MULTI-OBJECTIVE RESOURCE OPTIMIZATION FOR LARGE SCALE MACHINE LEARNING SYSTEMS

BY

HONGPENG GUO

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2024

Urbana, Illinois

ABSTRACT

The recent advancements in machine learning (ML) have marked significant progress across various fields, delivering high-quality solutions in computer vision, natural language processing, and virtual reality, among others. This leap forward is largely due to the innovations in deep learning and neural networks, which have opened new avenues in data analysis and decision-making processes, profoundly affecting people's lives and how society functions.

However, ML systems encounter considerable challenges in terms of resource efficiency, grappling with complex issues such as network bandwidth exhaustion, computational intensity, energy consumption, and hardware diversity. These challenges are interconnected, making the task of managing resources efficiently even more daunting. To enhance the effectiveness of machine learning models, it's crucial that these systems are optimized across multiple dimensions to strike a balance between performance, efficiency, and scalability, thereby ensuring sustainable operation at a larger scale.

In this thesis, we introduce a multi-objective resource optimization framework aimed at addressing the overarching resource challenges in large-scale machine learning systems. Leveraging the optimization opportunities presented by data redundancy and hardware configurability, we detail three initiatives that demonstrate optimizations for resource constraints within large-scale ML systems. Specifically, CROSSROI tackles network bandwidth and computational intensity by leveraging data redundancy in video streams, significantly reducing the amount of data required for processing and transmission. BOFL targets energy consumption and the timeliness of learning tasks, employing dynamic hardware configuration to enhance the power efficiency of devices involved in time-sensitive federated learning, which in turn prolongs battery life and lowers operational expenses. FEDCORE addresses the straggler effect in federated learning through the implementation of distributed coresets, minimizing the data processed by slower devices and thus boosting the overall efficiency of the system without sacrificing accuracy.

Collectively, these frameworks embody a comprehensive approach to multi-objective resource optimization, illustrating their effectiveness through significant enhancements across various resource dimensions. Moreover, our experiments confirm that adopting a holistic design that leverages both data and hardware opportunities can substantially elevate the efficiency of resource usage in machine learning systems.

"To my parents, for their love and support."

ACKNOWLEDGMENTS

I extend my deepest gratitude to my advisor, Professor Klara Nahrstedt, whose unwavering guidance and support have been pivotal throughout my Ph.D. journey. It has been an honor to be a part of her research group, where her profound knowledge, insightful feedback, and encouragement have been instrumental in navigating the challenges of my research projects. This thesis would not have been possible without her steadfast support.

My sincere thanks also go to my committee members, Professor Deming Chen, Professor Tianyin Xu, and Professor Baochun Li, for their valuable patience and constructive critiques that have significantly enhanced my thesis. Their involvement in my committee has been a great privilege.

I extend my deepest gratitude to my undergraduate advisor, Professor King-Shan Lui at the University of Hong Kong, who led me on the journey of academic research with a strong foundation. I also extend my deepest gratitude to my senior collaborators for their invaluable guidance and stimulating discussions, which have unveiled many intriguing problems my research addresses. Special acknowledgments go to Professor Haiming Jin at Shanghai Jiao Tong University, Professor Shuochao Yao at George Mason University, Dr. Eun Kyung Lee, and Dr. Tamar Eilam at IBM Research, as well as Dr. Claudiu Danilov at Boeing Research. Special thanks to Dr. Srikant Sharma, Dr. Ali Fakeri-Tabrizi, and Gordon Murphy for hosting my internships at Google, Meta, and Jane Street, respectively, where I gained tremendous happiness and invaluable experiences.

I would like to thank the US Army Research Laboratory (under cooperative agreement W911NF17-2-0196), the Ralph M. and Catherine V. Fisher Professorship, and the IBM-Illinois Discovery Accelerator Institute, for their financial support and assistance.

I want to thank my peers and friends in the MONET research group who have accompanied me along this journey: Dr. Haiming Jin, Dr. Wenyu Ren, Dr. Tuo Yu, Dr. Tarek Elgamal, Dr. Phuong V. Nguyen, Dr. Tianyuan Liu, Dr. Bo Chen, Dr. Jounsup Park, Dr. Qian Zhou, Dr. Rui-Xiao Zhang, Xiaoyang Wang, Beitong Tian, Mingyuan Wu, Ragini Gupta, Yinjie Zhang, Lingzhi Zhao, and Jiaxi Li. They are all close friends who have supported me in both my research and private life.

I also want to thank all my friends who have supported and helped me throughout my PhD studies. Since it is impossible to mention everyone, I can only list some of them: Zhihan Chen, Jinhui Song, Houxiang Ji, Xiaojun Xu, Huichen Li, Zhijian Yang, Shengzhong Liu, Tianshi Wang, Jinyang Li, Chaoqi Yang, Xuan Zhang, Yunqi Li, Yongzhou Chen, etc., who shared in my happiness and sadness during this journey. Additionally, special thanks to Haotian Gu, Fangjun Zhang, Yang Bai, Yuming Chen, Junjie Wang, Xiaoshi Guan, Jian Shi, Kaidong Wang, Shuaiyi Yue, etc., whom I have known for years and who have accompanied me through the PhD journey and beyond, bringing joy and growth.

Last but most importantly, I would like to express my most sincere gratitude and appreciation to my parents, Dianchao Guo and Yinfeng Liu, for their unconditional love and support. Special thanks to my girlfriend, Yufei Ruan, who brings me joy and courage to face any challenges in life. Their love is my most precious treasure.

TABLE OF CONTENTS

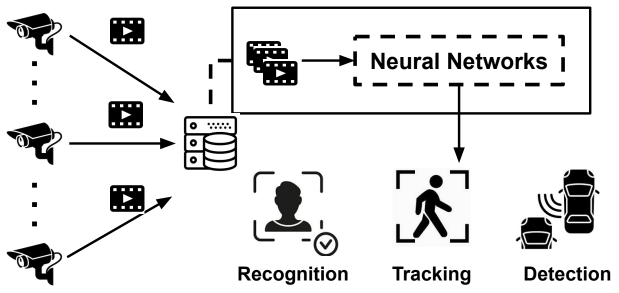
CHAPTER 1: INTRODUCTION

Recent advancements in machine learning (ML) have heralded unprecedented success across various domains, including computer vision, natural language processing, and virtual reality, among others. These achievements have been propelled by the utilization of advanced machine learning models, such as deep neural networks (DNNs). To harness the full potential of these sophisticated models, researchers and engineers have developed large-scale distributed systems. These systems are designed to train and deploy DNNs for a myriad of applications. For instance, DNN-powered computer vision solutions have been extensively implemented in numerous cities, utilizing real-time video data from thousands of surveillance cameras to enhance public safety and contribute to societal well-being [1, 2, 3, 4, 5, 6, 7].

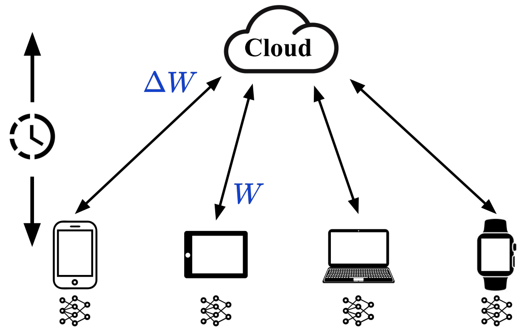
Prominent examples include Google and Meta, which have rolled out vast systems across millions of devices worldwide to refine their keyboard query suggestion and social network recommendation applications [8, 9], showcasing the broad applicability and impact of these technologies. The architecture of such machine learning systems typically encompasses a wide array of heterogeneous hardware, including cameras, edge devices, and cloud clusters, all interconnected through networks. From a functional perspective, these systems can be classified into two main categories: (a) *inference systems*, and (b) *training system*.

(a) **Inference system:** Inference system is a serving system where a well-trained model is deployed to generate inference results. For example, real-time video analytic system [10, 11, 12, 13, 14, 15, 16, 17, 18, 19] is a typical inference system. Figure 1.1(a) presents a typical architecture for video analytics systems. Cameras and edge devices are responsible for data collections. The videos collected at the edges are streamed to the cloud server through networks, where the DNN models are deployed to generate ML-based video analytical results for various computer vision tasks, such as human recognition, object detection & tracking and beyond.

(b) **Training system:** The training system is designed for collecting data to either initiate the training of a machine learning model from the ground up or to enhance the performance of an existing model through fine-tuning. Among these, federated learning (FL) systems have emerged as a focal point of recent studies [20, 21, 22, 23, 24]. As illustrated in Figure 1.1(b), federated learning architecture enables millions of end devices to use their private data for the collaborative training of a unified model



(a) Video Analytics System



(b) Federated Learning System

Figure 1.1: Examples of Machine Learning Systems.

with a cloud server. This approach addresses privacy concerns by allowing only the transmission of local model training gradients to the cloud, rather than the training data itself. This gradient exchange continues through multiple rounds until a model with satisfactory performance is achieved.

The remarkable progress in machine learning, both in inference and training systems, has unlocked capabilities previously deemed unattainable. However, this advancement comes at a cost: it generates a significant increase in data traffic and imposes a massive computational workload. These challenges are particularly pronounced at scale, where the sheer volume of data and the complexity of computations can strain resources, leading to inefficiencies that hinder the broader application and effectiveness of machine learning systems.

In summary, while the rapid evolution of machine learning systems has brought significant benefits to society, it also introduces new challenges in terms of resource efficiency, particularly at a large scale. This thesis aims to delve into these challenges, proposing methods and solutions to address the resource constraints faced by large-scale machine learning systems, ensuring their sustainable and efficient operation.

1.1 MOTIVATION AND CHALLENGES

The conventional design of large-scale machine learning systems, as depicted in Figure 1.1, faces significant challenges stemming from their substantial resource demands. For instance, video analytics systems necessitate considerable network bandwidth to stream real-time video data to cloud-based infrastructures for processing. Concurrently, cloud servers encounter limitations in computational capacity due to the voluminous requests for video analysis. These challenges are further exacerbated in federated learning systems, where the burden of intensive model training workloads and the bandwidth-intensive task of exchanging parameters are placed on mobile and edge devices. This can lead to rapid depletion of device power. Additionally, the diversity in hardware across the federated learning network introduces another layer of complexity, as devices with slower processing capabilities can bottleneck the overall system efficiency and delay the training process.

Indeed, the resource challenges faced by these machine learning systems are diverse and include *network bandwidth exhaustion*, *computational intensity*, *energy consumption*, *hardware heterogeneity*, and the *potential degradation of model performance*. Often, these challenges are interrelated, compounding the difficulty of managing resources efficiently. For instance, video analytics systems may simultaneously encounter network and computational constraints, amplifying the resource challenge. To better scale the power of machine learning models, such systems must be jointly optimized from multiple perspectives to remedy the resource intensive challenges. In this thesis, Wepresent a multi-objective resource optimization solution, aiming to enhance the efficiency of machine learning training and inference systems across the board.

1.2 OPTIMIZATION OPPORTUNITIES IN ML SYSTEMS

Achieving resource efficiency in machine learning (ML) systems is a complex challenge that necessitates a holistic approach to tackle the multifaceted, resource-intensive issues inherent in these systems. By exploring and integrating the following various optimization opportunities, it is possible to significantly enhance the efficiency of ML systems.

- The first opportunity lies in addressing the **inference data redundancy**. In many cases, data, particularly video, inherently contains significant redundancy that can be exploited for optimization. For instance, in video analytics systems, reducing frame rates and resolutions has been shown to effectively save network bandwidth and enhance system efficiency without compromising analytical outcomes [6, 25]. Furthermore, in systems analyzing multiple video streams, leveraging cross-camera data redundancy can further diminish data intensity, showcasing the potential for substantial efficiency improvements in broader contexts [5, 26].

- The second opportunity revolves around **hardware configurability**. Modern computing hardware offers various power configurations to accommodate different workloads, presenting a viable pathway to improve energy efficiency. Techniques like dynamic voltage frequency scaling (DVFS) exemplify how adjusting the power and performance settings of hardware components, such as CPUs and GPUs, can lead to significant energy savings in ML tasks [27, 28, 29, 30, 31]. Optimizing hardware configurations through strategies like DVFS enables a balance between operational speed and energy consumption, crucial for sustainable ML operations [32, 33].

- The third opportunity focuses on the **training data redundancy**. The vast quantities of data typically required for training ML models often contain redundant information, which, if effectively managed, can reduce the necessary data volume without impacting training efficacy. Techniques such as Coreset methods have proven effective in this regard. By selecting a representative subset of the original dataset, these methods retain critical information while substantially lowering both computational complexity and memory requirements. This approach has been successfully applied across various domains, including image classification, natural language processing, and federated learning. [34, 35, 36, 37, 38, 39].

In summary, by systematically addressing these optimization opportunities, it is feasible to construct ML systems that are not only resource-efficient but also maintain, or even enhance, their performance and effectiveness.

1.3 THESIS STATEMENT

In this thesis, We claim the following argument is true that *The design of resource-efficient large scale machine learning systems must consider the multi-objective optimizations of network bandwidth, computation throughput, execution latency, energy consumption and model accuracy via opportunities of data redundancy and hardware configurability*.

1.4 ROADMAP OF RESEARCH

In this thesis, We demonstrate a framework that describes how we utilize the aforementioned optimization opportunities to solve the resource efficiency challenges in machine learning systems, as shown in Figure 1.2. In the remainder of this section, We will present a roadmap of the thesis research, which focuses on resource-efficient optimizations for video analytics systems and federated learning systems.

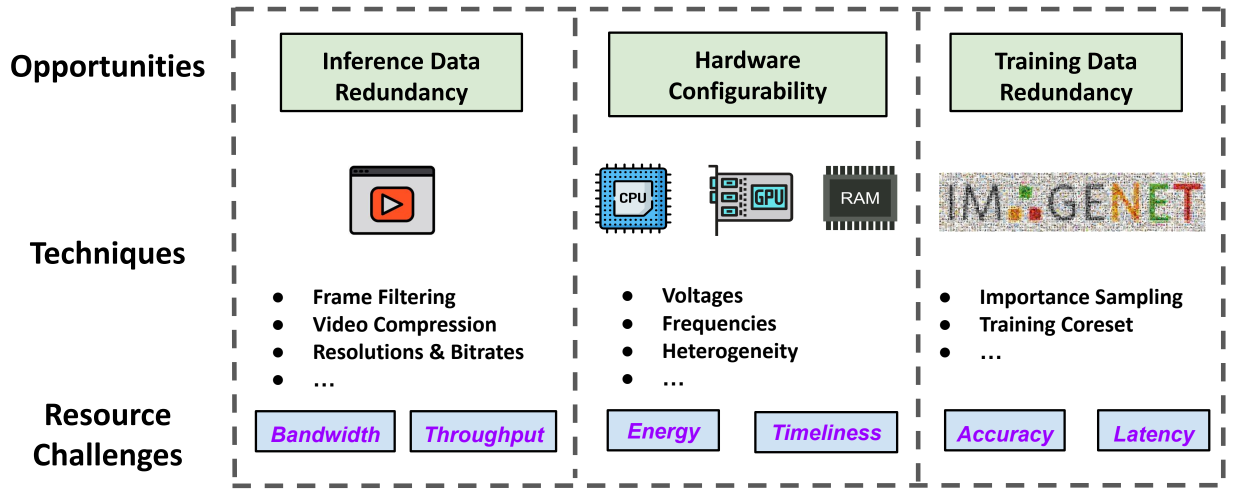


Figure 1.2: Overview of Thesis Works.

1.4.1 CrossRoI: Towards Resource-Efficient Real Time Video Analytics at Scale

Video cameras are pervasively deployed in city scale for public good or community safety (i.e. traffic monitoring or suspected person tracking). However, analyzing large scale video feeds in real time is data intensive and poses severe challenges to today's network and computation systems.

In the first part of this thesis (Figure 1.2, left), We present CrossRoI, a resource-efficient system that enables real time video analytics at scale via harnessing the videos content associations and redundancy across a fleet of cameras. CrossRoI exploits the **inference data redundancy** of cross-camera viewing fields to drastically reduce the **network bandwidth** and improve **computational throughput**.

CrossRoI removes the repentant appearances of same objects in multiple cameras without harming comprehensive coverage of the scene. CrossRoI operates in two phases - an offline phase to establish cross-camera correlations, and an efficient online phase for real time video inference. Experiments on real-world video feeds show that CrossRoI achieves $42\% \sim 65\%$ reduction for network overhead and $25\% \sim 34\%$ reduction for response delay in real time video analytics applications with more than 99% query accuracy, when compared to baseline methods. If integrated with SotA frame filtering systems, the performance gains of CrossRoI reaches $50\% \sim 80\%$ (network overhead) and $33\% \sim 61\%$.

1.4.2 BoFL: Bayesian Optimized Energy-Efficient Federated Learning

Federated learning is a machine learning paradigm that enables a cluster of decentralized edge devices to collaboratively train a shared machine learning model without exposing users' raw data. However, the intensive model training computation is energy-demanding and poses severe challenges to end devices' battery life.

In the second part of this thesis research (Figure 1.2, middle), we present BoFL, a training pace controller deployed on edge devices. It leverages **hardware configurability** to adjust chips' frequencies, aiming for **energy-efficient** federated learning. This approach ensures learning is both efficient and accomplished in a **timely manner**.

BoFL operates in an *explore-then-exploit* manner within limited rounds of FL tasks. BoFL first explores the large hardware frequency space strategically with a tailor designed Bayesian optimization algorithm to find a set of good operational configurations within few task training rounds. BoFL then exploits these configurations in the remaining rounds to achieve minimized energy consumption for model training. Experiments on multiple edge devices with different FL tasks suggest that BoFL can reduce energy consumption of model training by around 26%, and achieve near-optimal energy efficiency.

1.4.3 FedCore: Straggler Free Federated Learning with Distributed Coresets

In federated learning, the distributed nature of model training across client devices is often impeded by the straggler effect, where slower clients delay the overall process. In the third part of this thesis research (Figure 1.2, right), we introduce FedCore, a novel algorithm that addresses the straggler issue using distributed coresets, representative subsets of training data on each client. We show that FedCore, by utilizing the **training data redundancy**, can greatly reduce the federated learning training **latency** without degrading the machine learning **models' accuracy**.

FedCore addresses FL's heterogeneity by creating coresets that adapt to evolving models, effectively reducing the data processed by slower clients. This solution is integrated seamlessly into FL systems with minimal overhead, transforming coreset optimization into a tractable k-medoids clustering problem. Theoretical analysis validates FedCore's convergence, while practical evaluations demonstrate an **8x** reduction in FL training time without sacrificing model accuracy. FL training time, without compromising model accuracy.

1.5 THESIS ORGANIZATION

In the following chapters, We will introduce the the technical details of each of the four aforementioned works. To be more specific,

- Chapter 2 delves into the CROSSROI framework, illustrating the utilization of *inference data redundancy* to mitigate network and computational demands in video analytics systems.
- Chapter 3 explores the BOFL framework, explaining how *hardware configurability* enhances energy efficiency and timeliness in federated learning applications.
- Chapter 4 outlines the FEDCORE framework, presenting algorithms and theoretical insights on addressing *training data redundancy* to alleviate straggler problems in federated learning systems without compromising model performance.

Finally, in Chapter 5, We conclude the thesis and provide some future research directions for multi-objective resource optimizations for large scale machine learning systems.

CHAPTER 2: CROSSROI: CROSS-CAMERA REGION OF INTEREST OPTIMIZATION FOR EFFICIENT REAL TIME VIDEO ANALYTICS AT SCALE

In this chapter, we delve into the intricacies of the first thesis work, CROSSROI. We lay out a multi-objective resource optimization framework designed to **reduce network communication** and **enhance server computational throughput** for real time video analytics systems. This is achieved by capitalizing on the **inference data redundancy** opportunity, as detailed in Section 1.2.

2.1 INTRODUCTION

Driven by plummeting camera prices and advances of intelligent video inference algorithms, video cameras are being deployed ubiquitously in recent days. For example, many cities in the world now deploy tens of thousands of cameras at key locations, such as highway entrances or roads intersections, to collect rich video data for applications ranging from traffic monitoring, public safety and suspected target tracking [1, 2, 3, 40, 41, 42, 43]. As tremendous data are being generated by the cameras in every second, organizations usually rely on live video analytics to retrieve key information, such as objects locations and identities, in real time. Two key enablers for fast and accurate video inference are the rapid development of deep neural networks, especially Convolutional Neural Networks (CNNs), and their hardware accelerators which empower fast and large-scale neural networks training and inference.

However, live video analytics in large scale are usually *network-exhaustive* and *compute-intensive*. In a typical video analytics pipeline, real time video feeds from widely deployed cameras are streamed to a cloud server or geographically close edge clusters where powerful hardware (e.g. GPUs) and fine-trained CNNs (e.g. YOLO [44]) are prepared. The server immediately loads videos into the inference pipelines and aims for accurate and low latency analytic results. The high network demands for video streaming and computation demands for CNN-based inference pose severe challenges to such video analytics framework, especially when organizations are steadily increasing their deployment scale, which amplifies the problems.

Significant work has been presented to improve the efficiency of video analytics pipelines, which can be categorized into two groups: (1) frame filtering on single camera [6, 10, 11, 12, 45], and (2) target oriented cross-camera analytics [4, 5, 13, 46, 47]. Works in group one (e.g. Reducto [6]) optimize the cost/accuracy tradeoffs of single-video analytics with frame sampling or CNN-based filters for discarding frames. Reductos' optimiza-

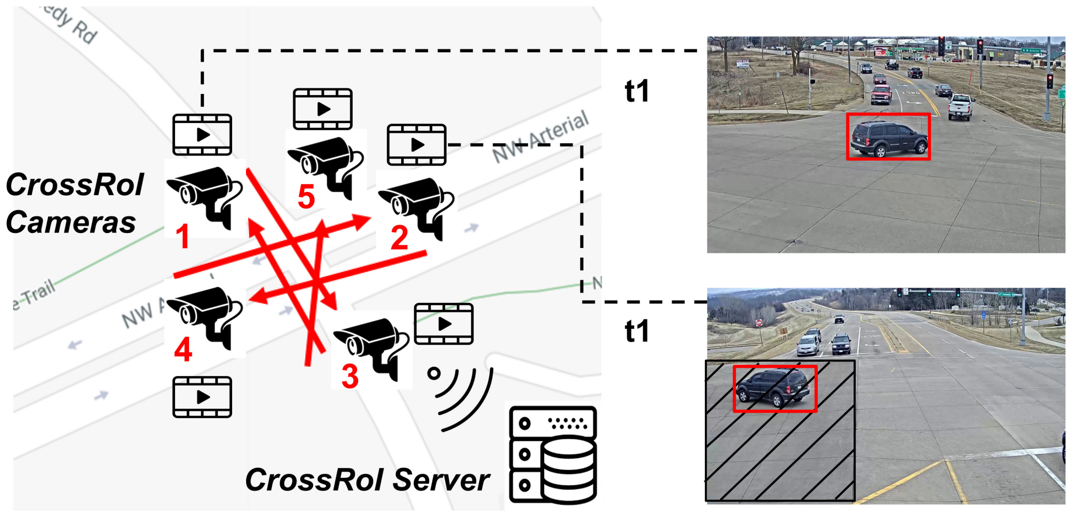


Figure 2.1: An application scenario of CROSSROI. Red arrows show the viewing angles of cameras.

tions are within single video streams independent of other streams, resulting in linear growth of resource demands and limitations to scaling. Works in group two (e.g. Spatula [5]) schedule the on and off of geographically distributed cameras to track a predefined target object across cameras. While Spatula substantially reduces network/computation demands by turning off the majority of cameras at any time, it fails to provide a comprehensive coverage to every scene where cameras are deployed.

In this work, we present CROSSROI to address the resource-intensive challenges for real time video analytics on a fleet of closely located cameras (e.g. the cameras installed at a traffic intersection) via harnessing the video content associations and redundancy across the group of cameras. As shown in Figure 2.1, 5 cameras are deployed at a road intersection with their viewing field overlapped. An object in the scene may appear in the *field-of-view* of multiple cameras at the same time. In many video analytics tasks (e.g. vehicle or suspect person detection), any capture of the interesting target is effective to fulfill the mission. For example in Figure 2.1 either detection of the black car in camera 1 or 2 is enough to locate it at this traffic crossing at this moment (t_1). Removing the lower left region (shadow region) of camera 2's frame at t_1 does not influence the comprehensiveness of vehicle detection results at all [1]. We argue that both network traffic and computation demands can be substantially reduced without harming inference ac-

[1]Different applications may have different requirements to define a detection as effective. For example in a vehicle plate detection scenario, only the detection of the front or the back view is effective. In this chapter, we assume an application that a detection from any viewpoint is sufficient to fulfill the mission. However, our system can be easily scaled to other scenarios given clear effectiveness definition, i.e., we only take the front/back views into our system in the vehicle plates detection scenario.

curacy for video analytics pipelines if the intrinsic associations across cameras could be discovered and harnessed properly.

CROSSROI highlights three **challenges** to discover the intrinsic data associations and harness the content redundancy across multiple cameras as follows.

(**C1**) How to establish data association among a fleet of cameras on unlabeled video data *automatically* and *accurately*?

(**C2**) How to calculate cross-camera region-of-interest (RoI) collectively to *remove redundancy* without harming the *comprehensiveness* of detection coverage?

(**C3**) How to leverage cameras' regions-of-interest to drastically *reduce network overhead* and *boost sever inference throughput* in the video analytics pipeline?

To tackle these challenges, we design CROSSROI to operate in two distinct phases-an offline phase and an online phase. In the offline phase, CROSSROI establishes the data association and calculates the optimized RoI information. In the online phase, cameras filter their real time video streams according to the RoI information to reduce overall system data intensity. To establish cross-camera data association, we augment existing re-identification solutions with statistical filters to generate highly-accurate ReID results, and hence, use the cross-camera appearances of same objects to represent data correlations among cameras (**C1**). We slice camera frames into fine-grained tiles and develop a combinatorial optimization framework to calculate least-sized regions of interest among the camera fleet collectively without missing detection of any object (**C2**). To best alleviate resource-intensive challenges, we apply the optimized RoI information in each camera as a filter to prevent non-interesting data being dumped into the analytics pipeline. We further specially design video compression module and RoI based CNN inference pipeline to boost overall system performance(**C3**). Overall, this thesis chapter makes the following **contributions**:

1. We augment existing re-identification (ReID) algorithms to establish cross-camera data association automatically and accurately.

2. We develop an multi-objective optimization framework to harness cross-camera data redundancy and significantly reduce the data intensity of the video analytics pipeline.

3. Our specially designed video compression module and RoI-based CNN inference pipeline boost the overall system performance even further.

4. Evaluations on real-world traffic videos suggest our system achieves network overhead reduction up to 65% and end-to-end response latency reduction by 34% compared to baselines.

5. Compared to most frame-filtering based existing solutions, e.g., Reducto [6], CROSSROI exemplifies an extra layer of optimization from spatial domain. When integrated with frame-filtering module, CROSSROI outperforms original frame-filtering systems by $2\times$, and outperforms baselines up to $5\times$, in terms of network usage.

The rest of the chapter is organized as follows. In Section 2.2, we survey related literature and present backgrounds about video analytics, streaming and ReID frameworks. In Section 2.3, we present our ReID based cross-camera data associations and optimization framework to generate RoI masks for each camera. We present CROSSROI system workflow and design details in Section 2.4. Section 2.5 shows our evaluations of CROSSROI system. Finally, Section 2.6 concludes the chapter.

2.2 BACKGROUNDS AND RELATED WORK

2.2.1 Multi-Objective Optimizations on Video Analytics Systems

Video analytics systems have been widely studied in recent literature. [4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 25, 48, 49, 50]. While all these works focus on solving the resource-intensive challenge, different approaches have been proposed. We categorize all existing works in terms of (1) their system architectures, (2) capabilities to fulfill real-time processing, and (3) processing multiple-camera videos independently or collectively, as follows.

Most works fells in either a three-layered *camera-edge-cloud* [5, 7, 11, 12, 16, 17, 48, 50] or a two-layered *camera-cloud* [6, 10, 15, 25, 51] architecture. The first class of work, exemplified by Focus [12], deploys close-to-camera edge devices to augment the processing power of cameras, and hence, prune redundant data using neural networks accurately before sending videos to the cloud for deep analysis. While the two-layered works, i.e. Glimpse [10], use heuristics and lower level features to remove video redundancy, which fits better into current real-word deployment where cameras are usually cheap and the edge servers are not available.

Real time video analytics systems [5, 6, 10, 14, 25] optimize the whole pipeline, including camera processing delay, network overhead and server inference latency, to reduce end-to-end respond time for the inference tasks. For example, Reducto [6] assigns tiny workload to the cameras to avoid exaggerating camera processing delay, and hence, fulfill real time missions. Non-real-time systems [12, 13] try to answer "after the fact" types queries from large scale stored videos. The latter class of systems usually focus on the efficiency of key frame searching and high inference throughput.

The majority of video analytics systems are designed to process video streams independently [6, 10, 11, 12]. All optimization and redundancy pruning are within a single video stream, which leads to its linear growth resource requirements. The other systems [4, 5, 13] focus on the cross-camera analytics on a group of cameras. But they either fail to achieve real time inference, i.e. Caesar [13], or fails to provide comprehensive coverage to the surveillance scene, i.e. Spatula [5].

Different from all existing works, CROSSROI achieves real-time cross-camera video analytics over a fleet of cameras and fulfills comprehensive scene coverage. CROSSROI fits into a two-layered architecture which only assumes normal surveillance cameras without the needs of advanced edge devices.

2.2.2 Classic & Tile-Based Video Compression

The vanilla video compression standard, i.e. H.264 AVC [52] and HEVC [53], are widely applied to significantly reduce data sizes in video storage/streaming applications. These compressors usually encode videos with two steps. (1) The encoders first split every frame into many small pixel *blocks* (for example, $16\ pixels \times 16\ pixels$ block size for H.264 standard). For every block in a video frame, the compressor searches similar blocks either within the already-encoded portion of current frame or in nearby frames that are buffered by the encoder. When a closely matched block is identified, it encodes the position of this similar block in a motion vector. (2) The encoder calculates the pixels level difference between current block and the reference block, and encodes this sparse residual difference with quantization and entropy encoding. Video compression efficacy can be impacted by the frame size in such codecs. For example, a block has more reference block options when the frame size is large, and hence, more easily get encoded into space-efficient motion vector and sparse residual difference.

Tile-based video compression are widely used in data-intensive applications, for example, panoramic video streaming. [54, 55, 56, 57] Tile-based video encoder splits the whole frame spatially into several rectangular *tiles*. Every tile of the video is processed independently by a classic video compressor (i.e. H.264) and can be encoded into different qualities. For example, a compressor can encode the region-of-interest parts of a video with high bitrates and the other parts with low bitrates. While tile-based compression reduces video size through the semantic region-of-interest information, splitting a large video into several smaller ones degrades the overall efficacy of the compressors.

In CrossRoI, we applied tile-based video compression to include only the interesting regions of the surveillance videos. To alleviate the compression efficacy degradation, we design a tile grouping algorithms to merge small tiles into larger ones, which reduces network overhead even further compared to existing tile-based approaches (Section 2.4.4).

2.2.3 Computer Vision Based Object Re-Identification

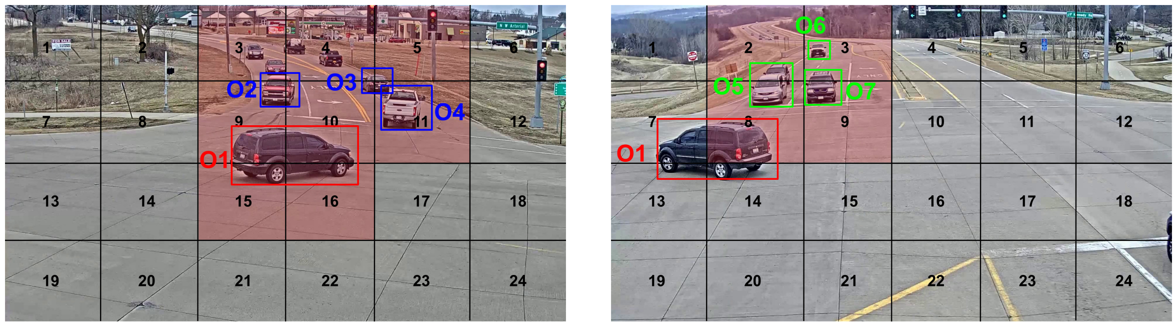
CrossRoI establishes the cross-camera region associations through profiling object re-identification (ReID) results among the group of cameras. ReID is a challenging problem in computer vision [58, 59, 60]. A typical ReID pipeline starts with automatic object detection with object detectors, i.e. YOLO [44], FasterRCNN [61] and SSD [62]. ReID algorithms then extracts deep image features from the detected objects and computes the similarity of two detection based on their feature distance [62, 63, 64, 65]. Some works [63, 64] apply object movement patterns as spatial-temporal cues to further improve the identification accuracy. Although many ReID algorithms are proposed, the ReID results are still not perfect, especially in crowded scenes and large camera networks where ablations and significantly different lighting conditions and viewing angles are common. Different from computer vision communities, we do not reinvent new ReID algorithms in this chapter, but apply statistical filters to augment existing ReID algorithms to obtain highly-confident region associations from error prone ReID results (Section 2.4.3).

2.3 CROSSROI BASIC MODELS, CONCEPTS AND PROBLEM DEFINITION

The CrossRoI system has two entities, which are CrossRoI *cameras* and CrossRoI *server*. CrossRoI cameras are the video providers. They capture videos of the mission scene and transmit them back to CrossRoI server for inference and analytics, e.g., car detection and counting. The workflow of CrossRoI contains an *offline phase* and an *online phase*. In offline phase, the CrossRoI server collects synchronized video clips from each CrossRoI camera. Through profiling and analyzing these clips, the server can calculate optimal RoI masks for the cameras. These RoI masks will then be applied in the online phase to reduce network overhead and boost inference throughput at the CrossRoI server. In the rest of this section, we focus on the offline video profiling process of CrossRoI to explore the following two questions:

- How to establish the data associations between multiple cameras covering the same scene?
- How to calculate the optimal RoI masks for these cameras without losing any interesting object?

2.3.1 CrossRoI Data Model



(a) C_1 (b) C_2

Figure 2.2: Video captures from two different cameras at timestamp t_1. Each frame is divided into 24 tiles. Red shadow represents the optimized RoI masks generated for these two cameras based on profiling on t_1 only.

We consider a CrossRoI system containing N cameras, named $C_1, C_2, \dots, C_N$. In the offline phase, the CrossRoI server collects synchronized video clips from all the CrossRoI cameras for profiling. "Synchronized" here refers that all N video clips have the same frame rate f, start at the same time[2] t_1, as well as the same video length. The k-th frame of every video clip is then corresponding to the same timestamp $t_k = t_1 + \frac{k-1}{f}$, for any k within the length of the videos. In this manner, the frames from CrossRoI cameras with the same indices are just image captures of the same scene at the same time from different perspectives. We further define the profile time window being a list of discrete timestamp $\mathcal{T} = \{t_1, t_2, \dots, t_L\}$, where t_i is the timestamp of the i-th frames in the video clips and L is the index of the last frame.

In order to study the fine-grained data associations between cameras, we further cut every video into *tiles*. Tiles are smaller rectangular spatial regions which cumulatively cover the whole frame. As shown in Figure 2.2(a), the whole frame area of C_1 is divided into 24 tiles indexed from 1 to 24 in an top-to-bottom, left-to-right order. We formally

[2]We consider two cross-camera timestamps as the same if their difference is small enough for frame alignment, i.e. $< \frac{1}{2f}$, which can be achieved by NTP protocol.

Timestamps	Detected Objects	Appearance Regions
t_1	$\mathcal{O}_{t_1} = \{O_1, O_2, O_3, O_4, O_5, O_6, O_7\}$	$\mathcal{R}^1_{t_1} = \{\{\mathcal{G}_{1,9}, \mathcal{G}_{1,10}, \mathcal{G}_{1,15}, \mathcal{G}_{1,16}\}, \{\mathcal{G}_{2,7}, \mathcal{G}_{2,8}, \mathcal{G}_{2,13}, \mathcal{G}_{2,14}\}\}$ $\mathcal{R}^2_{t_1} = \{\{\mathcal{G}_{1,3}, \mathcal{G}_{1,4}, \mathcal{G}_{1,9}, \mathcal{G}_{1,10}\}\}$ $\mathcal{R}^3_{t_1} = \{\{\mathcal{G}_{1,4}, \mathcal{G}_{1,5}, \mathcal{G}_{1,10}, \mathcal{G}_{1,11}\}\}$ $\mathcal{R}^4_{t_1} = \{\{\mathcal{G}_{1,11}\}\}$, $\mathcal{R}^5_{t_1} = \{\{\mathcal{G}_{2,2}, \mathcal{G}_{2,8}\}\}$ $\mathcal{R}^6_{t_1} = \{\{\mathcal{G}_{2,3}\}\}$, $\mathcal{R}^7_{t_1} = \{\{\mathcal{G}_{2,3}, \mathcal{G}_{2,9}\}\}$
$\cdots$	$\cdots$	$\cdots$

Table 2.1: Cross-camera association lookup-table for the example in Figure 2.2.

define $\mathcal{G}_i$ as the set of tiles for camera C_i, where $1 \le i \le N$.[3] The j-th tile of C_i can then be referred as $\mathcal{G}_{i,j}$. For example, the left top tile of C_1 in Figure 2.2(a) is $\mathcal{G}_{1,1}$. It is worth mentioning that a tile is not corresponding to any specific frame or timestamp. Tiling is a spatial description of how we divide the field of views of cameras into finer granularity.

In CROSSROI, we define *RoI mask* as the region in camera frames that may contain interesting objects, for example, vehicles or trucks. The regions outside of a RoI mask are ignored in the video analytics pipeline because no interesting targets may appear in these areas. In our system, a tile is the smallest spatial unit to constitute a RoI mask. The RoI mask for camera C_i, denoted as $\mathcal{M}_i$, is a subset of all its tiles, i.e., $\mathcal{M}_i \subset \mathcal{G}_i$. For example, in Figure 2.2(a), if we want RoI region to only include the four detected cars, then the minimum-sized RoI mask will be as follows.

$$\mathcal{M}_1 = \{\mathcal{G}_{1,3}, \mathcal{G}_{1,4}, \mathcal{G}_{1,5}, \mathcal{G}_{1,9}, \mathcal{G}_{1,10}, \mathcal{G}_{1,11}, \mathcal{G}_{1,15}, \mathcal{G}_{1,16}\} \tag{2.1}$$

2.3.2 Cross-Camera Regions-Association Concept

We establish cross-camera region associations based on existing object re-identification (ReID) algorithms, which take visual or geographical features to associate common objects across multiple frames/ cameras. ReID algorithms assign an ID for every detected object. Detection of the same object across different cameras will be assigned to the same ID. For example, Figure 2.2(a) and Figure 2.2(b) are two synchronized frames from C_1 and C_2, respectively. Every detected car in both frames is assigned an ID by the ReID algorithm. Cars O_2, O_3, and O_4 are unique to C_1. Cars O_5, O_6, and O_7 are unique to C_2. While car O_1 appears at the overlapping region of both views and can be identified in C_1 and C_2 simultaneously.[4]

[3] In this chapter, we use "tiles of C_i" and "tiles of the video generated by C_i" interchangeably.

[4] We only show detection of large objects in the two frames in Figure 2.2 for clarity of illustration.

We establish the cross cameras association by profiling the ReID results over the whole time window $\mathcal{T}$. At any timestamp t_m, we record the following two elements:

- All objects being detected at this timestamp, denoted as $\mathcal{O}_{t_m}$,
- The *appearance regions* for each object being detected at this timestamp.

We define the *appearance region* of an object O_k on camera C_i at timestamp t_m as the least set of tiles that can cover O_k, denoted as R^k_{i,t_m}. As the object may appear on multiple cameras simultaneously, we further define the *appearance regions* of O_k at timestamp t_m, denoted as $\mathcal{R}^k_{t_m}$, as the collection of its appearance regions over all CrossRoI cameras, s.t.,

$$\mathcal{R}^k_{t_m} = \{R^k_{i,t_m} | 1 \leq i \leq N \text{ and } R^k_{i,t_m} \neq \emptyset\} \tag{2.2}$$

Take Figure 2.2 as an example. There are seven objects being detected at t_1 in total, s.t., $\mathcal{O}_{t_1} = \{O_1, O_2, O_3, O_4, O_5, O_6, O_7\}$. O_1 appears in both frames, therefore its appearance regions contains two elements, s.t.,

$$\mathcal{R}^1_{t_1} = \{\{\mathcal{G}_{1,9}, \mathcal{G}_{1,10}, \mathcal{G}_{1,15}, \mathcal{G}_{1,16}\}, \{\mathcal{G}_{2,7}, \mathcal{G}_{2,8}, \mathcal{G}_{2,13}, \mathcal{G}_{2,14}\}\} \tag{2.3}$$

The other objects appear only once and thus have single-length appearance regions, e.g. $\mathcal{R}^5_{t_1} = \{\{\mathcal{G}_{2,2}, \mathcal{G}_{2,8}\}\}$. Profiling through the whole time window, we can build a lookup-table which ensembles the ReID based region associations, as shown in Table 2.1.

ReID algorithms are still not perfect. In order to achieve accurate region associations based on the error prone ReID results, we apply statistical filters on the raw ReID results to obtain highly-confident ReID results and establish the region association with the selected data instead.

2.3.3 RoI Masks Optimization

The optimization objective is to *include least number of tiles into the RoI masks cumulatively across all the N cameras*. In order to avoid missing any object at any timestamp in the time window, we set the optimization constraints as *any object occurred at timestamp t_m has at least one appearance region included by the RoI masks, for any $t_m \in \mathcal{T}$*. We define variable $\mathcal{M}$ as the union set of all the N RoI masks, s.t. $\mathcal{M} = \cup_{i=1}^{N} \mathcal{M}_i \subset \cup_{i=1}^{N} \mathcal{G}_i$. The optimization problem can be formally presented as follow:

$$\min \quad |\mathcal{M}| \tag{2.4}$$

$$\text{s.t.} \sum_{R \in \mathcal{R}^k_{t_m}} \big(\mathbb{1}(R \in \mathcal{M})\big) \geq 1, \qquad \forall\, t_m \in \mathcal{T},\ \forall\, k,\ \text{s.t.},\ O_k \in \mathcal{O}_{t_m} \tag{2.5}$$

Note that $\mathbb{1}(\cdot)$ in equation 2.5 is an indicator function with its range being $\{0, 1\}$. It will return 1 if the input condition, e.g., $R \in \mathcal{M}$ in equation 2.5, is true and return 0, otherwise. Solving the above combinatorial optimization will generate the optimal RoI masks which include least number of tiles while ensuring every object being detected. In the example of Figure 2.2, if we set the time window to include t_1 only, the optimized RoI masks $\mathcal{M}$ will be $\{\mathcal{G}_{1,3}, \mathcal{G}_{1,4}, \mathcal{G}_{1,5}, \mathcal{G}_{1,9}, \mathcal{G}_{1,10}, \mathcal{G}_{1,11}, \mathcal{G}_{1,15}, \mathcal{G}_{1,16}, \mathcal{G}_{2,2}, \mathcal{G}_{2,3}, \mathcal{G}_{2,8}, \mathcal{G}_{2,9}\}$, which are shown in the Figures with pink shadow. All the appearance of $O_2, \ldots, O_7$ are covered by the RoI masks. As O_1 appears simultaneously on both cameras, the algorithms will only include one of its appearance regions that introduce least overheads, e.g., its appearance region in C_1 in this example. The optimized RoI masks will then be applied in the online phase to boost overall system performance.

2.4 CROSSROI DESIGN AND IMPLEMENTATION

As mentioned in Section 2.3, the CrossRoI system has an offline phase and an online phase. In offline phase, the server generates optimal RoI masks for each camera through profiling synchronized video clips. In online phase, the server runs video analytics tasks, e.g. car detection, on the streams from CrossRoI cameras in real time, where the RoI masks serve as a guidance to reduce network burden and boost server throughput. Figure 2.3 depicts the high-level framework of CrossRoI. We show more details of the workflow as follows.

2.4.1 Offline Phase

Offline Server Re-Identification ①. The CrossRoI server first applies re-identification algorithms over several minutes of synchronized raw videos collected from all the CrossRoI cameras to characterize the view relations among different cameras. We choose DiDi-MTMC [63] algorithm as the CrossRoI server ReID module, which integrates vision features together with geographic information to achieve best accuracy on our experiment dataset (more descriptions about the dataset are presented in Section 2.5). At the end of

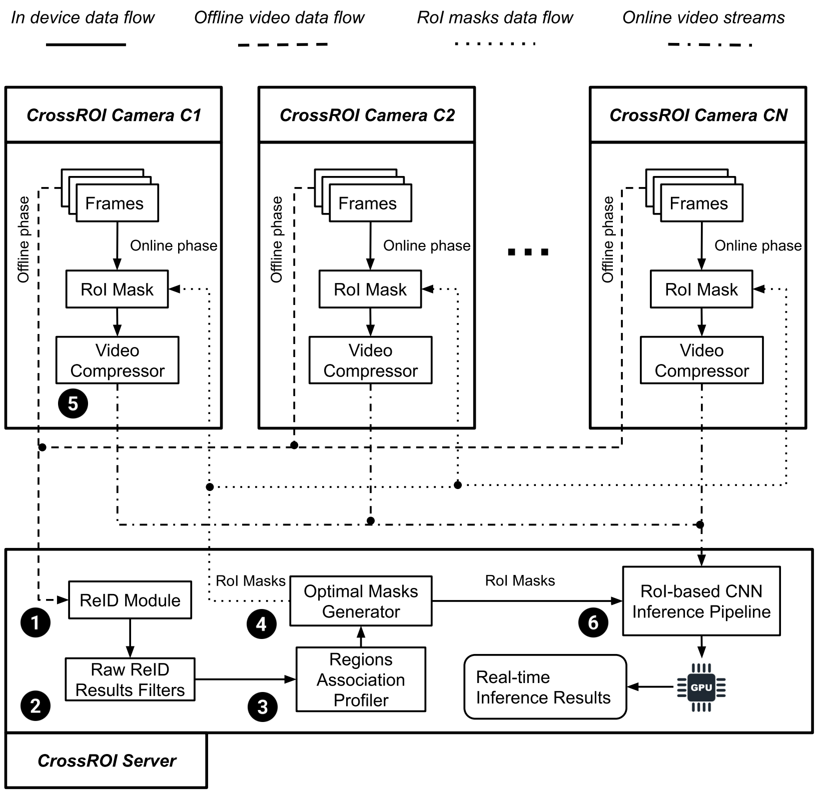


Figure 2.3: System Overview of CROSSROI

the re-identification step, every interesting object (cars and vehicles in our scenario) in every frame of the videos will be associated with a bounding box and an ID, in the form of (*left, top, width, height, id*). *Left* and *top* information locate the top left corner of the bounding box, while *width* and *height* information characterize the bounding box size. All four values are measured in terms of pixel(s). These ReID results will be further processed in modules ②, ③ and finally be used to generate the optimized RoI masks.

Raw ReID Results Filtering ②. Although DiDi-MTMC algorithm achieves state-of-the-art accuracy in object ReID, its ReID results still contain a lot of errors and mismatches. In order to establish accurate region associations among the CROSSROI cameras, we pass the ReID results through two tandem statistical filters to remove the "suspicious" ID assignments and only keep highly-confident ReID results. Specifically, we first apply a *regression filter* to decouple any two different objects that are mismatched together by the ReID algorithm, then apply an *SVM filter* to remove the "falsely isolated objects" in every frame. A "falsely isolated object" here refers to the case when the same object being assigned different IDs in different cameras or frames, then each of its appearance becomes a falsely isolated object. At the end of this step, we get a selected set of highly-confident

ReID results, which will be used in module ③ for cross camera region associations. It is worth mentioning that the two filters do not improve overall ReID accuracy compared to existing ReID algorithms. The design goal of statistical filters is to select a subset of highly-confident object identification results, which are effective to represent the cross-camera associations.

Regions Association & RoI Masks Generation ③, ④. Based on the filtered ReID results obtained in step ②, the server build a lookup table, e.g. Table 2.1, that ensembles the region associations across all the cameras over the whole profiling time window. The CROSSROI server then takes the table as data source into the optimization framework and generates the optimal RoI masks for every camera by solving the optimization problem, as described in Section 2.3. We use commercial optimization solver (i.e. Gurobi [66]) to obtain the optimal RoI masks in the offline phase. At the end of the offline phase, CROSSROI server sends the corresponding RoI mask to each CROSSROI camera. Hence, the cameras can use RoI masks to crop and further compress their video streams in online phase, as shown in ⑤. The CROSSROI server will also keep the RoI masks in memory and applies them onto the CNN inference tasks (e.g., YOLO object detection) to boost its execution speed, as shown in ⑥.

2.4.2 Online Phase

Online Phase Video Compression and Streaming ⑤. In the online phase, CROSSROI cameras stream their video feeds to the server in real time. The server runs neural network based inference algorithm on these video streams for the query tasks (e.g., vehicles detection or counting). In order to reduce the huge server side bandwidth consumption caused by receiving so many videos streams at the same time, all the CROSSROI cameras will (1) crop their videos and only stream the areas included by RoI masks, and (2) apply modern video compressor (e.g., H.264) to greatly reduce the video size.

Tile is the smallest spatial unit for tile-based video streaming. As our RoI masks are designed to be tile-based, it is natural for the CROSSROI cameras to stream videos in a tile-based manner, which (1) avoids transmitting non-interesting tiles and (2) can directly apply existing video compressors. However, a naive tile-based streaming may still be sub-optimal. As the compression efficacy of video compressors will drop significantly when being applied to small tiles. In order to best utilize the power of well-engineered video compressors, we develop a *tile grouping* algorithm which merges the fine-grained small tiles into larger ones, and hence, further reduces the videos size.

D	C_1				C_2				C_3				C_4				C_5			
S	**TP**	**FP**	**FN**	**TN**	**TP**	**FP**	**FN**	**TN**	**TP**	**FP**	**FN**	**TN**	**TP**	**FP**	**FN**	**TN**	**TP**	**FP**	**FN**	**TN**
C_1		\			335	253	263	7542	358	22	560	7453	162	15	336	7880	101	0	642	7650
C_2	333	253	291	4317			\		161	81	397	4551	242	56	401	4497	50	2	773	4371
C_3	358	22	977	8246	161	81	868	8558			\		434	40	951	8243	155	24	1871	7618
C_4	162	15	512	6784	242	56	917	6258	434	40	809	6190			\		138	22	1402	8583
C_5	101	0	694	8568	50	2	1074	8237	155	24	1552	7632	138	22	1328	7875			\	

Table 2.2: Characterization of raw ReID results. **S/D** represents the source/destination camera. For each pair of cameras, we count the number of identifications with four different *labels*, which are **TP, FP, FN, TN**, representing *true positive*, *false positive*, *false negative* and *true negative*, respectively.

RoI Based Real-Time CNN Inference ⑥. Once the tile streams are received by the CrossRoI server, they will be merged together to reconstruct frames. Note that the non-RoI regions of a frame will be empty (purely black) as the corresponding tiles are not streamed to the server. These recovered frames are then pushed into a RoI based CNN inference pipeline. In our system, we choose YOLO [44] as the inference handler for object detection task. Different from traditional object detection tasks, where an interesting target may appear everywhere in the whole frame, a RoI based object detection task has prior knowledge of the RoI regions, and thus, can greatly reduce the detection space (i.e. only run YOLO on the RoI regions). In CrossRoI system, we build an RoI-YOLO detector based on SBNet [67] which takes the RoI masks as cues to boost YOLO detection speed by 1.2x.

2.4.3 Raw RoI Results Analysis and Filtering (Offline)

Raw ReID Results Analysis We first present a comprehensive analysis towards the raw ReID results, which sheds light to our design of the two tandem statistical filters to remove "suspicious" ReID data points. We investigate pairwise ReID results between two different cameras to understand the structure of these raw results and where mistakes happen. In the study of a pair of cameras, we categorize any ID assignment of a *source camera*, into one of the two types, *positive* and *negative*, in terms of whether the detected object has an appearance in the *destination camera* at the same timestamp.

To better illustrate the concepts, we use C_1 and C_2 in Figure 2.2 as an example pair of cameras in the rest of this paragraph. We set C_1 as the source camera and C_2 as the destination camera. Every object being detected in this frame will be assigned a positive or negative label, i.e. O_1 is positive and O_2, O_3, O_4 are negative. In our example, every object is identified correctly. However, the ReID algorithm may make mistakes, e.g. assigning same ID to two different objects or assigning different IDs to multiple appearances of the same one. To illustrate the correctness of the identification, we further associate a correctness label, being either *true* or *false*, to each detected object in the source camera. Hence, there are four types of labels associated with all the identification results as follows:

- **True Positive (TP)**. A true positive label is assigned to an object in the source camera which has a corresponding appearance in the destination cameras, and these two appearances are given the same ID, e.g., O_1 in C_1 is a true positive data point.
- **False Positive (FP)**. A false positive label covers either of the following two cases: (1) a negative object being matched to an object in the destination camera, i.e. in case O_2 in C_1 and O_5 in C_2 being assigned the same ID, and (2) a positive object being matched to a wrong object in the destination camera, e.g., in case O_1 in C_1 being matched O_7 in C_2.
- **True Negative (TN)**. A true negative label refers to an object which has no appearance in the destination camera and that its identification is correct, e.g., O_2, O_3, O_4 in C_1.
- **False Negative (FN)**. A false negative label is corresponding to the case when a positive object being mistakenly identified as a negative object. For example, in case the ReID algorithms fails to find the appearance of O_1 in C_2 and assign different IDs to these two appearance of the same object.

Both false data (FP and FN) will sabotage the optimized RoI generation framework as mentioned in Section 2.3. Specifically, (1) the false positive data will make the generated RoI masks incorrect, as it introduces wrong region associations between cameras, and (2) the false negative will significantly degrade the efficacy of non-RoI tiles reduction, as we try to ensure every object has a least one appearance at any time. For example, if O_1 in C_1 and C_2 are assigned different IDs at t_1, we must include both $\{\mathcal{G}_{1,9}, \mathcal{G}_{1,10}, \mathcal{G}_{1,15}, \mathcal{G}_{1,16}\}$ and $\{\mathcal{G}_{2,7}, \mathcal{G}_{2,8}, \mathcal{G}_{2,13}, \mathcal{G}_{2,14}\}$ into the RoI masks forever no matter what identification happens in later timestamps.

To better understand the distribution of the above four types of ReID results. We profile a dataset containing synchronized videos from five traffic cameras watching the same crossing (we will describe more details about the dataset in Section 2.5). We compare DiDi-MTMC ReID algorithm to the ReID ground truth of the dataset and get the distributions of the pairwise ReID results as shown in Table 2.2. It can be observed that there are large amount of falsely identified cases, especially the false negative identifications which usually outweigh the total number of true/false positive samples. Applying raw

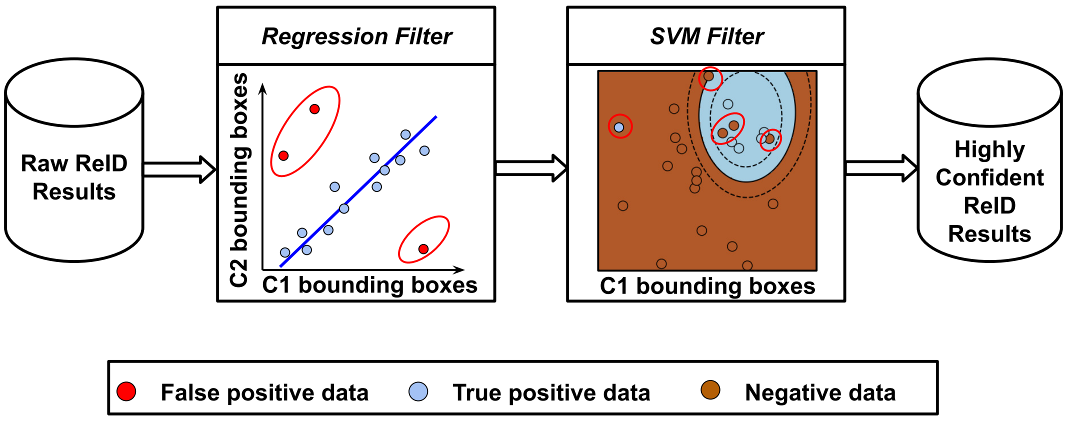


Figure 2.4: Statistical filters for raw ReID results. Outliers are circled out in red color.

ReID results to the optimization pipeline will definitely lose many optimization opportunities and degrade the system efficacy.

Although the raw ReID results are error prone, we make two important observations after close scrutiny of the application scenario and results distribution (Table 2.2), which can help remove the false ReID results significantly. The two **observations** are as follows:

(**O1**) The region-associations between two cameras have intrinsic physical relation. For example, the two appearances of O_1 at t_1 suggest that region $\{\mathcal{G}_{1,9}, \mathcal{G}_{1,10}, \mathcal{G}_{1,15}, \mathcal{G}_{1,16}\}$ in C_1 and region $\{\mathcal{G}_{2,7}, \mathcal{G}_{2,8}, \mathcal{G}_{2,13}, \mathcal{G}_{2,14}\}$ in C_2 are actually the same area in physical means. In any future frames, this mapping relation will also work.

(**O2**) In both positive and negative identifications of the ReID results, the number of true samples is always greater than that of false samples, and usually greater in several times or magnitudes.

Based on the above two observations, we decide to apply statistical filters to remove the false ReID results. Specifically, we design a *regression filter* to remove the false positive samples and a *SVM filter* to remove the false negative samples.

Regression Filter Design & Implementation As shown in Figure 2.4, we push raw ReID data through two tandem filters to get cleaned. The first filter is a regression filter. We dump all the positive results into a regression module to learn the intrinsic region mappings between a pair of cameras. We use regression method here for its reliable and successful applications to model correlations between a pair of dependent variables, e.g., appearances of same objects in source/destination cameras. The outliers of the trained model are regarded as false positive samples and will be rectified.

Specifically, we feed the two bounding boxes of a positive object in its source camera and its destination camera to the regression function, i.e. ($bbox^1_{C_1}$, $bbox^1_{C_2}$) for our example at t_1, where $bbox^1_{C_1}$ represents the bounding box of O_1 from C_1 and $bbox^1_{C_2}$ represents that from C_2. All the bounding boxes are 4D vectors in the form of *(left, top, width, height)*. We apply regression filter mechanism based on our observation that similarly localized bounding boxes with similar sizes are objects at the same physical locations and their corresponding appearances on the destination camera should also be homogeneous. An outlier of the regression results is very likely to be a false positive sample. After the regression filter, we get a subset of positive data outliers and regard them as the false positive samples. Instead of directly removing these data, we choose to decouple the incorrect association between its counterpart in destination by assign it a new ID. This data point will then be regarded as a negative data sample to go through the SVM filter.

In our system implementation, we use the robust regression module of `sklearn` [68] as our regressor. As the mapping relation between two cameras may not be simply linear, we apply higher order features of the data to make the filter fit ReID results better. Specifically, we use `RANSAC` [69] algorithm as the kernel algorithms of regression as its regression process naturally splits data samples into inliers and outliers, and hence, fits the purpose of our regression filter design. We fine-tune its `residual-threshold` parameter, which determines threshold distance for a sample to be regarded as an outlier, to find the best performance. We will show more evaluations about our filter mechanism in Section 2.5.

SVM Filter Design & Implementation After the regression filter, we push all the raw ReID data, both positive and negative samples, into the SVM [70] filter. In this step , we want to learn an accurate two-class clustering between positive and negative ReID data samples based on their position-and-shape features (i.e., bounding box position and size). We choose SVM as the second step filtering model for its widely successful application in two or multiple class classification.

In our case, we feed positive data to SVM in the form of ($bbox$, 1) and negative data in form of ($bbox$, 0). We push all data samples into SVM to train a model and apply this model back to the ReID data to obtain outliers. It is worth mentioning SVMs are usually trained and tested with different data. However, we train and use the SVM model on the same data because we are not generating a classifier for future data but applying it as a filter on existing samples. We fine tune hyper parameters in SVM to avoid model overfitting, and hence, generate no outliers. The outliers here refer to negative samples appeared in positive regions and positive samples in negative regions, as shown in Figure

2.4. As we have much less positive data and have already removed positive outliers in the regression filter, we do not further remove positive outliers in SVM filter. We regard the negative outliers as false negative samples and directly remove these data from entering the optimization process. We choose to remove false negative samples only because (1) it is impossible to correctly make this sample "positive" by locating its counter part in destination camera, which is not achieved even by the state-of-the-art ReID algorithms, and (2) due to the redundancy of region associations, i.e. different objects at different timestamp usually convey the same regions mapping, the region associations usually do not change without several pairs of data samples. At the end of SVM filtering, we remove the false negative data samples. The remaining ReID results are highly confident and will go through the profiling and optimization framework in ③ and ④.

In our system implementation, we use the SVM module with `sklearn` [68] as our filter. We fine-tune its γ parameter, which determines the SVM kernel non-linearity, to explore the best performance.

Discussion As both regression filter and SVM filter are statistical, it is impossible to ensure the filtering is perfectly accurate. It is possible that we can not remove all the false identification. The filtering mechanism may even remove true identification results, either true positive or true negative. However, due to the redundancy of region associations, especially when we profile through videos long enough (containing thousands of frames), the CROSSROI system accuracy will not be degraded by the harsh filtering, while the system efficacy gets boosted significantly. We will show more CROSSROI system evaluations in Section 2.5.

2.4.4 Tile Based Video Compression and Streaming (Online)

Characterizing Tile-Based Video Compression In online phase, the CROSSROI cameras apply RoI masks on their video captures to crop the videos and remove all the non-RoI tiles. The tiles included in RoI masks will be further compressed by video compressors to reduce their file sizes before being streamed over the network, as shown in ⑤. However, applying video compressor on each tile of video separately greatly degrades the efficacy of modern video compressors, e.g. H.264. As mentioned in Section 2.2, compressors reduce video size by exploring the content similarity among existing blocks, cutting videos into small tiles reduces the number of references each block may refer to and thus degrades compression efficacy. To better illustrate the performance degradation, we profile on our dataset (Section 2.5) by cutting five different videos into different-sized

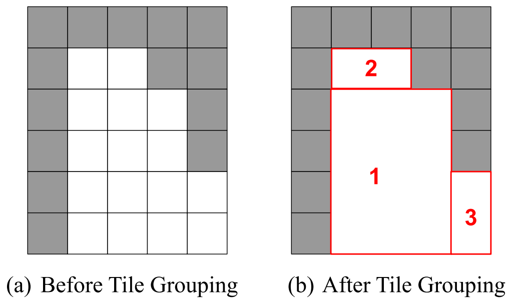
	original	2×2	2×4	4×4	4×8	8×8
C_1	82.7 **(1)**	85.9 **(1.03)**	86.2 **(1.04)**	89.0 **(1.07)**	90.4 **(1.09)**	97.3 **(1.17)**
C_2	121.2 **(1)**	124.5 **(1.03)**	124.8 **(1.03)**	127.6 **(1.05)**	129.6 **(1.07)**	136.2 **(1.12)**
C_3	102.2 **(1)**	103.3 **(1.01)**	103.6 **(1.01)**	105.2 **(1.03)**	106.4 **(1.04)**	112.9 **(1.10)**
C_4	97.9 **(1)**	99.3 **(1.01)**	99.5 **(1.01)**	100.0 **(1.02)**	101.7 **(1.04)**	108.6 **(1.11)**
C_5	40.9 **(1)**	41.1 **(1.01)**	41.4 **(1.01)**	42.0 **(1.03)**	43.2 **(1.06)**	47.4 **(1.16)**

Table 2.3: Efficacy characterization of tile-based video compression. Videos are either compressed with original H.264 standard or split into $m \times n$ tiles (e.g., 2×4) and compressed with tile-based method accordingly. Video-sizes are measured in unit of MB. **Bold** numbers represent the video size amplifications compared to *original* video compression without tiling.

tiles and encoding them in H.264 format to characterize the compression efficacy of the video compressor. As shown in Table 2.3, we split the videos according to five settings, each split the videos into $m \times n$ tiles evenly (e.g. 2×4). As we split the video in finer-grained tiles, the total video sizes grow larger, which indicates a degradation of video compression efficacy.

Tile Grouping Algorithm In order to improve video compression efficacy, we develop a straight-forward greedy-based *tile grouping* algorithm to merge fine-grained small tiles in RoIs masks into larger ones to further reduce the video-sizes being sent to the CROSS-ROI server over network. As shown in Figure 2.5(a), the video is cut into 6×5 small tiles. The white tiles are included in the RoI mask, while the shadow tiles are in non-RoI region. The tile grouping algorithm interactively find the largest inscribed rectangular in the RoI masks and merge all small tiles in this rectangule into a large tile until every tile in RoI mask is processed. For example, in Figure 2.5(b), we first merge all the 12 tiles covering region 1 into a large tile, and then merge the remaining 4 tiles into two large tiles, respectively. In this way, we merge the original 16 small tiles into 3 larger ones, and hence, improve the compression efficacy.

Finding largest inscribed rectangular in a binary grid can be easily solved with dynamic programming in $\mathcal{O}(M)$ time, where M is the number of small tiles in the video. The overall time complexity of the tile grouping is hence upper bounded by $\mathcal{O}(M^2)$. Furthermore, the tile grouping results can be calculated in offline phase once the RoI masks are generated. Therefore, the tile grouping algorithm will introduce zero overhead to the



(a) Before Tile Grouping (b) After Tile Grouping

Figure 2.5: *Tile grouping algorithm.* White tiles are corresponding to the RoI mask regions. Shadow tiles are out of the RoI mask.

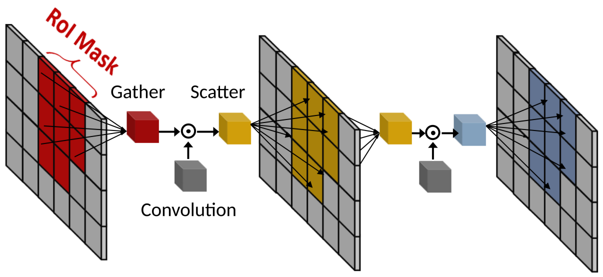


Figure 2.6: SBNet architecture illustrated with the RoI mask as shown in Figure 2.2(a). This figure is modified based on the SBNet chapter [67].

CrossRoI cameras in online phase. It is worth mentioning that our tile grouping algorithm is a heuristic greedy algorithm which cannot ensure the generated groups is exactly the optimal way to merger tiles. However, we show the significant improvement of video compression efficacy when applying our algorithm through experimental evaluations in Section 2.5.

Implementation In CrossRoI cameras, we choose `ffmpeg` [71] H.264 implementation as our video compressor. The video compressor will queue a segment of video frames, i.e. 2s or 20 frames, and compress these images into a short video before sending it to the server. A longer segment benefits video compression efficacy, as the more temporal redundancy can be reduced, but increases server response delay for detecting objects in video. We will show more evaluations on video segment length in Section 2.5.

2.4.5 RoI Based CNN Inference (Online)

Once the CrossRoI server receives video feeds from the cameras, it will dump these videos into the video analytics pipeline, which loads both video data and CNN-based machine inference models (e.g. YOLO object detector) to GPU and finally return the detection results (e.g. bounding boxes of vehicles) ⑥. Traditional CNN models usually have a respective field of the whole frame, which is not optimized for our case where the prior knowledge of RoI masks is available. In CrossRoI server, we prefer a RoI-based inference pipeline, where the CNN model works on the RoI covered data only, and hence, boosts the system inference speed.

In the CrossRoI server, we choose to implement the RoI-based CNN inference pipeline based on SBNet [67], which is an optimized CUDA kernel specially designed for RoI based CNN inference tasks. Image data is usually transformed into 4D tensors, in the form of *¡batch, height, width, channel¿*, when being processed in GPU. SBNet divides the input tensor into small tiles in the *height* and *width* dimensions. It gathers all the RoI "tensor-tiles" and stacks them together to generate a small and deep tensor constituting of RoI-covered data only. As presented in Figure 2.6, SBNet kernel adds a *gather* module before each convolutional layer of a CNN model to generate the "RoI tensor". SBNet then passes the new tensor through the convolutional module to get the data manipulated by the model. After the convolutional layer, SBNet adds another *scatter* module to transform the narrow tensor back to the original shape.

Based on SBNet, we build a RoI-YOLO object detector with `Tensorflow` [72]. It is worth mentioning that although SBNet can boost the system inference speed significantly (i.e. $1.5 \sim 2.5\times$) when the RoI area is small ($10\% \sim 20\%$) compared to the whole frame, SBNet introduces computational overhead (i.e. gather and scatter) compared to traditional CUDA kernel and may not perform as well when the RoI area is close to the whole frame. In practice, we load both RoI-YOLO and normal YOLO into GPU and push large RoI-area videos to normal YOLO model instead to achieve best performance. More evaluations about our CNN inference model will be presented in Section 2.5.

2.5 EVALUATION

2.5.1 Methodology

Dataset We evaluate our system with *AI City Challenge 2020* traffic video dataset published by NVIDIA [73]. The dataset consists of two types of scenes where the traffic cameras are deployed either along long streets or around a traffic intersection, in a northern American city. We choose the most challenging scene of type two to evaluate CrossRoI, where 5 cameras are deployed around a traffic crossing with complicated inter-camera viewpoint overlapping. We present the locations and viewing angles of the five cameras in Figure 2.1. The dataset provides 5 synchronized videos taken from five cameras with 10 fps frame rate. The length of the videos ranges from $193 \sim 215$ seconds. We choose their overlapped 180s to evaluate the CrossRoI system. All the five videos have $1920\ pixels \times 1080\ pixels$ resolutions (1080p) except the video generated by C_5, which is $1280\ pixels \times 960\ pixels$.

The five videos in scene 1 capture more than 30K vehicle bounding boxes over 3 min-

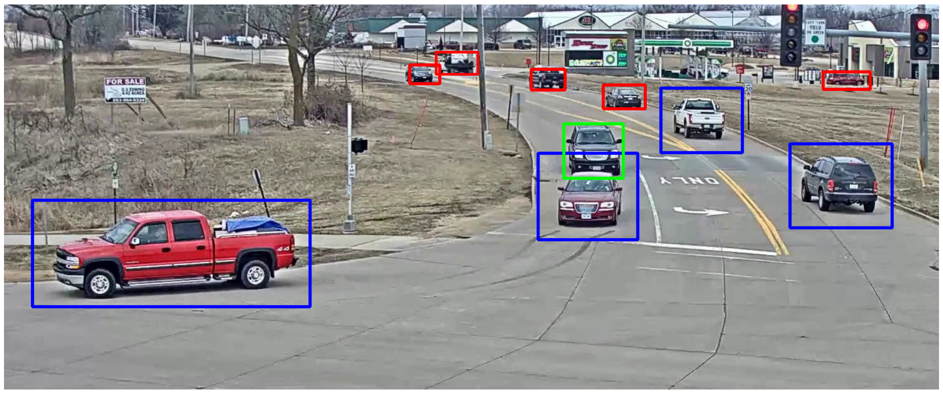


Figure 2.7: Illustration of ReID ground truth augmentation. *Blue bboxes* represent original detection provided by the dataset ground truth. The *green bbox* is missing in the ReID ground truth due to ablation, we use Kalman filter to recover its occurrence and the corresponding vehicle id. *Red bboxes* are not included in original ground truth because they are out of the view-overlapping region of the cameras. We recover these vehicle appearances with YOLO object detection results and assign each of these vehicles (vehicles with red bboxes) a unique id.

utes. Ground truth for vehicle re-identification (ReID) is provided with the dataset. However, the ReID ground truth has a shortcoming that it is very sensitive to occlusion, i.e. when vehicle A occludes vehicle B slightly, the ground truth will miss the detection and identification of B, while B could actually be detected by object detectors clearly. This usually leads to the "disappearance" of a vehicle for several frames, when it is partially occluded by other cars, in its continuous occurrence over the scene. Hence, we apply Kalman filter to fill the disappearance gaps in vehicles consecutive appearance. Another shortcoming of the ReID ground truth is that it only detects vehicles passing through multiple cameras and misses those vehicles appearing in a single camera only. We solve this limitation by augmenting the ReID ground truth with YOLO object detection results and assign unique ids to the vehicles not originally included in ReID ground truth. Figure 2.7 shows an illustrative example of our ground truth augmentation method.

Evaluation Scenario & Metrics In our evaluation, we consider the query scenario as *unique vehicle detection*. Specifically, we want to detect every unique vehicle across all cameras at the scene in real time. As shown by the example in Figure 2.2, there are 7 vehicles across the scene covered by C_1 and C_2 with 8 appearance bounding boxes. Our query scenario requires at least one detection bounding box for each unique object. Therefore, reporting either one of the two bounding boxes of O_1 fulfills the query requirement. As the CrossRoI system has two phases, we apply first 60s of the five videos as the input of offline phase to generate the RoI masks and evaluate the online phase system performance with the last 120s. The performance evaluation consists of the following four metrics:

1. **Results Accuracy**. We define accuracy error as the absolute value of the percentile difference on the number of detected unique cars between the correct and system returned value. Hence, the accuracy is defined as one minus the error. As the dataset does not provide ground truth for vehicle detection, we fuse the ReID ground truth and raw YOLO detection results as the correct reference in our evaluation.

2. **Network Overhead**. We define network overhead as the average bandwidth usage for CROSSROI server to download online video feeds from the cameras in real time.

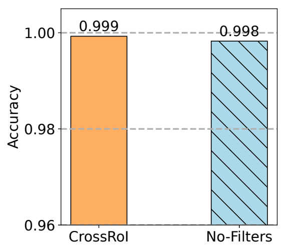
3. **System Throughput**. We define the system throughput as two parts: (1) the speed for the CROSSROI server to run vehicle detection inference in the online phase, and (2) the speed for CROSSROI cameras to compress video streams in real time.

4. **End-to-end Respond Latency**. The average delay for CROSSROI server to generate vehicle detection results in the online phase. This latency includes camera side processing delay, networked latency and CROSSROI server processing overhead.

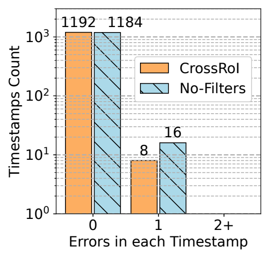
Hardware & Implementation We deploy CROSSROI service on a server with 2 GeForce RTX 2080 Graphics Card, each with 2944 CUDA cores. The server has an Intel i7-9700K 8-core CPU and 64GB memory. The CROSSROI cameras are emulated on a laptop computer with an Intel i7-8850H 6-core CPU with 16GB memory. The laptop achieves 23 fps throughput for H.264 video compression on 1080p videos. Its performance is similar to most surveillance cameras which can achieve $25 \sim 30$ fps throughput on 1080p videos (e.g. Arecont Vision MegaVideo [74] and Logitech C930e [75]). The recorded videos are stored onto the laptop and streamed out to the server in real time with `ffmpeg`. The cameras and server are connected with emulated WiFi networks of 30 Mbps bandwidth and 10 ms round-trip-time.

2.5.2 Ablation Studies

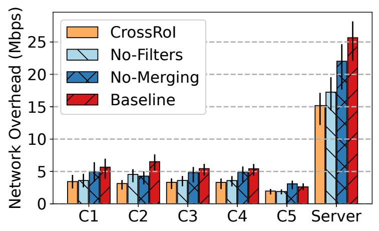
We compare CROSSROI with four alternative methods to verify its merits and some of our design choices. Each alternative achieves "partial" functionality of CROSSROI by turning off one or some of CROSSROI's functional modules. The details of the alternatives are as follows.



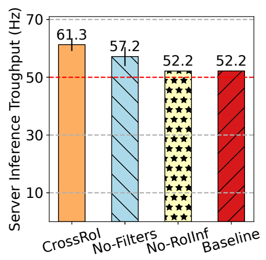
(a) Accuracy.



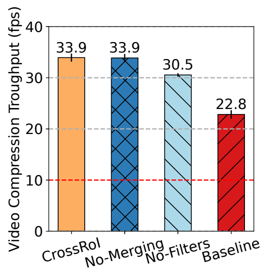
(b) Timestamp Distribution.



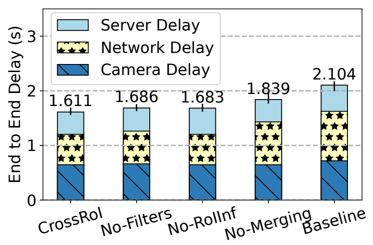
(c) Network Overhead.



(d) Severs Inference Throughput.



(e) Compression Throughput.



(f) End-to-end Response Latency.

Figure 2.8: Performance Evaluations between CROSSROI and alternative methods.

1. **Baseline**: *All* CROSSROI *functions* are turned off. Video streams are compressed with original H.264 compressor. The server runs off-the-shelve YOLOv3 [76] model as object detector to handle vehicle detection queries.

2. **No-Filters**: *Regression & SVM Filters* ② are turned off, but the other modules remain. Raw ReID results are dumped into the RoI masks generation framework. Cameras crop their video streams in online phase based on the corresponding RoI masks.

3. **No-Merging**: *Tile Grouping Algorithm* ⑤ is turned off, but the other modules remain. The cameras compress their video streams into fine-grained tiles without merging them into larger ones.

4. **No-RoIInf**: *RoI-based CNN inference* ⑥ is turned off, but the other modules remain. RoI-YOLO model is replaced with traditional off-the-shelve YOLOv3 model in the GPU inference step ⑥.

The evaluation results between CROSSROI and the four alternatives are shown in Figure 2.8.

Accuracy As the dataset does not provide ground truth for vehicle detection, we set the detection results generated by *Baseline* method as the correct reference and fuse it with ReID ground truth to obtain the correct baseline for *unique vehicle detection* task. The *Baseline* method achieves 100% accuracy naturally as it sends full video data. We present the accuracy achieved by CROSSROI and *No-Filters* in Figure 2.8(a). CROSSROI achieves an accuracy of 99.9% that only 8 vehicles are missed in total 15424 vehicle appearances over the 1200 timestamps. We plot the distribution of the 1200 timestamps in Figure 2.8(b) in terms of how many vehicles are missed at each timestamp. It is easily observed that CROSSROI achieves correct detection for 1192 timestamps over the two minutes interval. There is at most one vehicle missed in the other 8 timestamps. The accuracy of *No-Filters* method is 99.8%. CROSSROI achieves a higher accuracy than *No-Filters* with less video data because the regression filter rectifies false positive associations in raw ReID results and improves the overall accuracy.

Network Overhead We present the network overhead for each camera and the server in Figure 2.8(c). CROSSROI consumes least network bandwidth compared to all other alternatives. The network overhead of CROSSROI (15.2 Mbps) is 42% reduced compared to *Baseline* method (26.2 Mbps). Comparing with *No-Filters* (16.5 Mbps), CROSSROI reduces more video redundancy by applying the SVM filter, which removes false negative samples in raw ReID results and generates smaller-sized RoI masks. CROSSROI reduces 30% network overhead compared to *No-Merging* method due to applying tile grouping algorithm to further improve the video compression efficacy.

System Throughput We present the server inference throughput in Figure 2.8(d) and camera video compression throughput in Figure 2.8(e). The red lines represent the the minimum requirements for real time execution. That is, the server inference speed needs to be at least 50 Hz and the camera H.264 encoding throughput should be no less than 10 fps.[5] It can be observed that CROSSROI achieves highest throughput on both server (61.3 Hz) and camera (33.9 fps) sides. The RoI-based YOLO model improves overall server inference throughput by 18%. Compared with *No-Merging* method (33.9 fps), CROSSROI improves compression efficacy (i.e., reducing video sizes) without degrading compression processing speed.

End-To-End Respond latency As shown in Figure 2.8(f), CROSSROI generates least end-to-end response delay (1.61 s) comparing to all the other alternatives. Compared with the

[5]We reduce the video resolutions to *540* p for server inference due to the lack of strong GPUs.

Baseline case (2.104 s), CrossRoI reduces the latency by 25%. CrossRoI achieves less latency compared to the other alternatives because either less network overhead turns out to be less network delay or RoI-YOLO design boosts server inference speed. In this evaluation, we set the video streaming segment length as $1s$. We notice that segment length is a critical parameter to system end-to-end response delay. We will provide more detailed discussion shortly in Section 2.5.3.

2.5.3 Sensitivity to Parameters

We investigate how three hyperparameters influence the performance of CrossRoI as follows:

1. **SVM Model Non-Linearity**. We fine tune the γ parameter to manipulate the non-linearity of the SVM filter model. A small γ associates to a low non-linearity SVM kernel which usually can not fit training data perfectly and generates more outliers. A large γ corresponds to a kernel model of high non-linearity which usually fits all the training data and cannot find outliers from the training samples.

2. **`RANSAC` Threshold Distance**. In the regression filter, we manipulate the `residual threshold` parameter of `RANSAC`, which determines the threshold distance for a sample to be regarded as an outlier. Specifically, we set `residual threshold` $= \theta * mad$, where mad is the median absolute deviation of the training data and the default residual-threshold value of `RANSAC` algorithm. We fine tune different θ in the following evaluations instead.

3. **Segment Length**. Segment length is the smallest temporal unit when cameras stream live videos to the CrossRoI server. Cameras compress all frames captured in the last segment in one shot before send it to the server. Segment length has significant influence on the network overhead and end-to-end latency.

The evaluation results are shown in Figure 2.9, 2.10 and 2.11.

SVM Model Non-Linearity As shown in Figure 2.9, the system accuracy, network overhead and end-to-end response latency increase as γ increases. A very small γ causes the SVM Filters to remove too much negative outliers, which usually includes true negative samples. Hence, accuracy gets hurt when SVM non-linearity is very low. On the other hand, a small γ leads to a smaller RoI mask for each camera as it removes negative ReID

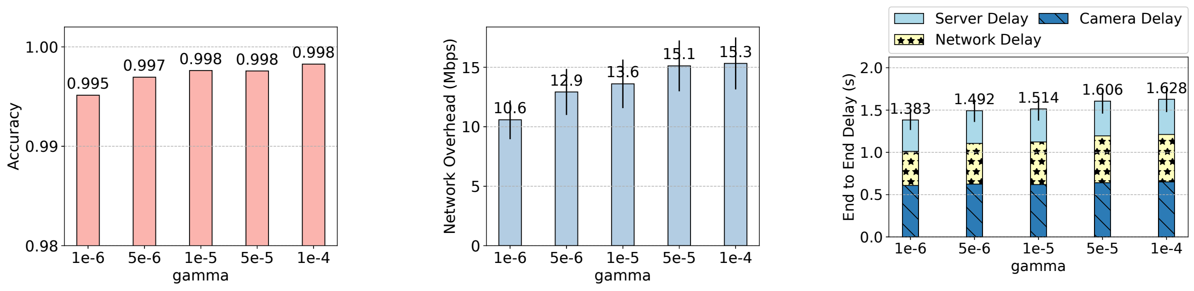


Figure 2.9: CROSSROI performance with different hyperparameter γ, which represents SVM model non-linearity.

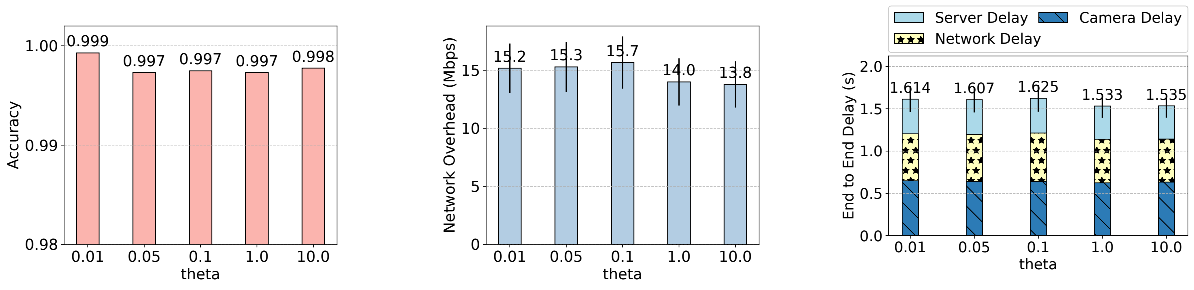


Figure 2.10: CROSSROI performance with different hyperparameter θ, which represents `RANSAC` threshold distance.

results fiercely, and thus, performs most significantly in reducing network overhead and end-to-end delay. We choose $\gamma = 10^{-4}$ in our system to achieve best system accuracy.

`RANSAC` Threshold Distance As shown in Figure 2.9, the system accuracy, network overhead and end-to-end response latency decrease as θ increases. A very low `residual threshold` causes more positive ReID samples being detected as outliers, which usually leads to larger RoI regions for the cameras, and hence, improves the system accuracy but hurts its efficiency. We use $\theta = 0.01$ in our system to achieve highest system accuracy.

Segment Length We present the network-latency trade-offs in Figure 2.11 by tuning segment length parameter. segment length is a very significant impact factor for end-to-end response latency due to the queuing mechanism for video compression and streaming. Comparing to frame-by-frame image sending, chunked-video-based streaming leads to data being queued at cameras memory, network interfaces and the server, and hence, increases the end-to-end latency. However, longer segment size provides better chance for cameras to compress the videos and significantly reduces the network overhead. We

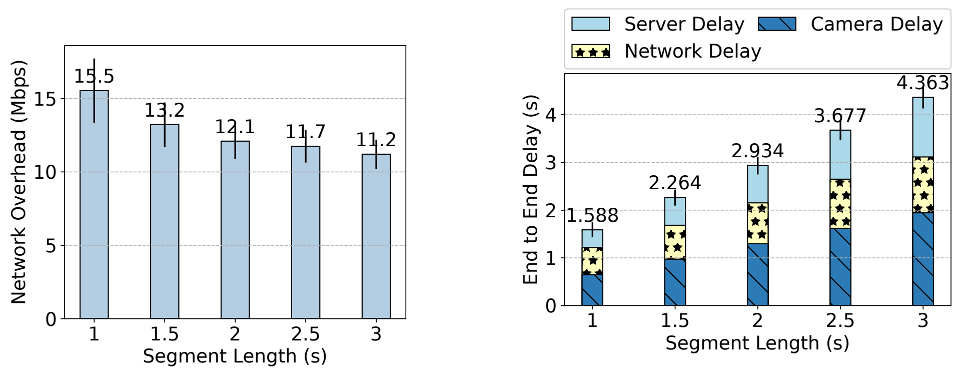


Figure 2.11: CROSSROI performance with different segment length.

choose 1s segment length in CROSSROI to achieve least end-to-end delay.

2.5.4 Comparison & Integration with Frame Filtering Systems

As mentioned in Section 2.2, significant frame filtering works have been presented to alleviate resource contention for video analytics. For example, REDUCTO [6], the SotA frame filtering system, optimizes the cost/accuracy trade-offs by discarding frames in each segment when streaming videos from camera to the server. Such systems usually perform well when the query accuracy requirement is not high, e.g. counting vehicle numbers roughly to understand current traffic condition.

As CROSSROI exploits spatial redundancy between closely located cameras fleets, we treat frame filtering as an extra layer of optimization to augment our system when the query accuracy requirement is not very high (i.e. $\leq 95\%$). Specifically, we integrate REDUCTO into our system to build CROSSROI-REDUCTO. Similar to CROSSROI, REDUCTO also operates in two phases. It profiles video clips in offline phase to learn video patterns and applies the learned-patterns as frame filters to discard frames at the cameras in online phase. It is natural to merge the two systems and generate CROSSROI-REDUCTO, which also operates in an offline-online mode.

The system workflow of CROSSROI-REDUCTO is shown in Figure 2.12. In the offline phase, the CROSSROI module profiles offline video clips to generate *RoI masks*. REDUCTO module profiles "masks-cropped" offline video clips to learn the video patterns and generates frame filters for each camera. In the online phase, the video frames first go through RoI masks to remove spatial repentant content and then go through the frame filter to eliminate temporal redundancy. The remaining data is compressed by the video compressor as described in○and sent to server for CNN inference.

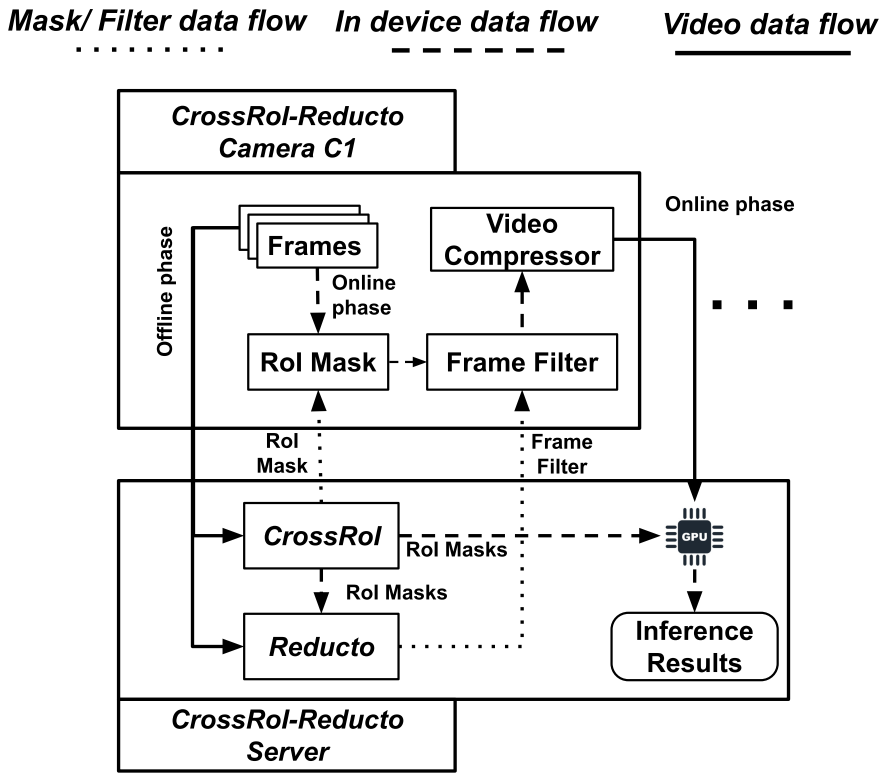


Figure 2.12: CROSSROI-REDUCTO system overview.

Accuracy Target		Accuracy Achieved	Frames Reduced	Network Overhead (Mbps)	Server Throughput (Hz)	End-to-end Respond Latency (s)
REDUCTO	**1.00**	1.000	0	26.48	52.07	2.104
	0.95	0.971	979	23.85	62.32	1.884
	0.90	0.947	2098	19.29	80.19	1.602
	0.85	0.902	4116	10.16	166.01	1.063
CrossRoI-Reducto	**1.00**	0.999	0	15.73 (**-40.6%**)	61.28 (**1.18** $\times$)	1.601 (**-23.9%**)
	0.95	0.962	1072	13.28 (**-44.3%**)	74.17 (**1.19** $\times$)	1.406 (**-25.4%**)
	0.90	0.943	2389	10.48 (**-45.7%**)	101.22 (**1.26** $\times$)	1.189 (**-25.8%**)
	0.85	0.893	4483	5.25 (**-48.3%**)	240.95 (**1.45** $\times$)	0.821 (**-22.8%**)

Table 2.4: Performance Comparison between REDUCTO and CROSSROI-REDUCTO. **Bold** number represents performance gains (server throughput) or resource reduction (network or latency overhead) of CROSSROI-REDUCTO compared to REDUCTO.

REDUCTO can adjust how fiercely to filter the frames based on a given accuracy target (e.g. 90%). We set different accuracy targets from 85% to 100% and compare the system performance between REDUCTO and CROSSROI-REDUCTO. The evaluation results are presented in Table 2.4. As shown in Table 2.4, we measure the the frame-filtering capabilities of the two systems by showing how many frames are removed, from total 6000 frames (5 cameras $\times$ 120 seconds $\times$ 10 fps), in the video analytics process. REDUCTO and CROSSROI-REDUCTO removes different number of frames under same accuracy targets because full video and cropped videos exhibit different patterns, which REDUCTO depends on to filter frames. When we set the accuracy target as 100%, the frame filtering mechanism fails to work. REDUCTO degenerates to be the *Baseline* scenario and CROSSROI-REDUCTO degenerates to the original CROSSROI. In all other scenarios, both REDUCTO and CROSSROI-REDUCTO achieves the corresponding accuracy targets. While CROSSROI-REDUCTO outperforms REDUCTO in all three system performance metrics significantly, i.e. network overhead reduction by 48.3%, server throughput boosting by $1.45\times$ and end-to-end response latency reduction by 25.8%.

2.6 CONCLUDING REMARK

In this chapter, we present CROSSROI, a resource-efficient system that enables real-time video analytics at scale by removing video content redundancy across a fleet of cameras. We develop a two-phase workflow in CROSSROI. In the offline phase, CROSSROI establishes cross-camera region associations to generate optimized RoI masks. In the online phase, CROSSROI applies these RoI masks to enhance real-time analytics performance. Experiments on real-world traffic videos demonstrate that CROSSROI reduces network overhead by $42\% \sim 65\%$ and end-to-end response latency by $25\% \sim 34\%$, compared to baseline methods, while maintaining 99.9% detection accuracy. Through this work, we have shown that **inference data redundancy** is a pivotal opportunity to achieve **multi-objective resource optimization** for large-scale video analytics systems.

CHAPTER 3: BOFL: BAYESIAN OPTIMIZED LOCAL TRAINING PACE CONTROL FOR ENERGY-EFFICIENT FEDERATED LEARNING

In this chapter, we explore the nuanced aspects of the second thesis project, FEDCORE. We introduce a multi-objective resource optimization framework aimed at **reducing energy consumption in edge devices** and **ensuring the timely delivery of federated learning tasks**. This framework leverages the **hardware configurability** to optimize resource use efficiently, as elaborated in Section 1.2.

3.1 INTRODUCTION

Federated learning (FL) is a privacy-preserving machine learning architecture that performs collaborative model training with large amount of resource-constrained edge devices (e.g., IoT devices, smartphones, etc.) in a distributed way, while keeping all the raw data locally on each device [20, 21, 22]. With FL, diverse privacy-sensitive domains have been drastically improved by the advancement of AI power. Such domains include cancer diagnosis [23, 24], human action detection [78], surveillance video analytics [18, 26, 79], and clinical decision support for COVID-19 [80]. Google also deployed a large-scale federated learning system over millions of real-world devices to improve their keyboard query suggestion model [8].

In a typical FL system, as depicted in Figure 3.1, all the edge devices are organized around a central server, which orchestrates the distributed model training in a round-by-round manner. Within each round, the central server first selects a group of participants from the large client pool to train the model. All selected devices will download a shared model from the server, and train it with their private-owned data. The participants are required to compute the gradient updates timely before a server-specified deadline, and then upload their local gradients back to the server. The server will aggregate the gradients from all the devices into a synchronized update to the global model, and initiate a new round of training accordingly.

Despite the success of FL across various domains, it poses critical energy challenges to the edge devices, which are usually resource-constrained with limited battery power. For each edge device, the on-device model training involves intensive cooperation across multiple hardware processing units, i.e. CPU, GPU, and memory controller. This procedure is energy-consuming and usually burns out the device's battery in short time. Several works [32, 81, 82, 83] propose to pursue better energy efficiency in FL, but most of them approach this problem from the server's perspective. For example, AutoFL [81]

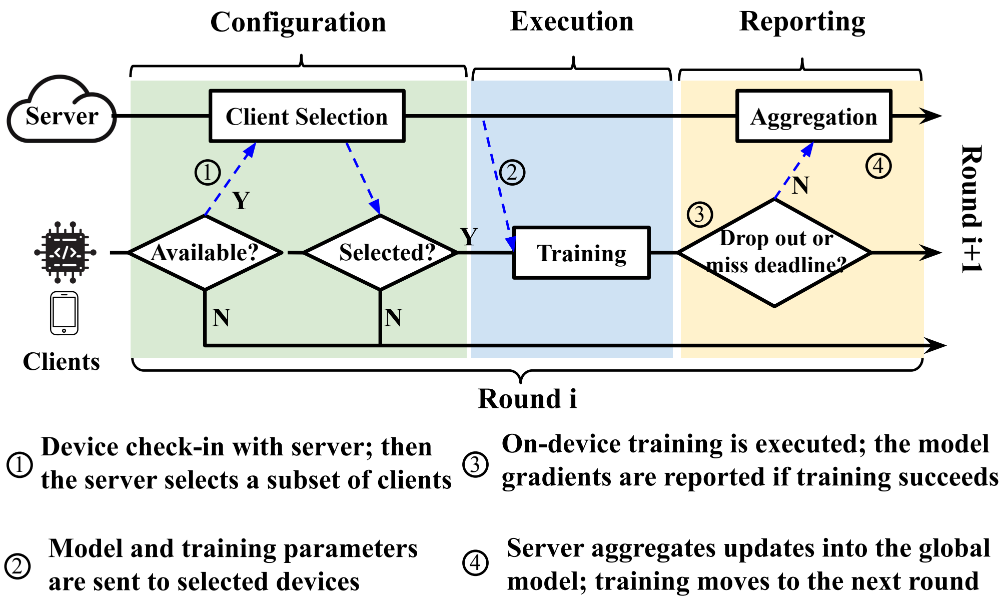

Configuration
Execution
Reporting
Server
Client Selection
Aggregation
Round i+1
① Y
Available?
Selected?
Y
② Training
③ Drop out or miss deadline?
N
④
N
N
Clients
Round i
① Device check-in with server; then the server selects a subset of clients
② Model and training parameters are sent to selected devices
③ On-device training is executed; the model gradients are reported if training succeeds
④ Server aggregates updates into the global model; training moves to the next round

Figure 3.1: Standard Federated Learning Workflow. This figure is modified based on the FedScale paper [77].

reduces the overall energy consumption by selecting a smaller group of participants in each round that are more likely to complete the model training before deadline. Although such design achieves global energy efficiency, it does not solve the energy challenge for individual devices. SmartPC [32] proposes an energy-aware training pace control solution based on CPU dynamic voltage and frequency scaling (DVFS) for edge devices (e.g., slow down CPU clock rate to achieve better energy efficiency). But this solution only works for CPU devices, and cannot be extended to modern edge devices that train neural networks jointly through GPU and CPU. Meanwhile, SmartPC models a linear relation between the training speed and CPU operational frequency. Such linear assumption fails to generalize to modern edge devices with multi-axes DVFS configurations, such as Nvidia Jetson AGX board [84], and the actual relation between performance and hardware configurations is highly-nonlinear (Section 3.2.2).

In this work, we present BoFL (Bayesian Optimized Federated Learning), a **system** deployed on each FL client locally to achieve energy-efficient training pace[6] control over multi-axes of DVFS configurations. As shown in Figure 3.2, model training performance, i.e., training speed and energy efficiency, can be drastically affected by different operational frequencies of CPU, GPU and memory controller. A proper DVFS configuration may lead to $8\times$ faster training speed and $4\times$ less energy consumption. However, the

[6]In this chapter, *training pace* refers to the hardware processing speed, i.e, the operational frequencies of CPU, GPU, and memory controller, when training neural networks.

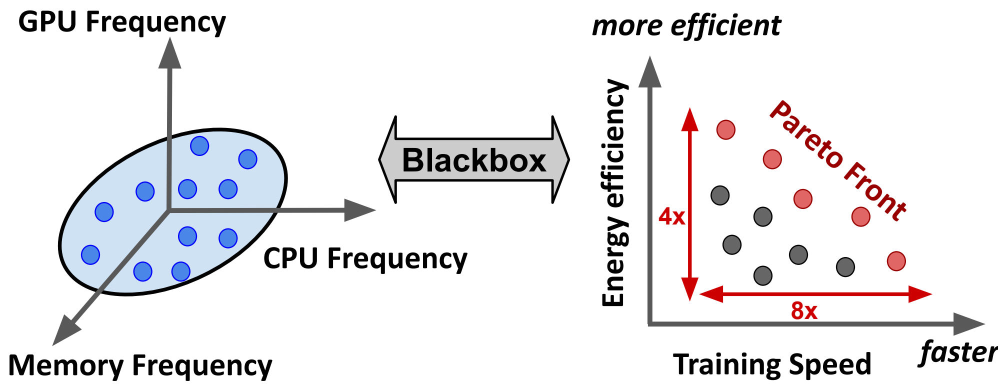


Figure 3.2: Complicated relation between hardware frequencies and model training performance (i.e. energy efficiency v.s. speed).

'configuration-to-performance' correspondence is highly non-linear and task-dependent (Section 3.2.2), thus, it is difficult to find and apply good configurations directly. Different from works such as [32] that builds explicit performance models on 1-D DVFS configurations, BoFL treats the multi-dimensional performance metric on multi-axes DVFS space as a blackbox function, and searches for the Pareto set with blackbox optimization tools. BoFL operates in a fully online manner within limited rounds of a FL task, and achieves near-optimal energy efficiency.

BoFL highlights three **challenges** to find the Pareto optimal configurations over multiple various-length training rounds and achieves optimal energy efficiency, as follows.

(**C1**) How to search for the set of Pareto optimal configurations in terms of energy efficiency and training speed *efficiently* in an *online* manner?

(**C2**) How to *balance* the effort between *exploring* the Pareto front and *exploiting* local optimal configurations within limited number of task rounds?

(**C3**) How to embed the *non-trivial* exploration algorithms with time-critical training jobs, while satisfying *multifaceted* system constraint and requirements?

To tackle these challenges, we design BoFL to operate in an *explore-then-exploit* manner. In the limited rounds of FL tasks, BoFL first explores the DVFS configuration space with a few trials, and then exploits the remaining rounds with the best configurations observed. Specially, BoFL strategically explores the large configuration space with multi-objective Bayesian optimization (MBO) framework, which searches for a set of Pareto trade-offs in the *energy-latency* 2-D performance space efficiently in just a few steps. The obtained Pareto optimal configurations can be exploited in later rounds adaptively with respect to the various deadlines (**C1**). To balance exploration and exploitation, we categorize the multi-round FL task into three phases: two short phases for exploration and one long phase for exploitation. We use hypervolume improvement indicator (Section 3.4) to determine the length of exploration phases, so that BOFL can construct near-optimal Pareto front within few rounds of exploration (**C2**). We run MBO calculation and the model training separately to avoid introducing extra latency overhead to the time-critical FL tasks. To cope with the uncertainty caused by exploring the whole configuration space (for example, BOFL may explore a straggler configuration and exacerbate the deadline challenge), we design a *safe exploration algorithm* to make sure every training deadline is being met (**C3**). Overall, this chapter makes the following **contributions**:

(**1**) We depict the complex relation between hardware operational frequencies and Neural Network (NN) training performance through a comprehensive measurements with multiple network models over different devices (Section 3.2.2).

(**2**) We develop a blackbox optimization framework for the energy-aware training pace control problem, and propose a MBO based solution which achieves near-optimal energy efficiency in an online manner (Section 3.3).

(**3**) We tailor-design the MBO workflow to embed it into the time-critical FL tasks (Section 3.4), and implement the BOFL solution which achieves both smart exploration and efficient exploitation, despite multifaceted system constraints (Section 3.5).

(**4**) We perform comprehensive experiments over multiple neural networks and devices to evaluate BOFL's effectiveness. Evaluation results suggest that BOFL can reduce more than 20% of energy consumption compared to a performant baseline. The energy overhead of BOFL is as low as 1.2% - 3.4% compared to an optimal oracle target (Section 3.6).

The rest of the chapter is organized as follows. In Section 3.2, we present backgrounds about federated learning and Bayesian optimization. We also depict the complexity of NN training performance with different hardware frequency settings to motivate our solution. In Section 3.3, we present the BOFL system model and optimization framework. We present BOFL system workflow and design details in Section 3.4. Section 3.5 and Section 3.6 present the implementation and evaluations of BOFL system. We survey related literature in Section 3.7. Finally, Section 3.8 concludes the chapter.

3.2 BACKGROUND AND MOTIVATION

3.2.1 Energy Efficient Federated Learning

Federated learning is a machine learning paradigm that enables collaborative model training from a large pool of edge devices with locally stored data. While a lot of works have been proposed to achieve fast model convergence [85, 86, 87, 88], protect client data privacy [89, 90, 91], defend against malicious attacks [92, 93, 94], and address source of data bias [95, 96], not many works have been presented to improve the energy efficiency of federated learning tasks.

SmartPC [32] and AutoFL [81] are two representative works that aim to optimize the energy efficiency of federated learning clients. They both adopt a two-level energy optimization solution, where (1) the cloud server, from global level, strategically selects a small group of devices with high energy efficiency, and assigns them a well-designed training deadline[7]. (2) The selected devices, from local level, adapt their training paces (e.g., CPU frequencies) to minimize energy consumption and finish all training workloads before the assigned deadline. While this two-level solution can successfully reduce energy usage, SmartPC and AutoFL both oversimplify the complexity of local pace control on edge devices. They model the training speed as a linear dependent variable of CPU or GPU frequency, which is not accurate for modern edge devices where the clock rates of CPU, GPU and memory controller jointly influence the training performance in a highly nonlinear way (Section 3.2.2).

In BOFL, we focus on an efficient and effective local training pace control algorithm that jointly controls hardware frequencies over multiple axes to achieve energy-efficient federated learning. BOFL is deployed on each edge device locally, and can smartly find the best DVFS configurations for this hardware within a few training rounds. BOFL assumes a cloud server which assigns a training deadline for each training round. Any deadline assignment algorithm, either strategically designing round deadlines [32, 81, 82, 83] or using a static timeout value [8], as shown in the vanilla system design [22], can function well with BOFL.

[7]The training deadline is referred to as execution target in AutoFL [81].

[8]There are two types of deadline definitions in the FL literature: (1) a training deadline before which the clients must finish the gradient calculation; and (2) a reporting deadline before which the server must receive the model updates from clients, which includes the model training delay and parameter uploading latency. In BOFL, we assume the first deadline model. For servers that only specify a reporting deadline, BOFL can be easily extended to work well with a network bandwidth measurement module that can infer its training deadlines from the reporting deadlines.

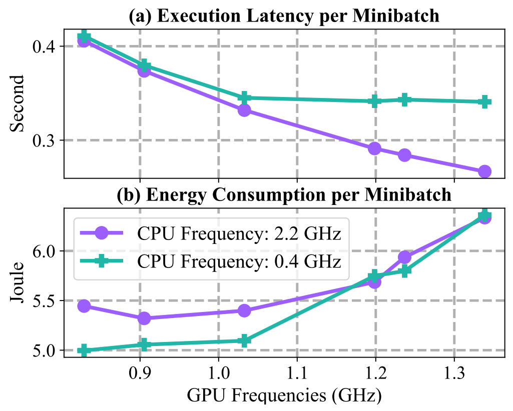


Figure 3.3: Training performance of ViT model with increasing GPU frequencies.

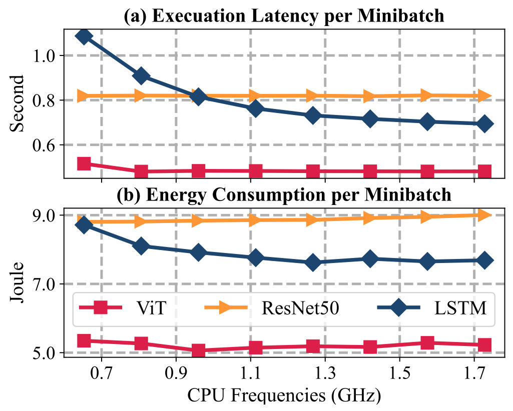


Figure 3.4: Training performance of three models with increasing CPU frequencies.

3.2.2 Complicated Correspondence Between DVFS Configurations and Training Performance

Dynamic voltage and frequency scaling (DVFS) technique has been widely applied in modern computers for energy efficient purpose, especially for resource constrained mobile or edge devices. For example, Nvidia Jetson AGX, a newly released edge device for AI workloads, has a large discrete DVFS configuration space, i.e., CPU (0.4-2.3GHz), GPU (0.1-1.4GHz) and memory controller (0.2-2.7GHz), which leads to more than 2K unique combinations.

In federated learning applications, it is important to make good choices in the large configuration space to achieve high-performance model training. For example, a good frequency choice can increase the training speed by $8\times$, and energy efficiency by $4\times$. However, it is non-trivial to find the good configurations due to the complicated correspondence between the configuration and the model training performance. Specially, we observe that the correspondence has **three-fold complexities** through a measurement experiment, which trains three representative neural networks (e.g. ViT, ResNet50 and LSTM) on two different devices (Nvidia Jetson AGX and Jetson TX2), as follows.

(1) Non-linearity. As shown in Figure 3.3(a), when the CPU clock is set to 0.4 GHz, the training speed of the ViT model sees a diminishing improvement after 1.0 GHz GPU frequency. This is because the job does not benefit from faster GPU clocks, when the slow CPU becomes the bottleneck. The energy consumption curve in Figure 3.3(b) presents an even higher complexity that it is neither linear nor monotonic. When the GPU frequency is low, i.e., 0.7 GHz, ViT achieves better energy efficiency with 0.4 GHz CPU clock than that of 2.2 GHz CPU setting. While the GPU is configured to high frequencies, i.e., 1.4 GHz, a slow CPU saves no more energy and slows down the training speed by half. In general, we observe that the training performance is influenced by the hardware frequencies in a *non-linear* way. Different configurations may have different bottlenecks over multiple axes, which leads to a complicated correspondence.

(2) NN-model dependence. As shown in Figure 3.4(a), the execution latencies of the three neural networks show different patterns as the CPU frequency increases. The training speeds of ViT and ResNet50 almost remain the same, while LSTM reduces its execution latency by half when increasing CPU clock rate from 0.6 GHz to 1.7 GHz. For energy consumption as shown in Figure 3.4(b), we can see ResNet50 exhibits a steadily increasing curve, while LSTM shows a consistently decreasing curve. In general, we observe that the relation between DVFS configurations and the training performance is *NN-model dependent*. Different network models may be influenced by the hardware configurations differently.

(3) Hardware dependence. Figure 3.5 shows the normalized training performance of the three models on Jetson AGX compared to that of Jetson TX2 (unit performance as the red line shows). Both devices are configured with maximum operational frequencies. As a newer version with stronger hardware and updated architectures, the AGX board can significantly reduce the training time as well as the energy consumption, compared to TX2. However, the performance improvement does not apply uniformly to all three models. E.g., ResNet50 reduces its training time by 70% on AGX, while LSTM only achieves 20% execution latency reduction. The above measurements suggest that the correspondence between hardware configurations and the training performance is *hardware dependent*. E.g., it is non-trivial to estimate the performance curve of AGX board based on measurements from TX2.

In summary, the correspondence between DVFS configurations and neural network training performance is complicated and hard to be accurately modeled in an explicit way. This motivates us to model this correspondence as a blackbox function and search for the good configurations with Bayesian optimization. Our solution can search for good configuration points efficiently, and can be generally applied to any NN model on any hardware.

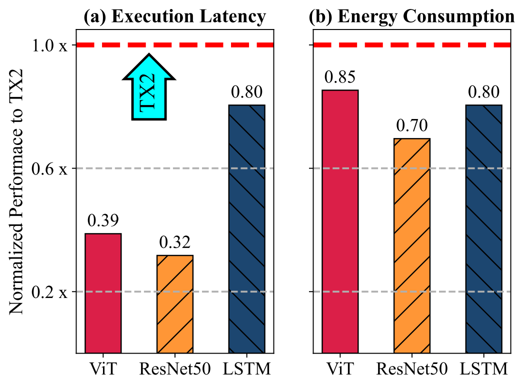


Figure 3.5: Normalized training performance on Jetson AGX compared to Jetson TX2.

3.2.3 Multi-Objective Bayesian Optimization for Blackbox Optimization Problems

The search for desirable DVFS configurations can be formulated as an optimization problem involving a multi-dimensional blackbox performance metric function (see Section 3.3.1 for details). A popular and sample-efficient solution for such optimization problem is multi-objective Bayesian optimization (MBO). MBO is an optimization framework that leverages a probabilistic surrogate model to solve optimization problems involving multiple blackbox objectives. The goal of MBO is to construct an approximated Pareto set (see Section 3.3.1) for those conflicting objectives, within a limited budget of function evaluations.

MBO sequentially selects new points to evaluate the blackbox function based on the surrogate model, and updates the model to incorporate new observations. To decide which point to evaluate next, MBO employs an *acquisition function* that specifies the utility of evaluating a new point based on the surrogate model's predictive distribution. A good choice of the acquisition function helps to balance the trade-off between exploration of unknown regions and exploitation of the current best-performing ones [97, 98].

MBO has been extensively studied in the literature [99, 100], and has enjoyed substantial successes in many applications, including environmental engineering [101], structural design [102] and high-energy physics [103].

We design BOFL with MBO embedded as the core algorithm for balancing exploration and exploitation in the Pareto search. The MBO workflow in BOFL as well as the choice of acquisition function is carefully designed to better fit the MBO module with other ingredients in the federated learning task. More details and reasoning of BOFL system design can be found in Section 3.4.

3.3 PROBLEM DEFINITION AND PRELIMINARIES

3.3.1 Problem Formulation

Federated Learning Task A federated learning task is initialized by the server and trained with a pool of edge devices in a round-by-round manner. At each round, the server selects a small group of participants from the device pool and trains the machine learning model for E epochs of the Stochastic Gradient Descent algorithm (SGD) with minibatch size of B on the selected devices using their private-owned data. The participants are required to finish the gradient calculation before a server specified training deadline, and then upload the model gradients back to the server where the gradients get averaged into a shared model. The deadline is calculated by the server as regards to the training data size and computation capabilities of the selected participants in this round. E.g., a training round using devices with stronger hardware or less training data may be assigned a shorter deadline by the server. The server usually forms different group of devices for each round to make sure the model is trained with heterogeneous data and thus not biased, which leads to various deadlines for different rounds.

In BOFL, a federated learning task can be formally defined as $(B, E, \mathbf{T}, N)$ from the perspective of a local device, where (1) B and E represent the aforementioned global parameters, minibatch size and training epoch number; (2) $\mathbf{T}$ is a vector, representing the training deadlines for the rounds when this device is selected to participate; and (3) N represents the number of minibatches of training data available on this device. E.g., a client with 1k images, joining a federated classifier training task of minibatch size 10, has $N = 1k/10 = 100$ in this case.

DVFS Configuration Space Our goal is to find the optimal or near-optimal device training pace for every round of the federated training that satisfies the corresponding training deadlines and minimizes the total energy consumption.The training pace can be controlled by configuring the operational frequencies of the device's CPU, GPU and memory controller. We thus define the DVFS configuration space as $\mathbf{X} = \mathbf{F}_{CPU} \times \mathbf{F}_{GPU} \times \mathbf{F}_{MC}$, where the $\mathbf{F}_{(\cdot)}$s represent the discrete operational frequencies of the three hardware units (CPU, GPU and memory controller), respectively.

Energy Optimization Problem For any given federated learning task on a specific device, the training speed and energy efficiency can be characterized as functions of the DVFS configurations. Formally, we define the two metrics as follows:

- $\mathcal{T}(\mathbf{x})$ is the execution **latency** to compute one minibatch of the training data under configuration x;
- $\mathcal{E}(\mathbf{x})$ is the **energy** consumed on one minibatch of data when trained with configuration x.

$\mathbf{x} \in \mathbf{X}$ is a three-element tuple encoding the operational frequencies of CPU, GPU and memory controller. In our system, we define the processing of a minibatch of data as a *job*. E.g., for an FL task that trains a ResNet50 model, a *job* refers to the process of feeding a minibatch of data (images) into the ResNet50 model and generating the gradient updates. We can always apply a different DVFS configuration for the next job, but no more than one configuration in the same job. We further define $W = E \times N$, which is the number of jobs in any single training round. The optimization problem can, thus, be formally presented as follows:

$$\begin{aligned} \min_{\mathbf{x}_{r,i} \in \mathbf{X}} \quad & \sum_{r=1}^{|\mathbf{T}|} \sum_{i=1}^{W} \mathcal{E}(\mathbf{x}_{r,i}) \\ \text{s.t.} \quad & \sum_{i=1}^{W} \mathcal{T}(\mathbf{x}_{r,i}) \leq \mathbf{T}_r \quad \forall\, 1 \leq r \leq |\mathbf{T}| \end{aligned} \tag{3.1}$$

The variable $\mathbf{x}_{r,i}$ is the DVFS configuration applied in the r-th training round on the i-th job. The optimization goal of BoFL is to minimize the overall energy consumption during the $|\mathbf{T}|$ rounds of training by carefully selecting DVFS configurations for every job, while satisfying the unique training deadline for each round.

3.3.2 Solution Sketch with MBO Searched Pareto Set

While the whole configuration space is large, the optimizers of Eqn. (3.1) can be chosen from a small **Pareto set**, defined as the set of Pareto optimal points in $\mathbf{X}$, in terms of the two metric functions, $\mathcal{E}$ and $\mathcal{T}$. More formally, let $\mathcal{M}(\mathbf{x}) = (\mathcal{E}(\mathbf{x}), \mathcal{T}(\mathbf{x}))$ be a two-objective function defined on $\mathbf{X}$, a point $\mathbf{x_1} \in \mathbf{X}$ is Pareto dominated by another point $\mathbf{x_2} \in \mathbf{X}$, iff $\mathcal{E}(\mathbf{x_1}) \geq \mathcal{E}(\mathbf{x_2})$ and $\mathcal{T}(\mathbf{x_1}) \geq \mathcal{T}(\mathbf{x_2})$, and either $\mathcal{E}(\mathbf{x_1}) > \mathcal{E}(\mathbf{x_2})$ or $\mathcal{T}(\mathbf{x_1}) > \mathcal{T}(\mathbf{x_2})$. We denote this by $\mathcal{M}(\mathbf{x_1}) \prec \mathcal{M}(\mathbf{x_2})$. A point is Pareto optimal if it is not Pareto dominated by any other point. We use $\mathbf{P} \subseteq \mathbf{X}$ to denote the set of Pareto optimal points for function $\mathcal{M}$. The images of the Pareto optimal points $\mathbf{P}_f := \mathcal{M}(\mathbf{P})$ are called the Pareto front.

It is intuitive that the variable space in Eqn. (3.1) can be reduced from $\mathbf{X}$ to $\mathbf{P}$ without affecting the solution: any DVFS configuration outside the Pareto front could be replaced

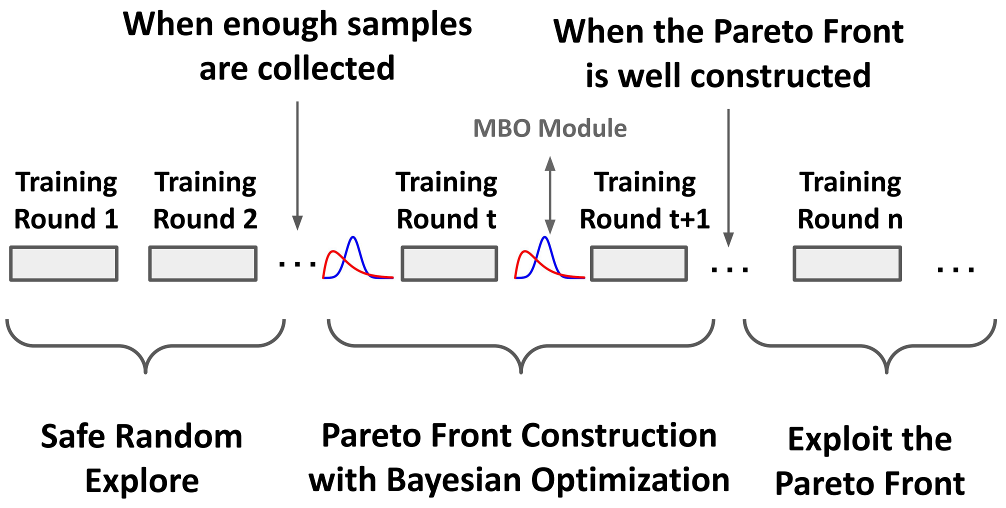


Figure 3.6: BoFL Workflow

by its dominant point in **P** to further minimize the energy objective without causing extra delay. Once the Pareto set **P** is given, it is straightforward to reformulate Eqn. (3.1) as an integer linear programming problem, which can be solved efficiently.

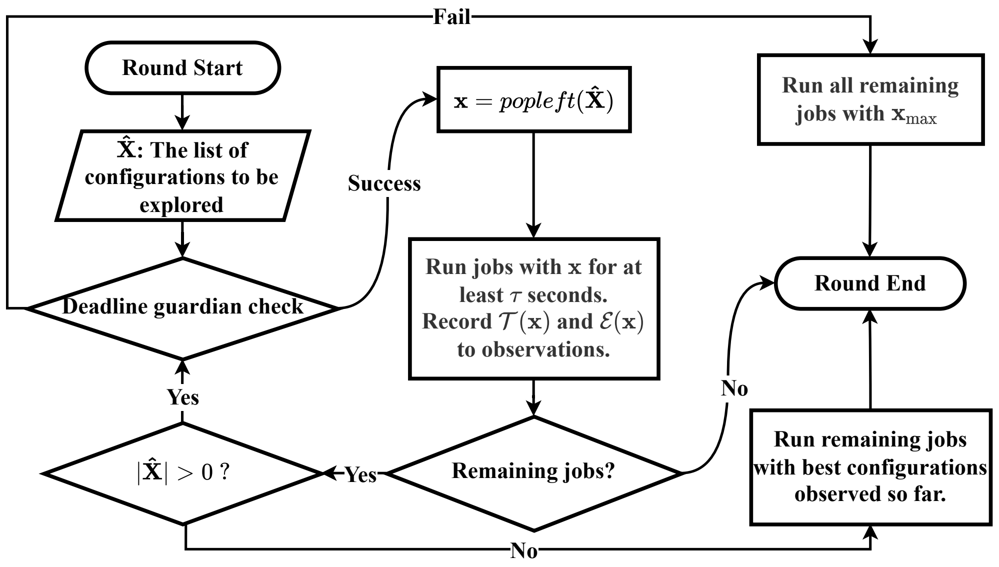
However, searching for the Pareto set **P** is a challenging task since the two metric $\mathcal{E}$ and $\mathcal{T}$ are blackbox functions that are expensive to evaluate. Due to the computational cost for configuration evaluation and the deadline requirement for training tasks, it is crucial to obtain an approximated Pareto set within limited rounds. In BoFL, we use multi-objective Bayesian optimization (MBO) as a sample-efficient method to search for the Pareto optimal points with just a few trials, and get near-optimal solution for Eqn. (3.1).

3.4 BOFL SYSTEM DESIGN

3.4.1 Overview

As shown in Figure 3.6, BoFL operates in **three** phases spanning all the training rounds as follows:

(1) *Safe random exploration* samples and tries candidates from the configuration space uniformly to collect the first group of observations for MBO model initialization. It will continue for one or a few rounds at the beginning of the FL task. We design a *safe exploration algorithm* to make sure the training deadlines are being satisfied, and the observed metrics (e.g., latency and energy consumption) are being accurate.



Fail
Round Start
$\hat{\mathbf{X}}$: The list of configurations to be explored
Deadline guardian check
Success
$\mathbf{x} = popleft(\hat{\mathbf{X}})$
Run jobs with $\mathbf{x}$ for at least τ seconds. Record $\mathcal{T}(\mathbf{x})$ and $\mathcal{E}(\mathbf{x})$ to observations.
Run all remaining jobs with $\mathbf{x}_{\max}$
Round End
No
Yes
$|\hat{\mathbf{X}}| > 0$?
Yes
Remaining jobs?
Run remaining jobs with best configurations observed so far.
No

Figure 3.7: Safe exploration algorithm

(2) *Pareto front construction* tries on the candidates suggested by MBO. This phase will continue for several rounds (e.g., 3 to 5 rounds) to search for the Pareto optimal configurations. The MBO update is separated from the model training, using the configuration and reporting time window, to minimize system overhead generated by co-locating training and MBO calculation at the same time. To accelerate Pareto searching, the MBO algorithm will propose *multiple* candidates (in batched form) to be explored for the next round.

(3) *Exploitation* is a long phase, which usually takes more than 90% of the time in a FL task. In this phase, we solve Eqn. (3.1) with the approximated Pareto optimal points constructed from the second phase. The energy consumption is thus minimized with the sweet spot configurations.

In the rest of this section, we present the design details of the three operational phases in Section 3.4.2, Section 3.4.3 and Section 3.4.4, respectively.

3.4.2 Safe Random Exploration

The goal of the random exploration phase is to collect some starting points, uniformly from the space, for the MBO model to estimate the objective functions. As we sample the starting points randomly over the whole space, it is inevitable to select some bad DVFS configurations that introduce longer delays than we expect, which exacerbates the challenge to meet every training deadline in this phase. We design a safe exploration algorithm, as depicted in Figure 3.7, to ensure the training deadlines met. We present the detailed design decisions as follows.

Sample Selection We sample a small group (e.g., 1% of the whole space) of starting points, uniformly distributed over $\mathbf{X}$, using a quasi-random number generator. This uniform exploration strategy helps the Bayesian model avoid making wrong assumptions over the sample space.

Instead of trying the starting points from the very beginning, we manually choose $\mathbf{x}_{\max}$ as the first DVFS configuration to be applied in the FL task, where

$$\mathbf{x}_{\max} = (\max(\mathbf{F}_{CPU}), \max(\mathbf{F}_{GPU}), \max(\mathbf{F}_{MC})) \tag{3.2}$$

refers to the DVFS configuration with the highest operational frequencies on all three processing units (e.g., CPU, GPU and the memory controller). $\mathbf{x}_{\max}$ is a configuration with maximum processing capability where the training can be quickly finished. After $\mathcal{T}(\mathbf{x}_{\max})$ is observed, $\mathbf{x}_{\max}$ could be used as a guardian configuration, so that we can always speed up our pace to $\mathbf{x}_{\max}$ in the middle of any exploration round to catch the training deadline before it is too late.

Workload Assignment In each exploration round, we can try multiple DVFS configurations through the W training jobs. It is important to assign a balanced workload to each configuration. A transient workload (e.g., trying the configuration for only one job) will lead to the execution being finished before the hardware voltage gets stable, and will generate large energy measurement error. Contrarily, a heavy workload prolongs exploration phase, and squeezes the exploitation phase.

In practice, we define τ as a reference measurement duration (e.g., 5s). BOFL will keep assigning a new job to configuration $\mathbf{x}$ until it has been explored for at least τ seconds. When the current job finishes and the configuration has been measured for more than τ seconds, BOFL will switch to explore the next candidate point.

Deadline Guardian Strategy As we randomly sample configurations to explore in the first phase, we will inevitably meet some bad points that delay our training progress, and exacerbates the challenge to catch the training deadlines. In BOFL, we make sure the deadline requirements never violated by using the known guardian configuration $\mathbf{x}_{\max}$. $\mathbf{x}_{\max}$ is tested first so that $\mathcal{T}(\mathbf{x}_{\max})$ is known before any exploration point is tried. Before exploring an unknown configuration $\mathbf{x}$, we run a quick *deadline guardian check* first to see if the remaining jobs could still be finished with $\mathbf{x}_{\max}$, even if the τ seconds of exploration on $\mathbf{x}$ fails to finish any job. Formally, let W_{remain} and T_{remain} represent the number of remaining jobs and the remaining time before deadline when configuration $\mathbf{x}$ is about to be explored, the deadline guardian check criterion could be expressed as follows,

$$T_{\text{remain}} - \tau \geq W_{\text{remain}} \times \mathcal{T}(\mathbf{x}_{\max}) \tag{3.3}$$

Configuration $\mathbf{x}$ would be explored only if Eqn. (3.3) satisfies. In case Eqn. (3.3) fails, configuration exploration will terminate in this round, and all the remaining jobs will be executed under configuration $\mathbf{x}_{\max}$. The random exploration phase will continue for several rounds until all the uniformly sampled starting points are explored.

Last Round Exploitation In the last round of the random exploration phase, we may finish exploring all the starting points early, leaving some jobs not executed. While we can play safe to apply $\mathbf{x}_{\max}$ on the remaining jobs, this method may consume more energy than needed, as $\mathcal{E}(\mathbf{x}_{\max})$ could be high. In BoFL, we apply an exploitation strategy to finish the last random exploration round with observed configurations. We calculate the best profile of configurations from the observed starting points to minimize energy consumption, while satisfying the deadline requirement. More details of the exploitation algorithm will be presented in Section 3.4.4.

3.4.3 Pareto Front Construction

The Pareto front construction phase is the main module that we apply MBO algorithms to search for Pareto optimal configurations. As shown in Figure 3.6, this phase continues for several rounds with the following two components.

(**1**) A MBO module that runs between two consecutive training rounds. It updates the function estimation with the observations from the previous rounds, and provides *a batch of suggested configurations* to explore for the next round.

(**2**) A FL task round in which the suggested configurations are explored. It ensures to satisfy the corresponding deadline with the safe exploration algorithm as presented in Section 3.4.2.

We present the reasoning and design decisions of the Pareto front construction phase as follows.

Separation of MBO Computation and Model Training The MBO calculation involved in Pareto searching is nontrivial, and usually takes several seconds to complete. Running the MBO module and training the learning task simultaneously can result in unexpected running-time overhead, and increase the risk of missing deadline. Alternatively, in BOFL, the MBO module is executed only during the configuration and reporting time window as shown in Figure 3.1, while the module will idle during the training time. Such separation will effectively minimize the running-time influence of the MBO module on the training task.

MBO Prior Function Without the loss of generality, in this project, two objective functions $\mathcal{T}$ and $\mathcal{E}$ are modeled by two independent Gaussian processes [104], each of which has a prior distribution with mean function $m(\mathbf{x}) = 0$ and kernel function $k(\mathbf{x}, \mathbf{x}') = C_{5/2}(\|\mathbf{x} - \mathbf{x}'\|_2)$. Here $C_{5/2}(\|\mathbf{x} - \mathbf{x}'\|_2)$ is the widely-used Matérn-$5/2$ kernel function [104] that can capture a large variety of function properties.

Pareto Front Approximation with Hypervolume Improvement Recall that the goal of our MBO algorithm is to identify a finite approximated Pareto front. To measure the quality of an approximated Pareto front, hypervolume indicator (HV) is the most commonly used metric in MBO. It quantifies the hypervolume of the region in the performance space that is dominated by the approximated Pareto front and bounded from below by a reference point. Mathematically, given an approximated Pareto front

$$\mathbf{P}' = \{\mathbf{p}_i = \mathcal{M}(\mathbf{x}_i) : \mathbf{x}_i \in \mathbf{X}, i = 1, \ldots, n\} \tag{3.4}$$

and a reference point $\mathbf{r} \in \mathbb{R}^2$, the hypervolume indicator $\mathrm{HV}(\mathbf{P}', \mathbf{r})$ is defined as

$$\mathrm{HV}(\mathbf{P}', \mathbf{r}) = \int_{\mathbb{R}^2} \mathbb{1}_{\mathrm{H}(\mathbf{P}', \mathbf{r})}(\mathbf{z}) d\mathbf{z}, \tag{3.5}$$

where $\mathrm{H}(\mathbf{P}', \mathbf{r}) := \cup_{i=i}^{n} \{\mathbf{z} \in \mathbb{R}^2 : \mathbf{r} \preceq \mathbf{z} \preceq \mathbf{p}_i\}$ is the region dominated by $\mathbf{P}'$ and bounded from above by $\mathbf{r}$. The higher the $\mathrm{HV}(\mathbf{P}', \mathbf{r})$ is, the better the $\mathbf{P}'$ approximates the true Pareto front $\mathbf{P}$. The reference point can be selected as the combination of the worst performances, for $\mathcal{T}$ and $\mathcal{E}$, we observed in phase 1, i.e., $\mathbf{r} = (\max(\mathcal{E}(\mathbf{x})), \max(\mathcal{T}(\mathbf{x}'))), \forall \mathbf{x}, \mathbf{x}' \in \widehat{\mathbf{X}}$, where $\widehat{\mathbf{X}}$ is the set of starting points explored in the random exploration phase.

To determine how much the hypervolume would increase if a set of new points $\mathbf{Q} = \{\mathbf{q}_j = \mathcal{M}(\mathbf{x}'_j) : \mathbf{x}'_j \in \mathbf{X}, j = 1, \ldots, m\}$ is added to the current Pareto front approximation

$\mathbf{P}'$, we define the hypervolume improvement (HVI) of $\mathbf{Q}$ with respect to $\mathbf{P}'$ as

$$\mathrm{HVI}(\mathbf{Q}; \mathbf{P}', \mathbf{r}) = \mathrm{HV}(\mathbf{Q} \cup \mathbf{P}', \mathbf{r}) - \mathrm{HV}(\mathbf{P}', \mathbf{r}). \tag{3.6}$$

EHVI Acquisition Function In general, the performance metric $\mathcal{M}(\mathbf{x})$ at any unobserved point $\mathbf{x} \in \mathbf{X}$ is unknown in the blackbox optimization. However, the GP surrogate model in the Bayesian framework provides a posterior distribution $\mathbb{P}(\mathcal{M}(\mathbf{x})|\mathbf{D})$ for any unobserved point $\mathbf{x} \in \mathbf{X}$, where $\mathbf{D}$ is the set of all historical observations. This allows one to define and compute the expected hypervolume improvement (EHVI) acquisition function, conditioned on historical observations $\mathbf{D}$, current approximated Pareto front $\mathbf{P}'$, and reference point $\mathbf{r}$:

$$\alpha_{\mathrm{EHVI}}(\mathbf{x}|\mathbf{D}, \mathbf{P}', \mathbf{r}) = \mathbb{E}_{\mathcal{M}(\mathbf{x}) \sim \mathbb{P}(\cdot|\mathbf{D})}[\mathrm{HVI}(\{\mathcal{M}(\mathbf{x})\}; \mathbf{P}', \mathbf{r})]. \tag{3.7}$$

The algorithm will select the point with maximal EHVI to evaluate in the next iteration. In practice, the 2-D EHVI value can be computed efficiently calculated in $\mathcal{O}(|\mathbf{D}| \log(|\mathbf{D}|))$ time complexity [105].

Batch Selection Strategy The classical formulation of EHVI (3.7) proposes only a single point to evaluate. However, practically, the time scale of each round in the Pareto front construction phase in BoFL allows the system to explore multiple configurations within one round. Therefore, the MBO algorithm is required to propose a batch of configurations to evaluate in the next round.

Note that the definition of EHVI (3.7) can be extended to a batch setting, by simply replacing the input point $\mathbf{x}$ by a batch of points, and the expectation is taken over the posterior distribution on the entire batch. However, finding the optimal batch based on such acquisition function will suffer from the high computational cost, especially when the batch size is large [106].

In BoFL, inspired by [106, 107], we adopt a batch selection strategy to select K points for the next batch in a *sequential greedy fashion* with the following three steps:

(**1**) Choose the next point $\mathbf{x}$ to be explored based on the acquisition function (3.7);

(**2**) Fantasize the observation on $\mathbf{x}$ from our surrogate model, i.e. $\widehat{\mathcal{M}}(\mathbf{x}) = \mathbb{E}[\mathcal{M}(\mathbf{x})|\mathbf{D}]$, and update posterior estimation accordingly;

(**3**) Repeat step (1) and (2) until K configurations are selected.

Our batching strategy leads to a diverse batch selection and scales well for large batches.

MBO Batching Size Selection In every run of the MBO model, we generate a batch of $K = T_{avg}/\tau$ suggestions, which roughly estimates the average number of explorations in each round. T_{avg} here refers to the average deadline length that we have observed in the safe random exploration phase. In practice, we can also set an upper threshold for the MBO batch size (e.g. 10 suggestions) to avoid the MBO calculation running too long, and affecting the proceeding training rounds.

MBO Stopping Condition The Pareto construction phase will continue until the following stopping condition is satisfied: when at least a certain number of configurations (e.g. 3% of the whole space) are explored and the EHVI value increase is less than a threshold (e.g., 1%). This stopping criterion ensures that the MBO module has explored enough configurations before stopping and does not struggle too much for small improvements.

Training Round Execution Details After the MBO module generates a batch of K suggestion points, they will be explored in the proceeding training round. As each round has its unique deadline which is unknown beforehand, BOFL may not have enough time to explore all the K configurations when the deadline length is short. Meanwhile, it is also possible that there are jobs remained after the K configurations are all explored.

In the Pareto front construction phase, we still follow the *safe exploration algorithm* (Figure 3.7) to explore the Bayesian suggestion points. With deadline guardian checking, we can drop extra suggestions to make sure the deadlines are caught. In case there are remaining jobs, we exploit the best profile of configurations we have observed to minimize energy consumption. We will present the exploitation details in Section 3.4.4.

3.4.4 Exploitation

After the Pareto construction phase, we have explored sufficiently many DVFS configurations, from which an approximated Pareto set $\mathbf{P}^{'}$ can be selected. In the exploitation phase, we use $\mathbf{P}^{'}$ as a surrogate of the actual Pareto set $\mathbf{P}$ to solve for the energy minimization problem in Eqn. (3.1).

Although we reduced the input space from $\mathbf{X}$ to $\mathbf{P}^{'}$, the optimization problem of Eqn. (3.1) is still nontrivial to solve. In general, the Pareto front curve induced by $\mathbf{P}^{'}$ is not convex, and consequently, the minimizers of Eqn. (3.1) could be a combination of multiple configurations. In BOFL, we solve the Integer Linear Programming (ILP) problem with *branch-and-bound algorithm* [108], which estimates the lower and upper bounds of the search space regions efficiently, and is widely applied in many discrete optimization problems.

	Jetson AGX	**Jetson TX2**
CPU	8-core ARM v8.2	2-core Nvidia Denver2 + 4-core ARM Cortex-A57
Frequencies	**0.42GHz → 2.26GHz (25 steps)**	**0.34GHz → 2.03GHz (12 Steps)**
GPU	512-core Volta GPU	256-core Pascal GPU
Frequencies	**0.11GHz → 1.38GHz (14 steps)**	**0.11GHz → 1.30GHz (13 steps)**
Memory	32GB 256-bit LPDDR4x	8GB 128-bit LPDDR4
Frequencies	**0.20GHz → 2.13GHz (6 steps)**	**0.41GHz → 1.87GHz (6 steps)**

Table 3.1: BOFL Testbed Hardware Specifications

As mentioned in Section 3.4.2 and Section 3.4.3, the exploitation algorithm is also applied in the exploration phase (3.7) when the candidates are fully explored, but the jobs are not finished. In such cases, we build Pareto front based on existing observations, and solve for the minimizers on the remaining jobs before the training deadline.

3.5 IMPLEMENTATION

3.5.1 Hardware Testbed

We implement BOFL on two devices, Nvidia Jetson AGX [84] and Nvidia Jetson TX2 [109]. BOFL controls the CPU, GPU and memory controller frequencies on these testbeds whose specifications are shown in Table 3.1. E.g., The CPU of Jetson AGX has a clock frequency range of 25 discrete steps from 0.42GHz to 2.26GHz, and the GPU of Jetson TX2 has 13-stepped operational frequencies ranging from 0.11GHz to 1.30GHz. Overall, the DVFS configuration spaces of Jetson AGX and Jetson TX2 have 2100 and 936 unique configurations, respectively.

3.5.2 Software Implementation

Figure 3.8 depicts an overview of BOFL's software implementation. We implement BOFL with `Python` in around 2K lines of code. It has five main modules as follows.

FL Task Executor The FL task executor ① takes a deep learning model (e.g., ResNet50) and executes the training loops with its local data. It follows the training parameters,

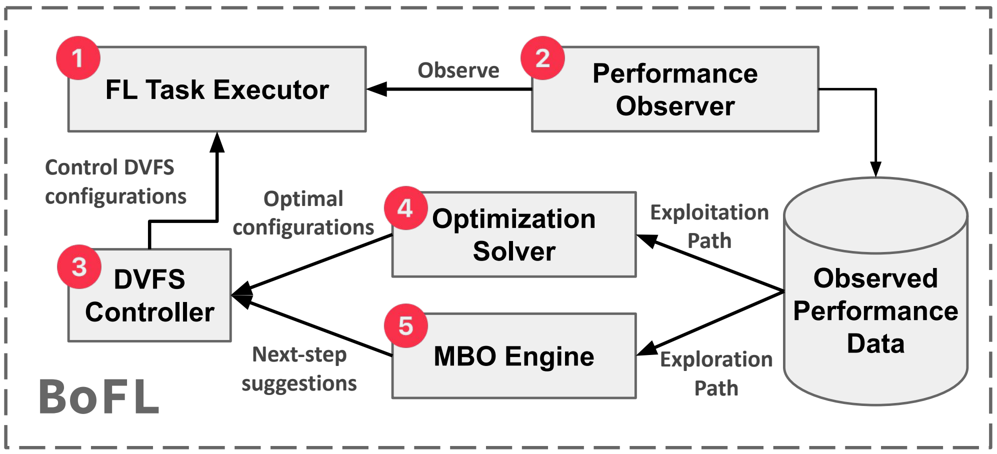


Figure 3.8: Architecture of BoFL's Implementation

e.g., minibatch size and number of epochs, specified by each FL task. We implement this module based on `Pytorch-1.10` [110], and it could be easily generalized to other machine learning platforms, such as `Tensorflow` [72]and `MXNet` [111].

Performance Observer We implement the performance observer ② to read the execution latency $\mathcal{T}(\mathbf{x})$ and energy consumption $\mathcal{E}(\mathbf{x})$ when any DVFS configuration x is applied. We use CUDA event recording APIs[9] to accurately measure the execution latency. We read the energy consumption with the built-in INA3221 power sensor [112], which can be easily accessed through the `sysfs` in Linux kernel.

DVFS Controller The DVFS controller ③ implements the main workflow in Section 3.4 and directly actuates the hardware frequencies. During exploration, it takes the Bayesian suggestions as the next-step exploration configurations. During exploitation, it actuates the operational frequencies according to the optimization results (solution for the ILP as shown in Eqn. 3.1). We modify the hardware frequencies by directly writing into the corresponding `sysfs` kernel files[10].

Optimization Solver The optimization solver ④ solves the ILP problem, Eqn.(3.1), with observed Pareto optimal configurations. We build this module with `Gurobi` optimization engine [66], which implements the *branch-and-bound algorithm* and solves Eqn. (3.1) efficiently, i.e., within 20ms.

[9]`torch.cuda.Event()` and `torch.cuda.synchronize()`.

[10]e.g., writing into "`/sys/devices/*/devfreq/*/min(max)_freq`" to modify the corresponding GPU frequencies.

MBO Engine Our MBO engine ⑤ is built on top of `Trieste` [113] which is a Bayesian optimization library implemented in `Python`. `Trieste` implements the standard BO priors, as well as a wide range of acquisition functions and batching rules, including the *EHVI* function and *sequential greedy* rule adopted by BoFL, as discussed in Section 3.4. The MBO engine is triggered before each training round in the Pareto construction phase to update the posterior estimation of $\mathcal{T}(\cdot)$ and $\mathcal{E}(\cdot)$, and generates the next-step suggestions for the DVFS controller, ③, to achieve efficient exploration.

3.6 EVALUATION

3.6.1 Methodology

Datasets & Neural Network Models We evaluate BoFL with three different federated learning tasks spanning both computer vision (CV) and natural language processing (NLP) applications. The three tasks cover three major types of neural network models, i.e., CNN, RNN and Transformer, as follows:

(1) *CIFAR10-ViT* trains Vision-Transformer model [114] (ViT) for image classification on the CIFAR10 dataset [115]. The CIFAR10 dataset contains $32p \times 32p$ color images of 10 different classes. This dataset is widely applied in image classification tasks for its lightweight.

(2) *ImageNet-ResNet50* trains ResNet50 model [116] on ImageNet dataset [117] for image classification. Compared to CIFAR10, ImageNet contains image data of more diverse classes and higher resolutions. When training with ImageNet, the images are usually cropped to $224p \times 224p$ for normalized and convenient data loading.

(3) *IMDB-LSTM* trains LSTM-RNN model [118] for text semantic analysis with IMDB movie review dataset [119]. This dataset contains more than $50K$ movie reviews with ground truth binary semantic labels, i.e., positive or negative. It is a widely used dataset for NLP model training.

Experiment Setup & FL Task Specifications To evaluate the performance of BoFL, we train the above three FL tasks on the two testbeds, i.e., Jetson AGX and Jetson TX2, each for 100 rounds. Table 3.2 presents the detailed specifications of the three FL tasks. For each task, we first specify its global parameters B and E, representing the minibatch size and number of training epochs in each round. We then load part of the datasets into

		CIFAR10-ViT	ImageNet-ResNet50	IMDB-LSTM
B		32	8	8
E		5	2	4
N	**AGX**	40	90	40
	TX2	15	30	20
$\lvert\mathbf{T}\rvert$		100		
$\mathbf{T}_{min}$	**AGX**	37.2s	46.9s	46.1s
	TX2	36.0s	49.2s	55.6s
$\mathbf{T}_{max}/\ \mathbf{T}_{min}$		$\{2.0,\ 2.5,\ 3.0,\ 3.5,\ 4.0\}$		

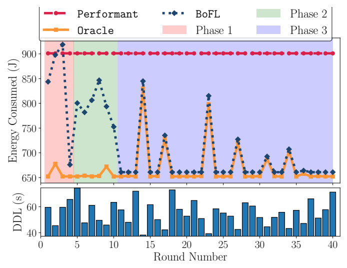
Table 3.2: Federated Learning Task Specifications

the two devices as their private training data. As mentioned in Section 3.3, N refers to the number of minibatches. For example, we load 40 batches of data, each containing 32 images into the AGX device for task *CIFAR10-ViT*. Finally, We sample 100 deadlines uniformly from the range $[\mathbf{T}_{min},\ \mathbf{T}_{max}]$. $\mathbf{T}_{min}$ is the execution latency to finish one round of training when the device is configured with maximum operational frequencies, e.g., $\mathbf{T}_{min} = \mathcal{T}(\mathbf{x}_{max}) \times W$. The $\mathbf{T}_{min}$ values in Table 3.2 are experiment measurements on the two testbeds when $\mathbf{x}_{max}$ is applied. A FL task can be finished on a device in time only if the assigned deadline is no less than $\mathbf{T}_{min}$. $\mathbf{T}_{max}$ is the deadline sampling upper bound. To evaluate the performance sensitivity of BOFL with different range of deadlines, we select a wide spectrum of $\mathbf{T}_{max}$ ranging from $2.0 \times \mathbf{T}_{min}$ to $4 \times \mathbf{T}_{min}$.

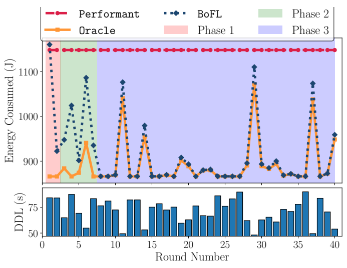
Comparison Targets To evaluate the effectiveness of BOFL, we compare it with two other designs as follows:

(1) PERFORMANT. The Performant design is the default DVFS configuration for real-time tasks. It turns all the hardware units into maximum operational frequencies, i.e., $\mathbf{x}_{max}$, to maintain stable performance, and make sure the deadlines will not miss. We compare BOFL with the PERFORMANT design to show that our algorithm can significantly reduce the energy consumption.

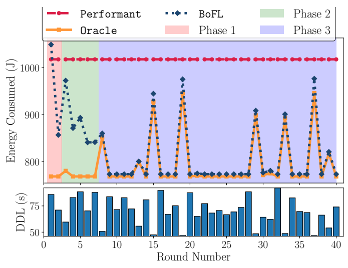
(2) ORACLE. In the ORACLE design, we profile $\mathcal{T}$ and $\mathcal{E}$ over the whole configuration space offline, and only run exploitation over the FL training rounds to achieve optimal energy usage. Note that ORACLE can not be achieved in practice as it requires long-lasting offline profiling. We compare BOFL with the ORACLE design to show that we achieve near-optimal energy efficiency with little regret.



(a) CIFAR10-ViT

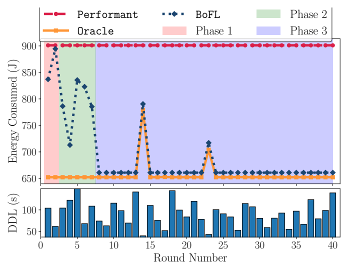


(b) ImageNet-ResNet50

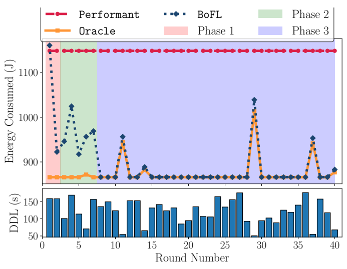


(c) IMDB-LSTM

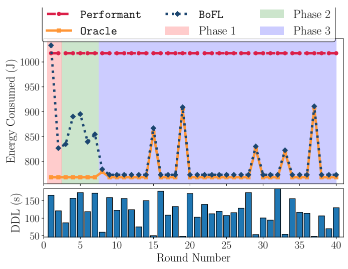
Figure 3.9: Energy consumption for the first 40 rounds of FL training on AGX testbed with $\mathbf{T}_{max}/\mathbf{T}_{max} = 2$



(a) CIFAR10-ViT



(b) ImageNet-ResNet50



(c) IMDB-LSTM

Figure 3.10: Energy consumption for the first 40 rounds of FL training on AGX testbed with $\mathbf{T}_{max}/\mathbf{T}_{max} = 4$

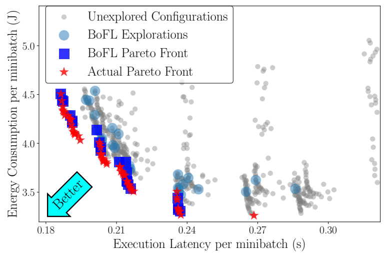
3.6.2 BoFL Energy Efficiency

We evaluate the energy efficiency of BoFL as shown in Figure 3.9 and 3.10. We plot the energy consumption of BoFL, as well as the two baselines, for the first 40 training rounds. The deadlines for each round are presented together with the energy usage. We also highlight the three algorithm phases of BoFL to better illustrate the *exploration and exploitation* trade off in our solution.

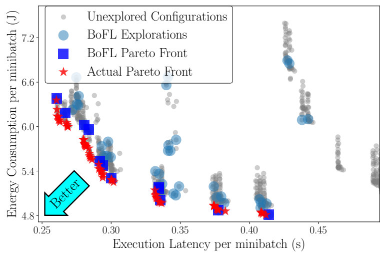
Figure 3.9(a) depicts our experiment results for *CIFAR10-ViT* task measured on the AGX testbed. As the figure shows, BoFL can reduce the overall energy consumption substantially comparing to the PERFORMANT baseline, and achieves pretty close energy usage comparing to the ORACLE target in the exploitation phase. BoFL takes the first 10 training rounds (phase 1 & 2) to explore the configuration space, and runs exploitation in all the remaining rounds. BoFL outperforms PERFORMANT consistently over the whole training process, except one round in phase 1 when BoFL inevitably meet some bad configurations during random exploration. Comparing to the ORACLE design, it is clear that the two energy curves, of BoFL and ORACLE, almost coincide in the exploitation phase. The major energy overhead comes from phase 1 and 2 when BoFL focuses on the exploration but not energy optimization. Note that in practice, a FL model may take $500 \sim 10000$ rounds to converge [20]. An exploration phase of around 10 rounds generates negligible overhead comparing to that of the dominantly long exploitation phase.

Overall, BoFL reduces energy consumption by 22.3% compared with Performant and generates 3.48% energy overhead compared to Oracle for the experiment as shown in Figure 3.9(a). Similar results can be observed from all remaining plots in Figures 3.9 and 3.10. Comparing Figure 3.9 and Figure 3.10, it is clear the longer deadlines reduce the spikes in the energy curves, as it provides more space to pace down for energy optimization. We will present more results to showcase how deadline length influences the performance of BoFL in Section 3.6.4.

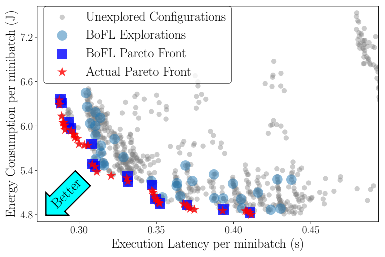
3.6.3 BoFL Pareto Construction



(a) CIFAR10-ViT



(b) ImageNet-ResNet50



(c) IMDB-LSTM

Figure 3.11: A comparison between BoFL searched Pareto fronts and the actual Pareto fronts on AGX testbed.

In Figure 3.11, we present the Pareto front constructed by BoFL (as shown in blue squares), as well as the actual Pareto front derived from offline profiling (as shown in red stars). It is clear that BoFL can successfully find a close approximation to the actual Pareto front over all three tasks. We further plot all remaining BoFL explorations in blue circles, and the unexplored configurations in gray dots. As the figure shows, BoFL makes most of its explorations around the Pareto front while seldom trials in the less performant area, because our solution can smartly search the configuration space with Bayesian optimized suggestions and skip a lot of sub-optimal points. Note that we only plot a small part of the whole configuration space around the Pareto front for presentation clarity. There are much more sub-optimal and unexplored points than those as shown in Figure 3.11. In our experiments, the Pareto front can be efficiently constructed after exploring just 3% of the whole configuration space.

	CIFAR10-ViT		ImageNet-ResNet50		IMDB-LSTM	
Round	**# Exp**	**# Pareto**	**# Exp**	**# Pareto**	**# Exp**	**# Pareto**
1	**9**	**1**	**17**	**2**	**16**	**1**
2	**2**	**0**	**4**	**0**	**5**	**0**
3	**8**	**1**	**10**	**2**	**5**	**0**
4	**2**	**0**	**10**	**0**	**10**	**3**
5	**10**	**1**	**10**	**5**	**10**	**4**
6	**6**	**4**	**7**	**2**	**10**	**3**
7	**10**	**2**	**10**	**2**	**10**	**3**
8	**7**	**2**				
9	**8**	**6**				
10	**8**	**3**				
Total	**70**	**20**	**68**	**13**	**66**	**14**

Table 3.3: The number of Explorations and searched Pareto points for each round in the first two phases. **Red** numbers are for the safe random exploration phase. **Blue** numbers are for the Pareto construction phase. E.g., In the first round of CIFAR10-ViT task, BOFL is in the random safe exploration phase. It explores 9 configurations with one of them being in the ultimate Pareto front.

We further present an example to walk through how BOFL explores the space and searches for the Pareto optimal configurations, as shown in Table 3.3. For the *CIFAR10-ViT* task, BOFL first randomly searches the whole configuration space for 4 rounds and explores 21 different configurations. After that, BOFL switches into the Pareto construction phase and starts taking exploration suggestions from the MBO module. The second phase continues for 6 rounds and explores 49 points before the ending criteria (Section 3.4.3) is satisfied. During the whole exploration process, 70 configurations are explored, in which 20 of them constitute the Pareto front, i.e., the blue squares as shown in Figure 3.11(a). It can be easily observed that most of Pareto front points, i.e., 18 out of 20, are searched in the second phase, because the Bayesian optimization algorithm focuses on exploring more promising regions and searches for Pareto front more efficiently than random sampling. Similar patterns can also be observed in the other two tasks in Table 3.3.

3.6.4 Sensitivity to Deadline Length

We evaluate how deadline length influences BOFL's effectiveness as follows. We run 6 set of experiments with different ranges of deadline length, i.e., $\mathbf{T}_{max}/\ \mathbf{T}_{min} \in [2, 4]$, as shown in Table 3.2, measuring the following two metrics to evaluate BOFL's performance:

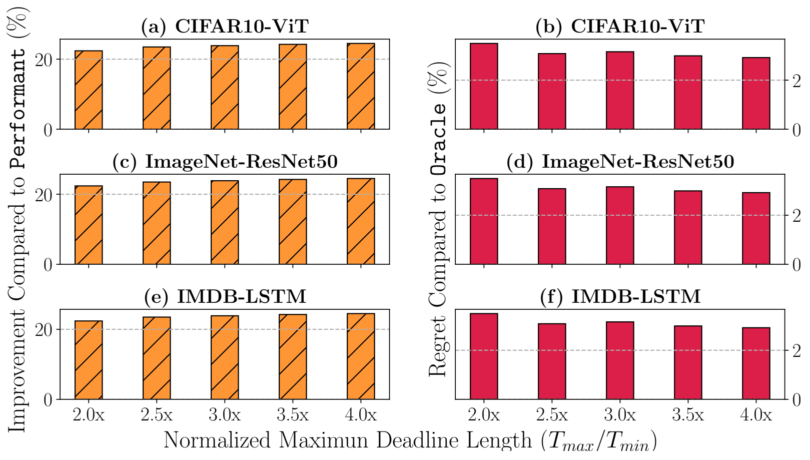


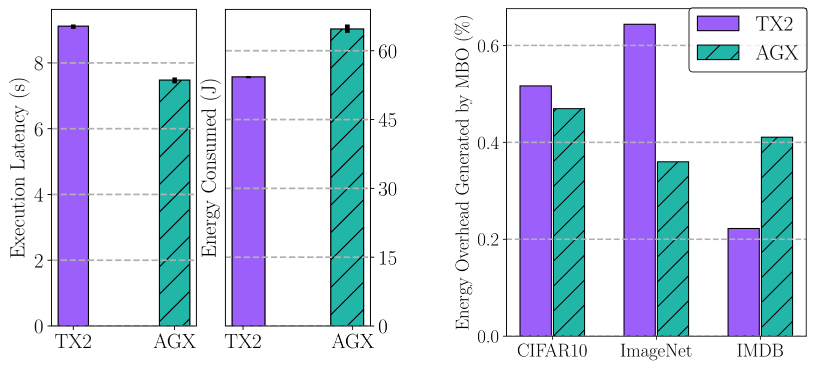
Figure 3.12: BoFL's effectiveness with different deadline length.

(1) *Improvement compared to* PERFORMANT. A metric represents the energy being reduced by BoFL compared to PERFORMANT, i.e., $1 - \frac{\text{BoFL energy usgae}}{\text{Performant energy usgae}}$.

(2) *Regret compared to* ORACLE. A metric represents how much energy overhead is generated by BoFL compared to ORACLE, i.e., $\frac{\text{BoFL energy usgae}}{\text{Oracle energy usgae}} - 1$.

The evaluation results are presented in Figure 3.12. As the figure shows, BoFL's improvement compared to PERFORMANT steadily increases as the FL tasks are assigned with longer deadlines. As the deadlines are getting longer, BoFL has larger optimization space for pace control, it can train the model slower to benefit from less energy consumption, which explains the increasing curve. As the deadlines are getting longer, BoFL's strategy will gradually converge to choosing the most energy-efficient configuration, i.e., the bottom point in the Pareto front as shown in Figure 3.11, which makes the overall energy consumption stable. Overall, BoFL can reduce energy consumption by $20.3\% \sim 25.9\%$ compared to the PERFORMANT baseline.

As the deadline increases, BoFL's regret compared to ORACLE steadily decreases. In case the deadlines are short, BoFL tends to spend more time in exploration, which generates more energy regret. As shown in Figure 3.9(a) and 3.10(a) for *CIFAR10-ViT* task, BoFL explores 10 rounds before exploitation when $\frac{\mathbf{T}_{max}}{\mathbf{T}_{min}} = 2$, while only explores 6 rounds when $\frac{\mathbf{T}_{max}}{\mathbf{T}_{min}} = 4$. The reason is that longer deadlines allow BoFL to explore more configurations in one round, which leads to a well constructed Pareto in fewer rounds. Overall, BoFL generates energy consumption regret by $1.2\% \sim 3.4\%$ compared to the ORACLE target.



(a) MBO overhead per round. (b) Overall energy overhead.

Figure 3.13: Overhead of the MBO module

3.6.5 MBO Module Overhead

In BoFL, we use MBO to smartly generate the next-step exploration points. However, calculating these suggestions does not come for free. In the rest of this section, we evaluate the overhead of the MBO module in terms of two metrics: (1) MBO calculation latency, and (2) MBO energy consumption. We present the evaluation results in Figure 3.13. As shown in Figure 3.13(a), the MBO module may take $6 \sim 9$ seconds to update its estimation and generate next-step suggestions. Note that the MBO calculation happens outside the time critical model training and introduces zero overhead to the FL workloads. In practice, the Bayesian optimization calculation can be co-located when the client is sending or receiving models from the server, which usually takes $10 \sim 20$ of seconds, and is long enough for the MBO calculation. For weak devices that takes longer time to update their Bayesian models, we can always speed up the MBO calculation by reducing its suggestion batch size. As Figure 3.13(a) depicts, one round of MBO calculation usually consumes $50 \sim 70$ Joule energy, which is much smaller than that of model training, which usually takes $600 \sim 1200$ Joule, as shown in Figures 3.9 and 3.10. As MBO only happens a few times during the Pareto construction phase, the overall energy overhead generated by MBO is as small as $0.4\% \sim 0.7\%$, as presented in Figure 3.13(b).

3.7 RELATED WORK

Federated learning Federated learning brings collaborative intelligence into multiple domains, including health care [120, 121, 122], smart transportation [123, 124], localization service [125, 126], recommendation system [127, 128] and beyond. Significant work has been proposed to improve the performance of FL in multiple perspectives, which contains but is not limited to model optimization [85, 129, 130], privacy preserving[91, 131], and model personalization [132, 133]. In this chapter, we present BOFL to improve FL from an energy efficient perspective.

Dynamic voltage frequency scaling Dynamic voltage frequency scaling (DVFS) is a widely applied technique for managing power and performance of processors, such as CPU and GPU. A large amount of research has been shown to achieve better energy efficiency with the benefit of DVFS [27, 28, 29, 30, 31]. In this chapter, we design a model training pace controller based on multi-axes DVFS to minimize the energy consumption of edge devices during federated learning tasks.

Bayesian optimization and its applications Bayesian optimization (BO) is a sample-efficient optimization framework for many blackbox functions that are expensive to evaluate. BO has been applied across multiple fields, i.e., A/B Testing [134, 135], robotics motion planing [136, 137], drug discovery [138], cherry picking cloud configurations [139], and more recently as an important ingredient for neural architecture search [140, 141, 142], which automates the process of neural network design. CherryPick [139] is the first work applying BO on system configuration selection for cloud clusters. It applies single-object BO to minimize the overall operational cost. In this work, we apply BO to efficiently search the vast DVFS design space for energy-aware federated learning. Compared to CherryPick, our problem is more challenging due to the multi-dimensional optimization targets, i.e., a joint minimization of latency and energy consumption.

3.8 CONCLUDING REMARK

In this chapter, we present BOFL, a local training pace controller for edge devices to achieve energy efficient federated learning. Experiments on multiple edge devices over multiple neural network models show that BOFL can reduce energy consumption of model training by more than 20% compared to the PERFORMANT baseline, and achieve close to optimal energy efficient compared to the ORACLE target with as low as 1.2% - 3.4% energy regret. Through this work, we have illustrated that **hardware configurability** is a crucial avenue for achieving **multi-objective resource optimization** in energy-efficient federated learning systems.

CHAPTER 4: FEDCORE: STRAGGLER-FREE FEDERATED LEARNING WITH DISTRIBUTED CORESETS

In this chapter, we examine the complexities of the third thesis project, FEDCORE. We introduce a multi-objective resource optimization framework aimed at mitigating the stragglers' issue in federated learning while **enhancing the training speed** and **maintaining the models' performance**. This achievement is made possible through the strategic utilization of **training data redundancy**, as detailed in Section 1.2.

4.1 INTRODUCTION

Federated learning (FL) enables multiple clients to collaboratively train a shared machine learning model while retaining their data locally. It has greatly enhanced various privacy-sensitive domains by harnessing the power of AI and providing tailored solutions, including cancer diagnosis [23, 24, 143], urban transportation surveillance [78, 124], financial services [144, 145] and beyond. Federated learning has given rise to several research areas, including model convergence optimization [85, 129, 130], FL system efficiency [33, 146, 147], privacy preservation [91, 131], and robustness against adversarial attacks [148]. Among these areas, the straggler problem, caused by slow or unresponsive clients, hinders overall training efficiency and scalability. Meta's million-client FL system, Papaya [9], demonstrated that per-client training time distribution spans over two orders of magnitude, and the round completion time is 21x larger than the average training time per client due to stragglers' delays. Thus, efficient straggler mitigation is vital to unlock FL's full potential across diverse applications.

Motivations. Existing solutions like client selection mechanisms [146, 147, 149] and asynchronous frameworks [9, 129, 150, 151, 152, 153] aim to mitigate the straggler issue in federated learning (FL). However, these methods inherently treat the symptoms rather than the cause. Client selection can result in biased training *data* due to the exclusion of slower clients, while asynchronous approaches can encounter staleness and inconsistency due to laggard updates from stragglers with slow *hardware*. These strategies don't directly address the root cause of the straggler issue, which is due to the *system and data volume heterogeneity* among clients in FL. The disparities in both computational capacity and data volume lead to varied training times, impacting overall efficiency.

Instead of sidestepping this fundamental challenge, our approach confronts it directly by *aligning each client's data volume with its computational capability*. Recognizing that upgrading clients' hardware is impractical, we propose adjusting the amount of data processed by each slow client. These straggler clients often hold more data than can be efficiently processed within the allotted round time. To address this, we propose creating a representative **coreset**, a compact subset of the full dataset that encapsulates essential learning information. This strategy offers a more precise and direct solution to the straggler problem in FL.

In contrast to existing coreset generation solutions [38, 154], where training data are collected on a central server to create a single coreset, we propose a *distributed* approach that forms training coresets on each client independently, maintaining the *privacy* integral to FL. This task is challenging, particularly when dealing with heterogeneous data distribution across numerous clients, each requiring different coreset sizes based on their computational capabilities. Further complexity arises from the dynamic nature of machine learning models, which is constantly updated during the training process, necessitating the creation of adaptive coresets that can be adjusted according to different model parameters and training phases. To tackle these issues, we designed FEDCORE which addresses two key **questions**:

(**Q1**) How can we select statistically unbiased coresets that adapt to continuously updated models?

(**Q2**) How to seamlessly integrate coreset generation with minimal overhead into FL frameworks?

Methods and Results. To generate statistically unbiased coresets that adapt to the evolving ML models, we design FEDCORE, which is applied independently to each client. FEDCORE operates by periodically searching for a coreset at the start of each FL round, ensuring that the selected coresets may differ between training rounds. This adaptability allows for the provision of the most suitable learning samples, taking into account the varying model parameters at different stages of training (**Q1**). To minimize coreset generation overhead, we employ gradient-based methods that leverage the per-sample gradients obtained during the gradient descent model training. By repurposing these gradients as input for our coreset algorithm, we optimize the use of available resources and eliminate the need for additional computations. Furthermore, we tackle the intricate coreset optimization problem by transforming it into a more manageable k-medoids clustering problem. This transformation allows for a more efficient resolution of the optimization task, streamlining the overall process and minimizing the system overhead (**Q2**). Overall, this chapter offers the following **contributions**:

(**1**) We design and implement the FEDCORE algorithm, a pioneering solution that leverages distributed coreset training to address the straggler problem in FL with minimal system overhead.

(**2**) We provide a theoretical convergence analysis for the FEDCORE algorithm, which manages to incorporate the coreset gradient approximation error with the federated optimization error, proving that federated model training with per-client coresets results in highly accurate models.

(**3**) We extensively evaluate FEDCORE against existing solutions and baselines. Evaluation results indicate an 8x reduction in FL training time without degrading model accuracy compared to baseline FEDAVG. In comparison to FEDPROX, which handles stragglers through fewer local training epochs, FEDCORE consistently achieves faster convergence and high model accuracy.

The rest of the thesis chapter is organized as follows. We survey related literature in Section 4.2. In Section 4.3, we present the problem setups. In Section 4.4, we present detailed FEDCORE algorithms and system framework. We provided convergence analysis for FEDCORE in Section 4.5. Section 4.6 presents the implementation and evaluations of FEDCORE system. Finally, Section 4.8 concludes the chapter.

4.2 RELATED WORK

Coreset Methods for Deep Learning Coreset methods are effective in reducing computational complexity and memory requirements in deep learning. They are based on selecting a representative subset, or coreset, from the original dataset to retain essential information while significantly reducing data size. Coresets have been successfully applied to tasks like image classification [34, 35, 36], natural language processing [37, 38], and reinforcement learning [155, 156, 157]. Several approaches for efficient coreset creation include:

1. *Geometry Based Clustering* [158, 159, 160], assuming data points in close proximity share similar properties and forming a coreset by removing clustered redundant data points;
2. *Loss Based Sampling* [161, 162, 163], prioritizing training samples based on their contribution to the error or loss reduction during neural network training and selecting the most important samples to form the coreset;

3. *Decision Boundary Methods* [164, 165], focusing on selecting data points near the decision boundary as the coreset, as they carry more informative content for model training; and

4. *Gradient Matching Solutions* [38, 154, 166], aiming to select a coreset that closely approximates the gradients produced by the full training dataset during deep model training, ensuring minimal gradient differences.

In this chapter, we adopt gradient matching methods to construct distributed coresets across federated learning clients. By utilizing per-sample gradients produced during model training, coresets can be efficiently computed with minimal overhead.

Straggler Prevention in Federated Learning. Stragglers, slow or unresponsive clients in federated learning, can significantly impact training efficiency and model convergence. Various strategies have been proposed to address this challenge, including:

1. *Client Selection* methods [146, 147, 149, 149] mitigate the impact of stragglers by adaptively selecting a subset of clients based on their performance, training speed, or other criteria. However, this approach may introduce bias in heterogeneous settings, as stragglers with unique and important learning samples could be excluded;

2. *Asynchronous FL* techniques [9, 129, 150, 151, 152, 153] eliminate the need for synchronized communication, enabling clients to update local models and communicate with the server independently. Although asynchronous FL can reduce straggler impact, it may suffer from staleness and inconsistency issues affecting the model performance;

3. *Accommodating Partial Work from Stragglers* approaches [86, 167, 168] adjust local epoch numbers or allow clients to perform partial updates. FEDPROX [86] introduces a proximal term in the optimization process, accommodating partial updates without severely affecting model convergence.

In this chapter, we propose FEDCORE, a novel straggler-resilient training method based on partial-work. Unlike most existing works reducing the number of local epochs, FEDCORE reduces the number of training samples by creating a coreset. This approach enables FEDCORE to perform more local optimization steps and explore gradients more deeply, resulting in faster convergence speed and better model accuracy.

4.3 PRELIMINARIES

4.3.1 Federated Leaning System Setup

Consider a set of clients, $U = \{1, \dots, n\}$. For each client u^i, we define $V^i = \{1, \dots, m^i\}$ as the index set of its training samples, where m^i represents the size of the training set. The j-th data point in the training set of client u^i is denoted as (x^i_j, y^i_j), with $j \in V^i$. x^i_j and y^i_j represent the data and label, respectively. In Federated Learning (FL), the primary objective is to minimize an empirical risk function using the training data from each client. Given a loss function L, a machine learning model f, and the model parameter space $\mathcal{W}$, the FL problem can be formulated as:

$$w_* = \operatorname*{argmin}_{w \in \mathcal{W}} \mathcal{L}(w), \text{ where } \mathcal{L}(w) := \sum_{i \in U} p^i \mathcal{L}^i(w), \;\; \mathcal{L}^i(w) := \frac{1}{m^i} \sum_{j \in V^i} \mathcal{L}^i_j(w), \tag{4.1}$$

Here, $\mathcal{L}^i_j(w) := L(f(w, x^i_j), y^i_j)$ represents the empirical loss for each sample (x^i_j, y^i_j), and $p^i = \frac{m^i}{\sum_{i \in U} m^i}$ is the weight proportional to the training set size. However, privacy concerns prevent a central server from directly accessing the clients' data and solving Eq.(4.1). As an alternative, FL algorithms require each client to solve a local problem, $w^{i,*} = \operatorname{argmin}_{w \in \mathcal{W}} \mathcal{L}^i(w)$, using their data independently. Through iterative rounds of communication, the central server aggregates the local models of each client and approximates the solution to Eq.(4.1).

In FL, clients typically use gradient descent based algorithms like SGD and ADAM for local training. The objective is to provide an unbiased estimate of the full gradient, denoted as $\nabla \mathcal{L}^i(w) = \sum_{j \in V^i} \frac{\partial \mathcal{L}^i_j}{\partial w}$. SGD optimizers calculate model-gradients based on randomly selected mini-batches of training samples through all the training samples in V^i. This constitutes one *epoch* of training. In conventional FL, each client performs SGD for multiple epochs, i.e., E epochs, before sending its gradients to the central server for global model synchronization. This entire process constitutes one FL *round*. The central server then aggregates the received gradients from participating clients and updates the global model. After multiple rounds, i.e., R rounds, of training and synchronization, the global model converges to a satisfactory performance.

The heterogeneity of client training data size and computational capabilities leads to considerable variation in per round training times in Federated Learning. To illustrate, let c^i represent the computational capability of the i-th client, which can be inferred from their hardware specifications. Here, u_i takes $1/c^i$ seconds to train one data sample. Hence, the per-round training time is $E\frac{m^i}{c^i}$, where E is the number of epochs per round. Due to the synchronous nature of FL, slower clients can significantly delay the overall training process, resulting in the *straggler problem*.

4.3.2 Distributed Coresets for Federated Learning.

In FEDCORE, our goal is to address the straggler problem by strategically selecting a small subset $S^i \subseteq V^i$ of the full training set V^i for each u^i. This enables the model to be trained only on the subset S^i while still approximately converging to the globally optimal solution (i.e., the model parameters that would be obtained if trained on the entire V^i).

Inspired by existing works in gradient-based coreset construction [38, 154], the key idea in FEDCORE is identifying a small subset S^i with the weighted sum of its elements' gradients closely approximating the full gradient over V^i. Unlike previous works, our approach generates distributed coresets across all clients $u^i, i \in U$, while still providing global model convergence properties.

To further resolve the straggler problem, we impose a training deadline τ on every client, ensuring that each u^i can complete one round of training within τ seconds using the coreset S^i. Consequently, $c^i\tau$ represents the maximum number of data samples that can be processed by u^i within a single training round. We specifically formulate the distributed coresets generation problem as follows:

$$(S^{i,*}, \delta^{i,*}) = \underset{S^i \subseteq V^i, \delta^i \in \mathbb{R}_+^{|S^i|}}{\operatorname{argmin}} \mathcal{E}^i(w, S^i, \delta^i), \text{ s.t. } |S^i| \le c^i\tau/E, \ \forall i \in U. \tag{4.2}$$

Here $\mathcal{E}^i(w, S^i, \delta^i) := \left\| \sum_{j\in V^i} \nabla\mathcal{L}^i_j(w) - \sum_{k\in S^i} \delta^i_k \nabla\mathcal{L}^i_k(w) \right\|$ is the 2-normed distance between the full-set gradient and the coreset gradient when the model parameter is w. δ^i is the weight vector of the coreset elements with $\dim(\delta^i) = |S^i|$.

Unfortunately, directly solving the aforementioned optimization problem is infeasible due to three main obstacles:

a) Finding the optimal coreset $(S^{i,*}, \delta^{i,*})$ is an NP-hard task, due to the combinatorial nature of the problem, even when the per-element gradient, $\mathcal{L}^i_j$, can be calculated through SGD training.

b) Deep machine learning models have high-dimensional model gradients, containing millions of parameters. Solving the above optimization problem with high-dimensional vectors is practically unmanageable.

c) Eq.(4.2) needs to be recalculated for every time the model parameter w gets updated, which further intensifies the computational complexity.

In the following sections, we introduce the design of FEDCORE and illustrate how it effectively addresses these challenges.

4.4 FEDCORE ALGORITHM AND SYSTEM

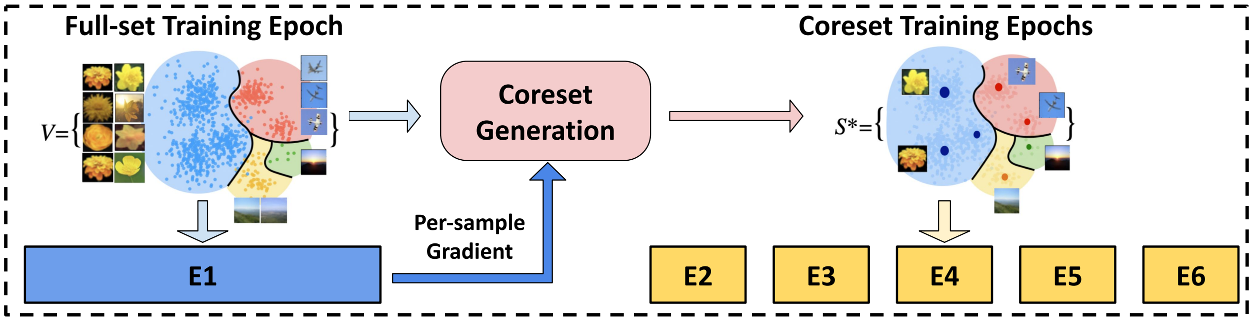


Figure 4.1: An example workflow of FEDCORE encompasses a single training round consisting of 6 epochs.

Algorithm 4.1: FEDCORE Algorithm

```
Input: $K$: # selected clients per round; $R$: # training rounds; $w_0$: initial model parameter.
    $E$, $\tau$, and $V^i$, $c^i$, $m^i$, $p^i$ for all $i \in U$, as defined in Section 4.3.
for $r = 0, 1, \cdots, R$ do
    Server randomly selects a subset of $K$ clients $U_r$. Each $u^i$ is chosen with probability $p^i$.
    Server sends current model $w_r$ and round deadline $\tau$ to all chosen clients $u^i, i \in U_r$.
    for each $i \in U_r$ do
        if $E \cdot m^i < c^i\tau$ then
            Client $u^i$ executes $E$ epochs of local training with its full-set $V^i$.
        else
            Client $u^i$ generates approximated gradient distance, either $\widetilde{d^i_{j,k}}$ or $\widehat{d^i_{j,k}}$ for convex
              models and neural networks, respectively, over the full-set $V^i$ in the first epoch.
            Client $u^i$ constructs coreset $(S^{i,*}, \delta^{i,*})$ by solving the k-medoids problem.
            Client $u^i$ executes $E - 1$ epochs of local training with its coreset $(S^{i,*}, \delta^{i,*})$.
        end
        Client $u^i$ sends its round-end local parameter $w^i_r$ back to the server.
    end
    Server aggregates the new global model: $w_{r+1} = \frac{1}{K}\sum_{i \in U_r} w^i_r$.
end
```

4.4.1 FedCore Algorithm Overview.

We present the FedCore workflow in Algorithm 4.1. FedCore operates in multiple training rounds, denoted by R. Like most existing works [86], the server selects K clients randomly, with probabilities proportional to their training set size, i.e., $p^i = \frac{m^i}{\sum_{i \in U} m^i}$ (line 4.1). The server sends the current model parameter and round deadline τ to the selected clients for distributed training (line 4.1). Clients assess if they can complete full-set training within τ. If possible, they execute E epochs of SGD training over its full-set V^i (line 4.1). Otherwise, they generate a training coreset and train using it (line 4.1 - 4.1). Finally, clients send their local parameters to the server at the end of each training round, which aggregates them to form a new global model (line 4.1).

To circumvent the need to solve Eq.(4.2) for every different model parameter w (i.e., every epoch), we design FedCore to search for a suitable coreset periodically at the beginning of each FL round. Figure 4.1 illustrates the workflow of FedCore during one FL round. In the first epoch, FedCore processes the entire training set, taking a comprehensive initial optimization step and generating per-sample gradients for coreset creation. For the remaining epochs, FedCore operates on a coreset, significantly reducing training time and mitigating the effects of stragglers.

By minimizing the upper bound of gradient estimation dissimilarity (i.e., $\mathcal{E}^i$), we transform Eq.(4.2) into a *k-medoids problem*, which can be solved approximately in polynomial time (Section 4.4.2). We also use low-dimensional gradient approximations as input for the coreset algorithm instead of high-dimensional model gradients, making coreset generation more efficient and lightweight (Section 4.4.3). The distributed coresets generated through our approach provide strong global convergence guarantees (Section 4.5). In the following sections, we detail the design of these algorithms and discuss practical techniques to accelerate FedCore.

4.4.2 Upper Bounding Dissimilarity Estimation with K-Medoids

We aim to construct a coreset S^i with b^i elements for each client u^i. In order to allow for the first epoch of every training round to be full-set with m^i training elements, we set $b^i = \lfloor \frac{c^i \tau - m^i}{E-1} \rfloor$ to meet the computational capability $c^i \tau - m^i$ of u^i in the remaining $E - 1$ epochs.

To upper bound the dissimilarity between the full-set gradient and the weighted coreset gradient on S^i, first consider a mapping function $\Phi^i : V^i \to S^i$ that, for every possible model parameter $w \in \mathcal{W}$, assigns each data point $j \in V^i$ to one of its coreset elements

$k \in S^i$, i.e., $\Phi^i(j) = k \in S^i$. Let $C_k^i := \{j : \Phi^i(j) = k\} \subseteq V^i$ represent the set of data points assigned to data point $k \in S^i$, and let $\delta_k^i := |C_k^i| \in \mathbb{N}_+$ denote the number of such points. Thus, for any arbitrary $w \in \mathcal{W}$, we have

$$\sum_{j \in V^i} \nabla \mathcal{L}_j^i(w) - \sum_{k \in S^i} \delta_k^i \nabla \mathcal{L}_k^i(w) = \sum_{j \in V^i} (\nabla \mathcal{L}_j^i(w) - \nabla \mathcal{L}_{\Phi^i(j)}^i(w)) \tag{4.3}$$

By applying the triangle inequality on both sides, we derive an upper bound for the normed error between the full-set gradient and the weighted coreset gradient, i.e.,

$$\mathcal{E}^i(w, S^i, \delta^i) = \left\| \sum_{j \in V^i} \nabla \mathcal{L}_j^i(w) - \sum_{k \in S^i} \delta_k^i \nabla \mathcal{L}_k^i(w) \right\| \leq \sum_{j \in V^i} \left\| \nabla \mathcal{L}_j^i(w) - \nabla \mathcal{L}_{\Phi^i(j)}^i(w) \right\|. \tag{4.4}$$

Note that the upper bound in Eq.(4.4) is minimized when Φ^i assigns every data point $j \in V^i$ to the element $k \in S^i$ with the most similar gradient, i.e., $\Phi^i(j) = \operatorname{argmin}_{k \in S^i} d_{j,k}^i(w)$, where $d_{j,k}^i(w) = \left\| \nabla \mathcal{L}_j^i(w) - \nabla \mathcal{L}_k^i(w) \right\|$. Hence,

$$\min_{S^i \subseteq V^i, \delta^i \in \mathbb{N}_+^{|S^i|}} \mathcal{E}^i(w, S^i, \delta^i) \leq \min_{S^i \subseteq V^i} \left\{ \sum_{j \in V^i} \min_{k \in S^i} d_{j,k}^i(w) \right\}. \tag{4.5}$$

Recall that the minimum value of Eq.(4.2) is further upper bounded by the left hand side of Eq.(4.5), as it has a larger feasible set for its weight vector $\delta^i \in \mathbb{R}_+^{|S^i|}$. Hence, we can adjust the optimization objective of Eq.(4.2) to the right hand side gradient dissimilarity upper bound as follows:

$$(S^{i,*}, \delta^{i,*}) = \operatorname*{argmin}_{S^i \subseteq V^i} \left\{ \sum_{j \in V^i} \min_{k \in S^i} d_{j,k}^i(w) \right\}, \quad \text{s.t. } |S^i| \leq b^i, \ \forall i \in U, \tag{4.6}$$

where $\delta^i \in \mathbb{N}_+^{|S^i|}$ is the weight vector associated with S^i, given by

$$\delta_k^{i,*} = \left| \left\{ j \in V^i : k = \operatorname*{argmin}_{l \in S^{i,*}} d_{j,l}^i(w) \right\} \right|. \tag{4.7}$$

Note that Eq.(4.6) is a *k-medoids problem* with a budget size of b^i. The goal is to minimize the objective function by finding the b^i medoids of the entire training set in the gradient space.

The *k-medoids problem* is a clustering technique forming k clusters based on data point similarities. K-medoids use actual data points as cluster centers, i.e., the medoids, minimizing dissimilarities between data points and their respective medoids. These medoids form a coreset for our Federated Learning problem. Multiple algorithms [38, 169, 170] have been proposed for this problem, offering diverse computational efficiency and quality trade-offs. In our case, we employ the FasterPAM algorithm, known for its speed and accuracy in identifying optimal medoids, efficiently minimizing our equation Eq.(4.6). In essence, FasterPAM quickly solves the k-medoids problem, generating coresets for large datasets within one second.

4.4.3 Accelerating Coreset Generation with Gradient Approximation.

Solving the k-medoids problem for each $w \in \mathcal{W}$, as illustrated in Eq.(4.6), requires calculating every pairwise gradient difference for the entire training set (i.e., $d^i_{j,k}(w), \forall j, k \in V^i$). Nonetheless, directly computing the gradient-distances is computationally costly due to the typically high-dimensional nature of the full model gradient, especially in the case of deep neural networks with millions of parameters. This leads to a computationally burdensome k-medoids clustering process. Following the approach in [38], we tackle this challenge by substituting the full gradient differences with lightweight approximations for two general types of machine learning models as below.

Convex Machine Learning Models. We utilize the method from [171] that allows for effective gradient distance approximation in convex machine learning models like linear regression, logistic regression, and regularized SVMs. This method approximates the gradient difference between data points using their Euclidean distance, a principle that uniformly applies across the entirety of the parameter space, $\mathcal{W}$. By substituting $d^i_{j,k}(w)$ with $\widetilde{d}^i_{j,k}(w) = \|x^i_j - x^i_k\|$ in Eq.(4.6), the coreset problem is reframed into a 2-norm k-medoids clustering within the original data space. This adjustment facilitates coresets formation using pre-calculated pairwise Euclidean distances, eliminating per-round generation and reducing training-time cost.

Deep Neural Networks. In deep neural networks, gradient changes primarily reflect the loss function's gradient relative to the last layer's input [172]. The normed differences of gradients between data points can be effectively bounded as below:

$$\forall i, j, k, \quad d^i_{j,k}(w) \leq \widehat{d}^i_{j,k}(w) = c_1 \cdot \left\|\partial \mathcal{L}^i_j(w)/\partial z^i_j - \partial \mathcal{L}^i_k(w)/\partial z^i_k\right\| + c_2, \tag{4.8}$$

Here, z_j^i is the input to the last neural network layer from data point x_j^i, and c_1 and c_2 are constants. We substitute $d_{j,k}^i$ with $\widehat{d_{j,k}^i}$ in Eq.(4.6) for optimization. $\left\|\partial\mathcal{L}_j^i(w)/\partial z_j^i\right\|$ is attainable from the first epoch of full-set training and requires no extra computation. In FEDCORE, we derive $\widehat{d_{j,k}^i}$ for all pairs $j, k \in S^i$ in the first FL epoch, thus alleviating the load of high-dimensional k-medoids clustering.

4.4.4 Discussions of Design Choices.

In designing FEDCORE, we intentionally set the first epoch to train on the entire dataset, generating (approximated) per-sample gradients for k-medoids coreset generation. However, heavy loaded straggling clients may struggle to complete the initial epoch[11], i.e., $c^i\tau < m^i$. In such cases, FEDCORE can use faster coreset methods not requiring a full epoch of forward and backward propagation. As explained in Section 4.4.3:

- Convex FL models can use static coresets to achieve model convergence and train with pre-computed coresets in any epoch, i.e., calculate corsets with pre-computed $\widetilde{d_{j,k}^i}$;
- Deep neural networks compute approximated pairwise gradient distance, $\widehat{d_{j,k}^i}$ that is attainable almost as cheap as calculating the loss (with only one step of gradient calculation for the last layer input), instead of a full epoch of forward and backward propagation.

As long as the training deadline allows, FEDCORE prefers to retain the initial full-set epoch, since it offers a more comprehensive representation of the training status by utilizing the entire dataset and establishing a more accurate, well-informed step in beginning of each round of model training.

4.5 CONVERGENCE ANALYSIS

The convergence of FEDCORE is established for strongly convex functions $\mathcal{L}$ under mild assumptions. It is important to note that most existing works on the convergence analysis of federated learning (e.g., [86, 130, 173]) assume that local gradient estimations at the client level are unbiased since the data is directly sampled from the full-set. However, in FEDCORE, gradients computed from coresets are biased approximations to full-set gradients. As a result, the main technical contribution of our convergence analysis is

[11] Existing solutions like FEDPROX also fail in extreme cases.

to meticulously incorporate the coreset gradient approximation error with the federated optimization error.

Theorem 4.1. Assume that for any $i \in U$, the loss function $\mathcal{L}^i$ is L-smooth and μ-strongly convex, and the coreset $(S^{i,*}, \delta^{i,*})$ constructed in FEDCORE is an ϵ-approximation to the full-set, i.e.

$$\forall w \in \mathcal{W}, \quad \frac{1}{m^i}\left\|\sum_{j\in V^i} \nabla \mathcal{L}_j^i(w) - \sum_{k\in S^{i,*}} \delta_k^{i,*} \nabla \mathcal{L}_k^i(w)\right\| \leq \epsilon, \tag{4.9}$$

Consider FEDCORE with R rounds with each round containing E epochs. Set the learning rate $\eta_t = \Omega(1/t)$ for $t \in \{1, 2, \cdots, ER\}$. The model w_{out} output by FEDCORE after R rounds satisfies

$$\mathbb{E}\left[\mathcal{L}(w_{\text{out}}) - \mathcal{L}(w_*)\right] \leq \mathcal{O}(\epsilon) + \mathcal{O}(1/R), \tag{4.10}$$

where $w_* = \operatorname{argmin}_{w\in\mathcal{W}} \mathcal{L}(w)$ is the global optimum of $\mathcal{L}$ in Eq.(4.1), and the expectation is taken over the randomness in client selection, coreset construction and model initialization.

The comprehensive collections of the technical assumptions and the detailed statement of Theorem 4.1 can be found in Section 4.7.2 and Section 4.7.3. The bound in Theorem 4.1 indicates that FEDCORE converges to the global optimum at the rate $\mathcal{O}(1/R)$, with an additional cost of $\mathcal{O}(\epsilon)$ attributed to the coreset gradient approximation error. It is worth noting that the rate $\mathcal{O}(1/R)$ aligns with the existing convergence results for federated learning [86, 130, 173]. The trade-off between full-set FL and coreset FL is explicitly characterized in Theorem 4.1. While learning on the full-set may circumvent the gradient approximation error, the straggler problem in full-set FL can lead to a small number of training rounds R under a limited time budget. On the other hand, FEDCORE reduces the impact of the straggler problem and allows for more training rounds to achieve a smaller optimization error $\mathcal{O}(1/R)$, while keeping the gradient approximation error low (only $\mathcal{O}(\epsilon)$), enabling both efficient and accurate optimization. The proof of Theorem 4.1 is deferred to Section 4.7.3.

4.6 EVALUATIONS

4.6.1 Experimental Setups

FL Datasets and Benchmarks We assess FEDCORE using three widely recognized federated learning benchmarks from computer vision, natural language processing, and

Dataset	**Clients**	**Samples**	**Samples / Client**	
			mean	std
MNIST	1,000	69,035	69	106
Shakespeare	143	517,106	3,616	6,808
Synthetic	30	20,101	670	1,148

Table 4.1: Statistics of the benchmarks

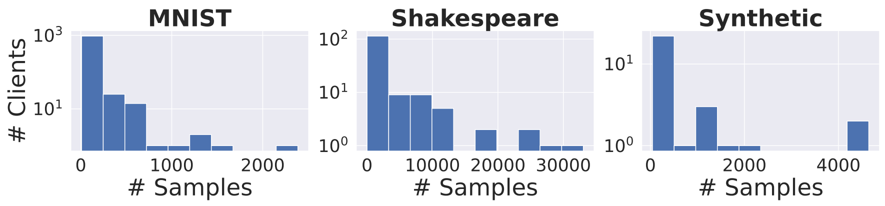


Figure 4.2: Distribution of training samples per client

feature-based classification domains. These tasks encompass major machine learning model types, including CNN, RNN, and Logistic Regression (LR), as detailed below:

1) **MNIST Dataset** [174]: This dataset features a digit classification task using a three-layer CNN for training. To create statistical heterogeneity, the data is allocated among 1,000 clients, where each client has samples of just two distinct digits. The quantity of samples per client adheres to a power-law distribution, highlighting the diversity among clients.

2) **Shakespeare Dataset** [85]: This dataset represents a next-character prediction task trained on *the Complete Works of William Shakespeare* using an LSTM model. Each of the 143 speaking roles in the plays is associated with a distinct client. And

3) **Synthetic Dataset** [86]: This dataset involves a feature-based classification task with 30 clients training an LR model. Each client's training data is generated from a random function $G(\alpha, \beta)$, where α and β control the cross-client and within-client data heterogeneity. Following the approach in [86], we evaluate our method with three different parameter settings: (α, β) equals to $(0, 0)$, $(0.5, 0.5)$ and $(1, 1)$, respectively.

In our evaluation, we train MNIST, Shakespare and Synthetic benchmarks for 100, 30 and 100 rounds, respectively. For all three tasks, each round comprises 10 local epochs. Detailed statistics for these three datasets can be found in Table 4.1 and Figure 4.2.

Experimental Harware and Hyper-Parameters In our evaluations, we utilize a physical server equipped with an Intel Core X Series Core i9 10920X CPU [175] and a NVIDIA

Hyper-parameters	MNIST	Shakespeare	Synthetic
Optimizer	SGD	SGD	SGD
Learning Rate	0.03	0.03	0.001
Batch Size	8	8	8
Local Epoch	10	10	10
Communication Round	100	30	100
Number of Clients	1000	143	30
Number of Clients per Round	100	10	10
μ in FEDPROX	0.1	0.001	0.1

Table 4.2: FEDCORE Evaluation Hyper-parameters

GeForce RTX 2080 Ti GPU [176]. The server runs on the Linux Ubuntu 20.04 operating system. The hyper-parameters used in our evaluations are detailed in Table 4.2.

Comparision Baselines We compare FEDCORE with the following three baselines.

a) FEDAVG [85] updates the global model by averaging local model updates from participating clients. However, it does not consider training deadlines, and thus, is prone to the stragglers issue.

b) FEDAVG-DS [85] is a variant of FEDAVG enforces training deadlines for each round by excluding stragglers. This strategy, however, may negatively impact its overall training performance.

c) FEDPROX [86] is designed to handle partial results from stragglers that might complete fewer local epochs than anticipated, FEDPROX incorporates a quadratic proximal term that explicitly limits the magnitude of local model updates to accommodate stragglers.

Implementations We develop FEDCORE along with all the baseline algorithms using PyTorch [177], extending the simulation framework proposed in FedML[178]. For each client u^i, we sample its computational capability from a normal distribution, i.e., $c^i \sim \mathcal{N}(1,\, 0.25)$. As discussed in Section 4.3, the per-round training time for a client is proportional to $\frac{m^i}{c^i}$. To emulate the stragglers problem, we designate the slowest $s\%$ of clients as stragglers by setting a per-round training deadline that these clients cannot complete all their training tasks within the allotted time. When the training deadline is reached, FEDAVG-DS simply excludes all stragglers and aggregates a global model using the non-stragglers' gradients. In contrast, FEDPROX and FEDCORE employ different strategies such as reducing local training epochs or training with coresets. In our evaluation, we consider two different stragglers' settings by choosing s to be 10 and 30, respectively.

4.6.2 Evaluation Results

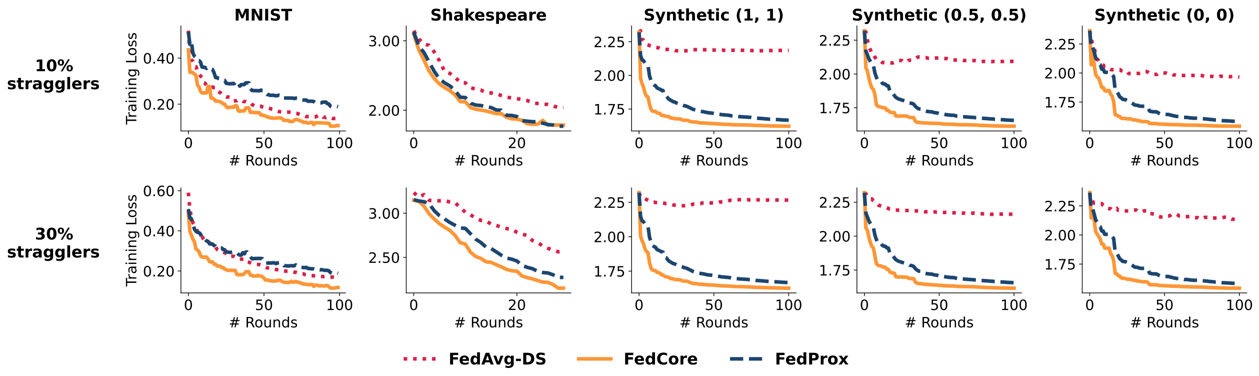


Figure 4.3: The training loss curves for FEDAVG-DS, FEDCORE, and FEDPROX at 10% and 30% stragglers.

		MNIST		Shakespeare		Synthetic (1, 1)		Synthetic (0.5, 0.5)		Synthetic (0, 0)	
		10%	30%	10%	30%	10%	30%	10%	30%	10%	30%
Test Accuracy	FedAvg	94.7		44.9		71.8		73.7		88.2	
	FedAvg-DS	94.1	93.1	39.0	25.2	23.0	19.9	32.2	23.6	36.3	34.6
	FedProx	92.6	92.7	44.1	31.3	**72.3**	72.2	74.1	74.1	87.2	87.2
	FedCore	**94.6**	**94.5**	**44.7**	**34.8**	72.2	**72.8**	**75.2**	**75.1**	**88.5**	**88.3**
Mean Training Time per Round (normalized)	FedAvg	**3.27**	**8.48**	**1.38**	**4.09**	**1.37**	**4.80**	**1.37**	**4.80**	**1.37**	**4.80**
	FedAvg-DS	0.94	0.95	0.60	0.67	0.69	0.79	0.69	0.79	0.69	0.79
	FedProx	0.98	0.99	0.85	0.94	0.86	0.95	0.86	0.95	0.86	0.95
	FedCore	0.99	0.99	0.90	0.99	0.93	0.99	0.93	0.99	0.93	0.99

Table 4.3: Comparison of test accuracy and training time for FEDAVG, FEDAVG-DS, FEDPROX, and FEDCORE at 10% and 30% stragglers. **Bold**: top accuracy; **Red**: exceeded deadline. Normalized time of 1 is round deadline.

Model Performance We present the training loss curves in Figure 4.3 and model accuracy, along with normalized training time, in Table 4.3. For model training loss, FEDCORE consistently achieves the fastest convergence speed and yields the lowest model loss. In contrast, FEDAVG-DS struggles to converge well under synthetic benchmarks due to its approach of dropping stragglers, which contain unique training samples essential for learning. FEDPROX presents competitive performance, but with slower convergence and higher loss compared to FEDCORE. Concerning test accuracy, FEDCORE consistently achieves the highest or near-highest values across all datasets and stragglers' settings, highlighting its superior performance in maintaining or improving model accuracy even

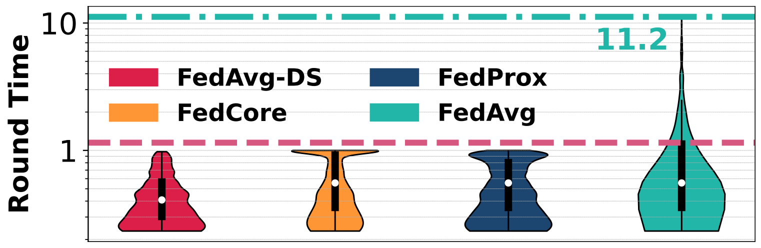


Figure 4.4: Round length distribution on MNIST benchmark, 30% stragglers. The y-axis is presented in log-scale for better illustration.

with stragglers. FEDPROX also demonstrates competitive performance, owing to its ability to accommodate partial results from stragglers, which contain a significant amount of unique training samples that improve model accuracy. However, FEDAVG-DS often results in lower accuracy, particularly in the 30% stragglers setting, as its approach of dropping straggler clients negatively impacts training performance. In terms of training time, FEDCORE, FEDPROX, and FEDAVG-DS are deadline-aware, ensuring they do not exceed the round deadlines. While FEDCORE does not always achieve the fastest training time, it strikes a balance between efficiency and maintaining high accuracy. Conversely, FEDAVG exhibits the longest training times, indicated in red, showcasing its vulnerability to stragglers and lack of deadline-awareness.

Stragglers Handling Figure 4.4 presents the distribution of clients' round times for the MNIST benchmark with 30% stragglers. As the figure illustrates, FEDAVG, which is oblivious to round deadlines, generates a tail distribution that can exceed 11 times the allotted training time for a round. In contrast, deadline-aware algorithms like FEDCORE, FEDAVG-DS, and FEDPROX consistently ensure that each training round is completed before the deadline. Interestingly, the FEDCORE distribution is more tightly clustered around the round deadline in comparison to FEDAVG-DS and FEDPROX, which signifies a more effective utilization of the allotted training time to accurately follow the gradient direction. Table 4.3 shows that although FEDCORE requires slightly longer time than the other two deadline-aware algorithms, it successfully meets the deadline requirements and ultimately achieves the best model performance.

As depicted in Figure 4.5, FEDCORE takes advantage of coresets to perform more epochs of local optimization and deeper gradient exploration, as opposed to FEDPROX's fewer epochs of full-set training. This approach leads to a faster convergence rate and improved model accuracy, demonstrating the effectiveness of the FEDCORE algorithm in addressing the straggler problem in federated learning.

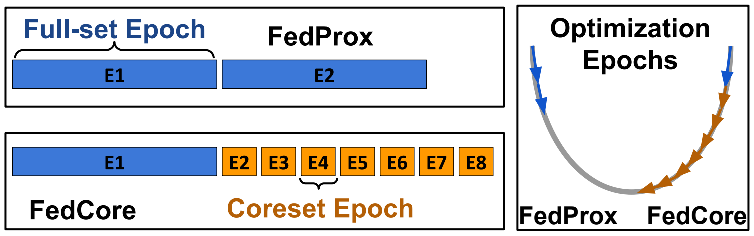


Figure 4.5: Faster FEDCORE convergence vs. FEDPROX, due to more coreset-based gradient steps compared to FEDPROX's fewer epochs of full-set training.

4.7 MATHEMATICAL PROOF FOR FEDCORE CONVERGENCE

4.7.1 Problem Settings and Notations

Problem Set-up First recall the notations defined in section 4.3. The federated learning problem is to solve

$$w_* = \operatorname*{argmin}_{w \in \mathcal{W}} \mathcal{L}(w), \quad \text{where } \mathcal{L}(w) := \sum_{i \in U} p^i \mathcal{L}^i(w), \quad \mathcal{L}^i(w) := \frac{1}{m^i} \sum_{j \in V^i} \mathcal{L}^i_j(w). \tag{4.11}$$

with $\mathcal{L}^i_j(w) := L(f(w, x^i_j), y^i_j)$ representing the empirical loss for each sample (x^i_j, y^i_j) from the i-th client, under the model $f(w,\cdot)$. Here $|U| = n$ is the total number of clients, and p^i is the weight of the i-th client, proportional to the size m^i of its training set with $\sum_{i=1}^n p^i = 1$.

The proposed federated learning algorithm FEDCORE consists of R rounds, each of which contains E epochs. We use the time index $t \in \{0, 1, 2, \cdots, ER\}$ to denote the time step of each epoch, where $t = 0$ corresponds to the model initialization. Meanwhile, denote w^i_t to be the model parameter of client i at time step t. One typical round in FEDCORE is described as follows.

Let $t = (r-1)E$ be the beginning at the r-th round for some $r = 1, 2, \cdots, R$. The central server broadcasts the latest model, w_t, to all the devices:

$$w^i_t \longleftarrow w_t, \quad \forall i \in U. \tag{4.12}$$

After that, the central server selects a set U_t of K clients randomly from U, according to the sampling probabilities $p^i, i \in U$. The coreset is then constructed for each client $(S^{i,*}, \delta^{i,*}), i \in U_t$. Each client $i \in U_t$ performs local updates on its model w^i_t for the remaining epochs in the current round, based on the data in its coreset $(S^{i,*}, \delta^{i,*})$:

$$w^i_{t+k+1} \longleftarrow w^i_{t+k} - \eta_{t+k} g^i_{t+k}, \text{ for } k = 0, 1, \cdots, E-1, \tag{4.13}$$

where η_{t+k} is the learning rate and g^i_{t+k} is the gradient computed from $(S^{i,*}, \delta^{i,*})$:

$$g^i_{t+k} = \frac{1}{m^i} \sum_{j \in S^{i,*}} \delta^{i,*}_j \nabla \mathcal{L}^i_j(w^i_{t+k}). \tag{4.14}$$

Finally, at the end of the r-th round, the server aggregates the local models $\{w^i_{t+E}\}_{i \in U_t}$ to produce the new global model w_{t+E}:

$$w_{t+E} \longleftarrow \frac{1}{K} \sum_{i \in U_t} w^i_{t+E}. \tag{4.15}$$

Note that the current update in Eq.(4.13) is written in the form of gradient descent (GD), meaning that the model will be updated once based on the full gradient computed from $(S^{i,*}, \delta^{i,*})$. In practice, however, within one epoch, the update in Eq.(4.13) is usually conducted sequentially using stochastic gradient descent (SGD): the entire coreset will be randomly split into several mini-batches, and the parameter will be updated on each mini-batch. In the following analysis, we focus on the gradient descent setting in Eq.(4.13) for the ease of presentation. Convergence guarantees for SGD updates can be established by using the similar arguments as in the proofs of our main results.

Notations In subsequent analysis, we use

$$G^i_t := \nabla \mathcal{L}^i(w^i_t) = \frac{1}{m^i} \sum_{j \in V^i} \nabla \mathcal{L}^i_j(w^i_t) \tag{4.16}$$

to denote the full gradient from the full-set V^i of client i at time t. And denote

$$G_t = \sum_{i \in U} p^i G^i_t = \sum_{i \in U} p^i \nabla \mathcal{L}^i(w^i_t) \tag{4.17}$$

as the full gradient of the population at time t. Meanwhile, denote

$$g^i_t = \frac{1}{m^i} \sum_{j \in S^{i,*}} \delta^{i,*}_j \nabla \mathcal{L}^i_j(w^i_t), \tag{4.18}$$

where $(\delta^{i,*}, S^{i,*})$ is defined in Eq.(4.6), as the gradient computed from the coreset of client i at time t. And denote $g_t = \sum_{i \in U} p^i g^i_t$ as the population coreset gradient at time t.

In practice, within one round, only a subset of K randomly selected clients will update their parameters, and the choices of clients vary each round. In order to facilitate the analysis under the random selection scheme, the following thought trick is introduced to circumvent the difficulty: we assume that FEDCORE always activates **all** devices at the beginning of each round, while only aggregates parameters from those sampled devices an the end of one round. It is clear that this updating scheme is equivalent to the original. More specifically, the updating scheme in FEDCORE is given by, $\forall i \in U$,

$$\begin{aligned} v_{t+1}^i &= w_t^i - \eta_t g_t^i, \\ w_{t+1}^i &= \begin{cases} v_{t+1}^i & \text{if } t+1 \notin \mathcal{I}_E, \\ \frac{1}{K}\sum_{k\in U_t} v_{t+1}^k & \text{if } t+1 \in \mathcal{I}_E, \end{cases} \end{aligned} \tag{4.19}$$

where $\mathcal{I}_E = \{rE \mid r = 1, 2, \cdots, R\}$ is the set of global synchronization steps, and U_t is the set of K selected clients at time t. An additional variable v_{t+1}^i is introduced to represent the immediate result of one step GD update from w_t^i, and w_t^i is the final model parameters maintained by client i at time t, (possibly after the global synchronization).

In addition, two virtual sequences are introduced in the subsequent analysis to denote the population-averaged model parameters, following the ideas from [130, 173, 179]:

$$\overline{v}_t = \sum_{i\in U} p^i v_t^i, \text{ and } \overline{w}_t = \sum_{i\in U} p^i w_t^i, \tag{4.20}$$

where $\overline{v}_{t+1}$ results from an single GD step of from $\overline{w}_t$:

$$\overline{v}_{t+1} = \overline{w}_t - \eta_t g_t. \tag{4.21}$$

4.7.2 Assumptions and Convergence Results

The following are the detailed assumptions required for the convergence analysis.

Assumption 4.1 (L-smoothness). $\forall i \in U, \mathcal{L}^i$ is L-smooth: for all $v, w \in \mathcal{W}$,

$$\mathcal{L}^i(v) \le \mathcal{L}^i(w) + (v-w)^\top \nabla \mathcal{L}^i(w) + \frac{L}{2}\|v-w\|_2^2.$$

Assumption 4.2 (μ-strong convexity). $\forall i \in U, L^i$ is μ-strongly convex: for all $v, w \in \mathcal{W}$,

$$\mathcal{L}^i(v) \ge \mathcal{L}^i(w) + (v-w)^\top \nabla \mathcal{L}^i(w) + \frac{\mu}{2}\|v-w\|_2^2.$$

Assumption 4.3 (ϵ-coreset). For any client i and time step t, with probability one, the coreset gradient g_t^i in Eq.(4.18) is an ϵ-approximation to the full-set gradient G_t^i in Eq.(4.16):

$$\left\| g_t^i - G_t^i \right\| \leq \epsilon,\ \forall i \in U, \text{ and } t \in \{0, 1, \cdots, ER\}, \quad \text{with probability one.} \tag{4.22}$$

Assumption 4.4 (D-bounded gradient)**.** For any client i and time step t, with probability one, 2-norms of the coreset gradient g_t^i in Eq.(4.18) and the full-set gradient G_t^i in Eq.(4.16) are uniformly upper bounded by a constant $D > 0$:

$$\max\left\{ \left\| g_t^i \right\|, \left\| G_t^i \right\| \right\} \leq D, \quad \forall i \in U, \text{ and } t \in \{0, 1, \cdots, ER\}, \text{ with probability one.} \tag{4.23}$$

Assumption 4.5 (Γ-heterogeneity)**.** Let $\mathcal{L}_*$ and $\mathcal{L}_*^i$ be the minimum values of $\mathcal{L}$ and $\mathcal{L}^i$, respectively. Assume there is a positive constant $\Gamma > 0$ such that $\Gamma \geq \mathcal{L}_* - \sum_{i \in U} p^i \mathcal{L}_*^i$.

Assumption 4.6 (Random sampling)**.** For any time step t, assume U_t contains a subset of K indices randomly selected with replacement according to the sampling probabilities $\{p^i\}_{i \in U}$.

Comments on Assumptions Assumption 4.1 and 4.2 are standard assumptions in convex optimization [180]; typical examples are linear/ridge regression, logistic regression, and regularized support vector machines. Assumption 4.3 characterizes the approximation capability of the coreset to the full-set, which is standard in the theoretical works on coreset-based gradient descent methods [38, 166]. Assumption 4.4 on the bounded gradient is a widely adopted setting in the existing theoretical works for federated learning and coreset methods [38, 86, 130]. Meanwhile, note that Assumptions 4.3 and 4.4 are presented in a probabilistic form to account for the potential randomness resulting from the coreset construction steps in FEDCORE. Assumption 4.5 quantifies the degree of heterogeneity among different clients. In the special case when data from all the clients are i.i.d., then $\mathcal{L}_* - \sum_{i \in U} p^i \mathcal{L}_*^i \to 0$ as the number of samples grows. Assumption 4.6 assumes the K clients are selected from the distribution $\{p^i\}_{i \in U}$ independently and with replacement, which is a common set-up in both theoretical and empirical works [86, 130].

Randomness in FEDCORE Note that randomness in FEDCORE can be attributed to three sources: client selection, coreset construction and model initialization w_0. Throughout the subsequent analysis and statements, unless otherwise specified, the expectation $\mathbb{E}[\cdot]$ is be taken over all three sources of randomness. Meanwhile, the notation $\mathbb{E}_{U_t}[\cdot]$ is also introduced to denote the expectation over the random client selection at time t, conditioned on the other sources of randomness.

4.7.3 Proofs of Main Results

The convergence of FEDCORE is established by the following theorem, which can be considered as a more detailed version of Theorem 4.1.

Theorem 4.2. Assume Assumptions 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 hold with constants L, μ, ϵ, D and Γ. Consider FEDCORE with R rounds and each round contains E epochs. For $t \in \{0, 1, \cdots, ER\}$, set the learning rate

$$\eta_t = \frac{\alpha}{t+\beta}, \quad \text{with } \alpha = \frac{2}{\mu} \text{ and } \beta = \max\left\{E, \frac{8L}{\mu}\right\}. \tag{4.24}$$

The model w_{out} output by FEDCORE after R rounds of training satisfies

$$\mathbb{E}\left[\|w_{\text{out}} - w_*\|^2\right] \le A_1 + \frac{A_2}{ER+\beta}, \tag{4.25}$$

where the constants A_1 and A_2 are given by:

$$\begin{aligned} A_1 &= \frac{2\epsilon D}{\mu^2}, \\ A_2 &= \max\left\{\beta\mathbb{E}\left[\|w_0 - w_*\|^2\right], \frac{4}{\mu^2}\left[\frac{4E^2D^2}{K} + 8(E-1)^2D^2 + 6L\Gamma + \epsilon^2 + 2\epsilon D\right]\right\}. \end{aligned} \tag{4.26}$$

Here $w_* = \operatorname{argmin}_{w \in \mathcal{W}} \mathcal{L}(w)$ as defined in Eq.(4.11) and the expectation is taken over the randomness in client selection, coreset construction and model initialization w_0. Consequently,

$$\mathbb{E}\left[\mathcal{L}(w_{\text{out}}) - \mathcal{L}(w_*)\right] \le \frac{L}{2}\left(A_1 + \frac{A_2}{ER+\beta}\right). \tag{4.27}$$

The proof of Theorem 4.2 is based on the following three key lemmas, whose proofs are deferred to Section 4.7.3.

Lemma 4.1. Under the setting of Theorem 4.2, for $t+1 \in \mathcal{I}_E = \{rE \mid r = 1, 2, \cdots, R\}$, the set of global synchronization steps,

$$\mathbb{E}_{U_t}\left[\overline{w}_{t+1}\right] = \overline{v}_{t+1}. \tag{4.28}$$

Lemma 4.2. Under the setting of Theorem 4.2, for $t+1 \in \mathcal{I}_E = \{rE \mid r = 1, 2, \cdots, R\}$, the set of global synchronization steps, the expected difference between $\overline{v}_{t+1}$ and $\overline{w}_{t+1}$ is bounded by

$$\mathbb{E}_{U_t}\left[\|\overline{v}_{t+1} - \overline{w}_{t+1}\|^2\right] \le \frac{4}{K}\eta_t^2 E^2 D^2. \tag{4.29}$$

Lemma 4.3. Under the setting of Theorem 4.2, for any time step $t+1 \in \{1, 2, \cdots, ER\}$,

$$\mathbb{E}\left[\|\overline{v}_{t+1} - w_*\|^2\right] \leq (1 - \eta_t \mu)\, \mathbb{E}\left[\|\overline{w}_t - w_*\|^2\right] + \eta_t \cdot A_3 + \eta_t^2 \cdot A_4, \tag{4.30}$$

where

$$A_3 = \frac{2\epsilon D}{\mu}, \quad A_4 = 8(E-1)^2 D^2 + 6L\Gamma + \epsilon^2 + 2\epsilon D. \tag{4.31}$$

Proof of Theorem 4.2. First note the following decomposition:

$$\begin{aligned}\|\overline{w}_{t+1} - w_*\|^2 &= \|\overline{w}_{t+1} - \overline{v}_{t+1} + \overline{v}_{t+1} - w_*\|^2 \\ &= \underbrace{\|\overline{w}_{t+1} - \overline{v}_{t+1}\|^2}_{H_1} + \underbrace{2\left\langle \overline{w}_{t+1} - \overline{v}_{t+1}, \overline{v}_{t+1} - w_* \right\rangle}_{H_2} + \underbrace{\|\overline{v}_{t+1} - w_*\|^2}_{H_3}.\end{aligned} \tag{4.32}$$

For the first term H_1 in Eq.(4.32), note that when $t+1 \notin \mathcal{I}_E$, we have $\overline{v}_{t+1} = \overline{w}_{t+1}$, and H_1 vanishes. Additionally, if $t+1 \in \mathcal{I}_E$, then the expectation of H_1 is bounded by Lemma 4.2:

For the second term H_2 in Eq.(4.32), when $t+1 \notin \mathcal{I}_E$, H_2 vanishes since $\overline{v}_{t+1} = \overline{w}_{t+1}$. Additionally, when $t+1 \in \mathcal{I}_E$, H_2 vanishes under the expectation $\mathbb{E}_{U_t}[\cdot]$, due to the unbiasedness of $\overline{w}_{t+1}$ stated in Lemma 4.1.

For the third term H_3 in Eq.(4.32), its expectation is bounded by Lemma 4.3 for any time step $t+1 \in \{1, 2, \cdots, ER\}$.

Overall, combining the bounds on H_1, H_2 and H_3 together, we have for any $t+1 \in \{1, 2, \cdots, ER\}$,

$$\mathbb{E}\left[\|\overline{w}_{t+1} - w_*\|^2\right] \leq (1 - \eta_t \mu)\, \mathbb{E}\left[\|\overline{w}_t - w_*\|^2\right] + \eta_t \cdot A_3 + \eta_t^2 \cdot \left(\frac{4E^2D^2}{K} + A_4\right), \tag{4.33}$$

where A_3, A_4 are defined in Eq.(4.31). For simplicity, denote

$$A_5 ;= \frac{4E^2D^2}{K} + A_4. \tag{4.34}$$

Now we will prove by induction that under the diminishing step size $\eta_t = \frac{\alpha}{t+\beta}$ with $\alpha = \frac{2}{\mu}$ and $\beta = \max\left\{E, \frac{8L}{\mu}\right\}$, for any time step $t \in \{0, 1, \cdots, ER\}$,

$$\mathbb{E}\left[\|\overline{w}_t - w_*\|^2\right] \leq A_1 + \frac{A_2}{t+\beta}, \tag{4.35}$$

where A_1, A_2 are defined in Eq.(4.26).

First, note that the definition of A_2 in Eq.(4.26) ensures that Eq.(4.35) holds for $t = 0$.

Assume Eq.(4.35) holds for some time step t. Then for time step $t+1$, by Eq.(4.33), we have

$$\begin{aligned}\mathbb{E}\left[\|\overline{w}_{t+1}-w_*\|^2\right] &\le \left(1-\frac{\alpha\mu}{t+\beta}\right)\cdot\left(A_1+\frac{A_2}{t+\beta}\right)+\frac{\alpha}{t+\beta}\cdot A_3+\left(\frac{\alpha}{t+\beta}\right)^2\cdot A_5,\\ &= A_1+\left(1-\frac{\alpha\mu}{t+\beta}\right)\cdot\frac{A_2}{t+\beta}+\left(\frac{\alpha}{t+\beta}\right)^2\cdot A_5+\frac{\alpha(A_3-\mu A_1)}{t+\beta} \qquad (4.36)\end{aligned}$$

Note that by the definitions of A_1 in Eq.(4.26) and A_3 in Eq.(4.31),

$$A_3=\mu A_1. \qquad (4.37)$$

Meanwhile,

$$\begin{aligned}\left(1-\frac{\alpha\mu}{t+\beta}\right)\cdot\frac{A_2}{t+\beta}+\left(\frac{\alpha}{t+\beta}\right)^2\cdot A_5 &= \frac{(t+\beta-1)A_2}{(t+\beta)^2}+\left[\frac{\alpha^2A_5}{(t+\beta)^2}-\frac{(\alpha\mu-1)A_2}{(t+\beta)^2}\right]\\ &\le \frac{A_2}{t+\beta+1}+\left[\frac{\alpha^2A_5}{(t+\beta)^2}-\frac{(\alpha\mu-1)A_2}{(t+\beta)^2}\right]\\ &= \frac{A_2}{t+\beta+1}+\frac{1}{(t+\beta)^2}\left[\frac{4A_5}{\mu^2}-A_2\right]\\ &\le \frac{A_2}{t+\beta+1}. \qquad (4.38)\end{aligned}$$

Here the second equality in Eq.(4.38) is due to the fact that $\alpha=\frac{2}{\mu}$, and the second inequality in Eq.(4.38) comes from the fact that $A_2\ge\frac{4A_5}{\mu^2}$, which is a direct consequence of the definitions of A_2 in Eq.(4.26) and A_5 in Eq.(4.34).

Plugging Eq.(4.37) and Eq.(4.38) into Eq.(4.36) completes the proof of the induction hypothesis in Eq.(4.35). Specifically, the model $w_{\text{out}}=\overline{w}_{ER}$ output by FEDCORE after R rounds satisfies Eq.(4.25).

Furthermore, by the L-smoothness of $\mathcal{L}$ (Assumption 4.1),

$$\mathbb{E}\left[\mathcal{L}(w_{\text{out}})-\mathcal{L}(w_*)\right]\le\frac{L}{2}\cdot\mathbb{E}\left[\|w_{\text{out}}-w_*\|^2\right]\le\frac{L}{2}\left(A_1+\frac{A_2}{ER+\beta}\right). \qquad (4.39)$$

QED.

4.7.4 Proofs of Lemmas

Proof of Lemma 4.1. This lemma is a direct consequence of Assumption 4.6. More specifically, for $t+1 \in \mathcal{I}_E = \{rE \mid r = 1, 2, \cdots, R\}$,

$$\mathbb{E}_{U_t}\left[\overline{w}_{t+1}\right] = \mathbb{E}_{U_t}\left[\frac{1}{K}\sum_{k\in U_t} v_{t+1}^k\right] = \frac{1}{K}\cdot K \cdot \mathbb{E}_{k\in U_t}\left[v_{t+1}^k\right] = \sum_{i\in U} p^i v_{t+1}^i = \overline{v}_{t+1},$$

where the second equality comes from the linearity of expectation, and the third equality is due to Assumption 4.6. QED.

Proof of Lemma 4.2. Lemma 4.2 is a direct consequence of Lemma 5 in [130]. The proof is outlined as follows.

For $t+1 \in \mathcal{I}_E = \{rE \mid r = 1, 2, \cdots, R\}$, $\overline{w}_{t+1} = \frac{1}{K}\sum_{k\in U_t} v_{t+1}^k$. Taking expectation over U_t,

$$\begin{aligned}\mathbb{E}_{U_t}\left[\left\|\overline{w}_{t+1} - \overline{v}_{t+1}\right\|^2\right] = \mathbb{E}_{U_t}\left[\frac{1}{K^2}\sum_{k\in U_t}\left\|v_{t+1}^k - \overline{v}_{t+1}\right\|^2\right] &= \frac{1}{K}\mathbb{E}_{k\in U_t}\left[\left\|v_{t+1}^k - \overline{v}_{t+1}\right\|^2\right]\\ &= \frac{1}{K}\sum_{i\in U} p^i \left\|v_{t+1}^i - \overline{v}_{t+1}\right\|^2 \end{aligned} \tag{4.40}$$

where the first equality follows from Assumption 4.6 that $\left\{v_{t+1}^k\right\}_{k\in U_t}$ are independent and unbiased with $\mathbb{E}_{k\in U_t}\left[v_{t+1}^k\right] = \overline{v}_{t+1}$.

To bound Eq.(4.40), first note that since $t+1 \in \mathcal{I}_E$, $t_0 := t+1-E \in \mathcal{I}_E$ is also a synchronization time, which implies $\left\{w_{t_0}^i\right\}_{i\in U}$ is identical. Then,

$$\begin{aligned}\sum_{i\in U} p^i \left\|v_{t+1}^i - \overline{v}_{t+1}\right\|^2 &= \sum_{i\in U} p^i \left\|\left(v_{t+1}^i - \overline{w}_{t_0}\right) - \left(\overline{v}_{t+1} - \overline{w}_{t_0}\right)\right\|^2\\ &= \left(\sum_{i\in U} p^i \left\|v_{t+1}^i - \overline{w}_{t_0}\right\|^2\right) - \left\|\overline{v}_{t+1} - \overline{w}_{t_0}\right\|^2 \le \sum_{i\in U} p^i \left\|v_{t+1}^i - \overline{w}_{t_0}\right\|^2,\end{aligned} \tag{4.41}$$

where the second equality results from $\sum_{i\in U} p^i\left(v_{t+1}^i - \overline{w}_{t_0}\right) = \overline{v}_{t+1} - \overline{w}_{t_0}$. Combining

Eq.(4.40) and Eq.(4.41), we have

$$\begin{aligned}
\mathbb{E}_{U_t}\left[\|\overline{w}_{t+1}-\overline{v}_{t+1}\|^2\right] &\leq \frac{1}{K}\sum_{i\in U}p^i\left\|v_{t+1}^i-\overline{w}_{t_0}\right\|^2=\frac{1}{K}\sum_{i\in U}p^i\left\|v_{t+1}^i-w_{t_0}^i\right\|^2\\
&=\frac{1}{K}\sum_{i\in U}p^i\left\|\sum_{\tau=t_0}^{t}\eta_\tau g_\tau^i\right\|^2\leq\frac{1}{K}\sum_{i\in U}p^iE\sum_{\tau=t_0}^{t}\left\|\eta_\tau g_\tau^i\right\|^2\\
&\leq\frac{1}{K}E\sum_{\tau=t_0}^{t}\eta_\tau^2D^2\leq\frac{1}{K}E^2\eta_{t_0}^2D^2\leq\frac{4}{K}\eta_t^2E^2D^2. \qquad (4.42)
\end{aligned}$$

Here, the second inequality in Eq. (4.42) follows from the Cauchy-Schwarz inequality. The third inequality is a result of Assumption 4.4. The fourth inequality is justified by the fact that $\eta_t=\frac{\alpha}{t+\beta}$ is non-increasing. Lastly, the last inequality holds since, by definition, $\beta=\max\left\{E,\frac{8L}{\mu}\right\}\geq E$, and therefore $\eta_{t_0}\leq 2\eta_{t_0+E-1}$. QED.

Proof of Lemma 4.3. First, by Eq.(4.21), we have

$$\|\overline{v}_{t+1}-w_*\|^2=\|\overline{w}_t-w_*-\eta_tg_t\|^2=\|\overline{w}_t-w_*\|^2+\underbrace{\eta_t^2\|g_t\|^2}_{F_1}\underbrace{-2\eta_t\langle\overline{w}_t-w_*,g_t\rangle}_{F_2} \qquad (4.43)$$

To bound F_1 in Eq.(4.43), note that

$$\begin{aligned}
F_1&=\eta_t^2\|g_t\|^2=\eta_t^2\left\|\sum_{i\in U}p^ig_t^i\right\|^2\leq\eta_t^2\sum_{i\in U}p^i\left\|g_t^i\right\|^2=\eta_t^2\sum_{i\in U}p^i\left\|G_t^i+g_t^i-G_t^i\right\|^2\\
&=\eta_t^2\left(\sum_{i\in U}p^i\left\|G_t^i-g_t^i\right\|^2+2\sum_{i\in U}p^i\left\langle G_t^i,g_t^i-G_t^i\right\rangle+\sum_{i\in U}p^i\left\|G_t^i\right\|^2\right)\\
&=\eta_t^2\left(\sum_{i\in U}p^i\left\|G_t^i-g_t^i\right\|^2+2\sum_{i\in U}p^i\left\|G_t^i\right\|\left\|g_t^i-G_t^i\right\|+\sum_{i\in U}p^i\left\|G_t^i\right\|^2\right)\\
&\leq\eta_t^2\left(\epsilon^2+2\epsilon D+\sum_{i\in U}p^i\left\|G_t^i\right\|^2\right)\\
&\leq\eta_t^2\left(\epsilon^2+2\epsilon D+2L\sum_{i\in U}p^i\left(\mathcal{L}^i(w_t^i)-\mathcal{L}_*^i\right)\right). \qquad (4.44)
\end{aligned}$$

Here the first inequality in Eq.(4.44) is due to the convexity of $\|\cdot\|^2$. The second inequality comes from Assumption 4.4 and Assumption 4.3. The last inequality follows from the fact that for L-smooth $\mathcal{L}^i$ (Assumption 4.1),

$$\left\|G_t^i\right\|^2 \leq 2L\left(\mathcal{L}^i(w_t^i) - \mathcal{L}_*^i\right). \tag{4.45}$$

To bound F_2 in Eq.(4.43), note that

$$\begin{aligned} F_2 &= -2\eta_t \left\langle \overline{w}_t - w^\star, g_t \right\rangle = -2\eta_t \sum_{i\in U} p^i \left\langle \overline{w}_t - w_*, g_t^i \right\rangle \\ &= -2\eta_t \sum_{i\in U} p^i \left\langle \overline{w}_t - w_t^i, G_t^i \right\rangle - 2\eta_t \sum_{i\in U} p^i \left\langle w_t^i - w_*, G_t^i \right\rangle + 2\eta_t \sum_{i\in U} p^i \left\langle \overline{w}_t - w_*, G_t^i - g_t^i \right\rangle \\ &\leq -2\eta_t \sum_{i\in U} p^i \left\langle \overline{w}_t - w_t^i, G_t^i \right\rangle - 2\eta_t \sum_{i\in U} p^i \left\langle w_t^i - w_*, G_t^i \right\rangle + 2\eta_t \epsilon \cdot \left\|\overline{w}_t - w_*\right\| \end{aligned} \tag{4.46}$$

Here the last inequality in Eq.(4.46) is due to the Cauchy-Schwarz inequality and Assumption 4.3. Moreover, by the Cauchy-Schwarz inequality and the AM-GM inequality,

$$-2\left\langle \overline{w}_t - w_t^i, G_t^i \right\rangle \leq \frac{1}{\eta_t}\left\|\overline{w}_t - w_t^i\right\|^2 + \eta_t \left\|G_t^i\right\|^2 \leq \frac{1}{\eta_t}\left\|\overline{w}_t - w_t^i\right\|^2 + 2L\eta_t\left(\mathcal{L}^i(w_t^i) - \mathcal{L}_*^i\right), \tag{4.47}$$

where the last inequality in Eq.(4.47) follows from Eq.(4.45). In addition, by the μ-strong convexity of $\mathcal{L}^i$, Assumption 4.2,

$$-2\left\langle w_t^i - w_*, G_t^i \right\rangle \leq -\left(\mathcal{L}^i\left(w_t^i\right) - \mathcal{L}^i\left(w_*\right)\right) - \frac{\mu}{2}\left\|w_t^i - w_*\right\|^2. \tag{4.48}$$

The μ-strong convexity of $\mathcal{L}$, together with the optimality of w_*, also implies that

$$\left\|\overline{w}_t - w_*\right\| \leq \frac{1}{\mu}\left\|\nabla\mathcal{L}(\overline{w}_t) - \nabla\mathcal{L}(w_*)\right\| = \frac{1}{\mu}\left\|\nabla\mathcal{L}(\overline{w}_t)\right\| \leq \frac{D}{\mu}, \tag{4.49}$$

where the last inequality comes from Assumption 4.4.

Now combining Eq.(4.43) with Eq.(4.44), Eq.(4.46), Eq.(4.47), Eq.(4.48), and Eq.(4.49), it follows that

$$\begin{aligned}\|\overline{v}_{t+1}-w_*\|^2 \leq\ & \underbrace{\|\overline{w}_t-w_*\|^2-\eta_t\mu\sum_{i\in U}p^i\left\|w_t^i-w_*\right\|^2}_{F_3}+\sum_{i\in U}p^i\left\|\overline{w}_t-w_t^i\right\|^2\\ & \underbrace{+\,4L\eta_t^2\sum_{i\in U}p^i\left(\mathcal{L}^i(w_t^i)-\mathcal{L}_*^i\right)-2\eta_t\sum_{i\in U}p^i\left(\mathcal{L}^i(w_t^i)-\mathcal{L}^i(w_*)\right)}_{F_4}\\ & +\eta_t\cdot\frac{2\epsilon D}{\mu}+\eta_t^2\cdot\left(\epsilon^2+2\epsilon D\right) \end{aligned} \tag{4.50}$$

To bound F_3 in Eq.(4.50), it follows by the convexity of $\|\cdot\|^2$ that

$$F_3\leq\|\overline{w}_t-w_*\|^2-\eta_t\mu\left\|\sum_{i\in U}p^i\left(w_t^i-w_*\right)\right\|^2=(1-\eta_t\mu)\|\overline{w}_t-w_*\|^2. \tag{4.51}$$

Meanwhile, it is shown by Lemma 1 of [130] that F_4 in Eq.(4.50) is bounded by

$$F_4\leq\eta_t^2\cdot 6L\Gamma+\sum_{i\in U}p^i\left\|\overline{w}_t-w_t^i\right\|^2. \tag{4.52}$$

By combining Eq.(4.50) with Eq.(4.51) and Eq.(4.52), it follows that

$$\begin{aligned}\|\overline{v}_{t+1}-w_*\|^2\leq\ &(1-\eta_t\mu)\|\overline{w}_t-w_*\|^2+\eta_t\cdot\frac{2\epsilon D}{\mu}+\eta_t^2\cdot\left(6L\Gamma+\epsilon^2+2\epsilon D\right)\\ &+2\underbrace{\sum_{i\in U}p^i\left\|\overline{w}_t-w_t^i\right\|^2}_{F_5}.\end{aligned} \tag{4.53}$$

Finally, to bound F_5 in Eq.(4.53), one can apply the same argument used in bounding Eq.(4.40). More specifically, for any t, there exists a $t_0\leq t$ such that $t-t_0\leq E-1$ and $w_{t_0}^i=\overline{w}_{t_0}$ for all $i\in U$. Then, by following the same arguments as in Eq.(4.41) and Eq.(4.42), it is easy to verify that:

$$F_5\leq\eta_t^2\cdot 4(E-1)^2D^2. \tag{4.54}$$

By plugging Eq.(4.54) into Eq.(4.53), we complete the proof of Lemma 4.3. QED.

4.8 CONCLUDING REMARK

In this chapter, we introduce FEDCORE, an innovative algorithm addressing the straggler problem in federated learning using distributed coresets. FEDCORE effectively adapts to updated models and integrates coreset generation with minimal overhead, significantly outperforming traditional methods.

Our comprehensive analysis and evaluation demonstrate that FEDCORE substantially reduces FL training time while maintaining high accuracy. With regards to broader impacts, this research pioneers the use of coreset methods in efficient federated learning, paving the way for more scalable and robust systems, especially in privacy-sensitive domains where data protection is vital. Through this work, we have demonstrated that **training data redundancy** is a critical path to achieving **multi-objective resource optimization** for scalable, efficient, and reliable federated learning.

CHAPTER 5: CONCLUSION

5.1 SUMMARY OF CONTRIBUTIONS

This thesis presents innovative solutions to enhance resource efficiency in large-scale machine learning systems, through three key contributions:

- CROSSROI presents an innovative system for real-time video analytics that drastically cuts down on communication and computation costs through the exploitation of data redundancy across camera networks. By leveraging the **inference data redundancy** from cross-camera viewing fields, CROSSROI effectively **reduces network bandwidth** and **boosts computational throughput**, achieving up to a 65% decrease in network overhead and a 34% reduction in end-to-end response delay.

- BOFL offers an energy-efficient federated learning solution by fine-tuning hardware operational frequencies on edge devices. Employing a Bayesian optimization-based "explore-then-exploit" strategy for **hardware configurability** in a **timely manner**. This work achieves a significant reduction in energy use during model training by 26%, showing a sustainable federated learning approach on edge devices

- FEDCORE addresses federated learning's straggler effect with distributed coresets that reduce data processing for slower clients, seamlessly enhancing system integration. This approach can slash training times by 8x without compromising the model accuracy, thanks to **training data redundancy** that **boosts training speed** and **preserves model performance**.

Overall, the thesis validates the importance of multi-objective resource optimizations in the design of large scale machine learning systems, showcasing significant advancements in managing network bandwidth, computation, energy consumption, and model accuracy through data redundancy and hardware configurability. These contributions not only push the boundaries of current technology but also lay the groundwork for future research in optimizing machine learning systems for sustainable and efficient real-world applications.

5.2 FUTURE WORK AND OPPORTUNITIES

This section outlines potential directions for extending the research on multi-objective resource optimization in machine learning systems. The comprehensive exploration of this thesis lays the groundwork for several promising areas of future inquiry:

ML Model Quantization. The escalating sizes of ML models, such as the GPT-3 [181] with its 175 billion parameters, challenge federated learning, particularly regarding deployment on user devices constrained by memory and computational demands for local gradient computations. Parameter quantization offers a viable solution by compressing model sizes (e.g., using 8-bit over 64-bit representations to achieve an 8x reduction). This technique aims to optimize **multiple system objectives** like *memory usage* and *communication overhead*. However, its efficacy in diverse federated learning settings, considering the variability in device hardware, remains to be thoroughly assessed.

Efficient ML Training Parallelization. The era of large ML models necessitates the use of multi-GPU DNN training to manage the models' substantial size and overcome single GPU memory constraints. Strategies like pipeline and tensor parallelization [182, 183, 184, 185] have been proposed to enhance **multiple performance objectives** including *training speed* and *energy efficiency*. Nonetheless, these parallelization strategies introduce new challenges, such as low GPU utilization and the "bubble" problem [182, 186], making the optimization of such systems for improved efficiency a complex, yet unresolved issue.

Gradient Compression Over Communication. The increase in ML model sizes has led to escalated gradient-parameter communication between federated learning clients and the cloud, exacerbating the communication bottleneck. Gradient compression (GC) methods [187, 188, 189] provide a solution by diminishing the data volume required for transmission, thereby optimizing **multiple system metrics**, including *network bandwidth* and *model performance*. While existing works, such as *HiPress* [189], concentrate on gradient compression within cloud clusters to reduce communication overhead, the application of GC techniques in federated learning—where clients possess heterogeneous hardware and might compress their local gradients to varying degrees—remains a significant challenge.

Addressing these areas not only builds upon this thesis's contributions but also opens new avenues for research, aiming to further enhance the scalability, efficiency, and effectiveness of large scale machine learning systems.

REFERENCES

[1] "Can 30000 Cameras Help Solve Chicago's Crime Problem?" https://www.nytimes.com/2018/05/26/us/chicago-police-surveillance.html, accessed: 2021-01-27.

[2] "Absolutely everywhere in Beijing is now covered by police video surveillance," https://qz.com/518874/, accessed: 2021-01-27.

[3] "45 Billion Cameras by 2022 Fuel Business Opportunities," https://www.ldv.co/insights/2017, accessed: 2021-01-27.

[4] S. Jain, G. Ananthanarayanan, J. Jiang, Y. Shu, and J. Gonzalez, "Scaling video analytics systems to large camera deployments," in *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2019, pp. 9–14.

[5] S. Jain, X. Zhang, Y. Zhou, G. Ananthanarayanan, J. Jiang, Y. Shu, V. Bahl, and J. Gonzalez, "Spatula: Efficient cross-camera video analytics on large camera networks," in *ACM/IEEE Symposium on Edge Computing (SEC)*, 2020.

[6] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2020, pp. 359–376.

[7] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, "Scaling video analytics on constrained edge nodes," *arXiv preprint arXiv:1905.13536*, 2019.

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

[9] D. Huba, J. Nguyen, K. Malik, R. Zhu, M. Rabbat, A. Yousefpour, C.-J. Wu, H. Zhan, P. Ustinov, H. Srinivas et al., "Papaya: Practical, private, and scalable federated learning," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 814–832, 2022.

[10] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (Sensys)*, 2015, pp. 155–168.

[11] S. P. Chinchali, E. Cidon, E. Pergament, T. Chu, and S. Katti, "Neural networks meet physical networks: Distributed inference between edge devices and the cloud," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks (HotNets)*, 2018, pp. 50–56.

[12] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu, "Focus: Querying large video datasets with low latency and low cost," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 269–286.

[13] X. Liu, P. Ghosh, O. Ulutan, B. Manjunath, K. Chan, and R. Govindan, "Caesar: cross-camera complex activity recognition," in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems (Sensys)*, 2019, pp. 232–244.

[14] Z. Fang, D. Hong, and R. K. Gupta, "Serving deep neural networks at the cloud edge for vision applications on mobile platforms," in *Proceedings of the 10th ACM Multimedia Systems Conference (MMsys)*, 2019, pp. 36–47.

[15] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *2018-IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2018, pp. 1421–1429.

[16] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *2017 ACM/IEEE Symposium on Edge Computing (SEC)*, 2017, pp. 1–13.

[17] S. Zhang, Y. Li, X. Liu, S. Guo, W. Wang, J. Wang, B. Ding, and D. Wu, "Towards real-time cooperative deep inference over the cloud and edge end devices," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 2, pp. 1–24, 2020.

[18] Z. Yang, K. Nahrstedt, H. Guo, and Q. Zhou, "Deeprt: A soft real time scheduler for computer vision applications on the edge," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2021, pp. 271–284.

[19] Y. Leng, R. Liu, H. Guo, S. Chen, and S. Yao, "Scaleflow: Efficient deep vision pipeline with closed-loop scale-adaptive inference," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 1698–1706.

[20] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings et al., "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[21] J. Wang, Z. Charles, Z. Xu, G. Joshi, H. B. McMahan, M. Al-Shedivat, G. Andrew, S. Avestimehr, K. Daly, D. Data et al., "A field guide to federated optimization," *arXiv preprint arXiv:2107.06917*, 2021.

[22] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan et al., "Towards federated learning at scale: System design," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 374–388, 2019.

[23] A. Jiménez-Sánchez, M. Tardy, M. A. G. Ballester, D. Mateus, and G. Piella, "Memory-aware curriculum federated learning for breast cancer classification," *arXiv preprint arXiv:2107.02504*, 2021.

[24] W. Li, F. Milletarì, D. Xu, N. Rieke, J. Hancox, W. Zhu, M. Baust, Y. Cheng, S. Ourselin, M. J. Cardoso et al., "Privacy-preserving federated brain tumour segmentation," in *International workshop on machine learning in medical imaging*. Springer, 2019, pp. 133–141.

[25] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2020, pp. 557–570.

[26] H. Guo, S. Yao, Z. Yang, Q. Zhou, and K. Nahrstedt, "Crossroi: Cross-camera region of interest optimization for efficient real time video analytics at scale," in *Proceedings of the 12th ACM Multimedia Systems Conference*, 2021, pp. 186–199.

[27] S. M. Nabavinejad, H. Hafez-Kolahi, and S. Reda, "Coordinated dvfs and precision control for deep neural networks," *IEEE Computer Architecture Letters*, vol. 18, no. 2, pp. 136–140, 2019.

[28] Q. Jiao, M. Lu, H. P. Huynh, and T. Mitra, "Improving gpgpu energy-efficiency through concurrent kernel execution and dvfs," in *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2015, pp. 1–11.

[29] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás, "Dvfs-aware application classification to improve gpgpus energy efficiency," *Parallel Computing*, vol. 83, pp. 93–117, 2019.

[30] Z. Tang, Y. Wang, Q. Wang, and X. Chu, "The impact of gpu dvfs on the energy and performance of deep learning: An empirical study," in *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, 2019, pp. 315–325.

[31] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Coordinated batching and dvfs for dnn inference on gpu accelerators," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[32] L. Li, H. Xiong, Z. Guo, J. Wang, and C.-Z. Xu, "Smartpc: Hierarchical pace control in real-time federated learning system," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, 2019, pp. 406–418.

[33] H. Guo, H. Gu, Z. Yang, X. Wang, E. K. Lee, N. Chandramoorthy, T. Eilam, D. Chen, and K. Nahrstedt, "Bofl: bayesian optimized local training pace control for energy efficient federated learning," in *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, 2022, pp. 188–201.

[34] C. Guo, B. Zhao, and Y. Bai, “Deepcore: A comprehensive library for coreset selection in deep learning,” in *Database and Expert Systems Applications: 33rd International Conference, DEXA 2022, Vienna, Austria, August 22–24, 2022, Proceedings, Part I.* Springer, 2022, pp. 181–195.

[35] S. Gang, N. Fabrice, D. Chung, and J. Lee, “Character recognition of components mounted on printed circuit board using deep learning,” *Sensors*, 2021.

[36] O. Sener and S. Savarese, “Active learning for convolutional neural networks: A core-set approach,” *arXiv preprint arXiv:1708.00489*, 2017.

[37] J. Bilmes, “Submodularity in machine learning and artificial intelligence,” *arXiv preprint arXiv:2202.00132*, 2022.

[38] B. Mirzasoleiman, J. Bilmes, and J. Leskovec, “Coresets for data-efficient training of machine learning models,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 6950–6960.

[39] H. Guo, H. Gu, X. Wang, B. Chen, E. K. Lee, T. Eilam, D. Chen, and K. Nahrstedt, “Fedcore: Straggler-free federated learning with distributed coresets,” 2024.

[40] “British transport police: CCTV,” http://www.btp.police.uk/advice_and_information/safety_on_and_near_the_railway/cctv.aspx, accessed: 2021-01-27.

[41] Q. Wang, J. Zhang, B. Guo, Z. Hao, Y. Zhou, J. Sun, Z. Yu, and Y. Zheng, “Cityguard: citywide fire risk forecasting using a machine learning approach,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 4, pp. 1–21, 2019.

[42] Q. Zhou, Z. Yang, H. Guo, B. Tian, and K. Nahrstedt, “360broadview: Viewer management for viewport prediction in 360-degree video live broadcast,” in *Proceedings of the 4th ACM International Conference on Multimedia in Asia*, 2022, pp. 1–7.

[43] Q. Zhou, B. Chen, Z. Yang, H. Guo, and K. Nahrstedt, “360viewpet: View based pose estimation for ultra-sparse 360-degree cameras,” in *2021 IEEE International Symposium on Multimedia (ISM)*. IEEE, 2021, pp. 1–8.

[44] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016, pp. 779–788.

[45] B. Chen, Z. Yan, H. Guo, Z. Yang, A. Ali-Eldin, P. Shenoy, and K. Nahrstedt, “Deep contextualized compressive offloading for images,” in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 2021, pp. 467–473.

[46] H. Guo, B. Tian, Z. Yang, B. Chen, Q. Zhou, S. Liu, K. Nahrstedt, and C. Danilov, “Deepstream: bandwidth efficient multi-camera video streaming for deep learning analytics,” *arXiv preprint arXiv:2306.15129*, 2023.

[47] S. Liu, T. Wang, H. Guo, X. Fu, P. David, M. Wigness, A. Misra, and T. Abdelzaher, "Multi-view scheduling of onboard live video analytics to minimize frame processing latency," in *2022 IEEE 42nd international conference on distributed computing systems (ICDCS)*. IEEE, 2022, pp. 503–514.

[48] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S.-W. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 159–173.

[49] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 377–392.

[50] B. Liu, Y. Li, Y. Liu, Y. Guo, and X. Chen, "Pmc: A privacy-preserving deep learning model customization framework for edge computing," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 4, pp. 1–25, 2020.

[51] A. Guo, A. Jain, S. Ghose, G. Laput, C. Harrison, and J. P. Bigham, "Crowd-ai camera sensing in the real world," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 3, pp. 1–20, 2018.

[52] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h. 264/avc video coding standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560–576, 2003.

[53] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[54] M. Xiao, C. Zhou, Y. Liu, and S. Chen, "Optile: Toward optimal tiling in 360-degree video streaming," in *Proceedings of the 25th ACM international conference on Multimedia (ACM MM)*, 2017, pp. 708–716.

[55] B. Chen, Z. Yan, H. Jin, and K. Nahrstedt, "Event-driven stitching for tile-based live 360 video streaming," in *Proceedings of the 10th ACM Multimedia Systems Conference (MMsys)*, 2019, pp. 1–12.

[56] Y. Guan, C. Zheng, X. Zhang, Z. Guo, and J. Jiang, "Pano: Optimizing 360 video streaming with a better understanding of quality perception," in *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2019, pp. 394–407.

[57] X. Feng, V. Swaminathan, and S. Wei, "Viewport prediction for live 360-degree mobile video streaming using user-content hybrid motion tracking," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 2, pp. 1–22, 2019.

[58] L. Zheng, Y. Yang, and A. G. Hauptmann, "Person re-identification: Past, present and future," *arXiv preprint arXiv:1610.02984*, 2016.

[59] X. Liu, W. Liu, H. Ma, and H. Fu, "Large-scale vehicle re-identification in urban surveillance videos," in *2016 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2016, pp. 1–6.

[60] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," in *European conference on computer vision (ECCV)*. Springer, 2016, pp. 17–35.

[61] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[62] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision (ECCV)*. Springer, 2016, pp. 21–37.

[63] P. Li, G. Li, Z. Yan, Y. Li, M. Lu, P. Xu, Y. Gu, B. Bai, Y. Zhang, and D. Chuxing, "Spatio-temporal consistency and hierarchical matching for multi-target multi-camera vehicle tracking." in *CVPR Workshops*, 2019, pp. 222–230.

[64] Z. He, Y. Lei, S. Bai, and W. Wu, "Multi-camera vehicle tracking with powerful visual features and spatial-temporal cue." in *CVPR Workshops*, 2019, pp. 203–212.

[65] E. Ristani and C. Tomasi, "Features for multi-target multi-camera tracking and re-identification," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR*, 2018, pp. 6036–6046.

[66] "Gurobi Solver," https://www.gurobi.com/, accessed: 2021-01-27.

[67] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, "Sbnet: Sparse blocks network for fast inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8711–8720.

[68] "Scikit Learn," https://scikit-learn.org/, accessed: 2021-01-27.

[69] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[70] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their applications*, vol. 13, no. 4, pp. 18–28, 1998.

[71] "FFmpeg," https://ffmpeg.org, accessed: 2021-01-27.

[72] "Tensorflow," https://www.tensorflow.org/, accessed: 2021-01-27.

[73] M. Naphade, Z. Tang, M.-C. Chang, D. C. Anastasiu, A. Sharma, R. Chellappa, S. Wang, P. Chakraborty, T. Huang, J.-N. Hwang et al., "The 2019 ai city challenge." in *CVPR Workshops*, 2019, pp. 452–460.

[74] "Arecont Vision MegaVideo UltraHD," https://sales.arecontvision.com/product/MegaVideo+UltraHD+Series/AV12ZMV-301, accessed: 2021-01-27.

[75] "Logitech C930e BUSINESS WEBCAM," https://www.logitech.com/en-us/products/webcams/, accessed: 2021-01-27.

[76] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[77] F. Lai, Y. Dai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Fedscale: Benchmarking model and system performance of federated learning," in *Proceedings of the First Workshop on Systems Challenges in Reliable and Secure Federated Learning*, 2021, pp. 1–3.

[78] J. Feng, C. Rong, F. Sun, D. Guo, and Y. Li, "Pmf: A privacy-preserving human mobility prediction framework via federated learning," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 1, pp. 1–21, 2020.

[79] A. Albaseer, B. S. Ciftler, M. Abdallah, and A. Al-Fuqaha, "Exploiting unlabeled data in smart cities using federated edge learning," in *2020 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 2020, pp. 1666–1671.

[80] I. Dayan, H. R. Roth, A. Zhong, A. Harouni, A. Gentili, A. Z. Abidin, A. Liu, A. B. Costa, B. J. Wood, C.-S. Tsai et al., "Federated learning for predicting clinical outcomes in patients with covid-19," *Nature medicine*, vol. 27, no. 10, pp. 1735–1743, 2021.

[81] Y. G. Kim and C.-J. Wu, "Autofl: Enabling heterogeneity-aware energy efficient federated learning," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 183–198.

[82] J. Shin, Y. Li, Y. Liu, and S.-J. Lee, "Sample selection with deadline control for efficient federated learning on heterogeneous clients," *arXiv preprint arXiv:2201.01601*, 2022.

[83] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1935–1949, 2020.

[84] "Jetson AGX Xavier Developer Kit," https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit, accessed: 2022-05-08.

[85] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[86] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[87] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5132–5143.

[88] X. Li, W. Yang, S. Wang, and Z. Zhang, "Communication efficient decentralized training with multiple local updates," *arXiv preprint arXiv:1910.09126*, vol. 5, 2019.

[89] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[90] H. B. McMahan, G. Andrew, U. Erlingsson, S. Chien, I. Mironov, N. Papernot, and P. Kairouz, "A general approach to adding differential privacy to iterative training procedures," *arXiv preprint arXiv:1812.06210*, 2018.

[91] A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor, and R. Rogers, "Protection against reconstruction and its applications in private federated learning," *arXiv preprint arXiv:1812.00984*, 2018.

[92] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[93] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[94] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 387–402.

[95] T. Hashimoto, M. Srivastava, H. Namkoong, and P. Liang, "Fairness without demographics in repeated loss minimization," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1929–1938.

[96] M. B. Zafar, I. Valera, M. G. Rogriguez, and K. P. Gummadi, "Fairness constraints: Mechanisms for fair classification," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 962–970.

[97] P. I. Frazier, "Bayesian optimization," in *Recent advances in optimization and modeling of contemporary problems*. Informs, 2018, pp. 255–278.

[98] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc., 2012.

[99] M. Zuluaga, G. Sergent, A. Krause, and M. Püschel, "Active learning for multi-objective optimization," in *International Conference on Machine Learning*. PMLR, 2013, pp. 462–470.

[100] S. Greenhill, S. Rana, S. Gupta, P. Vellanki, and S. Venkatesh, "Bayesian optimization for adaptive experimental design: A review," *IEEE access*, vol. 8, pp. 13 937–13 948, 2020.

[101] D. C. Manheim and R. L. Detwiler, "Accurate and reliable estimation of kinetic parameters for environmental engineering applications: A global, multi objective, bayesian optimization approach," *MethodsX*, vol. 6, pp. 1398–1414, 2019.

[102] A. Mathern, O. S. Steinholtz, A. Sjöberg, M. Önnheim, K. Ek, R. Rempling, E. Gustavsson, and M. Jirstrand, "Multi-objective constrained bayesian optimization for structural design," *Structural and Multidisciplinary Optimization*, vol. 63, no. 2, pp. 689–701, 2021.

[103] R. Roussel, A. Hanuka, and A. Edelen, "Multiobjective bayesian optimization for online accelerator tuning," *Physical Review Accelerators and Beams*, vol. 24, no. 6, p. 062801, 2021.

[104] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced Lectures on Machine Learning*. Springer, 2003, pp. 63–71.

[105] K. Yang, M. Emmerich, A. Deutz, and T. Bäck, "Multi-objective bayesian global optimization using expected hypervolume improvement gradient," *Swarm and evolutionary computation*, vol. 44, pp. 945–956, 2019.

[106] S. Daulton, D. Eriksson, M. Balandat, and E. Bakshy, "Multi-objective bayesian optimization over high-dimensional search spaces," *arXiv preprint arXiv:2109.10964*, 2021.

[107] D. Ginsbourger, R. L. Riche, and L. Carraro, "Kriging is well-suited to parallelize optimization," in *Computational Intelligence in Expensive Optimization Problems*. Springer, 2010, pp. 131–162.

[108] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, vol. 19, pp. 79–102, 2016.

[109] "Jetson TX2 Module," https://developer.nvidia.com/embedded/jetson-tx2, accessed: 2022-05-08.

[110] "Pytorch," https://pytorch.org, accessed: 2022-05-08.

[111] "Apache MXNET," https://https://mxnet.apache.org/versions/1.9.0/, accessed: 2022-05-08.

[112] "INA3221 Texas Instruments," https://www.ti.com/product/INA3221, accessed: 2022-05-08.

[113] "Secondmind-Trieste Document," https://secondmind-labs.github.io/trieste/index.html, accessed: 2022-05-08.

[114] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly et al., "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[115] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[116] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

[117] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[118] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[119] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011. [Online]. Available: http://www.aclweb.org/anthology/P11-1015 pp. 142–150.

[120] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "Fedhealth: A federated transfer learning framework for wearable healthcare," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 83–93, 2020.

[121] Z. Xiong, Z. Cheng, C. Xu, X. Lin, X. Liu, D. Wang, X. Luo, Y. Zhang, N. Qiao, M. Zheng et al., "Facing small and biased data dilemma in drug discovery with federated learning," *BioRxiv*, 2020.

[122] S. R. Pfohl, A. M. Dai, and K. Heller, "Federated and differentially private learning for electronic health records," *arXiv preprint arXiv:1911.05861*, 2019.

[123] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Distributed federated learning for ultra-reliable low-latency vehicular communications," *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 1146–1159, 2019.

[124] Y. Liu, J. James, J. Kang, D. Niyato, and S. Zhang, "Privacy-preserving traffic flow prediction: A federated learning approach," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7751–7763, 2020.

[125] B. S. Ciftler, A. Albaseer, N. Lasla, and M. Abdallah, “Federated learning for localization: A privacy-preserving crowdsourcing method,” *arXiv preprint arXiv:2001.01911*, 2020.

[126] F. Yin, Z. Lin, Q. Kong, Y. Xu, D. Li, S. Theodoridis, and S. R. Cui, “Fedloc: Federated learning framework for data-driven cooperative localization and location data processing,” *IEEE Open Journal of Signal Processing*, vol. 1, pp. 187–215, 2020.

[127] S. Duan, D. Zhang, Y. Wang, L. Li, and Y. Zhang, “Jointrec: A deep-learning-based joint cloud video recommendation framework for mobile iot,” *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 1655–1666, 2019.

[128] T. Qi, F. Wu, C. Wu, Y. Huang, and X. Xie, “Privacy-preserving news recommendation model learning,” *arXiv preprint arXiv:2003.09592*, 2020.

[129] C. Xie, S. Koyejo, and I. Gupta, “Asynchronous federated optimization,” *arXiv preprint arXiv:1903.03934*, 2019.

[130] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, “On the convergence of FedAvg on non-iid data,” in *International Conference on Learning Representations*, 2020.

[131] M. Nasr, R. Shokri, and A. Houmansadr, “Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning,” in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 739–753.

[132] Y. Mansour, M. Mohri, J. Ro, and A. T. Suresh, “Three approaches for personalization with applications to federated learning,” *arXiv preprint arXiv:2002.10619*, 2020.

[133] Q. Wu, K. He, and X. Chen, “Personalized federated learning for intelligent iot applications: A cloud-edge based framework,” *IEEE Open Journal of the Computer Society*, vol. 1, pp. 35–44, 2020.

[134] O. Chapelle and L. Li, “An empirical evaluation of thompson sampling,” *Advances in neural information processing systems*, vol. 24, 2011.

[135] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, “Controlled experiments on the web: survey and practical guide,” *Data mining and knowledge discovery*, vol. 18, no. 1, pp. 140–181, 2009.

[136] D. J. Lizotte, T. Wang, M. H. Bowling, D. Schuurmans et al., “Automatic gait optimization with gaussian process regression.” in *IJCAI*, vol. 7, 2007, pp. 944–949.

[137] R. Martinez-Cantin, N. De Freitas, E. Brochu, J. Castellanos, and A. Doucet, “A bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot,” *Autonomous Robots*, vol. 27, no. 2, pp. 93–103, 2009.

[138] E. O. Pyzer-Knapp, “Bayesian optimization for accelerated drug discovery,” *IBM Journal of Research and Development*, vol. 62, no. 6, pp. 2–1, 2018.

[139] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, “{CherryPick}: Adaptively unearthing the best cloud configurations for big data analytics,” in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 469–482.

[140] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, “Neural architecture search with bayesian optimisation and optimal transport,” *Advances in neural information processing systems*, vol. 31, 2018.

[141] C. White, W. Neiswanger, and Y. Savani, “Bananas: Bayesian optimization with neural architectures for neural architecture search,” *arXiv preprint arXiv:1910.11858*, vol. 1, no. 2, p. 4, 2019.

[142] H. Jin, Q. Song, and X. Hu, “Auto-keras: An efficient neural architecture search system,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 1946–1956.

[143] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin, and S. Bakas, “Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation,” in *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 4th International Workshop*. Springer, 2019, pp. 92–104.

[144] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, “Hybridalpha: An efficient approach for privacy-preserving federated learning,” in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 13–23.

[145] G. Long, Y. Tan, J. Jiang, and C. Zhang, “Federated learning for open banking,” in *Federated Learning: Privacy and Incentive*. Springer, 2020, pp. 240–254.

[146] Y. G. Kim and C.-J. Wu, “Autofl: Enabling heterogeneity-aware energy efficient federated learning,” in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 183–198.

[147] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, “Oort: Efficient federated learning via guided participant selection.” in *Operating Systems Design and Implementation*, 2021, pp. 19–35.

[148] L. Lyu, H. Yu, X. Ma, C. Chen, L. Sun, J. Zhao, Q. Yang, and S. Y. Philip, “Privacy and robustness in federated learning: Attacks and defenses,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[149] T. Nishio and R. Yonetani, “Client selection for federated learning with heterogeneous resources in mobile edge,” in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.

[150] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 3581–3607.

[151] M. van Dijk, N. V. Nguyen, T. N. Nguyen, L. M. Nguyen, Q. Tran-Dinh, and P. H. Nguyen, "Asynchronous federated learning with reduced number of rounds and with differential privacy from less aggregated gaussian noise," *arXiv preprint arXiv:2007.09208*, 2020.

[152] X. Li, Z. Qu, B. Tang, and Z. Lu, "Stragglers are not disaster: A hybrid federated learning algorithm with delayed gradients," *arXiv preprint arXiv:2102.06329*, 2021.

[153] Z. Chai, Y. Chen, A. Anwar, L. Zhao, Y. Cheng, and H. Rangwala, "Fedat: a high-performance and communication-efficient federated learning system with asynchronous tiers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–16.

[154] K. Killamsetty, S. Durga, G. Ramakrishnan, A. De, and R. Iyer, "Grad-match: Gradient matching based data subset selection for efficient deep model training," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5464–5474.

[155] S. Chakraborty, A. S. Bedi, A. Koppel, B. M. Sadler, F. Huang, P. Tokekar, and D. Manocha, "Posterior coreset construction with kernelized stein discrepancy for model-based reinforcement learning," *arXiv preprint arXiv:2206.01162*, 2022.

[156] Y. Huang, K. Xie, H. Bharadhwaj, and F. Shkurti, "Continual model-based reinforcement learning with hypernetworks," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 799–805.

[157] H. Bostani, M. Sheikhan, and B. Mahboobi, "A strong coreset algorithm to accelerate opf as a graph-based machine learning in large-scale problems," *Information Sciences*, vol. 555, pp. 424–441, 2021.

[158] Y. Chen, M. Welling, and A. Smola, "Super-samples from kernel herding," in *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'10, 2010.

[159] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A coreset approach," in *International Conference on Learning Representations*, 2018.

[160] C. Sohler and D. P. Woodruff, "Strong coresets for k-median and subspace approximation: Goodbye dimension," in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2018, pp. 802–813.

[161] M. Toneva, A. Sordoni, R. T. d. Combes, A. Trischler, Y. Bengio, and G. J. Gordon, "An empirical study of example forgetting during deep neural network learning," *arXiv preprint arXiv:1812.05159*, 2018.

[162] O. Bachem, M. Lucic, and A. Krause, "Coresets for nonparametric estimation-the case of dp-means," in *International Conference on Machine Learning*. PMLR, 2015, pp. 209–217.

[163] M. Paul, S. Ganguli, and G. K. Dziugaite, "Deep learning on a data diet: Finding important examples early in training," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 20 596–20 607.

[164] M. Ducoffe and F. Precioso, "Adversarial active learning for deep networks: a margin based approach," *arXiv preprint arXiv:1802.09841*, 2018.

[165] K. Margatina, G. Vernikos, L. Barrault, and N. Aletras, "Active learning by acquiring contrastive examples," *arXiv preprint arXiv:2109.03764*, 2021.

[166] O. Pooladzandi, D. Davini, and B. Mirzasoleiman, "Adaptive second order coresets for data-efficient machine learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 17 848–17 869.

[167] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smithy, "Feddane: A federated newton-type method," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2019, pp. 1227–1231.

[168] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 7611–7623.

[169] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 2009.

[170] W. Sheng and X. Liu, "A genetic k-medoids clustering algorithm," *Journal of Heuristics*, vol. 12, pp. 447–466, 2006.

[171] Z. Allen-Zhu, "Katyusha: The first direct acceleration of stochastic gradient methods," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, 2017, pp. 1200–1205.

[172] A. Katharopoulos and F. Fleuret, "Not all samples are created equal: Deep learning with importance sampling," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2525–2534.

[173] F. Haddadpour and M. Mahdavi, "On the convergence of local descent methods in federated learning," *arXiv preprint arXiv:1910.14425*, 2019.

[174] Y. LeCun, C. Cortes, and C. J. C. Burges, "Mnist handwritten digit database," http://yann.lecun.com/exdb/mnist, 2010.

[175] Intel, "Intel core x series core i9 10920x cpu specifications," Online, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/198012/intel-core-i910920x-xseries-processor-19-25m-cache-3-50-ghz/specifications.html

[176] NVIDIA, "Geforce rtx 2080 ti gpu specifications," Online, 2023. [Online]. Available: https://www.nvidia.com/en-us/geforce/20-series/

[177] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 8024–8035.

[178] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu et al., "Fedml: A research library and benchmark for federated machine learning," *arXiv preprint arXiv:2007.13518*, 2020.

[179] S. U. Stich, "Local sgd converges fast and communicates little," in *International Conference on Learning Representations*, 2019.

[180] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[181] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, no. 4, pp. 681–694, 2020.

[182] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu et al., "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[183] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[184] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[185] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E. P. Xing et al., "Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 559–578.

[186] S. Choi, I. Koo, J. Ahn, M. Jeon, and Y. Kwon, "{EnvPipe}: Performance-preserving {DNN} training framework for saving energy," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 851–864.

[187] J. Jiang, F. Fu, T. Yang, and B. Cui, "Sketchml: Accelerating distributed machine learning with data sketches," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1269–1284.

[188] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, M. Canini, and P. Kalnis, “Grace: A compressed communication framework for distributed machine learning,” in *2021 IEEE 41st international conference on distributed computing systems (ICDCS)*. IEEE, 2021, pp. 561–572.

[189] Y. Bai, C. Li, Q. Zhou, J. Yi, P. Gong, F. Yan, R. Chen, and Y. Xu, “Gradient compression supercharged high-performance data parallel dnn training,” in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 359–375.