity within a cohort, generating a larger number of cohorts may dilute available resources for each individual cohort when operating under fixed training resources. Consequently, this leads to a new trade-off between resource efficiency and model convergence. As shown in Figure 3.3, generating more cohorts has diminishing returns in terms of heterogeneity. In a setting where total training resources are fixed, allocating resources to a larger number of cohorts implies fewer resources for each, which may negatively affect model convergence. Conversely, having too few cohorts is insufficient for adequately addressing intra-cohort heterogeneity. Therefore, Auxo faces the challenge of optimally determining both the number of cohorts and the timing for their creation to balance resource efficiency and model performance effectively.

Intuitively, the decision to generate new cohorts should be based on the extent of client heterogeneity and the available training resource budget post-partition. When client heterogeneity is significant and the resource budget is sufficient, the creation of additional cohorts is warranted to further reduce client heterogeneity. On the other hand, when these conditions are not met, the creation of new cohorts should be deferred.

We next provide analytical insights to ground our strategy. Prior works in ML theory [75, 93, 180, 127] have shown that the convergence rate of CL training is largely dominated by

heterogeneity. We start by analyzing the relationship between heterogeneity and training resources in theory. Inspired by the convergence analysis of FedAvg [75], we establish the following Lemma.

Lemma 1. *If the population and training resources are partitioned into up to K cohorts, to theoretically achieve better model convergence, intra-cohort heterogeneity should be reduced by $\sqrt{K}$ times when the training resource $|\mathcal{P}|$ is larger than $\alpha\sqrt{\frac{|\mathcal{P}_0|}{J_0^2}}$. α is a constant setting that elaborates the relationship between model convergence and training resources.*

From Lemma 1, we notice that the number of generated cohorts rely on the expected reduced heterogeneity and a lower bound of training resources. As such, Auxo actively monitors the gradient divergence within each cohort at runtime to estimate the potential heterogeneity reduction.

When a sufficient decrease (e.g., $\frac{1}{\sqrt{K}}$) in intra-cohort heterogeneity and ample post-partition training resources are detected, Auxo autonomously partitions the population into a maximum of K cohorts, allotting equal training resources to each. This strategy theoretically enhances model convergence through cohort-based training in Auxo. As for some CL datasets with larger heterogeneity, CL developers can further improve model convergence by dynamically raising the resource budget to allow generating more cohorts.

In addition to deciding the right number of cohorts, the time to cohort partition is also critical to model convergence. As cohort partitioning may reduce the unique training data for each cohort model, the trade-off between model bias and variance can be affected by the time of partition. On the one hand, hard partitioning of the entire population at the beginning could reduce heterogeneity, but it could also reduce the amount of unique training data for each cohort model, leading to poor model generalizability. On the other hand, late partitioning exposes the model to diverse training data but leads to worse model variance due to high heterogeneity. These also guide the *reuse* of identified cohorts to facilitate other CL tasks.

From the sensitive analysis of cohort partition time (§3.7.4), we found the model convergence is not sensitive to exact partition time as long as cohorts are not partitioned at the beginning or the end of the training. We report more results about the effect of partition time on model convergence in Section 3.7.4.

Finally, the start time of gradient-based clustering can impact the efficiency of the process. In the early stages of training, gradients are often large and may not adequately capture the distributional features of the data. However, as the model approaches convergence, the gradients become more informative indicators of data similarities. Thus, it is crucial for Auxo to judiciously select the optimal starting point for clustering so as not to delay the

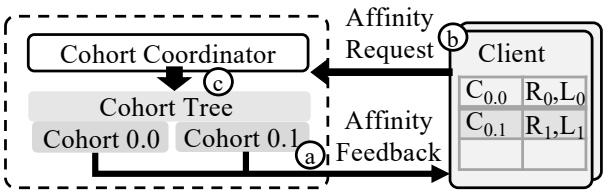


Figure 3.8: Scalable Design Overview ⓐ Cohort affinity feedback. ⓑ Client affinity request. ⓒ Cohort coordinator request match.

cohort identification. Detailed results discussing the effect of the clustering start time on model convergence can be found in Section 3.7.4.

3.5 Auxo System Design

In this section, we discuss how to design a practical and robust system on top of the clustering algorithm under real-world challenges.

3.5.1 Distributed Auxo

As the scale of training grows, the server faces more server challenges for tremendous storage, fault tolerance, and client privacy in order to maintain the cohort and client information. Thus, Auxo designs a solution to use a soft-state server that ofCLoads cohort-related information to individual clients to mitigate these challenges. In this subsection, we describe how to implement the proposed clustering algorithm in a distributed fashion, while achieving the same objective.

Firstly, we introduce *affinity message*, which is a lightweight message containing all necessary state information needed to identify cohorts in a distributed fashion. Affinity message consists of two pieces of information between a client and a cohort to enable efficient state transmission: (*Reward* $R \in \mathbb{R}$, *Cluster index* $L \in [0, K)$). The *reward* implies how well the client fits for this cohort. The *cluster index* expresses the client's cluster membership within this cohort and is used to indicate its future cohort index.

Through exchanging affinity messages between different components, Auxo encourages similar clients to collaborate more in a distributed fashion. As shown in Figure 3.8, we next describe **a**. how a cohort informs its relationship with its participants, **b**. how clients request for their preferred cohort based on the affinity feedback and **c**. how the cohort coordinator matches different requests.

Affinity Feedback At the end of each round, every cohort computes the affinity feedback to inform participants about their relationship with the cohort. These affinity feedback correspond to the clustering results returned by Algorithm 2 Line 6 (reward R) and Line 17 (cluster index L). These clustering results would be sent back to the participants respectively in the format of affinity messages, which informs the participant about whether the cohort is a good fit and which sub-cohort to select after partitioning.

Client Reaction After receiving the affinity feedback from the cohort, the client would update its affinity records itself based on the equation in Algorithm 2 Line 22- 27 and copy the cluster index directly. Following the same decaying $\epsilon-$greedy selection method (§3.4.3), clients select the cohort to train by themselves. Then, clients ready to participate would submit the corresponding affinity request to the cohort coordinator.

Request Match After receiving the affinity request, the cohort coordinator matches each client to the cohort it requests. Note that only the leaf cohort in the cohort tree would be returned as it conducts actual CL training inside. The requested cohort may not be the leaf cohort because some clients may not be aware of the cohort partitioning, which is not yet transparent to all clients. In this case, the cohort coordinator should assist clients to select their best-fit cohort through finding the closest leaf cohort indicated by the requested cohort and cluster index in the affinity message.

After finding a proper cohort, cohort coordinator would forward this affinity request to the corresponding cohort to initiate traditional CL rounds. Moreover, these forwarded affinity requests provide each cohort with all necessary input to conduct the clustering algorithms. Thus, after receiving the gradients from its participants, each cohort is able to run the Algorithm 2 independently to compute the aforementioned affinity feedback.

3.5.2 Resilient Auxo

Fault Tolerance Auxo enables fast recovery to minimize the impact on the model training. Upon a cohort process failure in the server, the cohort coordinator spawns a new cohort process. The new cohort loads the model from the latest checkpoint and restarts the incomplete round.

If the cohort coordinator fails, cohort processes would continue their current independent CL training and wait until a new cohort coordinator to be re-spawned. Clients checking in within that recovery period would be ignored.

Finally, Auxo is resilient to client failures just like traditional CL by design. Most client failure handlers, which are orthogonal to Auxo, can be applied directly. In addition, a failed

client, who may lose its own affinity records, would restart exploring again. We empirically show that Auxo can tolerate a certain amount of such client failures while continuing to benefit the CL training (§3.7.5).

Robustness Based on Auxo's threat model, Auxo can naturally cooperate with some existing privacy-preserving approaches [44, 110, 111] to address potential threats from both the server and clients. To handle the honest-but-curious server, Auxo can be used with local differential privacy (LDP), which is used to provide user-level privacy guarantees. Since differential privacy is immune to post-processing [42] and Auxo's clustering is post-processing, Auxo incurs no additional privacy loss.

To handle a small fraction of unreliable clients [15, 21], Auxo can be used with existing adversary-resilient solutions [31, 144]. For Auxo-specialized adversaries, such as fake affinity requests, Auxo detects anomalies by comparing its position in the cluster with its claimed affinity (Algorithm 2 Line 6). If a significant discrepancy is detected, Auxo will detect and blacklist it. In Section 3.7.5, we empirically evaluate Auxo's robustness under these scenarios.

3.6 Implementation

We have implemented Auxo as an independent Python library (1,664 lines) to serve existing CL frameworks (e.g., TFF [4] and PySyft [118]), and integrated it with FedScale [85] for evaluations. Auxo abstracts away the cohort identification and partition so that CL developers can easily try out their CL algorithms or datasets on top of Auxo without any modifications.

Auxo's implementation consists of the three components described in Section 4.3: The cohort coordinator manages and spawns cohort processes, which initiate CL training tasks. Clients continuously submit their training requests based on their availability and affinity records. Then, the cohort coordinator takes client training requests as input and forwards the requests to corresponding cohorts. Each cohort process conducts conventional CL training with the assigned available clients independently. At the end of each individual round, the Auxo clustering algorithm runs within every cohort and reports clustering results to each participant over the network. All training metadata and model weights are checkpointed periodically for fault tolerance. Meanwhile, the cohort coordinator continuously monitors the progress of cohorts for resource management and failure recovery.

Dataset	#Clients	#Samples
Google Speech [158]	2,618	105K
FEMNIST [34]	3,400	640K
OpenImage-Easy	10,133	1M
OpenImage [1]	13,771	1.3M
Amazon Review [77]	42,031	2M
Reddit [2]	63,058	5M

Table 3.2: Statistics of the six datasets in evaluation.

Task	Dataset	Model	Target Acc.	Speedup	Acc. Impr.
Image Classification	FEMNIST	ResNet-18	82.2%	1.2×	7.3%
	OpenImg	MobileNet	56.5%	1.3×	4.8%
		ShuffleNet	58.2%	2.2×	5.0%
	OpenImg-Easy	MobileNet	65.4%	1.4×	3.4%
		ShuffleNet	64.8%	1.2×	4.4%
Language Modeling	Amazon Review	LR.	65.3%	1.2×	8.2%
	Reddit	Albert	7 ppl	1×	0 ppl
Speech Recognition	Google Speech	ResNet-34	78.5%	1.5×	5.7%

Table 3.3: Summary of improvements of Auxo on time to accuracy and final accuracy. We target the highest accuracy attainable by YoGi.

3.7 Evaluation

We evaluate Auxo's effectiveness for six different ML tasks as well as different choices of CL algorithms. Our evaluation shows the following key highlights:

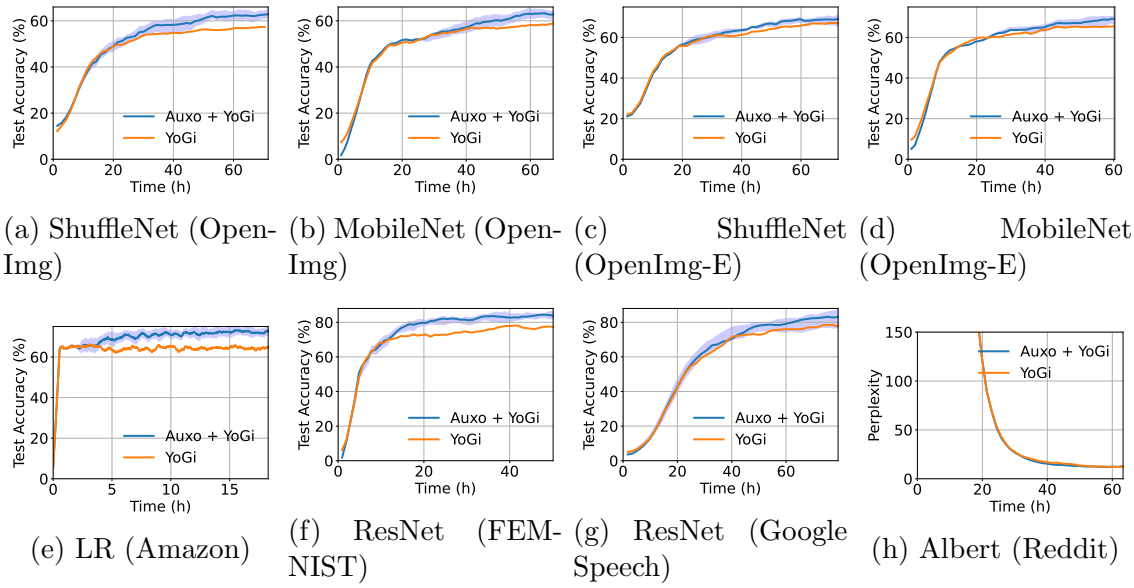
1. Auxo speeds up model convergence on different CL datasets up to 2.2×, while improving final test accuracy by 3.4%-8.2%. Auxo cooperates with existing CL efforts (e.g., personalization) and boosts final test accuracy by 2.1%–6.7%. Auxo can mitigate model bias across devices by 4.8% and 53.8% and improve resource efficiency (§3.7.2).

2. Auxo outperforms existing clustered CL solutions up to 4.8× in time and 5.2× in resources (§3.7.3).

3. Auxo performs well across a broad range of its parameter settings (§3.7.4).

3.7.1 Experiment Setup

Evaluation environment We use 24 NVIDIA Tesla P100 GPUs on CloudLab [41] to emulate the large-scale client training in our evaluations. The client data distribution follows



(a) ShuffleNet (OpenImg) (b) MobileNet (OpenImg) (c) ShuffleNet (OpenImg-E) (d) MobileNet (OpenImg-E)

(e) LR (Amazon) (f) ResNet (FEMNIST) (g) ResNet (Google Speech) (h) Albert (Reddit)

Figure 3.9: Time-to-Accuracy performance on different dataset. For the language modeling (LM) task, a lower perplexity is better. The solid line reflects the average test accuracy. The shaded portion covers the test accuracy performance among all cohorts generated by Auxo.

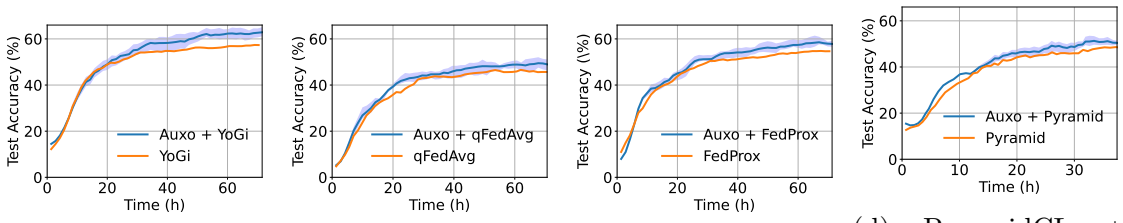
the real-world partition, where client data can vary in quantities, data, and label distribution. We use the open-source benchmark FedScale [85] with standardized setup including realistic device capacity, data, and client availability traces. We report the simulated wall clock time by relying on these realistic CL system and data traces.

Datasets and models We run three categories of applications with six CL datasets [85] of different scale factors using real-world partitions, whose statistics are reported in Table 3.2. The clients for all datasets can check-in with Auxo multiple times following the availability trace.

1. *Speech Recognition*: We train Resnet-34 [58] on a small-scale Google Speech dataset with 35 commands.

2. *Image Classification*: We train Resnet-18 on small-scale FEMNIST with 62 handwritten digits to classify. Also, we train ShuffleNet [177] and MobileNet [136] on middle-scale OpenImage with 596 classes to classify, whereas OpenImage-Easy only has 60 classes.

3. *Language Modeling*: We train logistic regression (LR) on middle-scale Amazon Review for ratings prediction, and Albert model [88] on large Reddit for word prediction.



(a) Yogi (b) q-Fedavg (c) FedProx (d) PyramidCL + YoGi

Figure 3.10: Auxo with different CL algorithms.

These applications are widely used in real end-device applications [165], and these models are lightweight.

Parameters We follow the standardized experiment and parameter settings in FedScale. We adopt an over-commitment strategy to mitigate stragglers which allow 25% failures or stragglers every round as in real CL deployments [23]. We set the number of participants per round to be 200, the local minibatch size to be 6, and the initial learning rate to be 4e-5 for the Albert model, and 0.05 for other models. And we use the linear scaling rule [51] to scale the learning rate.

Metrics The *time-to-accuracy* performance, *final test accuracy*, and *model bias* are our key metrics. We use the cohort member's test data, which follows the realistic data partition, to evaluate each cohort model. The test data would be the global test data if we end up with one global model. For each experiment, we report the average top-1 accuracy based on the results over 3 runs.

3.7.2 End-to-End Performance

Auxo's performance on different datasets. We first evaluate Auxo's performance on different real-world CL datasets. In the following experiments, we adopt YoGi as the CL algorithm because it outperforms other CL algorithms most of the time. Table 3.3 summarizes the key time-to-accuracy performance of all datasets, where we tease apart the overall improvement with statistical and system ones. We quantify the time-to-accuracy as speedup by Auxo, which measures how many times Auxo can speed up to achieve the target accuracy compared to the baseline time cost. Figure 3.9 reports the timeline of training to achieve different accuracy, as different cohorts perform CL asynchronously. The shaded portion covers the test accuracy among all cohorts generated by Auxo.

We notice that Auxo speeds up the wall-clock time to reach target accuracy up to 2.2× faster. Moreover, the final accuracy for different datasets is improved by 3.4%–8.2%. The benefit of Auxo varies over datasets. For most of the datasets, Auxo can achieve significant final accuracy improvement. Nevertheless, Auxo does not improve Reddit task because the clients' texting behavior is similar to each other that makes it hard to identify significant groups as shown in Figure 3.3. Hence, Auxo decides not to partition into multiple cohorts to maximize the benefit of clustering in CL training.

Auxo's performance on different CL algorithms. We then evaluate Auxo's performance on ShuffleNet-OpenImage with different CL Algorithms, which are complementary to Auxo. We refer to YoGi running atop Auxo as YoGi+Auxo, and similarly for FedProx, q-FedAvg and PyramidCL+YoGi.

As shown in Figure 3.10, Auxo speeds up the time to reach the target accuracy of baseline algorithms, from 1.2× to 2.2× faster and improve the final test accuracy by 3%–6.8%.

As for the personalization algorithm FTFA, we adopt the cohort models generated by Auxo to conduct local training using FTFA on corresponding cohort members. In addition to faster convergence of the initial model, Auxo also improves the average test accuracy of FTFA from 63.18% to 67.40% with local fine-tuning.

Auxo's benefit on resource efficiency. We finally show that Auxo can optimize resource efficiency on OpenImg dataset by saving 55% training resources. We also account for the affinity maintaining overhead into the client resource usage, which is around 0.02% of the total resource consumption.

Auxo's benefit on model bias. We show that Auxo can also mitigate the model bias due to smaller intra-cohort statistical heterogeneity. We report the variance of the final accuracy distributions, the worst and best 10% test accuracy in Table 3.4. Our experiment show that the variance of test accuracy is decreased for all the datasets by 4.8% and 53.8%.

3.7.3 Clustered CL Comparison

We compare Auxo with four existing clustered CL algorithms CCL, CL+HC, FlexCCL, and IFCA in terms of three metrics: time-to-accuracy, resource-to-accuracy, and final accuracy. Since these algorithms do not meet some real-world CL constraints (Table 3.1), we simplify the settings accordingly.

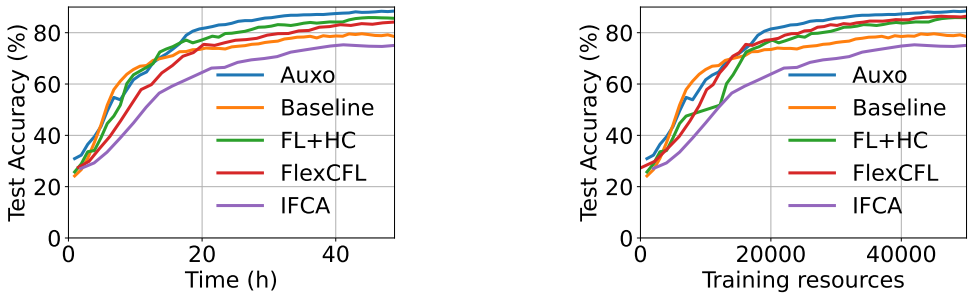
We compare with CCL in small-scale settings ($\sim$ 100 clients) from the FEMNIST dataset to meet their full participation assumption. We observe little difference between CCL, Auxo,

Dataset	Setting	Worst 10% (%)	Best 10% (%)	Variance
OpenImg	Auxo	38	83	267
	Baseline	34	79	296
OpenImg	Auxo	38	88	234
-Easy	Baseline	33	84	273
Review	Auxo	50	100	460
	Baseline	32	100	995
FEMNIST	Auxo	63	97	171
	Baseline	60	93	185
Speech	Auxo	57	100	479
	Baseline	52	100	503

Table 3.4: Summary of improvements on model bias.

and baseline (i.e., no cohorts) in terms of time and resources used due to the absence of significant clusters within small populations. However, this highlights the need for large-scale CL settings, where CCL cannot even be applied as it does not support partial participation.

To compare with CL+HC, FlexCCL and IFCA, we conduct experiments with the full FEMNIST and Amazon Review datasets *without* the client availability traces to align with their constraints. As shown in Table 3.5 and Figure 3.11, Auxo achieve better time efficiency $1.4 \times -4.8\times$ and better resource efficiency $1.3 \times -4.8\times$ compared to the related works especially for the large-scale Amazon dataset, due to our efficient and scalable algorithm design. Also, our result for IFCA are consistent with Motley [161].



(a) Accuracy-Time

(b) Accuracy-Resource

Figure 3.11: Comparison with clustered CL (FEMNIST).

		CL+HC	FlexCCL	IFCA	Auxo
FEMNIST	Speedup	1.7×	1.3×	0.5×	2.4×
	Efficiency	1.6×	1.8×	0.5×	2.4×
	Final acc.	5.8%	7.1%	1.3%	9.1%
Amazon Review	Speedup	0.4×	0.4×	0.5×	2.3×
	Efficiency	0.4×	0.5×	0.5×	2.1×
	Final acc.	-2.9%	-2.7%	0.6%	5.4%

Table 3.5: Summary of improvements over baseline (i.e., no cohorts) in terms of time, resource and accuracy.

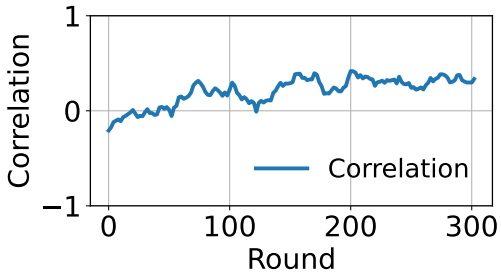


Figure 3.12: Impact of cluster start time.

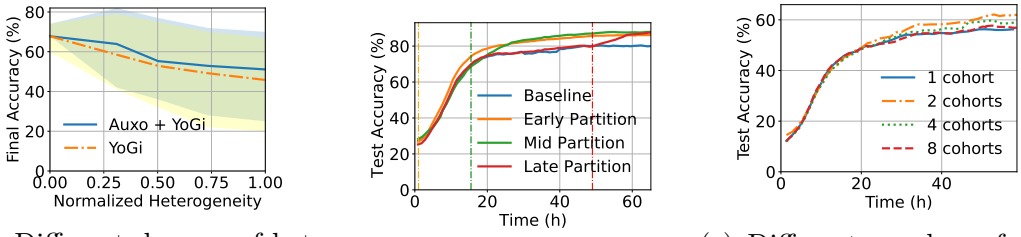
3.7.4 Sensitivity Analysis

Impact of different degrees of heterogeneity. We generate different statistical heterogeneity by applying affine shift [128] on OpenImage, then we evaluate Auxo with YoGi across different degrees of statistical heterogeneity. Figure 3.13a reports the final test accuracy as well as top 10% and worst 10% client test accuracy on different degrees of heterogeneity. We observe that Auxo can improve model accuracy and mitigate model bias under different degrees of heterogeneity. Moreover, similar to the previous experiment, Auxo achieves faster time-to-accuracy performance from 1.2× to 1.8×.

Impact of time to partition. We investigate the impact of different cohort partition times on the model convergence. As mentioned in Section 3.4.4, the partition time relates to the trade-off between model generalizability and intra-cohort heterogeneity. As shown in Figure 3.13b, we choose different partition times with the same cohort composition and report the test accuracy to time performance on FEMNIST. We observe that cohort-based training all outperform the baseline experiment with one cohort. However, early partitions such as FlexCCL and IFCA are worse than intermediate partitions, because it sacrifices the model generalizability. Similarly, late partition after convergence as CCL does not outperform intermediate partition, because it slows down the model convergence to a smaller heterogeneous population.

Impact of time to start clustering. We investigate the impact of the clustering start time on the model convergence. As mentioned in Section 3.4.4, different clustering start time may affect the clustering accuracy and efficiency. To quantify how well the gradient similarity correlates with data similarity, we use Pearson correlation coefficient $r = \frac{\sum_{i=1}^{n}(G_i-\bar{G})(D_i-\bar{D})}{\sqrt{\sum_{i=1}^{n}(G_i-\bar{G})^2 \sum_{i=1}^{n}(D_i-\bar{D})^2}}$, where D and G are pairwise data similarity and gradient similarity. As shown in Figure 3.12, the similarity correlation slightly increases over training rounds, which suggests a slightly later cluster start time.

Impact of the number of cohorts. We investigate the impact of the number of cohorts generated by Auxo, which relates to the trade-off between training resources and intra-cohort heterogeneity (§3.4.4). As shown in Figure 3.13c, we observe that the model convergence is negatively affected once the number of cohorts exceeds 4 under the same resource budget. By further comparing the reduce of heterogeneity with different number of cohorts indicated in Figure 3.3, this result verifies that better model convergence can be achieved as long as the heterogeneity can proportionally compensate the reduced training resources

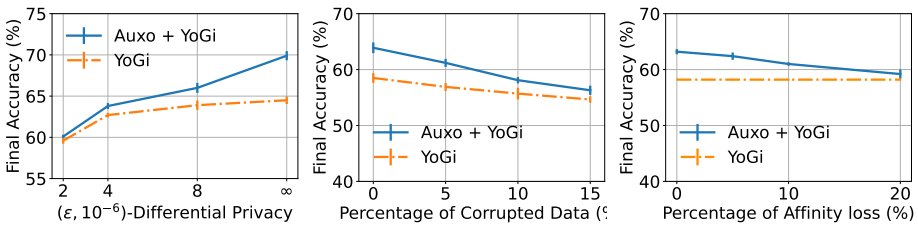


(a) Different degrees of heterogeneity. (b) Different partition time. (c) Different number of cohorts.

Figure 3.13: Sensitivity analysis.

3.7.5 Auxo Resilience

Impact of differential privacy. Auxo is robust to local differential privacy (LDP). LDP is used to protect user-level privacy by adding Gaussian noise to the client update before sending it to the server, but it hurts model accuracy. We evaluate Auxo's performance under LDP for a learning task on the Amazon Review dataset. To achieve (ϵ, δ)-differential privacy, where $\delta = 10^{-6}$ based on the training scale and $\epsilon = 2, 4, 8$, we set the noise scale $\sigma = 1.0, 0.77, 0.6$. As shown in Figure 3.14a, Auxo can still benefit CL training across different differential privacy guarantees.



(a) Local DP. (b) Corrupted client. (c) Client affinity loss.

Figure 3.14: Robustness of Auxo under different scenarios.

Impact of malicious attacks. We investigate the robustness of Auxo by manually involving corrupted clients. Following a popular adversarial ML setting that introduces local model poisoning [47], we randomly flip the ground-truth data labels for these corrupted clients. As shown in Figure 3.14b, we introduce different percentages of corrupted clients to the OpenImg task. We set the percentage of corruption below 15%, which is a practical percentage under the real-world setting [47]. We observe that Auxo still improves performance across different degrees of corruption through identifying malicious clients and eliminating their interference.

Impact of unstable client. Finally, we show Auxo is robust with unstable clients who fail to maintain their affinity records, which may result in less accurate clustering results. We consider loss rates from 0% to 20% and report the corresponding final test accuracy in Figure 3.14c. We notice Auxo outperforms the baseline across different affinity loss rates.

3.8 Related Work

Distributed Machine Learning Distributed ML in data centers has been well-studied [106, 113, 174], where homogeneous data and workers are assumed [53]. With the same training goal, CL raises its unique challenges including the data heterogeneity and system heterogeneity. As a result, Auxo aims at speeding up the training process through directly reducing the intra-cohort heterogeneity at scale.

Collaborative Learning CL is a distributed machine learning paradigm [23, 81] with two key challenges: statistical and system heterogeneity. State-of-the-art CL algorithms try to tackle these two challenges and optimize different targets including model convergence [127, 90, 87, 179, 99, 94], fairness [89, 91], privacy [25, 130, 131, 132], efficiency [109, 55, 164], and robustness [89, 54]. However, they underperform in CL because they do not tackle the root

cause of CL challenges but mitigate the negative effect caused by heterogeneity.

Federated Analytics There has been significant work on geo-distributed data analytics [64, 84, 153, 123]. They mainly optimize the execution latency [86] and resource efficiency [162, 63]. To further preserve privacy for distributed data, Orchard [132] and Honeycrisp [131] have been proposed to enable large-scale differentially private analytics. Helen [182] and Cerebro [181] allow multiple parties to securely train models without revealing their data.

Traditional Clustering Algorithms Clustering algorithms [166, 20] are used in popular data mining techniques, which usually assume access to all data. However, under CL setting, it is non-trivial to design a clustering algorithm because of the unavailability of data. Auxo proposes a clustering algorithm that can be applicable to the CL settings.

CL Client Clustering In order to leverage the nature of clusters in real-world CL dataset, many algorithms have been proposed to identify the clusters among CL clients. However, existing clustered CL solutions [137, 50, 26, 39] mainly suffer from scalability and practicality, which are hard to adapt to large-scale, low-participation, and resource-constraint CL training. Considering all real-world constraints, Auxo build a practical system to identify cohorts and benefit CL training.

3.9 Conclusion

We presented Auxo, which builds on top of the observation that there exist natural groups of statistically similar clients (cohorts) in large real-world CL populations. Auxo identifies cohorts with reduced intra-cohort heterogeneity at scale, addressing heterogeneity-borne CL challenges at their roots. Auxo proposes an efficient algorithm and practical system that can be applied under real-world CL constraints to significantly benefit CL training in terms of model convergence, final accuracy, and model bias.

CHAPTER 4

Optimizing Resource Sharing in Multi-Job Collaborative Learning Environments

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, so does resource contention among multiple CL jobs. The *ephemeral nature and resource heterogeneity*, coupled with the *overlapping resource requirements of diverse CL jobs*, complicate efficient device scheduling. Existing resource managers for CL jobs opt for random assignment of devices to CL jobs for simplicity and scalability, which hurts job efficiency.

In this paper, we present Venn, an CL resource manager, that efficiently schedules contended ephemeral, heterogeneous devices among many CL jobs, with the goal of reducing their average job completion time (JCT). Venn formulates the *Intersection Resource Scheduling (IRS)* problem to identify complex resource contention among multiple CL jobs. Then, Venn proposes a contention-aware scheduling heuristic to minimize the average scheduling delay. Furthermore, it proposes a resource-aware device-to-job matching heuristic that focuses on optimizing 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 availabile at https://github.com/SymbioticLab/Venn.

4.1 Introduction

Collaborative learning (CL) enables distributed edge devices to perform collaborative machine learning (ML) without moving raw data into the cloud [24, 125]. 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 [158], healthcare study [135], next-word

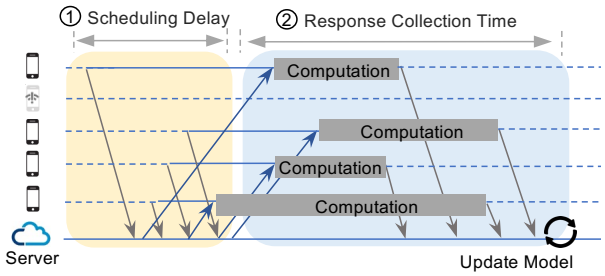


Figure 4.1: Composition of the completion time of one round of an CL job.

prediction [56, 167], emoji prediction [126], and query suggestion on keyboard [170]. Each CL training job in practice often requires 1000∼10000 device participants in each training round and takes 4∼8 days to finish [170]. As the number of CL applications continues to grow, efficient edge resource management has become the key to fast and resource-efficient CL.

In comparison to cloud resources, CL resources are not accessible all the time, they only become available for training when they are connected to WiFi and in charging [24]. Moreover, they are also highly heterogeneous in terms of their hardware capacity, software versions, and training data availability (§4.2.1). Even when running on the same population, different CL jobs often compete for different subsets of devices, based on their specific model characteristics and training objectives. Both of the resource characteristics lead to complex multi-resource contentions among multiple ongoing CL jobs, where the eligible resources for each job are not only limited and but also may *overlap, contain, or be within* those of one or more other jobs.

However, existing CL research primarily focuses on optimizing the resource efficiency within a *single* jobs through client selection [87, 7, 17]. They often ignore the resource contention among other CL jobs in the cluster and solely optimize the response collection time by assuming sufficient resources are always available. In practice, however, multiple CL jobs may run concurrently and compete for a subset of devices, leading to resource contention [24]. Hence, they miss an important factor to the job completion time (JCT) of an CL job as dipicted in Figure 4.1, which is the scheduling delay (§4.2.2). Scheduling delay refers to the time needed to acquire all resources required by the CL job, which is often prolonged due to resource contention.

With the proliferation of CL in production, large corporations such as Apple [120], Meta [66], and Google [24] have developed their own CL infrastructures to coordinate multiple CL jobs at production scale. However, despite low-level differences, these CL resource managers can all be described simply as *random device-to-job matching* in various forms.

We show that such random assignment can lead to higher scheduling delays and response collection times as more and more CL jobs vie for a shared device population.

In this paper, we present Venn, an CL resource manager that minimizes the average JCT of multiple CL jobs competing over a large number of heterogeneous devices. First, Venn addresses the intricate resource contention among CL jobs by formulating it as an *Intersection Resource Scheduling (IRS)* problem (§4.4.2), where the resources that one job contends for may overlap, contain, or be within those resources of other jobs. Venn then introduces a contention-aware scheduling heuristic that prioritizes small jobs requiring scarce resources to minimize the average scheduling delay. Second, Venn employs a resource-aware device-to-job matching heuristic to reduce response collection time (§4.4.3). Together, they jointly optimize the average JCT for a diverse set of 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 (§4.5). Compared to state-of-the-art CL resource allocation solutions [24, 66, 120], Venn improves the average JCT by up to 1.88×.

Overall, we make the following contributions in this paper:

1. We introduce Venn, an 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 scheduling and matching joint solution to optimize both the scheduling delay and response collection time.

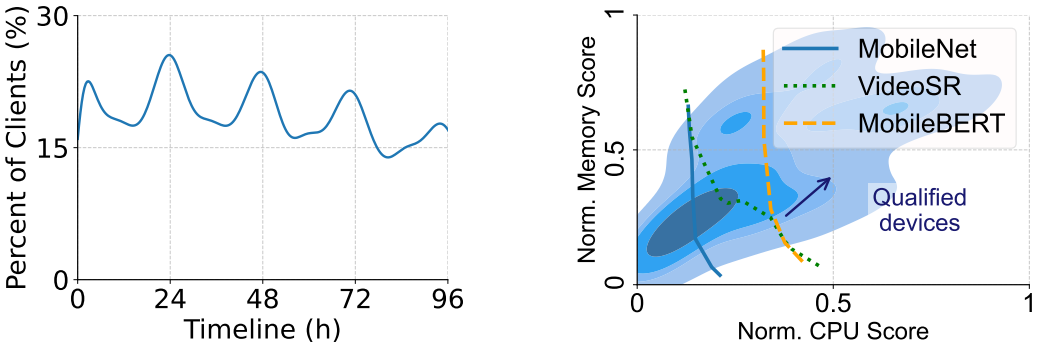
3. We have implemented and evaluated Venn, along with its scheduling and matching algorithms, demonstrating improvements in the average JCT compared to the state-of-the-art across various real-world CL workloads.

4.2 Background and Motivation

4.2.1 Collaborative Learning

CL has been widely adopted by the industry to train ML models on larger and more diverse on-device datasets without having to copy raw data into the cloud. Example applications include healthcare study, speech recognition, next-word prediction, etc.. CL is unique both in terms of resources it uses and jobs that use it, which pose unique challenges for CL resource managers.

mple of three resource s



(a) Diurnal device availability.

(b) Device hardware heterogeneity.

Figure 4.2: CL resources exhibit both high variance in availability and capacity.

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 comes from CL resource unique characteristics in terms of availability and heterogeneity.

Dynamic availability. Unlike cloud resources (e.g., GPUs) that are accessible at most of the time, CL resources only become available for training when certain condtions are met, such as being charged and connected to WiFi We analyze a real-world client availability trace [85], which encompass 180 million trace items of devices behavior over a week. Figure 4.2a shows that the number of available devices (charging and connected to WiFi) changes over time with diurnal pattern. Hence, there is an inherent scheduling delay for an CL job to acquire enough participating devices.

Device heterogeneity. The large number of devices involved in CL are heterogeneous in multiple dimensions including data availability, software version, and hardware capacity, which leads to varying response time among devices. Figure 4.2b showcases this heterogeneity, focusing on variations in memory and CPU capacities among edge devices, based on data from AI Benchmark [68]. It also annotates the minimum hardware requirements needed to execute three popular on-device ML models within a reasonable time. As jobs may have different learning objectives, they may compete for different subsets of devices. When more factors like data availability and software version are considered, these subsets can exhibit relationships that are *inclusive, overlapping, or nested*, leading to complex contention patterns.

All of these resource characteristics make the resource management in CL more challenging by slowing down the training process. Hence, many solutions have been proposed to optimize the training time, more specifically the response collection time, such as client

selection [87, 7, 17], quantization [129], and model aggregation [59].

4.2.2 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 [120] 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 [66] is centralized, where it randomly matches each client with one eligible CL job.

3. Google's CL resource manager [24] is driven by jobs, where each job independently samples from available clients.

Despite the seeming variety in their designs, existing resource managers all boil down to random device-to-job matching in different forms. Unfortunately, in the absence of extreme surplus of qualified devices compared to the demand, random matching fail to address the growing resource contention and yield longer scheduling delay.

4.2.3 Limitation and Opportunities

Limitations of the state-of-the-art. Consider the toy example in Figure 4.3 that compares three scheduling solutions: random matching as specified above, Shortest-Remaining-Service-First (SRSF) proposed for cloud ML scheduling [53], and the optimal solution. We assume all jobs arrive simultaneously and clients with different eligibilities become online constantly over time. Both random matching and SRSF fail to account for resource scarcity and contention, thereby allocating limited resources (i.e., Emoji clients) to jobs (i.e., the Keyboard job) that already have an abundance of resources, resulting in an average JCT of 12 and 11 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 Most of existing CL solutions that focus on single-job optimizations, such as client selection [87, 7, 17], often rely on the assumption that CL jobs have access to sufficient number of online devices. 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 [24]. 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 4.4. In this experiment, the resource pool is evenly partitioned and managed by each job, who aims to train a ResNet-18 model for the FEMNIST dataset [34]. We vary the number of concurrently running jobs to observe its impact. 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 partitioned 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 [129] and client selection [59], predominantly focus on optimizing *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 4.1.

To offer a holistic view of JCT, we dissect its components for a single round request, under varying degrees of resource contention, as shown in Figure 4.5. Utilizing the same experimental setup as in Figure 4.4, 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 4.5 indicate, scheduling delay can significantly impact overall JCT, especially when resource supply falls short of demand.

4.3 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 4.6 provides an overview of Venn's workflow, along with its role in the CL job's lifecycle.

In each execution round, an CL job submits resource requests to Venn, specifying its device requirements and associated resource demands (⓪). 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 (②), until the job's demand is satisfied. 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 computation [103, 36] (③ and ④). Finally, the device may either report the training result to the corresponding job upon completion, or drop off mid-process due to availability dynamics (⑤).

Note that Steps ③ through ⑤ adhere to conventional CL protocols between individual

CL jobs and devices. Venn's primary role is in optimizing the job-to-device assigning phase, denoted by ②. We elaborate more detailed division of tasks in Appendix .

4.4 Resource Scheduling in Venn

In this section, we first introduce the problem statement (§4.4.1). We then develop our scheduling algorithm in two steps: establishing the job scheduling order to minimize the scheduling delay (§4.4.2) and determining the device-to-job matching to co-optimize the response collection time (§4.4.3). Then, we discuss additional enhancements for real-world deployments (§4.4.4).

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 [45]. 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. We now formally define such resource scheduling problem.

Problem Definition. Assume we have m jobs $\mathbb{J} = \{J_1, J_2, ..., J_m\}$ with their resource demands $\mathbb{D} = \{D_1, D_2, ..., D_m\}$. Let $\mathbb{S} = S_1 \cup S_2 \cup ... \cup S_n$ be the available device set, where S_k is the eligible device subset in $\mathbb{S}$ that satisfies the device specification of job J_i, i.e., $f(J_i) = S_k$. 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. Jointly optimizing scheduling delay and response collection time is non-trivial. Intuitively, to reduce the average scheduling delay, it is desirable to promptly assign each device upon check-in to one eligible job. On the other hand, since the response collection time is usually determined by the final

responding device among the target number of participants, it can be minimized by mitigating stragglers and cherry picking ideal devices to jobs. However, such an approach takes a longer time to acquire a certain number of qualified devices, leading to longer scheduling 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 to decide the CL job scheduling order in order to minimize the average scheduling delay? (§4.4.2)*

2. *How to match devices with CL jobs in order to minimize average response collection time while also reducing average JCT? (§4.4.3)*

Algorithm 3 Intersection Resource Scheduling

Inputs: Job Groups $\mathbb{G}$, Devices $\mathbb{S}$
Output: Resource allocation for each job group
for all G_j in $\mathbb{G}$ **do**
    Sort J_i by D_i in ascending order, $\forall J_i \in G_j$ ▷ Sort within job group
$S = \cup_{j=1}^{n} S_j$
Sort G_j by $|S_j|$ in ascending order, $\forall G_j \in \mathbb{G}$ ▷ Generate initial allocation
for all G_j in $\mathbb{G}$ **do**
    $S'_j = S \cap S_j, S = S \setminus S'_j$
Sort G_j by $|S_j|$ in descending order, $\forall G_j \in \mathbb{G}$ ▷ Allocate resource
for all G_j in $\mathbb{G}$ **do**
    if $|S'_j| > 0$ **then**
        for all $G_k \in \mathbb{G} : |S_k| < |S_j|, S_k \cap S_j \neq \emptyset$ **do**
            $m'_j, m'_k =$ `get-queue-len`()
            if $\frac{m'_j}{|S'_j|} > \frac{m'_k}{|S_k|}$ **then**
                $S'_j = S'_j \cup (S_j \cap S_k)$
                $S'_k = S'_k - S'_j$
            else
                break
return $\{G_j[0], S'_j\}, \forall j \in [1, n]$

4.4.2 Intersection Resource Scheduling (IRS)

We now tackle the first question, minimizing the average 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 to optimally allocate resources to minimize scheduling delay (Appendix C.2) 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 requirements, 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 3: (i) Determining the job order **within a job group** to optimize local resource scheduling (§4.4.2.1). (ii) Then deciding how to merge the job order **across job groups** to ensure global scheduling efficiency (§4.4.2.2). Venn invokes Algorithm 3 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.3.

4.4.2.1 Intra Job Group Scheduling

Within each job group that shares the same device specifications, Venn prioritizes jobs based on their remaining resource demand (Algorithm 3 line 2). 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 [138]. 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.

4.4.2.2 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. At a high-level, it should prioritize jobs with scarce resources, i.e., those with stringent device requirements and limited eligible resource options, to avoid being blocked by jobs with more abundant resources. 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 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 3 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 job group with most scarce resources (line 4). This results in an initial allocation plan with no resource sharing across job groups (lines 3–6), setting the stage for subsequent cross-group allocations.

To determine how to allocate intersected resource across job groups, Venn greedily evaluates whether a job group with more abundant resources should acquire intersected resources from groups with scarcer resources with the objective of minimizing the average scheduling delay. This evaluation starts with the job group possessing the most abundant resources (line 7). If the resources allocated to this group remain unclaimed by other groups (line 9), 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 12). 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 13–14). Otherwise, the algo-

rithm ceases to allocate additional resources to the current job group from remaining groups (line 16). 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 11) 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 13–14).

The time complexity of Algorithm 3 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 C.4 to illustrate the effectiveness of Venn's approach.

4.4.3 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 4.4.1, there is 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 4.4.2.

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 4.

The matching algorithm is activated only for jobs that are currently served, as scheduled by Algorithm 3. For each 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

Algorithm 4 Device Matching

Input: Jobs J_i, Devices $S'_j \in Venn - SCHED(\mathbb{G}, \mathbb{S})$
Output: Matched jobs and devices

1 $S = \{S^1, S^2, ..., S^V\}$ ▷ Evenly partition devices
2 $g_v = \frac{t^v}{t^0}, \forall v \in [1, V]$ ▷ Response time speed-up
3 $c_i = \frac{t_{response}}{t_{schedule}}$ ▷ Assign tiers in a rotating manner
4 $u = \texttt{randint}(0, V)$
5 **if** $V + g_u \times c_i < c_i + 1$ **then**
6 Update $S'_j = S'_j \cap S^u$ ▷ Assign tier u devices to J_i
7 **return** $\{J_i, S'_j\}$

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 4 line 4). 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 4.7, 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 5). 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 [156]. 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 2).

4.4.4 Enhancements

Dynamic Resource Supply As shown in Figure 4.2a, 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 4.2a. Given that CL jobs often span multiple days and resource availability typically follows a diurnal pattern, a more accurate approach is to use the average eligible resource rate over a 24-hour period as a representative metric for each job's eligible resources. 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 4.4.2. 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 (§4.5.5).

4.5 Evaluation

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

- Venn speeds up the average JCT by up to 1.88X without affecting the model accuracy, compared to the state-of-the-art across various real-world CL workloads (4.5.2).
- Venn outperforms its counterparts through intelligent job scheduling and device-to-job matching using different design components (4.5.3).

- Venn 's benefits are robust under a wide range of CL workloads and environment setups (4.5.5).

4.5.1 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.

CL Resources: To faithfully emulate heterogeneous device runtimes, network throughput, and availability, we use device traces from FedScale [85] and AI Benchmark [68], as depicted in Figure 4.2 and Figure 4.8a. Each unique device trace is limited to one CL job per day for realism.

CL Jobs: Our focus is on synchronous CL jobs [170], 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. To assess the generalizability of Venn, we curate a diverse set of CL jobs drawn from diverse applications [167, 170, 56, 126, 158, 135], whose resource demand is depicted in Figure 4.8b. In the real CL experiment, each job aims to train a ResNet-18 [58] and MobileNet-V2 [136] on FEMNIST dataset.

Workloads: Our evaluation includes five workload scenarios that sample differently from the job trace in Figure 4.8b 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 4.8a) to create various resource contention pattern 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 4.1: Summary of improvements on average JCT over random matching on different CL workloads.

Baselines We compare Venn with FIFO, SRSF, and an optimized random matching. Both Venn and SRSF are agnostic to the total CL job rounds. Random matching algorithm is supposed to match one eligible job to each device at random. However, to reduce its round abortion rate under contention, we optimize it to schedule jobs in a randomized order, thereby setting a more challenging baseline. Note that we only run Venn with the starvation prevention strategy in Section 4.5.5.

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.

4.5.2 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 algorithms compared to the random scheduling in Table 4.1. 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 4.9, we observe 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 4.10 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 can be attributed to its time complexity $\max(O(m \log m), O(n^2))$, where m, n are the numbers of ongoing jobs and job groups,

	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 4.2: 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 4.3: 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.

respectively.

4.5.3 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 A.3, 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 4.8a), and then analyzed the average JCT improvement for each type. Table 4.2 and Table 4.3 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

	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 4.4: Average JCT improvement on four biased workloads.

demands and scarcer resources benefit the most from Venn.

4.5.4 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 4.8b. The outcomes of these experiments are systematically presented in Table 4.4, offering insights into the algorithm's performance under each workload scenario.

4.5.5 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 4.12, 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 4.13 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 4.14a. The results demonstrate that, as ϵ increases, the JCT speed-up tends to decrease. As shown in Figure 4.14b, 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.

4.6 Related Works

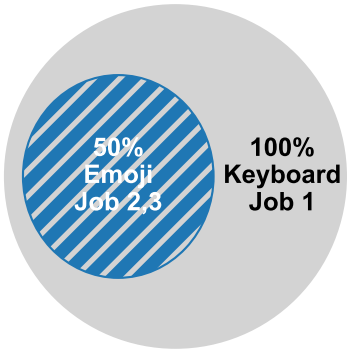
Cluster Resource Manager. There are many cluster resource managers that schedule resources with constraints [151, 49, 114]. Existing ML cluster resource managers mainly focus on managing the stable data-center resources like GPU and CPU [163, 112] in order to improve JCT, utilization and fairness [122, 30]. Some research delves into GPU-specific optimizations [53, 173, 67], while others co-design resource managers with the specific characteristics of ML workloads [124, 171]. 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 Selector. 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 [87, 7, 17, 69, 29, 59] to speed up the training. However, they solely focus on response collection time [95, 96] 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 Manager. 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.

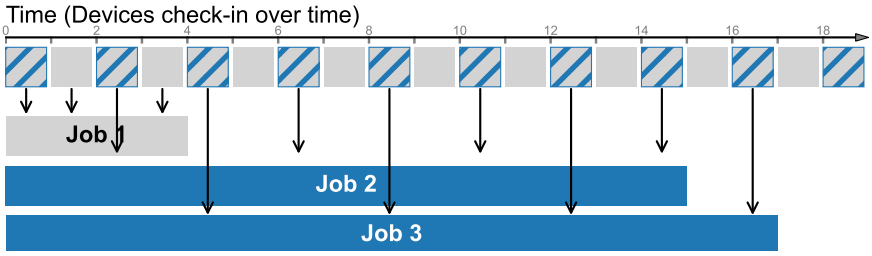
4.7 Conclusion

In this paper, we introduce our CL resource manager, Venn, to enable efficient sharing of a large amount of heterogeneous device resources among multiple CL jobs with diverse requirements. Venn incorporates a contention-aware job scheduling algorithm and a resource-aware device-to-job matching algorithm to minimize the average JCT for CL jobs. Our evaluation over various real-world CL workloads shows that Venn achieves up to 1.88X improvement on average JCT.

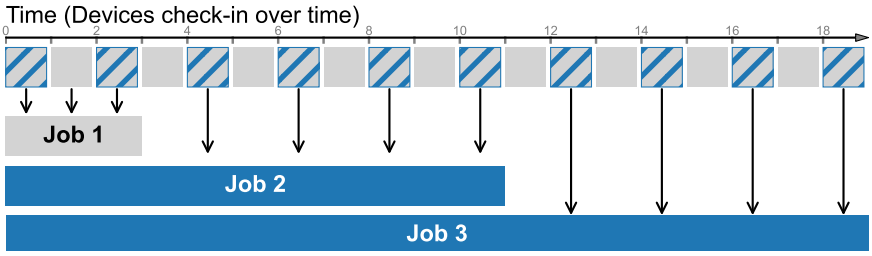


Job ID	Training Data	# Devices
1	Keyboard	3
2	Emoji	4
3	Emoji	4

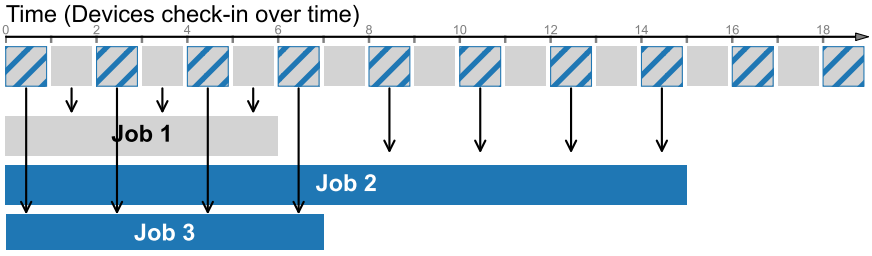
(a) Setup.



(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 4.3: Toy example of three resource schedules across 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. Conversely, the optimal schedule smartly allocates these scarce resources to Jobs 2 followed by Job 3, thereby minimizing average job completion time.

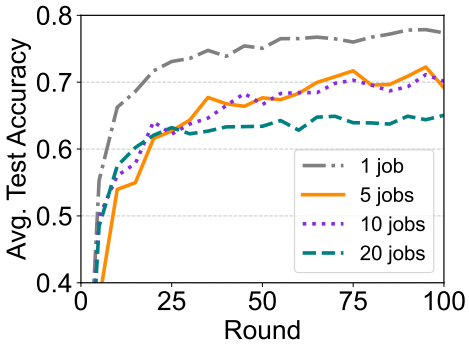


Figure 4.4: Impact of resource contention.

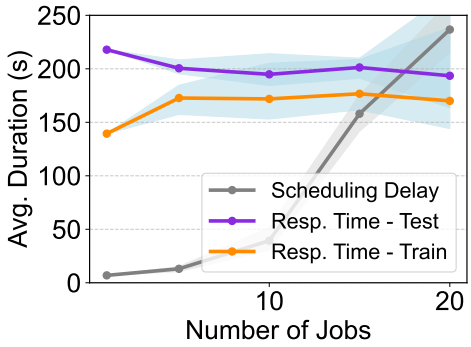


Figure 4.5: JCT breakdown in a single round.

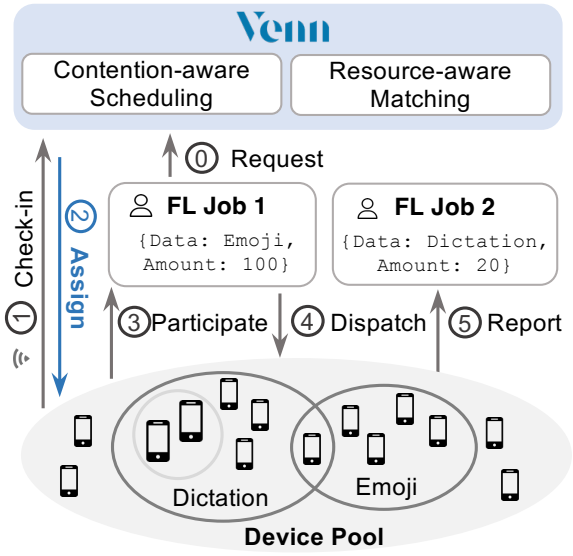


Figure 4.6: Venn System Overview.

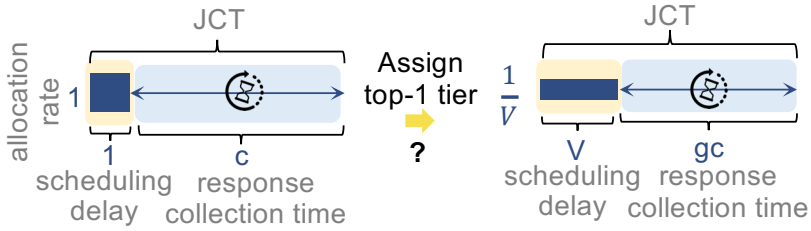
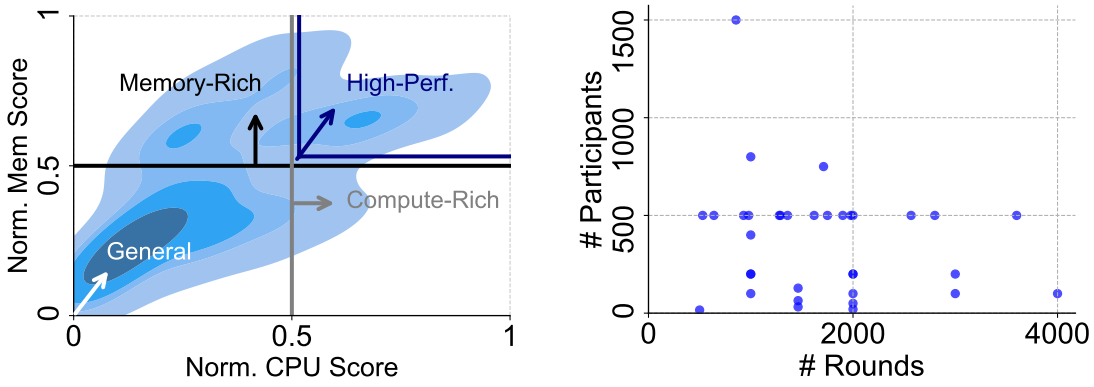


Figure 4.7: Visualize tier-based device-to-job matching condition.



(a) Device eligibility trace.

(b) CL job demand trace.

Figure 4.8: Device and job trace used in experiments. (a) Device 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.

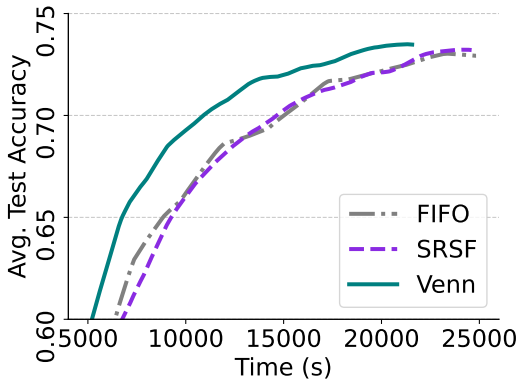


Figure 4.9: Venn does not affect the average test accuracy.

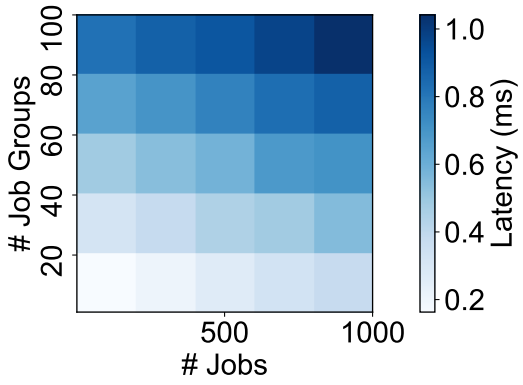
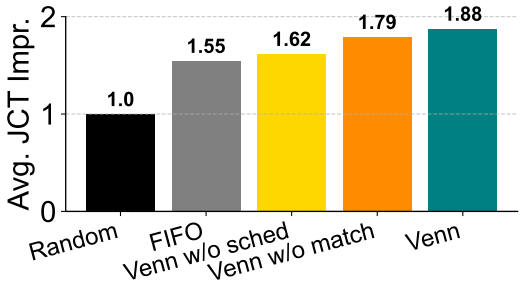
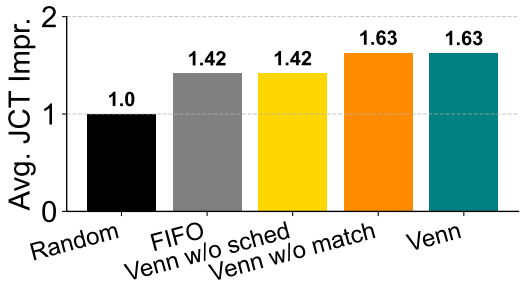


Figure 4.10: Venn introduces negligible overhead at scale.



(a) Low workload.



(b) High workload.

Figure 4.11: Average JCT improvement breakdown.

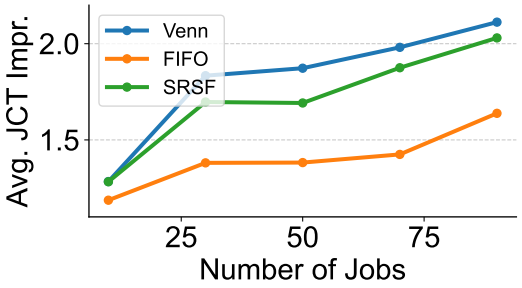


Figure 4.12: Venn outperforms FIFO and SRSF across different numbers of jobs.

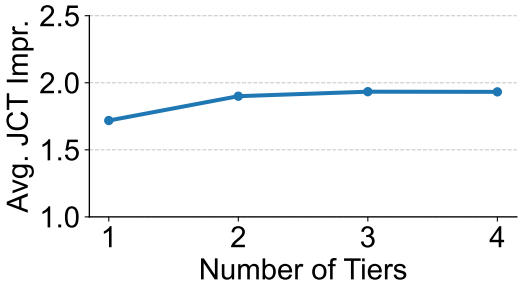
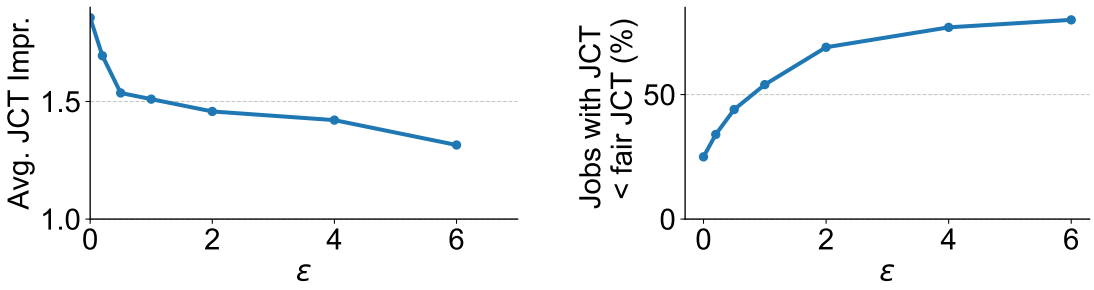


Figure 4.13: Venn's improvement across different numbers of tiers.



(a) Venn's improvement over different ϵ.

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

Figure 4.14: Fairness knob.

CHAPTER 5

Conclusion

As ML transitions from a specialized technology to an integral part of daily life for individuals everywhere, we need to rethink: How should we architect the new generation of ML systems to meet the evolving needs of pervasive AI? This dissertation proposes a solution for developing user-centric ML systems [173, 98, 97, 78, 80, 96, 85, 184, 79, 154] through sophisticated resource management, aiming to enhance both end-user experience and server-side efficiency. This approach goes beyond the singular pursuit of system efficiency to a holistic paradigm of user-centric system design.

To realize the vision towards pervasive AI, we need to address the challenge including the diversity of AI workloads—from conversational AI to large-scale training and collaborative learning—each with unique system demands. Compounding this is the heterogeneity of computing resources, spanning powerful cloud data centers to resource-constrained edge devices, and varying user data distributions. My research addresses these challenges by designing systems grounded in the core principles of user-centricity, workload-awareness, and synergistic server-client co-design.

This concluding chapter will first offer my perspectives on the overarching trends of AI and the crucial lessons learned for ML system design (§5.1). Subsequently, I will discuss the tangible impact of the research presented in this dissertation (§5.2). Finally, I will outline promising avenues for future work to continue advancing the field of user-centric machine learning systems (§5.3).

5.1 My Thoughts

5.1.1 The Trend of Pervasive AI

Over the past five years, we have witnessed the rapid expansion of AI from a technology confined to specialized domains within large corporations and research labs to a tool that is increasingly integrated into the everyday life of individuals. This shift, accelerated by

advancements in generative AI models such as ChatGPT, signifies a clear trend: AI is moving towards ubiquity, poised to become as commonplace in our daily existence as essential utilities like water, electricity, and the internet. With this trend, it necessitates a rethinking of its role in people's life and, consequently, how we design the underlying systems to support this integration.

Looking forward, the medium through which AI delivers its value to humans will likely evolve beyond current chat-based interfaces where users proactively submit requests. Two prominent future mediums emerge.

1. **Off-body AI:** Physical robots and virtual AI agents designed to proactively complete complex tasks in the physical or digital world in an autonomous and goal-oriented manner. Their functionalities extend beyond simple command execution to complex reasoning, planning, and execution across diverse domains. Imagine an advanced cleaning robot with the ability to understand spoken commands, sophisticated vision to identify objects and a dexterous arm to manipulate items.

2. **On-body AI:** Augmented reality (AR) and virtual reality (VR) running on wearable devices such as smart glasses. This form of AI would function more like an ever-present, invisible daily assistant, subconsciously learning user habits and contexts to offer timely suggestions and support. For instance, in a professional setting, it could understand an individual's tasks and provide context for upcoming meetings, summarize discussions, or even act as a co-pilot in various professions.

To enable these sophisticated off-body and on-body AI experiences, a confluence of advanced technologies is necessary. These include advanced LLMs for natural language understanding and high-level planning, multimodal models for perception (e.g., vision, audio processing), reinforcement learning for dynamic decision-making in complex environments, and sophisticated control systems for robotic manipulation.

These evolving interaction paradigms not only underscore the future where AI is deeply and proactively embedded in our daily activities but also directly inspire the future research directions in ML system design to support such pervasive AI, where I will discuss the underlying challenges and opportunities in §5.3.

5.1.2 Lessons Learned for ML Systems

The evolving landscape of ML underscores the critical role of ML systems, which are essential for enabling these emerging AI applications. Throughout my research journey, I continually reflect on the current ML systems landscape, asking:

- *What kind of ML systems research deserve us to dive deeper?*
- *How do we find promising ML systems problems to solve?*
- *How should we approach these problems?*

Below, I share my thoughts to the questions, hoping they are helpful for future researchers.

Conceptual Stages of ML Systems Innovation. Through my reflection of numerous ML systems projects, I've found three conceptual levels of research:

1. **0→1:** This represents the most pioneering research, often best pursued in academic settings. It involves addressing entirely new ML use cases, applications, or workloads for which no existing system adequately caters to their unique objectives, resource constraints, or workload characteristics. The novelty lies in being the first to identify these distinct system design requirements and to build a foundational solution. For instance, my work on *Venn* [97], *Curie* [78] and *Exp-Bench* [80] exemplify 0→1 research by identifying and addressing previously unarticulated system needs for emerging ML paradigms.

2. **1→2:** Once a foundational ($0 \rightarrow 1$) system or concept exists, the next critical step is to enhance its practicality and efficiency. This phase focuses on making the system scalable, robust, and more performant. It's about transforming an initial proof-of-concept into something that is not just usable but also efficient under realistic conditions. This stage of research is also valuable and well-suited for doctoral exploration, as it involves deep thinking and understanding of emerging ML workloads along with underlying resources to bridge the gap between novel concepts and practical applications. My contributions such as *Andes* [98], *Fluid* [173], and *FedScale* [85] fall into this category, building upon foundational ideas to deliver more performant systems capable of handling realistic workloads, and running with advanced resource scheduler and scaling to heterogeneous resources.

3. **2→ ∞:** This level involves the intensive effort required to make an ML system truly viable for large-scale, real-world industrial deployment. The primary focus here is on aggressively optimizing efficiency—cutting operational costs, maximizing resource utilization, and minimizing latency or training time. While critically important for widespread adoption, this phase often demands substantial engineering resources and domain-specific expertise, making it particularly well-suited for industry teams who possess the scale and focused incentives for such deep optimization. This distinction

is perhaps particularly salient in ML systems, where academia and industry often collaborate or drive parallel innovations, especially with the rapid, real-world impact seen in fields like Generative AI. My projects like *Auxo* [96] and *FedTrans* [184] further improve the system and model efficiency via techniques like customized ML model training based on the underlying compute resources and data characteristics.

Think ahead. Given the rapid pace of innovation in AI, it is crucial to think ahead and proactively position your research to address future needs. While ML systems play an indispensable role in enabling ML advancements—and can sometimes even drive new ML capabilities, akin to the 'hardware lottery' concept [61]—they often serve a supporting role, following the advancements of ML models and algorithms. In fast-moving and competitive areas like Generative AI, where new ideas can quickly reshape the landscape, a reactive approach can leave research feeling disempowered. Therefore, it's vital for systems researchers not only to address current popular ML challenges but also to explore ML technologies and use cases that are likely to emerge and become significant *in the next three to five years*, or even further out. This foresight, as demonstrated by my pivot towards supporting Generative AI during my PhD, can lead to more impactful and enduring research contributions. Proactively seeking and integrating insights from industry trends, where possible, can further sharpen this forward-looking perspective.

Specialized system design. As AI continues to diversify, we encounter an expanding array of ML use cases, each possessing unique objectives and resource characteristics [173, 97, 98]. These specialized demands present both unique challenges and significant opportunities for innovation. Therefore, when designing ML systems, it is often necessary to think from first principles—to challenge existing assumptions and identify the fundamental building blocks required for a given workload. For instance, my work on Fluid [173] highlighted the unique requirements of experimental model training, where the primary objective shifts from minimizing job completion time to optimizing makespan, as comprehensive evaluation across numerous training jobs is needed to identify optimal configurations. Similarly, Andes [98] was designed specifically for the emerging demands of AI conversational services, focusing on user-perceived Quality-of-Experience (QoE) rather than traditional system metrics such as request throughput. We discuss more challenges and opportunities in Section 5.3.1.

5.2 Impact

My research has aimed to design machine learning systems that enhance user-centricity and system efficiency. The impact of this work can be seen in immediate improvements to current technologies and in laying the groundwork for future advancements.

Elevating User Experience in Generative AI Systems. The way we interact with AI is undergoing a profound transformation. Before 2025, all LLMs chatbot interfaces presented text through an inconsistent token stream, often testing user patience with unevenly paced outputs. My research directly addressed this by aligning system design with user experience to ensure smooth token delivery at the user perspective, while also minimizing GPU resource usage particularly under peak load conditions. Therefore, right after 2025, it is gratifying to observe that a more considered, user-centric approach to token streaming has now become a widely adopted standard. While such advancements are invariably collaborative, I am pleased that my work contributed to raising awareness of user experience in Generative AI systems and making today's generative AI interfaces significantly more user-friendly and cost efficient.

Advancing Large-Scale Foundation Model Training During my internship at Meta, I tackled the challenges inherent in training increasingly massive foundation models. As the model training scaled to hundreds of thousands of GPUs (the exact number remaining confidential), issues like fault-tolerance, stragglers, and communication overhead became critical bottlenecks. I contributed by developing an asynchronous training paradigm and system. Building on this foundation, a broader team at Meta has worked to productionize my solution. This work has helped prepare Meta for even larger-scale training scale for future Llama models, which eventually benefit the broader open-source AI community.

Pioneering Future Directions in Private Machine Learning. My work in private machine learning is a crucial future investment, addressing persistent data isolation challenges in sensitive domains like healthcare and banking. Though practical adoption faces legacy hurdles and requires incentives for collaborative data sharing without compromising user privacy, my research in this area aims to provide solutions and inspire future work when the ecosystem matures. Auxo [96], Venn [97], FedScale [85] and FedTrans [184] realize this vision by providing practical, efficient, and scalable solutions for collaborative learning that address critical challenges in data heterogeneity, resource management, personalization, and multi-job coordination, thereby paving the way for more widespread adoption of privacy-preserving machine learning techniques.

5.3 Future Work

Looking ahead, the principles of user-centricity for pervasive AI (§5.1) and the lessons learned for ML systems research (§5.1.2) guide my vision for future work. The following directions focus on long-term opportunities that align with the anticipated trends in pervasive ML, primarily emphasizing foundational (0→1) and practicalization (1→2) research to pioneer and solidify the next generation of user-centric ML systems.

5.3.1 Advancing User-Centricity and Quality of Experience (QoE) Across Diverse Modalities

Inspired by the vision of pervasive AI (both off-body and on-body AI) and the proliferation of multimodal generative models, a primary direction for future work is to extend the principles of user-centric system design with the concept of Quality of Experience (QoE) as explored in Andes [98], to a broader spectrum of ML applications and modalities. As AI-driven generation and understanding of images, audio, and video become increasingly integrated into daily life, it is crucial to pivot system design objectives to prioritize the user's direct experience with these rich media.

Defining and Optimizing Modality-Specific QoE Metrics. Quantifying user experience for non-textual modalities requires thoughtful design. Future work must first formulate QoE metrics that align closely with the specific goals of underlying ML applications and the nuanced expectations of users. (1) *On the content quality side*, this includes the perceptual quality of generated images, the temporal coherence and narrative consistency of generated video, the fidelity of audio synthesis, and the accuracy and relevance of video understanding outputs—all from a user's perspective. (2) *On the system efficiency side*, this translates to optimizing for interactive responsiveness (e.g., time-to-first-token/image/frame, content delivery timeline), consistent content delivery across different modalities or transitions between them, overall system responsiveness under varying loads, and resource efficiency (e.g., energy, compute) in meeting target QoE levels.

Based on the defined QoE metrics, systems should ideally adapt QoE goals based on individual user preferences (e.g., tolerance for artifacts versus speed), task context (e.g., rapid prototyping versus final production), or even user expertise level. This necessitates research into adaptive scheduling algorithms that can dynamically adjust system behavior and resource allocation to meet these personalized QoE targets.

Resource Management for Dynamic Generative Workloads. The computational demands of generative models, especially during interactive use, can be highly variable and unpredictable. For instance, an on-body AI assisting with visual tasks typically consumes minimal resources during passive observation. However, its resource demand can spike dramatically when the user poses a complex query (e.g., 'Summarize the key activities in this busy street market') about a dynamic and intricate environment. The complexity of both the environment (e.g., number of objects, rate of change) and the user's request (e.g., level of detail, reasoning required) directly dictates the necessary computational power for perception, understanding, and response generation.

Additionally, the memory footprint, computational intensity, and access patterns across modalities vary significantly (e.g., LLMs versus image diffusion models versus vision encoder models), developing specialized system components and resource allocation strategies tailored to each modality is essential. This includes dynamic memory allocation for large models, adaptive batching strategies for variable arrival rates, specialized deployment strategies for different modalities, and efficient offloading mechanisms between edge and cloud resources. New resource management techniques and system designs will be needed to deliver responsive user interaction for such dynamic and resource-intensive generative tasks.

Cross-Modal QoE Synchronization. As applications increasingly blend multiple modalities (e.g., a robot assistant that visually perceives its environment, verbally plans its actions, and then physically interacts), ensuring a consistent and high-quality experience across these interconnected components will be a significant system design challenge. For instance, a system might need to ensure that visual understanding (e.g., identifying an object in a user's view) is tightly synchronized with concurrent auditory cues or interactive elements (e.g., highlighting the object on an AR display) to provide a seamless and coherent experience. This requires novel scheduling algorithms that understand inter-modal dependencies and optimize for a holistic and synchronized QoE.

5.3.2 AI Agents for Self-Evolving ML Systems Design

The emergence of AI agents capable of performing complex tasks autonomously presents a transformative opportunity, potentially facilitating scientific research and accelerating innovation. My last-year work on building a co-scientist AI agent to help automate research experimentation and optimize the research solutions- Curie [78] - has shown the potential of this direction. Key capabilities and research opportunities include:

1. **Automated System Design and Optimization:** AI agents could be tasked with

discovering optimal system configurations (i.e., 2→ ∞ research level tasks), such as the optimal degrees of different parallelisms (e.g., tensor, pipeline, data parallelism) for training LLMs, navigating vast and complex search spaces more effectively to find the optimal system configurations that maximize training throughput. Many 2→ ∞ tasks—those requiring significant, detailed engineering effort to squeeze out final percentages of performance or cost savings—could potentially be delegated to AI agents. This would free human researchers and engineers to focus on more foundational 0→1 or 1→2 challenges, accelerating the pace of innovation in ML systems. Beyond 2→ ∞ configuring systems, agents might propose and even implement novel system components or designs (contributing to 0→1 and 1→2 research levels). Such agents could propose innovative architectural components, autonomously adapt systems to evolving workloads and hardware, contributing to novel scheduling policies, kernel-level optimizations, or advanced strategies for computation-communication overlap.

2. **Autonomous Benchmarking, Analysis, and Refinement:** An AI agent could be empowered to deploy system changes, execute over diverse benchmark workloads, collect and analyze performance traces, and iteratively refine its designs based on observed outcomes, creating a closed loop of continuous system improvement. A fundamental challenge will be developing ways for AI agents to represent and understand the intricate components, traces, and performance characteristics of complex ML systems.

Achieving this level of autonomous capability requires a new generation of foundation models, which are imbued with deep, specialized knowledge of ML systems. In addition, reinforcement learning is needed to train agents to master the full lifecycle of ML systems research experimentation. This necessitates high-quality datasets that capture end-to-end experimentation processes—including hypothesis generation, system implementation, execution, and analysis—to provide effective training supervision for such agents.

5.3.3 Next-Generation Agentic AI Systems.

As AI agents become capable of tackling increasingly complex and long-running tasks, the underlying system frameworks must evolve significantly. Current AI agents often rely on relatively simple sequences of API calls, but future agents will need to perform more sophisticated tool use (e.g., dynamically composing software libraries, executing generated code), interact robustly with physical environments via sensors, manage long-horizon tasks involving intricate dependencies and error recovery, and strategically leverage heterogeneous compute resources. This necessitates re-designing agentic AI system frameworks from the

ground up to natively support these advanced agentic capabilities. A key focus will be on creating abstractions that simplify the programming and orchestration of complex agentic workflows. This includes:

1. **Intelligent Resource Management:** Agents will decompose high-level goals into many sub-tasks, each potentially requiring different computational resources (e.g., LLM inference, symbolic reasoning, code execution, physical actuators). The framework must provide abstractions to seamlessly dispatch these sub-tasks to appropriate resources, whether local, cloud-based, or on specialized hardware. To execute complex plans efficiently, agents will need to perform multiple sub-tasks, which may run in parallel or sequentially, often forming a Directed Acyclic Graph (DAG) of operations. The underlying system must offer intuitive ways to express and manage this parallelism, handling dependencies, data flow, and synchronization automatically where possible. Moreover, advanced scheduling are needed for interdependent sub-tasks and underlying resources to optimize for latency, cost, or other objectives, ensuring robust execution of long-running, multi-step agentic workflows. Finally, robust checkpointing and fault tolerance mechanisms will be essential to ensure the reliable execution of these potentially long-running and complex multi-step agentic workflows.

2. **Self-Evolving Agent Systems:** As agentic systems undertake longer-running tasks and operate in dynamic environments, they should possess the ability to learn and adapt continuously. The framework should support agents that learn from their past actions, environmental feedback, and direct human input. This involves integrating mechanisms akin to reinforcement learning, where agents can refine their policies, improve their planning abilities, and even discover new tools or strategies over time. Training AI agents that learn complex behaviors and adapt over long periods requires significant advancements in RL techniques and systems. This includes developing algorithms that are more sample-efficient to learn from limited interaction data, computationally scalable to handle complex state and action spaces, and capable of effective reward assignment over extended time horizons. From the systems perspective, this implies building resource-efficient infrastructure for distributed RL training and data management for agent experiences.

By tackling these system-level challenges, we can enable the development of more capable, adaptable, and reliable AI agents that can address complex, real-world problems across a multitude of domains.

APPENDIX A

Andes Appendix

A.1 Alternative Scheduling Objectives

In Section 2.4.1, we presented Venn in terms of maximizing the average QoE across all requests. However, text streaming services can have different quality goals under various deployment circumstances. More importantly, our defined QoE metric and proposed solution can be seamlessly adapted to different QoE objectives. In this section, we explore alternative scheduling objectives.

Maximizing the Minimum QoE. To maximize the minimum QoE across all requests (i.e., max-min QoE), the gain (item value in knapsack) of request i can be formulated as:

$$\max(Q_{\text{min}} - Q_{\text{wait},i}, 0), \tag{A.1}$$

where Q_{min} is the minimum QoE across all requests. This function prioritizes requests that, if not served within Δt, would further degrade the minimum QoE. By prioritizing these urgent requests, the overall QoE floor can be lifted, ensuring a more uniformly satisfying user experience.

Maximizing the Number of Requests with Perfect QoE. To optimize the number of requests that achieve perfect QoE, the gain (item value in knapsack) of request i can be formulated as:

$$[\mathbb{1}(Q_{\text{serve, i}} = 1) - \mathbb{1}(Q_{\text{wait, i}} = 1)] \cdot \mathbb{1}(Q_{\text{current},i} = 1), \tag{A.2}$$

where $\mathbb{1}(\cdot)$ is 1 if the given condition is true and 0 otherwise, and $Q_{\text{current},i}$ is the request's current QoE. The intuition behind this approach is that (1) there is no point in serving a request whose QoE is not perfect at the moment, and (2) if a request with currently perfect QoE will degrade QoE if not served for Δt, the request must be prioritized.

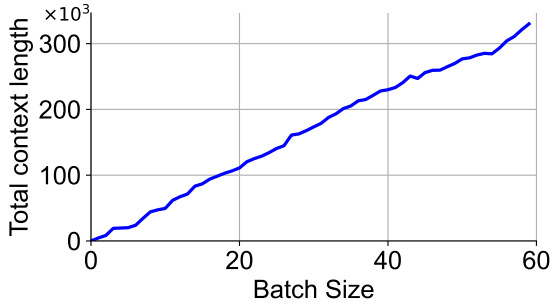


Figure A.1: Total context length distribution under different batch sizes using the Multi-Round ShareGPT dataset.

A.2 Modeling Token Generation Latency

In order to solve our knapsack formulation in Section 2.4.1, we need to be able to anticipate the QoE of a request after Δt if served (i.e., $Q_{\text{serve},i}$), which requires us to know the token generation latency, which is known to depend on batch size and the total number of tokens in the batch.

Figure A.1 shows the relationship between the batch size and the total number of tokens across all requests in the batch (i.e., total context length). It can be seen that batch size and total context length are nearly perfectly correlated, with Pearson correlation coefficient being 0.997. Moreover, with the increase of batch size, the total context length is more predictable as it averages out the variance in individual request context lengths. Therefore, we can drop total context length and estimate token generation latency simply as a function of batch size B.

A.3 Dynamic Programming Solution

In Algorithm 5, we give a 3D dynamic programming solution to Equation 2.5. The time complexity of the algorithm is $O(M \cdot N^2)$ as the largest batch size B is N in the worst case, and the problem needs to be solved for all feasible batch sizes B to find the optimal set of requests to serve. We note for clarity that our knapsack problem is *weakly NP-Hard* and the 3D DP algorithm is *not* polynomial time with respect to problem size (number of bits required to represent the problem). That is, when the problem size (number of bits) is scaled in terms of the number of requests N by adding more requests, runtime grows quadratically. However, when the problem is scaled in terms of available memory M by increasing the number of bits needed to represent M, the value of M and thus algorithm runtime grows exponentially. Therefore, the solution runs in *pseudo-polynomial* time, which is effectively

exponential time. For more details on weak NP-Hardness and pseudo-polynomial runtime, we direct the reader to [159].

A.4 Token Pacer In Action

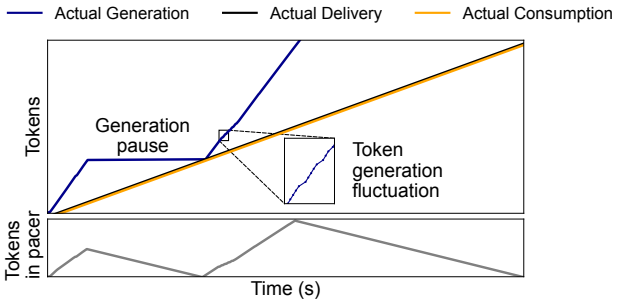


Figure A.2: The client-side token pacer holds excess tokens sent from the server to absorb token generation fluctuations and paces token delivery based on the user's ideal reading speed.

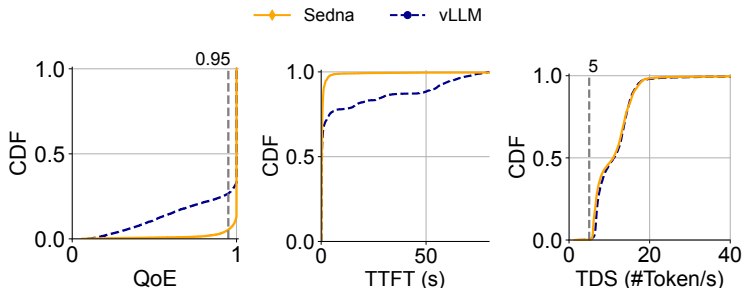
Figure A.2 visualizes the token pacer in action. With an initial burst generation faster than the user's token consumption speed, the pacer withholds excess tokens and paces token delivery, thus growing in size. Then, the server is aware of the token pacer and the user's QoE parameters, so knowing that this request would have sufficient tokens to deliver to the user for a while, the server preempts the request to serve other requests. While the request is waiting, the pacer continues to deliver tokens to users at their token consumption speed. Finally, the server resumes the request at the right timing and starts generating tokens again, and together with the token pacer, perfect QoE was achieved.

A.5 vLLM Configuration Details

To ensure a fair and optimized comparison, we modified vLLM's default configuration to maximize server utilization. By default, vLLM sets relatively conservative values for the maximum number of batched tokens and sequences, which may leave GPU resources under-utilized and exacerbate head-of-line blocking under load. We increased these limits to the maximum values supported by vLLM to better saturate the server and improve throughput. The non-default configuration parameters used in our evaluation are listed in Table A.1.

Configuration	Value
`max-num-batched-tokens`	200000
`max-num-seqs`	512

Table A.1: Non-default vLLM configurations used in evaluation.



(a) QoE (b) TTFT (c) TDS

Figure A.3: QoE, TTFT, and TDS CDFs of requests in BurstGPT.

A.6 Zooming In on Real-World Traces

Here, we take a closer look at a one-hour BurstGPT [157] slice from Section 2.6.2 to further understand the improvements Venn brings.

QoE Improvement. We report the CDF of QoE, TTFT, and TDS across all requests in Figure A.3. Overall, compared with vLLM using FCFS, Venn improves average QoE from 0.88 to 0.99 and reduces average TTFT from 10.5s to 1.8s. Notably, 97% of requests served by Venn achieve a QoE of 0.95 or higher, compared to only 75% under vLLM.

QoE improvement without additional resources means that Venn can serve more requests concurrently while maintaining high QoE levels, or conversely, significantly reduce the amount of GPUs needed to maintain the same level of QoE, directly leading to cost savings. Venn achieves this while reducing the average TDS of requests only by a marginal amount: from 11.2 tokens/s to 10.9 tokens/s. The sacrifice is small because Venn – particularly its preemption overhead-aware refiner – keeps the number of preemptions well under control, keeping its impact to overall system throughput low.

Significant Queue Length Reduction. To examine the system's real-time behavior, we visualize the state of Venn while serving requests in Figure A.4. Visualization of vLLM serving the same trace can be found in Figure 2.2. We observe that Venn can reduce peak waiting queue length during serving by a significant 85% through token-level preemptive

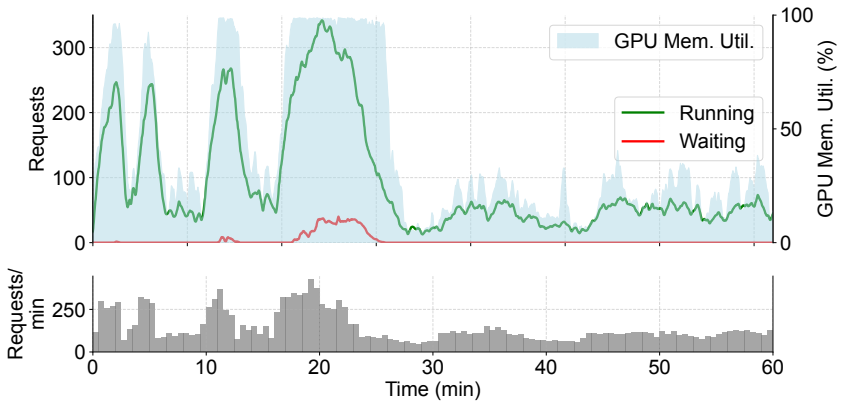


Figure A.4: Venn serving requests from BurstGPT. vLLM serving the same trace is shown in Figure 2.2. Queue length is significantly reduced under load surges, and GPU memory utilization is higher.

request scheduling. Furthermore, unlike vLLM, a large portion of the waiting requests in Venn are those that have been preempted by the scheduler after generating sufficient tokens for their users, explaining the high average QoE achieved by Venn.

A.7 More Details About Cyclic Burst Load Pattern

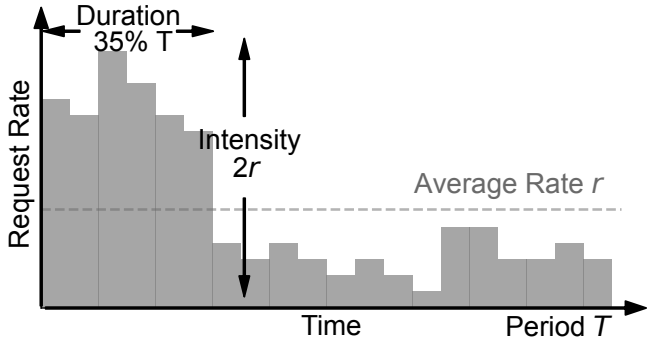


Figure A.5: One cycle of the cyclic burst load pattern.

We introduce the *Cyclic Burst Load Pattern* to resemble burst patterns in BurstGPT. Figure A.5 illustrates one cycle of burst in the cyclic burst load pattern. Bursts are characterized with two parameters: *intensity* (ratio of request rate from burst to the entire trace) and *duration* (percentage of time the burst takes up in the whole trace). During burst periods, the request arrival rate significantly exceeds the average, while it drops below the average in non-burst periods. Request arrivals in both burst and non-burst phases follow the Pois-

son process, and burst/non-burst periods alternate in a cyclic pattern, which we elaborate more in Appendix A.7. This cyclic burst pattern mimics the bursty and non-bursty phases commonly observed in production-level LLM serving environments, offering a controlled yet representative evaluation of serving system's performance.

To generate burstiness in the cyclic burst load pattern, we choose the average request arrival rate r (in requests per second) for each prompt dataset and model pair. The choice of average request arrival rate r is shown in Table A.2. For TTS service 2.6.4 with a slower expected TDS, we set the average request arrival rate r to 0.95 requests per second to create the burstiness.

Model	**ShareGPT**	**Arxiv**	**Coding**
Phi-3-mini 3.8B	1.8	0.28	0.6
Command R 32B	1.8	0.25	0.65
Phi-3.5-MoE 16×3.8B	0.45	2.3	1.1
Llama 3.1 70B	0.8	0.1	0.2

Table A.2: Average request arrival rates (r) for the Cyclic Burst Load Pattern across different prompt datasets and models.

A.8 More Results on Serving Capacity Gain

We further evaluate the serving capacity improvement of Venn on the Arxiv and Code dataset. We use the same setup as in Section 2.6.3 and compare Venn with vLLM under different target QoE levels. We show the results in Table A.3 and Table A.4. For Code dataset, since the baseline is not able to achieve the target high QoE for all models, we only report the results for the experiments that achieved the target QoE.

Target QoE	**Phi-3-mini 3.8B**	**Command R 32B**	**Phi-3.5-MoE 16×3.8B**	**Llama 3.1 70B**
0.95	1.41×	1.43×	1.15×	1.33×
0.96	1.40×	1.38×	1.12×	1.29×
0.97	1.36×	1.33×	1.09×	1.23×
0.98	1.32×	1.43×	1.05×	1.17×
0.99	1.27×	1.33×	1.14×	1.09×

Table A.3: Venn improves the serving capacity on the Arxiv dataset under different target QoE compared to vLLM.

Target QoE	Phi-3-mini 3.8B	Command R 32B	Phi-3.5-MoE 16×3.8B	Llama 3.1 70B
0.95	1.33×	1.14×	1.43×	–
0.96	1.37×	–	1.33×	–
0.97	1.42×	–	1.40×	–
0.98	1.47×	–	1.50×	–
0.99	1.41×	–	1.67×	–

Table A.4: Venn improves the serving capacity on the Code dataset under different target QoE compared to vLLM. We only report the results for the experiments that the baseline can achieve the target QoE.

Algorithm 5 Dynamic programming solution to Equation 2.5

Input:
Number of requests N and KV cache capacity M
Request context length array $l[N]$
Request QoE gain array $q[N]$
Target batch size B
Output: Solution array $x[N]$.

Initialize $dp[N+1][B+1][M+1]$ with $-\infty$
Initialize $choice[N+1][B+1][M+1]$ with 0
$dp[0][0][0] = 0$
for $i = 1$ to N **do**
    for $b = 0$ to $\min(i, B)$ **do**
        for $m = 0$ to M **do**
            if $dp[i][b][m] < dp[i-1][b][m]$ **then**
                ▷ Request i is not served.
                $dp[i][b][m] = dp[i-1][b][m]$
                $choice[i][b][m] = 0$
            if $b \geq 1$ & $m \geq l[i]$ **then**
                if $dp[i-1][b-1][m-l[i]] + q[i] > dp[i][b][m]$ **then**
                    ▷ Request i is served.
                    $dp[i][b][m] = dp[i-1][b-1][m-l[i]] + q[i]$
                    $choice[i][b][m] = 1$
$Q_{\max} = \max(dp[N][B][:])$
$m_{\text{current}} =$ Index of $Q_{\max}$ in $dp[N][B]$
$b_{\text{current}} = B$
Initialize $x[N+1]$ with zeros
for $i = N$ **downto** 1 **do**
    $x[i] = choice[i][b_{\text{current}}][m_{\text{current}}]$
    if $x[i] == 1$ **then**
        $m_{\text{current}} = m_{\text{current}} - l[i]$
        $b_{\text{current}} = b_{\text{current}} - 1$
return $x[1:]$

APPENDIX B

Auxo Appendix

B.1 Proof of Lemma

We first make precise some definitions that are related to the proof from SCAFFOLD and then see the proof of Lemma.

Assumption 1. $g_i(w)$ is unbiased stochastic gradient of f_i with bounded variance, where f_i represents the loss function on client i.

$$\mathbb{E}_{x_i}[||g_i(w) - \nabla f_i(w)||^2] \leq \sigma^2, \forall i, x.$$

where w is the aggregated server model. Note that σ only bounds the variance within clients not across clients.

Assumption 2. $\{f_i\}$ are β-smooth and satisfy:

$$||\nabla f_i(w) - \nabla f_i(v)|| \leq \beta||w - v||, \forall i, w, v.$$

Assumption 3. f_i is μ-convex for $\mu \geq 0$ and satisfies:

$$\langle \nabla f_i(w), v - w \rangle \leq -(\nabla f_i(w) - \nabla f_i(v) + \frac{\mu}{2}||w - v||^2), \forall i, w, v.$$

Assumption 4. (G, B)-BGD or Bounded Gradient Similarity: there exist constants $G \geq 0$ and $B \geq 1$ such that

$$\frac{1}{N}\sum_{i=1}^{N} ||\nabla f_i(w)||^2 \leq G^2 + B^2 \nabla f(w)), \forall w.$$

Theoretical Results

Lemma 1. If the population and training resources are partitioned into up to K cohorts, to theoretically achieve better model convergence, intra-cohort heterogeneity should be reduced by $\sqrt{K}$ times when the training resource $|\mathcal{P}|$ is larger than $\alpha\sqrt{\frac{|\mathcal{P}_0|}{J_0^2}}$. α is a constant setting specified in SCAFFOLD that elaborates the relationship between model convergence and training resources.

Proof. We first borrow the proof of convergence analysis on FedAvg (Theorem 1) from SCAFFOLD following the same assumptions mentioned above:

$$\mathbb{E}[f(w^R)] - f(w^*) \leq 3||w^0 - w^*||^2\mu e^{-\frac{\widetilde{\eta}}{2}R}$$

$$+\widetilde{\eta}(\frac{2\sigma^2}{kP}(1+\frac{P}{\eta_g^2}) + \frac{8G^2}{P}(1-\frac{P}{N})) + \widetilde{\eta}^2(36\beta G^2),$$

$$\forall \frac{1}{\mu R} \leq \widetilde{\eta} \leq \frac{1}{8(1+B^2)\beta}$$

where P denotes the training resources, k is the number local steps, η_l is the local step-size, η_g is the global step-size and $\widetilde{\eta} = k\eta_l\eta_g$ is the effective step-size

Since we only care about the effect of training resources P and heterogeneity G on the convergence analysis, we further simplify the right hand side equation to be

$$h(P, G) = \frac{\theta}{P} + \gamma\frac{G^2}{P} + \rho G^2 + \xi$$

where θ, γ, ρ and ξ are constant settings. Since we proportionally partition the population and training resources, we can assume $(1 - \frac{S}{N})$ to be constant before and after partition.

In order to have no worse model convergence bound after partitioning, we need $h(P, G)$ to be non-increasing with the reduction of training resources P. As proposed in Lemma 1, Auxo partitions K cohorts when the intra-cohort heterogeneity can be reduced by $\sqrt{K}$ times, which approximately give $\frac{G^2}{P}$ be constant as the one before partition $\frac{G_0^2}{P_0}$. By substituting this relationship into $h(P, G)$, we can derive the lower bound for the range of training resources required to achieve better convergence bound:

$$P \geq \sqrt{\frac{\theta P_0}{G_0^2\rho}} = \sqrt{\frac{\sigma^2}{18k\widetilde{\eta}^2\beta}\frac{P_0}{G_0^2}} = \alpha$$

APPENDIX C

Venn Appendix

C.1 Detailed Venn Responsibility

Venn delegates responsibilities such as device selection, device fault tolerance, and privacy protection to individual CLjobs. Device failures are both inevitable and difficult to predict in CL. Rather than imposing a one-size-fits-all solution, Venn empowers CLjobs to take the reins on fault tolerance based on their specific workloads and objectives. Therefore, Venn offloads handling device fault tolerance to CLjobs, who can better detect and react to device failures (e.g., deciding the amount of overcommit [24]). Similarly, Venn offers CLjobs the freedom to design their own device selectors [87], where they can incorporate customized resource criteria into their requests. Venn also does not interfere with job-specific privacy solutions such as secure aggregation [22, 66] or differential privacy [48, 167].

C.2 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:

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

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

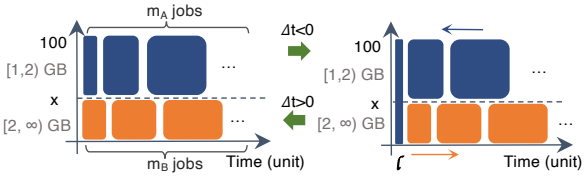
$$\sum_{i=0}^{q} x_{ij} = D_j, \forall j \in [1, m]$$

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:

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

(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 C.1: Venn scheduling algorithm.

C.3 Theoretical Insight to the Heuristic of IRS

Lemma 2. *Given a diverse set of CLjobs 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 [138], 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. □

C.4 Effectiveness of Venn Scheduling Heuristic

To illustrate the effectiveness of Venn's approach, we start with proving Lemma 3, 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 3. *Given two job groups with arbitrary resource contention patterns, the scheduling plan generated by Venn as in Algorithm 3 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 C.1a, 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 C.1c. 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 C.1c, 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 C.1a, 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 C.1a, it can be generalized to job group with intersected resource contention and give the same conclusion.

Based on Algorithm 3, the first step is to sort these jobs within each job group by job size in ascending order (Figure C.1b). 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 C.1c), 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 C.1c) 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 3 line 12. □

By leveraging the conclusion of Lemma 3, 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 3 to minimize the average scheduling delay between G_j and G_k.

BIBLIOGRAPHY

[1] Google Open Images Dataset. https://storage.googleapis.com/openimages/web/index.html.

[2] Reddit Comment Data. https://files.pushshift.io/reddit/comments/.

[3] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). https://eur-lex.europa.eu/eli/reg/2016/679/oj, 2016. Official Journal of the European Union, L 119, pp. 1-88.

[4] TensorFlow Federated. https://www.tensorflow.org/federated, 2018.

[5] Federated learning: Collaborative machine learning without centralized training data. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html, 2020. Accessed August 29, 2021.

[6] Ahmed M. Abdelmoniem, Atal Narayan Sahu, Marco Canini, and Suhaib A. Fahmy. Resource-efficient federated learning. 2023.

[7] Ahmed M Abdelmoniem, Atal Narayan Sahu, Marco Canini, and Suhaib A Fahmy. Resource-efficient federated learning. *EuroSys*, 2023.

[8] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[9] Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. INFERCEPT: Efficient intercept support for augmented large language model inference. In *ICML*, 2024.

[10] Vaibhav Aggarwal, Vaibhav Gupta, Prayag Singh, Kiran Sharma, and Neetu Sharma. Detection of spatial outlier by using improved z-score test. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 788–790, 2019.

[11] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *OSDI*, 2024.

[12] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. The Falcon series of open language models. *arXiv preprint arXiv:2311.16867*, 2023.

[13] Daniel An. Find out how you stack up to new industry benchmarks for mobile page speed. https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/, 2018.

[14] Kara Anderson. Who, what, and where of ai adoption in america, feb 2025. Accessed: March 28, 2025.

[15] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *AISTATS*, 2020.

[16] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. A quest for an internet video quality-of-experience metric. In *HotNets*, 2012.

[17] Ravikumar Balakrishnan, Tian Li, Tianyi Zhou, Nageen Himayat, Virginia Smith, and Jeff Bilmes. Diverse client selection for federated learning via submodular maximization. In *ICLR*, 2022.

[18] Nabajeet Barman and Maria G Martini. QoE modeling for HTTP adaptive video streaming–a survey and open challenges. *IEEE Access*, 2019.

[19] Dom Barnard. Average speaking rate and words per minute. https://virtualspeech.com/blog/average-speaking-rate-words-per-minute.

[20] P. Berkhin. *A Survey of Clustering Data Mining Techniques*. Springer Berlin Heidelberg, 2006.

[21] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *ICML*, 2012.

[22] K. A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[23] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In *MLSys*, 2019.

[24] Keith Bonawitz, Hubert Eichner, and et al. Towards federated learning at scale: System design. In *MLSys*, 2019.

[25] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *CCS*, 2017.

[26] Christopher Briggs, Zhong Fan, and Péter András. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. *IJCNN*, 2020.

[27] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

[28] Marc Brysbaert. How many words do we read per minute? a review and meta-analysis of reading rate. *Journal of memory and language*, 2019.

[29] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. In *Proceedings of the 29th international symposium on high-performance parallel and distributed computing*, 2020.

[30] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning. In *EuroSys*, pages 1–16, 2020.

[31] Yudong Chen, Lili Su, and Jiaming Xu. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *ACM SIGMETRICS*, 2019.

[32] Gary Cheng, Karan N. Chadha, and John C. Duchi. Fine-tuning is fine in federated learning. *CoRR*, abs/2108.07313, 2021.

[33] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A discourse-aware attention model for abstractive summarization of long documents. In *NAACL*, 2018.

[34] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. In *arxiv.org/abs/1702.05373*, 2017.

[35] Cohere For AI. c4ai-command-r-08-2024. https://huggingface.co/CohereForAI/c4ai-command-r-08-2024, 2024.

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

[37] Don Kurian Dennis, Tian Li, and Virginia Smith. Heterogeneity for the win: One-shot federated clustering. In *ICML*, 2021.

[38] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. In *SIGCOMM*, 2011.

[39] Moming Duan, Duo Liu, Xinyuan Ji, Yu Wu, Liang Liang, Xianzhang Chen, and Yujuan Tan. Flexible clustered federated learning for client-level data distribution shift. *IEEE TPDS*, 2021.

[40] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[41] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *ATC*, 2019.

[42] C. Dwork and A. Roth. *The Algorithmic Foundations of Differential Privacy*. Foundations and trends in theoretical computer science. 2014.

[43] Elfsight Team. Chatgpt usage statistics: How popular chatgpt is in 2025, feb 2025. Accessed: March 28, 2025.

[44] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *ACM SIGSAC*, 2014.

[45] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of time table and multi-commodity flow problems. In *16th annual symposium on foundations of computer science (sfcs 1975)*, pages 184–193. IEEE, 1975.

[46] Exploding Topics Team. 35+ ai statistics & trends [2025 update], 2025. Accessed: March 28, 2025.

[47] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. *SEC*, 2020.

[48] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. 2017.

[49] Ali Ghodsi, Matei Zaharia, Scott Shenker, and Ion Stoica. Choosy: Max-min fair sharing for datacenter jobs with constraints. In *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013.

[50] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. In *NeurIPS*, 2020.

[51] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[52] Grand View Research. Large language model (LLM) market size, share & trends analysis report by component, by application, by enterprise size, by end-use, by region, and segment forecasts, 2023 - 2030. https://www.grandviewresearch.com/industry-analysis/large-language-model-llm-market-report, 2023.

[53] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *NSDI*, 2019.

[54] Hanxi Guo, Hao Wang, Tao Song, Yang Hua, Zhangcheng Lv, Xiulang Jin, Zhengui Xue, Ruhui Ma, and Haibing Guan. Siren: Byzantine-robust federated learning via proactive alarming. In *Proceedings of the ACM SoCC*, 2021.

[55] Peizhen Guo, Bo Hu, and Wenjun Hu. Mistify: Automating DNN model porting for On-Device inference at the edge. In *NSDI*, 2021.

[56] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction, 2019.

[57] Jingrui He. Learning from data heterogeneity: Algorithms and applications. In *IJCAI*, pages 5126–5130, 2017.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[59] Shiqi He, Qifan Yan, Feijie Wu, Lanjun Wang, Mathias Lécuyer, and Ivan Beschastnikh. Gluefl: Reconciling client sampling and model masking for bandwidth efficient federated learning. *MLSys*, 2023.

[60] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps. *NeurIPS*, 2021.

[61] Sara Hooker. The hardware lottery, 2020.

[62] Andreessen Horowitz. The top 100 genai consumer apps. https://a16z.com/100-gen-ai-apps-4/.

[63] Charlie Hou, Kiran K Thekumparampil, Giulia Fanti, and Sewoong Oh. Reducing the communication cost of federated learning through multistage optimization. 2018.

[64] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. Gaia: Geo-distributed machine learning approaching LAN speeds. In *NSDI*, 2017.

[65] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

[66] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, et al. Papaya: Practical, private, and scalable federated learning. *MLSys*, 2022.

[67] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. Elastic resource sharing for distributed deep learning. In *NSDI*, 2021.

[68] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. Ai benchmark: All about deep learning on smartphones in 2019. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3617–3635. IEEE, 2019.

[69] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. Towards understanding biased client selection in federated learning. In *AISTATS*, 2022.

[70] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

[71] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. CFA: A practical prediction system for video QoE optimization. In *NSDI*, 2016.

[72] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. Pytheas: Enabling Data-Driven quality of experience optimization using Group-Based Exploration-Exploitation. In *NSDI*, 2017.

[73] Zhifeng Jiang, Wei Wang, Baochun Li, and Bo Li. Pisces: Efficient federated learning via guided asynchronous training. In *SoCC*, 2022.

[74] Eirini Kalliamvakou, Christian Bird, and Thomas Zimmermann. Research: Quantifying github copilot's impact on developer productivity and happiness, feb 2025. Accessed: March 28, 2025.

[75] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In *ICML*, 2020.

[76] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer Berlin Heidelberg, 2004.

[77] Phillip Keung, Yichao Lu, György Szarvas, and Noah A. Smith. The multilingual amazon reviews corpus. *CoRR*, abs/2010.02573, 2020.

[78] Patrick Tser Jern Kon, Jiachen Liu, Qiuyi Ding, Yiming Qiu, Zhenning Yang, Yibo Huang, Jayanth Srinivasa, Myungjin Lee, Mosharaf Chowdhury, and Ang Chen. Curie: Toward rigorous and automated scientific experimentation with ai agents. *arXiv preprint arXiv:2502.16069*, 2025.

[79] Patrick Tser Jern Kon, Jiachen Liu, Yiming Qiu, Weijun Fan, Ting He, Lei Lin, Haoran Zhang, Owen M. Park, George Sajan Elengikal, Yuxin Kang, Ang Chen, Mosharaf Chowdhury, Myungjin Lee, and Xinyu Wang. Iac-eval: A code generation benchmark for infrastructure-as-code programs. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

[80] Patrick Tser Jern Kon, Jiachen Liu, Xinyi Zhu, Qiuyi Ding, Jingjia Peng, Jiarong Xing, Yibo Huang, Yiming Qiu, Jayanth Srinivasa, Myungjin Lee, Mosharaf Chowdhury, Matei Zaharia, and Ang Chen. Exp-bench: Can ai conduct ai research experiments? *arXiv preprint arXiv:2505.24785*, 2025.

[81] Nicolas Kourtellis, Kleomenis Katevas, and Diego Perino. Flaas: Federated learning as a service. In *Proceedings of the 1st workshop on distributed machine learning*, 2020.

[82] Naveen Kumar. Chatgpt statistics (march 2025): Number of users and queries. *DemandSage*, feb 2025. Accessed: March 28, 2025.

[83] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, 2023.

[84] Fan Lai, Mosharaf Chowdhury, and Harsha Madhyastha. To relay or not to relay for inter-cloud transfers? In *HotCloud*. USENIX Association, 2018.

[85] Fan Lai, Yinwei Dai, Sanjay S. Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning at scale. In *ICML*, 2022.

[86] Fan Lai, Jie You, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Sol: Fast distributed computation over slow networks. In *NSDI*, 2020.

[87] Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Oort: Efficient federated learning via guided participant selection. In *OSDI 21*, July 2021.

[88] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *ICLR*, 2020.

[89] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *ICML*, 2021.

[90] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *MLSys*, 2020.

[91] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *ICLR*, 2020.

[92] Wenqi Li, Fausto Milletarì, Daguang Xu, Nicola Rieke, Jonny Hancox, Wentao Zhu, Maximilian Baust, Yan Cheng, Sébastien Ourselin, M. Jorge Cardoso, and Andrew Feng. Privacy-preserving federated brain tumour segmentation. Springer-Verlag.

[93] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *ICLR*, 2020.

[94] Zitao Li, Bolin Ding, Ce Zhang, Ninghui Li, and Jingren Zhou. Federated matrix factorization with privacy guarantee. *VLDB*, 2021.

[95] Ji Liu, Juncheng Jia, Beichen Ma, Chendi Zhou, Jingbo Zhou, Yang Zhou, Huaiyu Dai, and Dejing Dou. Multi-job intelligent scheduling with cross-device federated learning. *IEEE Transactions on Parallel and Distributed Systems*, 34(2):535–551, 2022.

[96] Jiachen Liu, Fan Lai, Yinwei Dai, Aditya Akella, Harsha Madhyastha, and Mosharaf Chowdhury. Auxo: Efficient federated learning via scalable cohort identification. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 87–102, 2023.

[97] Jiachen Liu, Fan Lai, Ding Ding, Yiwen Zhang, and Mosharaf Chowdhury. Venn: Resource management across federated learning jobs. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

[98] Jiachen Liu, Zhiyu Wu, Jae-Won Chung, Fan Lai, Myungjin Lee, and Mosharaf Chowdhury. Andes: Defining and enhancing quality-of-experience in llm-based text streaming services. *arXiv preprint arXiv:2401.12345*, 2024.

[99] Junxu Liu, Jian Lou, Li Xiong, Jinfei Liu, and Xiaofeng Meng. Projected federated averaging with heterogeneous differential privacy. *VLDB*.

[100] Xi Liu, Florin Dobrian, Henry Milner, Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. A case for a coordinated internet video control plane. In *SIGCOMM*, 2012.

[101] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: KV cache compression and streaming for fast large language model serving. In *SIGCOMM*, 2024.

[102] Guodong Long, Ming Xie, Tao Shen, Tianyi Zhou, Xianzhi Wang, and Jing Jiang. Multi-center federated learning: clients clustering for better personalization. *WWW*, 2022.

[103] Chengfei Lv, Chaoyue Niu, Renjie Gu, Xiaotang Jiang, Zhaode Wang, Bin Liu, Ziqi Wu, Qiulin Yao, Congyu Huang, Panos Huang, 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)*, pages 249–265, 2022.

[104] Julie Lythcott-Haims, Sarah Dixon, and Martin Casado. Navigating the high cost of ai compute, feb 2025. Accessed: March 28, 2025.

[105] J MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297. University of California Los Angeles LA USA, 1967.

[106] Luo Mai, Guo Li, Marcel Wagenländer, Konstantinos Fertakis, Andrei-Octavian Brabete, and Peter Pietzuch. KungFu: Making training in distributed machine learning adaptive. In *OSDI*, 2020.

[107] McKinsey & Company. Global survey: The state of ai in 2020, nov 2020. Accessed: March 28, 2025.

[108] Brendan McMahan and Daniel Ramage. Utilization of fate in risk management of credit in small and micro enterprises. https://www.fedai.org/cases/utilization-of-fate-in-risk-management-of/credit-in-small-and-micro-enterprises/, 2017. Accessed August 31, 2021.

[109] H. B. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

[110] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *ICLR*, 2018.

[111] Lorenzo Minto, Moritz Haller, Benjamin Livshits, and Hamed Haddadi. *Stronger Privacy for Federated Collaborative Filtering With Implicit Feedback*. 2021.

[112] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. Looking beyond {GPUs} for {DNN} scheduling on {Multi-Tenant} clusters. In *OSDI*, 2022.

[113] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *SOSP*, 2019.

[114] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 481–498, 2020.

[115] OpenAI. ChatGPT. https://chat.openai.com.

[116] OpenAI. Text to speech. https://platform.openai.com/docs/guides/text-to-speech.

[117] OpenAI. Chatgpt, 2025. Accessed: March 28, 2025.

[118] OpenMined.

[119] Tara Parachuk. Speaking rates comparison table. https://www.voices.com/blog/languages-in-usa-speaking-rates-per-minute.

[120] Matthias Paulik, Matt Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, et al. Federated evaluation and tuning for on-device personalization: System design & applications. *arXiv preprint arXiv:2102.08503*, 2021.

[121] Matthias Paulik, Matt Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier C. van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, Sudeep Agarwal, Julien Freudiger, Andrew Byde, Abhishek Bhowmick, Gaurav Kapoor, Si Beaumont, Áine Cahill, Dominic Hughes, Omid Javidbakht, Fei Dong, Rehan Rishi, and Stanley Hung. Federated evaluation and tuning for on-device personalization: System design & applications. *CoRR*, abs/2102.08503, 2021.

[122] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *EuroSys*, pages 1–14, 2018.

[123] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. Low latency geo-distributed data analytics. In *SIGCOMM*, 2015.

[124] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R Ganger, and Eric P Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *OSDI*, 2021.

[125] Daniel Ramage and Stefano Mazzocchi. Federated analytics: Collaborative data science without data collection. https://blog.research.google/2020/05/federated-analytics-collaborative-data.html, 2020.

[126] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329*, 2019.

[127] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *ICLR*, 2021.

[128] Amirhossein Reisizadeh, Farzan Farnia, Ramtin Pedarsani, and Ali Jadbabaie. Robust federated learning: The case of affine distribution shifts. In *NeurIPS*, 2020.

[129] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *AISTATS*, 2020.

[130] Edo Roth, Karan Newatia, Yiping Ma, Ke Zhong, Sebastian Angel, and Andreas Haeberlen. Mycelium: Large-scale distributed graph queries with differential privacy. In *SOSP*, 2021.

[131] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. Honeycrisp: Large-scale differentially private aggregation without a trusted core. In *SOSP*, 2019.

[132] Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C. Pierce. Orchard: Differentially private analytics at scale. In *OSDI*, 2020.

[133] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: communication-efficient federated learning with sketching. ICML, 2020.

[134] Nikola Roza. Dall-e statistics facts and trends for 2025 - all the crucial stats you must know about (dall-e 2, dall-e 3 and dall-e 4)!, feb 2025. Accessed: March 28, 2025.

[135] Adam Sadilek, Luyang Liu, Dung Nguyen, Methun Kamruzzaman, Stylianos Serghiou, Benjamin Rader, Alex Ingerman, Stefan Mellem, Peter Kairouz, Elaine O Nsoesie, et al. Privacy-first health research with federated learning. *NPJ digital medicine*, 4(1):132, 2021.

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

[137] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[138] Linus Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 1968.

[139] D. Sculley. Web-scale k-means clustering. In *WWW*, 2010.

[140] Elizabeth Seger, Aviv Ovadya, Divya Siddarth, Ben Garfinkel, and Allan Dafoe. Democratising ai: Multiple meanings, goals, and methods. In *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, pages 715–722, 2023.

[141] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *ICLR*, 2018.

[142] Ao Shen, Zhiyao Li, and Mingyu Gao. FastSwitch: Optimizing context switching efficiency in fairness-aware large language model serving, 2024.

[143] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E Gonzalez, and Ion Stoica. Fairness in serving large language models. In *OSDI*, 2024.

[144] Jinhyun So, Başak Güler, and A Salman Avestimehr. Byzantine-resilient secure federated learning. *IEEE Journal on Selected Areas in Communications*, 2020.

[145] Stefanini. Democratized ai: Potential benefits, risks, and glimpse to future, feb 2025. Accessed: March 28, 2025.

[146] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *OSDI*, 2024.

[147] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[148] Xueyang Tang, Song Guo, and Jingcai Guo. Personalized federated learning with clustered generalization. *IJCAI*, 2022.

[149] Gemma Team. Gemma 2: Improving open language models at a practical size. 2024.

[150] ShareGPT Team. ShareGPT. https://sharegpt.com/.

[151] Prashanth Thinakaran, Jashwant Raj Gunasekaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R Das. Phoenix: A constraint-aware scheduler for heterogeneous datacenters. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017.

[152] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[153] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. CLARINET: WAN-Aware optimization for analytics queries. In *OSDI*, 2016.

[154] Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. Efficient large language models: A survey. *Transactions on Machine Learning Research*, 2024.

[155] Ewen Wang, Ajay Kannan, Yuefeng Liang, Boyi Chen, and Mosharaf Chowdhury. FLINT: A platform for federated learning integration. In *MLSys*, 2023.

[156] Ewen Wang, Ajay Kannan, Yuefeng Liang, Boyi Chen, and Mosharaf Chowdhury. Flint: A platform for federated learning integration. *MLSys*, 2023.

[157] Yuxin Wang, Yuhan Chen, Zeyu Li, Zhenheng Tang, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. BurstGPT: A real-world workload dataset to optimize LLM serving systems. *arXiv preprint arXiv:2401.17644*, 2024.

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

[159] Wikepedia. Weak NP-completeness. https://en.wikipedia.org/wiki/Weak_NP-completeness.

[160] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. LoongServe: Efficiently serving long-context large language models with elastic sequence parallelism. In *SOSP*, 2024.

[161] Shanshan Wu, Tian Li, Zachary Charles, Yu Xiao, Ziyu Liu, Zheng Xu, and Virginia Smith. Motley: Benchmarking heterogeneity and personalization in federated learning. *FL-NeurIPS*, 2022.

[162] Zhe Wu, Michael Butkiewicz, Dorian Perkins, Ethan Katz-Bassett, and Harsha V. Madhyastha. SPANStore: Cost-effective geo-replicated storage spanning multiple cloud services. In *SOSP*, 2013.

[163] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *OSDI*, pages 595–610, 2018.

[164] Yuexiang Xie, Zhen Wang, Dawei Gao, Daoyuan Chen, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. Federatedscope: A flexible federated learning platform for heterogeneity. *VLDB*, 2023.

[165] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. When mobile apps going deep: An empirical study of mobile deep learning. *CoRR*, abs/1812.05448, 2018.

[166] Rui Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 2005.

[167] Zheng Xu, Yanxiang Zhang, Galen Andrew, Christopher A Choquette-Choo, Peter Kairouz, H Brendan McMahan, Jesse Rosenstock, and Yuanbo Zhang. Federated learning of gboard language models with differential privacy. *arXiv preprint arXiv:2305.18465*, 2023.

[168] Ye Xue, Diego Klabjan, and Yuan Luo. Aggregation delayed federated learning. *IEEE International Conference on Big Data*, 2022.

[169] Yihan Yan, Xiaojun Tong, and Shen Wang. Clustered federated learning in heterogeneous environment. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[170] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[171] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and optimizing gpu energy consumption of dnn training. *NSDI*, 2023.

[172] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *OSDI*, 2022.

[173] Peifeng Yu, Jiachen Liu, and Mosharaf Chowdhury. Fluid: Resource-aware hyperparameter tuning engine. In *Proceedings of Machine Learning and Systems*, volume 3, 2021.

[174] Peifeng Yu, Jiachen Liu, and Mosharaf Chowdhury. Fluid: Resource-aware hyperparameter tuning engine. In *MLSys*, 2021.

[175] Honglin Yuan, Warren Morningstar, Lin Ning, and Karan Singhal. What do we mean by generalization in federated learning? *arXiv preprint arXiv:2110.14216*, 2021.

[176] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[177] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.

[178] Xu Zhang, Yiyang Ou, Siddhartha Sen, and Junchen Jiang. SENSEI: Aligning video streaming quality with dynamic user sensitivity. In *NSDI*, 2021.

[179] Yuchen Zhao, Payam Barnaghi, and Hamed Haddadi. Multimodal federated learning on iot data. In *2022 IEEE/ACM Seventh International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2022.

[180] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[181] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. Cerebro: A platform for Multi-Party cryptographic collaborative learning. In *USENIX Security*, 2021.

[182] Wenting Zheng, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *IEEE S&P*, 2019.

[183] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *OSDI*, 2024.

[184] Yuxuan Zhu, Jiachen Liu, Mosharaf Chowdhury, and Fan Lai. Fedtrans: Efficient federated learning via multi-model transformation. In *Proceedings of Machine Learning and Systems*, volume 6, 2024.