National Technical University of Athens
School of Electrical and Computer Engineering

Division of Computer Science and Technology

HarmoniFL: Resource-Adaptive Federated Learning Tool

DIPLOMA THESIS

of

NIKOLAOS S. VLACHAKIS

Supervisor: Dimitrios Tsoumakos
Associate Professor

Athens, April 2024

National Technical University of Athens
School of Electrical and Computer Engineering

Division of Computer Science and Technology

HarmoniFL: Resource-Adaptive Federated Learning Tool

DIPLOMA THESIS

of

NIKOLAOS S. VLACHAKIS

Supervisor: Dimitrios Tsoumakos
Associate Professor

Issued by the three-member committee of inquiry in April 2024

.....	.....	.....
Dimitrios Tsoumakos	Minos Garofalakis	Dionysios Pneymatikatos

Athens, April 2024

.....
Nikolaos S. Vlachakis
3th April 2024

Abstract

In the realm of Federated Learning (FL), the presumption of uniform processing capacity among participating clients overlooks the reality of diverse client hardware. This diversity introduces system heterogeneity, leading to disparities in computational resources that ultimately compromise the efficient utilization of distributed data, thus hindering optimal learning outcomes. Addressing this challenge, this thesis introduces HarmoniFL, an open-source initiative engineered to navigate the complexities of resource heterogeneity within federated learning frameworks. By focusing on enhancing efficiency and inclusivity, HarmoniFL employs a dynamic client selection protocol that leverages real-time metrics such as CPU load, memory availability, and network bandwidth to optimize device participation in learning tasks. Adaptive strategies—ranging from data sampling adjustments to epoch reduction and batch size optimization for high-demand devices—address the challenges posed by resource diversity. Our experimental analysis aims at two primary goals: minimizing training duration for less capable devices and improving the accuracy of the aggregated global model. Results demonstrate that HarmoniFL effectively reduces training times and enhances model performance, underscoring its potential to foster more equitable device participation in federated learning tasks without sacrificing learning quality. The code repository can be found at `https://github.com/NikosVlachakis/harmoni-fl.git`.

Keywords and Phrases: Federated Learning, Device Heterogeneity, Resource-Adaptive, Flower, Prometheus, MLflow, Open-Source Tool Development

Εκτεταμένη Περίληψη

Στη σύγχρονη σφαίρα της μηχανικής μάθησης, η άνοδος της ιδιωτικότητας των δεδομένων ως κεντρικής σημασίας ζήτημα έχει προκαλέσει την επανεκτίμηση των συμβατικών πλαισίων, τοποθετώντας την ομοσπονδιακή μάθηση ως ένα καινοτόμο παράδειγμα που κυριαρχεί σήμερα στις συζητήσεις εντός του κλάδου. Βαθιά ριζωμένη στην αποκέντρωση της επεξεργασίας δεδομένων και της εκπαίδευσης μοντέλων, στοχεύει στην προστασία της ιδιωτικής ζωής του ατόμου, ενώ αξιοποιεί συλλογικά τους κατανεμημένους πόρους δεδομένων, αντιπροσωπεύοντας έτσι μια πρωτοποριακή προσέγγιση που διευρύνει τα όρια του πεδίου. Παρά τις δυνατότητές του, αυτό το πολλά υποσχόμενο πεδίο αντιμετωπίζει πολυάριθμες σημαντικές προκλήσεις που εμποδίζουν τη βέλτιστη εφαρμογή και αποτελεσματικότητά του, περιλαμβάνοντας προκαταλήψεις που απορρέουν από μεθόδους δειγματοληψίας δεδομένων, αλγοριθμικούς σχεδιασμούς και διαφοροποιήσεις στις δυνατότητες των συσκευών, οι οποίες ασκούν αξιοσημείωτη επίδραση στη δικαιοσύνη και την ακρίβεια των μοντέλων. Επιπλέον, οι περιπλοκές που σχετίζονται με την ετερογένεια των δεδομένων, η οποία χαρακτηρίζεται από διαφορές στους τύπους και τους όγκους δεδομένων μεταξύ των κόμβων, εισάγουν πολυπλοκότητες που εμποδίζουν τη διαδικασία μάθησης δημιουργώντας εμπόδια που απαιτούν προσεκτική πλοήγηση.

Ταυτόχρονα, οι γνωστικές προκαταλήψεις περιπλέκουν περαιτέρω την κατάσταση εισάγοντας ανθρώπινα σφάλματα στις διαδικασίες λήψης αποφάσεων και στο σχεδιασμό του συστήματος, θέτοντας έτσι πρόσθετα εμπόδια στην επιτυχή ανάπτυξη του. Επιπλέον, η εγγενής ποικιλομορφία στις ικανότητες των συσκευών εντός των δικτύων αποτελεί πρόκληση, εμποδίζοντας την πλήρη αξιοποίηση των δυνατοτήτων κάθε συσκευής. Οι ανισότητες στην υπολογιστική ισχύ, την αποθηκευτική ικανότητα και τις δυνατότητες δικτύωσης μεταξύ αυτών δεν δημιουργούν μόνο τεχνικά εμπόδια, αλλά εγείρουν και ηθικά διλήμματα σχετικά με την ισότιμη πρόσβαση και συμμετοχή. Μέσα σε αυτή τη συνεχιζόμενη συζήτηση, το HarmoniFL αποτελεί μια λύση προσαρμοσμένη να ξεπεράσει τις πολυπλοκότητες που συνδέονται με την ετερογένεια των συσκευών. Η προσέγγιση που ακολουθούμε εξασφαλίζει μια πιο περιεκτική συμμετοχή σε όλο το ευρύ φάσμα των συσκευών που συμμετέχουν στην διαδικάσια της εκπαίδευσης του τελικού μοντέλου.

Οι κυρίαρχες μεθοδολογίες συχνά αγνοούν την ετερογένεια ανάμεσα στις συσκευές, υποθέτοντας λανθασμένα ότι όλες έχουν παρόμοιες ικανότητες, και κατ΄ επέκταση αντιμετωπίζουν αναποτελεσματικά τις σχετικές προκλήσεις. Ως απάντηση σε αυτό το πρόβλημα, αναπτύχθηκαν καινοτόμες μεθοδολογίες που προσαρμόζονται δυναμικά στη διαφορετικότητα των συσκευών. Αυτές οι προσεγγίσεις εκμεταλλεύονται με ευελιξία το εύρος των υπολογιστικών δυνατοτήτων που υφίστανται στον σύγχρονο τεχνολογικό κόσμο. Μέσω της εφαρμογής εξελιγμένων τεχνικών όπως η προσαρμοστική μοντελοποίηση, η δομημένη εγκατάλειψη και η διαμόρφωση ευέλικτων τοπικών κύκλων εκπαίδευσης, ανοίγει ο δρόμος έτσι ώστε κόμβοι με ελάχιστους υπολογιστικούς πόρους να μπορούν να συμμετέχουν στην διαδικασία εκπαίδευσης του μοντέλου.

Εμπνευσμένοι από τις τελευταίες εξελίξεις, αναπτύξαμε ένα σύστημα που αναγνωρίζει και αξιοποιεί την ποικιλία δυνατοτήτων στο οικοσύστημα των συσκευών. Το σύστημά μας εφαρμόζει έναν μηχανισμό επιλογής κόμβων, ο οποίος βασίζεται σε πραγματικού χρόνου μετρήσεις απόδοσης για κάθε

συσκευή, όπως υπολογιστική ισχύς, κατάσταση μνήμης και δικτυακή απόδοση. Χρησιμοποιώντας αυτά τα δεδομένα σε συνδυασμό με ειδικούς κανόνες ορισμένους από τον διαχειριστή του συστήματος, το σύστημα μας προσαρμόζει τις παραμέτρους του μοντέλου για κάθε συσκευή ξεχωριστά, έτσι ώστε να ενσωματώσει επιτυχώς όλες τις συσκευές στην μαθησιακή διαδικασία.

Στο βασικό του επίπεδο, το HarmoniFL ενσωματώνει τρία κρίσιμα στοιχεία - τον Επιλογέα Πελατών, τον Εξαγωγέα Μετρήσεων και τον Ελεγκτή Στρατηγικής - τα οποία συνεργάζονται για να επιβλέπουν δυναμικά την επιλογή και τη ρύθμιση των συσκευών, ενισχύοντας τελικά τη συμμετοχή στη διαδικασία της εκπαίδευσης. Ο Επιλογέας Πελατών αξιολογεί την ετοιμότητα των πελατών και, καθοδηγούμενος από τον Ελεγκτή Στρατηγικής, εξάγει στρατηγικά κριτήρια που ειναι ορσιμένα από τον διαχειριστή της εκπαίδευσης, για να επιτρέψει την ενδελεχή αξιολόγηση της καταλληλότητας των πελατών με βάση μετρήσεις σε πραγματικό χρόνο που παρέχονται από τον Εξαγωγέα Μετρήσεων.

Η αρχιτεκτονική του HarmoniFL ενισχύεται περαιτέρω μέσω της ενσωμάτωσής του με το Flower, το Prometheus, το Grafana και το MLflow. Το Flower μας παρέχει την υποδομή για αποκεντρωμένη εκπαίδευση μοντέλων μηχανικής μάθησης, ενσωματώνοντας εκεί τη λογική επιλογής πελατών. Το Prometheus και το Grafana διαδραματίζουν κρίσιμο ρόλο στην παρακολούθηση και την οπτικοποίηση των δεδομένων, συλλέγοντας και παρουσιάζοντας μετρήσεις σε πραγματικό χρόνο δίνοντας αναλυτικές πληροφορίες σχετικά με τις αποδόσεις και την κατάσταση που βρίσκονται όλες οι συσκευές, όσον αφορά τις μετρήσεις σε επίπεδο υλικού. Με την ενσωμάτωση του MLflow, το HarmoniFL ενισχύει τη διαχείριση του κύκλου ζωής της διαδικασίας εκπαίδευσης, προωθώντας την ιχνηλασιμότητα και την αναπαραγωγιμότητα σε όλα τα πειράματα μάθησης.

Στην πράξη, το HarmoniFL εφαρμόζει μια διαδοχική και κυκλική μεθοδολογία κατά τη διάρκεια κάθε γύρου εκπαίδευσης. Αυτή η διαδικασία ξεκινά με τη συλλογή δεδομένων από τις συνδεδεμένες συσκευές, ακολουθούμενη από την εξατομικευμένη προσαρμογή των παραμέτρων του μοντέλου για κάθε συσκευή ξεχωριστά. Ακολούθως, κάθε συσκευή προχωρά στην εκπαίδευση του δικού της τοπικού μοντέλου, χρησιμοποιώντας τις ειδικά προσαρμοσμένες παραμέτρους. Μετά την ολοκλήρωση της εκπαίδευσης, οι παράμετροι του κάθε τοπικού μοντέλου αποστέλλονται πίσω στον κεντρικό διακομιστή, όπου πραγματοποιείται η συνάθροιση τους. Αυτή η διαδικασία επαναλαμβάνεται, μέχρι να επιτευχθεί ένα εκ των προτέρων καθορισμένο κριτήριο σύγκλισης ή ένας συγκεκριμένος αριθμός γύρων εκπαίδευσης. Το κριτήριο σύγκλισης ή ο αριθμός των γύρων μπορεί να καθοριστεί από τον διαχειριστή του συστήματος εκπαίδευσης με βάση τους στόχους απόδοσης και ακρίβειας που επιθυμεί να επιτύχει.

Η έρευνα που διεξήχθη αναδικνύει την σημαντική επίδραση του HarmoniFL στην ενίσχυση της συνολικής ακρίβειας και στη μείωση των χρόνων εκπαίδευσης. Με την ευέλικτη προσαρμογή σε διαφορετικές συσκευές, το HarmoniFL ξεπερνά τις συνήθεις υπολογιστικές προκλήσεις, όπως οι περιορισμοί μνήμης σε λιγότερο ισχυρές συσκευές, διασφαλίζοντας έτσι ότι όλες οι συσκευές μπορούν να συμμετέχουν στη διαδικασία μάθησης. Τα πειράματα, τα οποία χρησιμοποίησαν ένα ποικίλο σύνολο συσκευών και ένα προσεκτικά επιμελημένο σύνολο δεδομένων, επιβεβαιώνουν την αποτελεσματικότητα του HarmoniFL στη δυναμική προσαρμογή των παραμέτρων μάθησης.

Με βάση αυτά τα ευρήματα, μπορεί κανείς να συμπεράνει ότι η ενίσχυση των περιβαλλόντων ομοσπονδιακής μάθησης μέσω προσαρμογής ώστε να ταιριάζει με τις προδιαγραφές των συσκευών μπορεί να οδηγήσει σε σημαντική βελτίωση της ακρίβειας των μοντέλων και μείωση της διάρκειας των περιόδων εκπαίδευσης, ιδιαίτερα για συσκευές με περιορισμένες υπολογιστικές ικανότητες. Αυτή η έρευνα υπογραμμίζει την κεντρική σημασία της προσαρμοστικότητας στην προώθηση της ένταξης και τη βελτιστοποίηση της ακρίβειας εντός των ομοσπονδιακών δικτύων, διασφαλίζοντας ότι κάθε συσκευή, ανεξάρτητα από τις δυνατότητές της, μπορεί να συμβάλλει σημαντικά στην κοινή προσπάθεια μάθησης.

Στο άμεσο μέλλον, προτείνονται μελλοντικές κατευθύνσεις που στοχεύουν στην ενίσχυση της προσαρμοστικότητας του συστήματος μέσω της επέκτασης των κριτηρίων που χρησιμοποιούνται για την προσαρμογή των παραμέτρων μάθησης, όπως και η βελτίωση του αλγορίθμου επιλογής πελατών ενσωμα-

τώνοντας πιο εξελιγμένες τεχνικές ομαδοποίησης, εξαλείφοντας έτσι την ανάγκη συλλογής μετρήσεων για κάθε πελάτη σε κάθε γύρο. Επιπλέον, μια αξιόλογη κατεύθυνση είναι η επέκταση του μηχανισμού συνάθροισης ώστε να εντάσσει τις τοπικές παραμέτρους εκπαίδευσης για κάθε χρήστη, διευκολύνοντας έτσι την προσαρμογή στις αλλαγές που επιφέρει κάθε χρήστης στο μοντέλο του. Αυτές οι πρωτοβουλίες επιδιώκουν κυρίως να αυξήσουν την αποδοτικότητα και να εμπλουτίσουν τις λειτουργίες του HarmoniFL, βελτιστοποιώντας τη συνολική απόδοση και την αξιοποίηση των δεδομένων.

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Purpose of the Thesis

The genesis of this thesis is rooted in addressing a pivotal challenge within the sphere of federated learning (FL): the issue of resource heterogeneity among participating devices. Such heterogeneity, marked by diverse computational capabilities, network conditions, and memory availabilities, poses a substantial barrier to the fair and efficient operation of FL systems. The principal objective of this thesis is to forge a novel pathway through this challenge, making a substantial contribution to the federated learning domain.

To surmount this hurdle, we have developed a Resource-Adaptive Federated Learning Tool. The tool is devised with a dual-purpose strategy: firstly, to optimize the learning process by dynamically adapting to the fluctuating resource profiles of participating devices, thereby ensuring seamless and efficient learning engagements; and secondly, to bolster the inclusivity and efficacy of FL networks by facilitating equitable participation across a spectrum of devices.

Central to achieving these aims is the implementation of a sophisticated dynamic client selection mechanism. This strategy is grounded in the utilization of real-time resource metrics—such as processing power, memory usage, and network bandwidth—enabling the adaptive tool to fine-tune the learning protocol to the specificities of each device's resource landscape. Through this approach, the thesis introduces a solution to the perennial challenge of resource diversity in federated learning, thereby paving the way for more resilient, efficient, and inclusive FL ecosystems.

1.2 Structure of the Thesis

This thesis is organized into several chapters, each focusing on a different aspect of managing device heterogeneity in federated learning environments. The structure is as follows:

- **Chapter 1: Introduction** - This chapter introduces the thesis, outlining its purpose and structure.
- **Chapter 2: Background** - Provides an overview of federated learning, including its definition, key characteristics, and importance.
- **Chapter 3: Challenges in Federated Learning** - Discusses the various challenges faced in federated learning.

- **Chapter 4: Adaptivity in Federated Learning: A Review** - Offers a critical examination of existing adaptivity mechanisms within federated learning, setting the stage for the proposed system.
- **Chapter 5: Proposal for an Adaptive Federated Learning Tool** - Delivers a comprehensive description of the proposed adaptive federated learning system. It details the system's architecture, the integration of tools and frameworks, and the design strategies that enhance adaptivity and efficiency.
- **Chapter 6: Experiment Results** - Details the experimental setup, describes the methodologies employed to evaluate the proposed system, and analyzes the results, demonstrating the system's effectiveness.
- **Chapter 7: Conclusion** - Summarizes the key findings, discusses the contributions and implications of the research, and suggests directions for future exploration and enhancement.

Each chapter builds upon the previous one, culminating in a comprehensive understanding of the challenges, solutions, and implications of managing device heterogeneity in federated learning environments.

Chapter 2

Background

2.1 Definition of Federated Learning

Federated Learning (FL) is a groundbreaking approach in machine learning that deviates from the traditional centralized paradigm. In a stark contrast to methods where data is gathered and processed centrally, FL empowers model training directly on decentralized data sources, primarily located on users' devices. This shift is not just a technical leap but also a stride towards enhancing data privacy, utilizing the distributed computing power available across numerous devices.

The core of FL lies in its unique ability to train algorithms across a multitude of decentralized devices, like smartphones or IoT devices, without centralizing the data. This approach is a departure from traditional methods, presenting both opportunities and challenges. In FL, each device in the network, acting as a client, trains a model on its local data and computes updates to this model. These updates, rather than the raw data, are then transmitted to a central server.

This server plays a crucial role in the FL process. It aggregates the updates from various clients, using sophisticated algorithms such as Federated Averaging, as conceptualized by McMahan et al. [25]. The central server then updates the global model based on these aggregated insights. Such a process ensures that the sensitive data remains within its original location, effectively upholding user privacy.

The workflow and actors involved in this federated learning process are succinctly depicted in Figure 2.1. This figure illustrates the cyclical nature of FL, where each participating device trains the model on local data and sends only the model updates to the server. These updates are then synthesized by the server to refine the global model, which is subsequently shared back with the devices. This iterative process allows for continuous improvement of the model, leveraging the distributed nature of the data while maintaining its confidentiality.

One of the landmark contributions to this field by Bonawitz et al. [8] delineates a comprehensive system design tailored for federated learning, especially focusing on mobile devices. Their pioneering work addresses the practical challenges associated with FL, such as the variability in device availability, issues of connectivity, and limitations inherent to on-device resources. This work underscores the feasibility and scalability of FL in real-world scenarios.

In summary, FL heralds a significant evolution in the field of machine learning. It offers a framework that is not only privacy-preserving and efficient but also decentralized, aligning with the increasing concerns over data privacy and the widespread presence of edge computing devices.

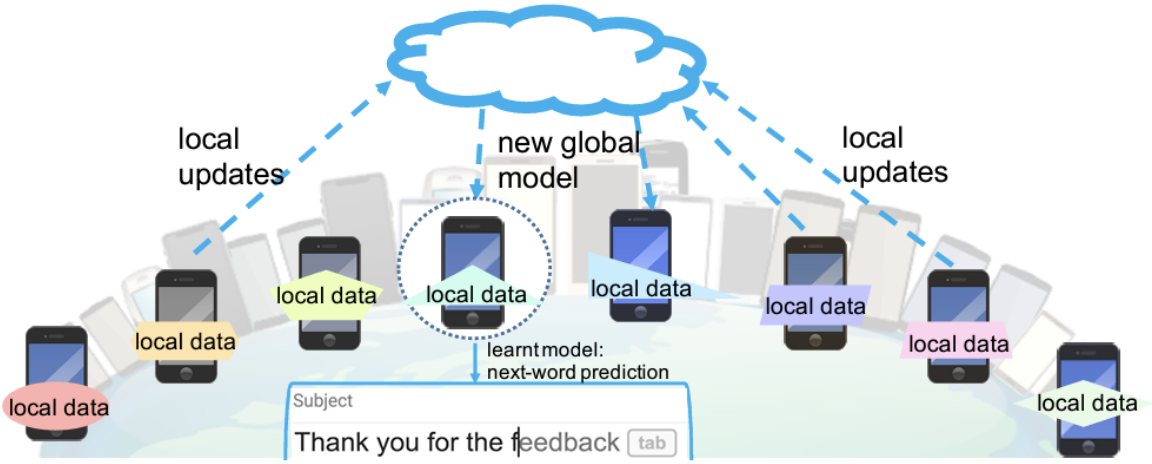

Figure 2.1: The lifecycle of an FL-trained model and the various actors in a federated learning system [20]

2.2 Key Characteristics of Federated Learning

Federated Learning (FL) revolutionizes the traditional machine learning landscape with its unique characteristics. These features address pivotal challenges in data privacy, decentralized processing, and collaborative model training, making FL an essential paradigm in today's data-driven world.

Decentralized Data Training Decentralization is the cornerstone of FL. In this model, training occurs directly on users' devices, a stark contrast to the traditional data-centralized approaches. Each participant in the FL network utilizes its local data to train an independent model. These local models are then synthesized into a global model through a process that involves sending only the model updates, not the data itself, to a centralized server. This process effectively addresses privacy concerns, as sensitive user data never leaves its original location. Bonawitz et al. [8] explore the intricate system designs necessary to facilitate this decentralized training, addressing challenges such as device availability, connectivity reliability, and computational constraints inherent in device-based model training.

Collaborative Model Building Collaboration is a defining feature of FL. It allows for the collective contribution of a diverse array of devices, each enriching the global model with its unique dataset. This process leads to a more robust and inclusive model that benefits from an expansive range of data characteristics, such as varying distributions and modalities. Konečnỳ et al. [18] delve into the collaborative nature of FL, highlighting how it leverages the strengths of distributed datasets while maintaining the privacy and security of user data. Their work underlines the effectiveness of FL in environments where data cannot be centralized due to privacy or logistical constraints.

Privacy by Design Privacy is not just a feature but a fundamental principle of FL. The architecture of FL inherently supports data privacy by training models on local devices without transferring raw data to external servers. This design philosophy aligns with the increasing global emphasis on data privacy and security, making FL an attractive approach for applications handling sensitive information. McMahan et al. [25] emphasize this aspect of FL, presenting it as a solution to the growing need for privacy-preserving data analysis in the digital age. Their work illustrates

how FL can be utilized in scenarios where traditional data sharing and centralized processing are not feasible or desired.

Scalability and Flexibility The scalability of FL is a critical attribute that allows it to handle a large number of participants efficiently. This scalability is complemented by the system's flexibility to work with various data structures and types. Sattler et al. [30] examine these aspects, discussing how FL can be scaled to accommodate the growing number of devices and data sources in modern networks. They also explore the adaptability of FL to different domains and applications, highlighting its potential to transform various sectors by providing scalable and flexible machine learning solutions.

Asynchronous Updates and Fault Tolerance Asynchronous updates represent a significant advantage in FL, enabling the system to manage updates from various devices that might not always be simultaneously connected to the network. This feature ensures the system's robustness, particularly in environments with unreliable or intermittent device connectivity. Bonawitz et al. [8] address the challenges and solutions related to asynchronous communication in FL systems. They discuss how FL can maintain consistency and reliability in the learning process despite the asynchronous nature of client updates and the potential for device dropouts.

Model Personalization Personalization is another hallmark of FL. Leveraging local datasets allows FL to tailor models to the specific characteristics and preferences of individual users or groups. This ability to customize models based on local data nuances enhances the relevance and effectiveness of the model for end-users. Kairouz et al. [17] explore the personalization potential within FL frameworks, illustrating how local training can lead to models that are finely tuned to the needs and contexts of specific user groups.

These characteristics collectively underscore the transformative potential of federated learning, making it a pivotal approach in the evolving landscape of machine learning.

2.3 Importance of Federated Learning

Federated Learning (FL) stands as a beacon in the realm of machine learning, reshaping how data intelligence is harnessed while safeguarding privacy and efficiency. Its significance in the contemporary technological landscape extends far beyond the conventional boundaries of data analysis, influencing a multitude of sectors and application areas.

Revolutionizing Privacy-Preserving Data Analysis FL represents a paradigm shift in the approach to data privacy. It inherently preserves user data confidentiality by allowing the data to stay where it is generated – on the user's devices. This methodology is crucial in today's world, where data breaches and privacy concerns are increasingly becoming a public issue. By processing data locally and only sharing model updates, FL provides a pathway to leverage data insights while adhering to stringent privacy standards. This characteristic is particularly valuable in areas where data sensitivity is paramount, such as in medical records analysis or personal finance management. McMahan et al. [25] delve into this aspect, illustrating how FL aligns with and enhances privacy preservation in machine learning.

Empowering AI in Edge Computing As computing continues to shift towards the edge, FL emerges as a key player in this transition. It capitalizes on the computational power available in modern edge devices, enabling localized AI processing. This not only reduces latency in AI applications but also minimizes the dependency on cloud computing infrastructures, leading to more efficient and responsive systems. Bonawitz et al. [8] discuss the synergy between FL and edge computing, highlighting its potential to transform the landscape of local data processing and real-time decision-making.

Diverse Applications Across Various Domains The adaptability of FL to various fields is one of its most compelling attributes. The methodology has been successfully applied in numerous domains, each benefiting from its decentralized, privacy-preserving nature. For instance: - *Healthcare*: FL enables the development of predictive models for disease diagnosis by leveraging data from various hospitals while maintaining patient confidentiality. - *Financial Services*: In banking and finance, FL aids in fraud detection and risk management by learning from transactional data across different branches without centralizing sensitive financial information. - *Smart Cities*: FL is utilized in smart city initiatives, optimizing traffic flow, and public safety by processing data from various urban sensors and devices. - *Retail and E-commerce*: It helps in personalizing customer experiences by analyzing shopping patterns directly on users' devices. - *Telecommunications*: FL enhances network optimization and predictive maintenance by learning from distributed network data. Kairouz et al. [17] provide a comprehensive overview of these applications, showcasing the versatility and widespread impact of FL.

Future Prospects and Technological Evolution The trajectory of FL points towards a future rich with innovation and enhanced capabilities. As technological advancements continue, particularly in the realms of IoT, 5G networks, and edge computing, the potential applications and effectiveness of FL are poised to expand exponentially. This evolution positions FL as a cornerstone technology in the future landscape of AI and machine learning, driving forward new frontiers in data-driven decision-making and intelligent system design.

In sum, the importance of Federated Learning in the current and future technological era is profound. It stands as a critical solution to privacy concerns, a facilitator of edge AI, and a versatile tool in a wide array of applications. Its continued development and adoption are set to redefine the paradigms of data analysis and machine learning.

Chapter 3

Challenges in Federated Learning

3.1 Bias

In the rapidly evolving domain of Federated Learning (FL), a profound understanding of bias becomes vital. Bias—systematic discrepancies leading to a deviation of model predictions from actual values—can permeate FL processes through various avenues, subsequently affecting model performance, fairness, and utility [12, 17].

Bias within FL proves to be multifaceted, originating from diverse elements including sampling techniques, heterogeneity in data and systems, fusion algorithm design, and cognitive aspects [12, 17, 23, 22]. Identifying these biases necessitates an exploration into complex factors like device selection criteria, connectivity logistics, computational capacity, data volume, and the impact of human decision-making [17, 8, 22]. Unchecked, these biases can significantly affect the global model, skewing predictions, enabling unequal representation, and reducing overall model applicability [12, 17, 23, 30].

The ensuing analysis is committed to elucidating these intricate aspects of bias in FL, scrutinizing its origins, manifestations, and implications.

3.1.1 Sampling Bias

In the domain of federated learning, sampling bias emerges as a significant and complex issue influencing the generation and performance of the global model. The complex web of factors contributing to this form of bias encompasses not only the volume and diversity of data nodes but also the intricate logistics of device connectivity, computational performance, and data availability.

Sampling bias materializes when the data nodes (or devices) utilized for model training do not uniformly represent the wider population of interest. This lack of representativeness can skew the learning process and exert disproportionate influence on the final model, potentially undermining its predictive power for underrepresented groups [12, 17, 30].

A notable facet of sampling bias in federated learning systems pertains to node selection. Nodes are often chosen for model updates based on their availability or charge status, resulting in a temporal dimension of bias. Specifically, if model updates predominantly occur at specific times, an unintentional correlation could form with factors like day-shift versus night-shift work schedules. Such correlations, though seemingly subtle, can result in the overrepresentation or underrepresentation of certain user groups in the aggregated model, thereby impacting its overall performance [17, 8, 23].

An additional contributing factor to sampling bias is the timing and frequency of device connectivity. As federated learning relies on real-time data from connected devices, those that maintain connectivity during off-peak times could potentially have a higher chance of selection due to lower network competition. This dynamic could lead to an overrepresentation of these off-peak devices in the aggregated output, thereby subtly influencing the characteristics of the global model [12, 17, 8].

The influence of device performance and data volume on sampling bias presents another critical dimension to consider. Devices endowed with faster processors or possessing smaller datasets may complete computations more rapidly, potentially leading to their overrepresentation in the aggregated model updates. Consequently, the global model could be skewed towards these higher-performance devices or those with fewer data, leading to an unintentional form of sampling bias [12, 17].

In sum, the nuanced nature of sampling bias in federated learning highlights the intricacy of attaining fair and representative learning in a decentralized environment [12, 17, 23]. This necessitates meticulous attention and mitigation strategies to ensure the federated learning process and the resultant global model genuinely represent the user population's diversity, thereby augmenting the model's predictive accuracy and fairness [12, 17, 8, 30].

3.1.2 Data Heterogeneity

Heterogeneity in data represents a formidable challenge within the context of federated learning systems. In these decentralized networks, participating nodes (clients) - each reflecting a distinct user or group of users - are characterized by diverse behaviors, preferences, and traits. As a consequence, the data harbored at each node can significantly differ, leading to a substantial degree of heterogeneity across the network. This disparity can stem from varying data types, volumes, and qualities across clients, and critically influences the effectiveness and fairness of federated learning implementations [23].

To elucidate, one can consider a healthcare scenario where multiple hospitals participate in a federated learning network to develop a predictive model for patient outcomes. Each hospital, located in a unique geographical setting, serves a distinct patient demographic, leading to disparities in data regarding age, ethnicity, and lifestyle diseases. Furthermore, hospitals specializing in certain medical domains will have an overrepresentation of data corresponding to those domains. Such inter-hospital variations culminate in a heterogeneous dataset across the federated learning network [17].

Similarly, variations in user engagement levels can cause data volumes to differ across nodes. In a federated learning setup involving a mobile application, for example, power users will generate larger volumes of data than casual users due to their higher interaction levels. This disparity in data volume contributes further to the overall data heterogeneity [20].

The inherent data heterogeneity has significant repercussions on the global model developed through federated learning. A common practice in fusion algorithms is to weigh node contributions proportional to their data volume. However, this might result in a global model unduly skewed by nodes with larger data volumes, compromising its generalizability [12, 17, 8, 23, 25]. For instance, in the earlier hospital scenario, if the fusion algorithm assigns more weight to a cardiology-specializing hospital, the resultant model might disproportionately reflect cardiac patient characteristics, potentially undermining its effectiveness for different ailments.

Further, sparse or infrequent data from certain nodes might deprive the global model of adequate training for corresponding user groups. Using the mobile application example, the model might underperform for infrequent users if their data isn't adequately represented during training [20, 22].

In light of these implications, it is vital that federated learning systems are designed with meticulous consideration of data heterogeneity. Ensuring the equitable representation of various data distributions in the global model is imperative for achieving optimal performance and fairness in these systems. Moreover, understanding the sources of this heterogeneity, whether from user behavior, geographic factors, specialized data collections, or variations in data volumes, is fundamental to its effective management in federated learning.

3.1.3 Device Heterogeneity

In the federated learning context, device heterogeneity refers to the variability in the computational, communication, and power capabilities of the devices participating in the learning process. This analysis involves understanding how these device-specific characteristics influence the federated learning process, potentially introducing bias and affecting the global model's performance [12, 17, 25].

Computationally, devices in a federated network may vary extensively in their hardware attributes, including processing speed and memory capacity. These variations dictate the rate at which devices can conduct model training and updating. More computationally potent devices might compute updates quicker, possibly leading to overrepresentation if faster updates are favored during the aggregation process. Conversely, devices with less computational power might lag, creating disparities in the timeline of contributions, and possibly causing their updates to be overlooked or undervalued [12, 17, 23].

Communication capabilities, determined by network connectivity (3G, 4G, 5G, WiFi, etc.), can significantly influence the federated learning process. Devices with more robust or reliable network connections might transmit updates more efficiently, leading to potential favoritism or prioritization of their updates. Conversely, devices with weaker or less reliable connections may face challenges transmitting updates, resulting in potential delays or data losses that could disproportionately impact their representation in the learning process [17, 20].

Power constraints, including battery life, are equally critical. Devices with longer battery life or access to a constant power supply might be more readily available for participation in the federated learning process, leading to more frequent or timely updates. Conversely, devices with shorter battery life or infrequent access to a power source might face interruptions or be less available, potentially reducing their contribution frequency or timeliness [8, 23].

Lastly, a variety of device-related constraints, often determined by network size and inherent device limitations, may exist. Given the vast scale of typical federated networks, potentially involving millions of devices, only a small fraction might be active at any given moment. This low participation rate, coupled with the unpredictability of device dropout due to connectivity or energy constraints, can further complicate the federated learning process [17, 22].

In sum, these elements collectively contribute to the overall system heterogeneity in federated learning, each presenting unique challenges and potential biases. A comprehensive understanding of device heterogeneity and its implications is fundamental for fair and efficient federated learning implementations [25, 18, 30].

3.1.4 Fusion Algorithm Bias

Fusion algorithms are the backbone of federated learning systems, acting as the central mechanism to merge updates from individual models into a comprehensive global model. The intricacies of these algorithms are pivotal in shaping the learning and prediction capabilities of the model, yet can simultaneously serve as a conduit for bias [12, 8, 25].

The main purpose of a fusion algorithm is the aggregation of diverse local updates. The manner in which these updates are weighted and amalgamated can foster disparities across nodes, which can subsequently transpire as bias in the resulting model. Consider, for instance, the case of unweighted aggregation. This approach treats all local model updates equivalently, regardless of their originating devices' unique attributes such as data volume, data diversity, or usage frequency. A node with sparse or low-quality data is awarded equal influence over the global model as a node with extensive, high-quality data, potentially diluting the accuracy of the overall model [12, 8, 23].

To illustrate, consider a federated learning ecosystem implementing an equal average fusion algorithm. This method affords equal weightage to all participating nodes, irrespective of the quality or volume of their data contributions. Consequently, the global model could be unduly influenced by nodes possessing poor-quality or unrepresentative data, leading to a decrease in the model performance or prediction accuracy [23].

On the other side of the spectrum lies the weighted average fusion algorithm, a technique that assigns weights to local model updates commensurate with their data volume or frequency of use. This approach seems to reward nodes that are more active or data-rich, theoretically driving the global model toward higher accuracy. However, this method bears its own set of complications. By disproportionately weighing data-rich nodes, the global model could develop a bias towards these nodes. This could result in a model that is well-tuned for a particular subset of nodes but falls short in generalizing to the wider network [17, 20].

This discussion highlights the potential for fusion algorithms to act as conduits for bias in federated learning. The selection and deployment of these algorithms demand careful consideration, taking into account their potential implications on the fairness and performance of the global model. Future advancements in federated learning should endeavor to tackle these challenges, devising strategies that ensure fair representation in the global model while preserving the overarching goal of model accuracy [8, 18, 30].

3.1.5 Human-Centric Biases

Cognitive Bias Cognitive bias arises from systematic errors in human decision-making, which can occur during the formulation of data collection procedures, the design of federated learning algorithms, and even in the process of selecting which nodes are eligible for participation. Such biases could implicitly seep into the federated learning process through the design of the applications generating the data, shaping the nature and range of data collected, and thus affecting the subsequent machine learning model. While the degree and impact of cognitive bias can be difficult to quantify due to its inherent human element, it remains a crucial factor to consider in the design and deployment of federated learning systems [17].

Reporting Bias Reporting bias in federated learning systems is primarily a consequence of differential data availability across user groups. For instance, populations that do not own devices due to socioeconomic constraints may not contribute data to the training dataset. This lack of representation in the dataset and subsequent model training and evaluation presents a form of reporting bias [17]. The system's ability to generalize and make accurate predictions for these underrepresented or unrepresented populations may be compromised as a result.

Confirmation Bias Confirmation bias could play a role in the interpretation and evaluation of federated learning outcomes. As an example, if there are pre-existing beliefs or assumptions about the behaviors of certain device owners or populations, these might influence the interpretation of results derived from the federated learning process [17]. Analysts may unconsciously seek out outcomes that confirm their preconceived notions, potentially overlooking evidence to the contrary.

3.2 Communication Efficiency

Efficient communication underpins the success of Federated Learning (FL) systems. This section explores the multifaceted aspects of communication efficiency, focusing on how network bandwidth and latency impact the performance and scalability of FL. These factors are crucial for the synchronization and timely exchange of data across distributed networks, directly influencing the learning process and model accuracy. The following subsections delve into specific challenges related to network bandwidth and latency, providing a comprehensive understanding of their individual and combined effects on FL efficiency.

3.2.1 Latency

The influence of network latency manifests in several challenges in Federated Learning. Notably, latency introduces a consequential delay in the convergence of the global model. In the distributed learning framework that FL adopts, rapid communication between the server and the participating clients is a crucial component of the learning process. Increased latency obstructs this communication, extending the time required for the global model to achieve convergence. This impact is particularly significant when the learning model necessitates a large number of communication rounds, thereby highlighting how latency can considerably decelerate the process [17, 8].

Furthermore, latency may precipitate a condition of asynchronous updates. In high-latency environments, updates originating from different clients are likely to reach the central server at varied intervals. This inconsistency in time can introduce irregularities in the global model as the server attempts to reconcile updates received at different time frames. This latency-induced asynchronicity has the potential to disrupt the orderly learning process and can contribute to a degradation in the overall performance of the global model [17, 25].

Another challenge in the context of latency is the potential increase in device dropout rates. For instance, in a high-latency environment, there is an increased risk that a device may disconnect before successfully communicating its update to the server, due to reasons such as user interference or power loss. Such disruptions in communication can lead to the transmission of irregular updates, impeding the continuity of the learning process [17, 22].

Lastly, scalability represents a significant challenge in the presence of network latency in Federated Learning. As the number of participating devices in FL increases, the system becomes increasingly susceptible to the impacts of latency-induced delays and asynchronous updates. Therefore, effective management of latency becomes an increasingly critical consideration as the system scales to accommodate a larger number of clients. In larger systems, high latency can introduce substantial delays in the transmission of model updates, thereby affecting the efficiency and operational speed of FL systems [8, 25].

In conclusion, network latency in Federated Learning introduces a range of challenges, including the delay in convergence of the global model, asynchronous updates, increased device dropouts, and scalability issues. These challenges highlight the importance of effectively managing network latency to ensure the operational efficiency and robustness of FL systems.

3.2.2 Network Bandwidth

In Federated Learning, the communication between the server and clients necessitates the transfer of a substantial volume of data. This data primarily comprises model parameters and gradient updates, which can significantly increase in number when handling intricate models such as deep

learning networks. As such, high network bandwidth becomes an integral requirement to efficiently facilitate this data exchange [8, 25].

In practice, FL systems typically involve a large number of clients, potentially reaching into the thousands or millions. Given that each client must transmit and receive updates, the overall demand for network bandwidth escalates proportionally with the count of participating clients. Hence, network bandwidth becomes a substantial bottleneck, posing a challenge to the scalability of FL systems [8, 25].

The requirement for network bandwidth is further intensified by the frequency of communication rounds. Each round involves clients sending local updates to the server, followed by the server disseminating the updated global model back to the clients. Therefore, in scenarios necessitating numerous communication rounds for model convergence, the demand for network bandwidth increases considerably [8, 25].

Additionally, the diverse network conditions in a real-world FL implementation introduce further complexity to the bandwidth challenge. Devices participating in the FL process may range from those with high-speed internet connectivity to those restrained by limited bandwidth networks. This heterogeneity may result in slower devices constraining the overall learning process due to their extended time to send or receive updates, attributable to their limited network bandwidth [8].

Network bandwidth challenges in FL are also magnified when considering data transfer costs, particularly in regions or circumstances where these costs are substantial. The consistent interchange of updates between the server and clients in FL systems can culminate in high data transfer costs. This could impose a financial burden on the clients, potentially making participation in FL unfeasible for some [25].

In conclusion, the high-dimensional nature of models, the large number of participants, the frequency of communication, and the diversity in network conditions among clients collectively contribute to network bandwidth being a significant challenge in FL. The ramifications on scalability and data transfer costs pose considerable obstacles to the practical implementation of FL systems.

3.3 Privacy Concerns

Federated Learning (FL) has gained significant attention as a privacy-preserving approach to collaborative machine learning. However, FL brings forth inherent privacy challenges that need to be addressed. The following analysis delves into the privacy challenges encountered in FL, aiming to identify and analyze the key obstacles to data privacy. By understanding these challenges, effective privacy-enhancing mechanisms can be developed to ensure the confidentiality and integrity of participant data in FL systems.

3.3.1 Data Leakage

Data Leakage is a critical challenge in Federated Learning (FL) which can compromise the privacy-preservation premise of the learning process. This form of leakage can surface in distinct yet interconnected ways, mainly through Membership Inference Attacks and Model Inversion Attacks. The analysis of each category can provide a nuanced understanding of the mechanisms of data leakage and the associated risks.

Membership Inference Attacks Membership inference attacks pose a significant risk to privacy in Federated Learning (FL) systems. An adversary during such an attack attempts to determine if a specific data point was part of the model's training set. This inference is possible by leveraging model predictions or shared updates during federated learning [17]. Real-world data distribution

among clients, often imbalanced and non-IID (Independent and Identically Distributed), contributes to the susceptibility of such attacks. Unique characteristics from skewed data or classes present with few participants could be reflected in the global model, enabling an adversary to make educated guesses about data point membership [30]. The risk of these attacks increases in Vertical Federated Learning scenarios where different entities own different features of the same sample. Consequently, shared model updates may reveal sensitive information about local data [22].

Model Inversion Attacks Model inversion attacks present another avenue for data leakage in FL systems. In these attacks, adversaries strive to reconstruct original data inputs from the model's outputs or shared updates. The decentralized nature of FL models exacerbates the potential impact of these attacks. Shared model parameters or gradients during aggregation may carry information about data distribution, indirectly revealing information if the client's data has unique statistical properties [23]. Both Vertical and Horizontal Federated Learning scenarios face the risk of model inversion attacks. In Vertical FL, despite entities possessing only a subset of features for each data point, shared model updates could provide sufficient information for an adversary to reconstruct original data points [22]. Similarly, in Horizontal FL, model inversion attacks are feasible based on shared model updates, even when each entity holds different samples of the same set of features [23].

Both of these attack vectors highlight the complexity and nuances of the data leakage challenge in FL. While these learning systems are designed with privacy as a priority, the potential for data leakage through these methods continues to pose substantial privacy threats. Addressing this requires the development of more sophisticated and privacy-preserving FL models and protocols, which can safeguard against these forms of data leakage without hampering the efficacy of the learning process.

3.3.2 Colluding Participants

Federated Learning (FL), despite its data privacy advantages, faces a significant challenge from colluding participants, who can pool their shared knowledge to infringe upon the privacy safeguards of the federated system [17]. Collusion among participants can enable them to aggregate and analyze more information than what is conventionally shared during the model update process, thereby posing a substantial threat to the privacy of data.

In FL, there are two primary models: Vertical Federated Learning (VFL) and Horizontal Federated Learning (HFL). VFL involves scenarios where different entities possess different features for the same set of samples, while HFL involves scenarios where each entity holds different samples of the same set of features. Each model faces unique challenges regarding collusion among participants.

For instance, in a VFL scenario, shared model updates amongst colluding entities could potentially help infer sensitive information about local data [23]. This is particularly concerning in VFL due to the diverse nature of the data features held by different entities, which, when combined, can reveal comprehensive information about the data.

Similarly, in an HFL scenario, shared model parameters amongst colluding entities might leak information about the unique statistical properties of a client's data, thereby revealing information about the data distribution [17]. In HFL, the risk lies in the potential for colluding participants to reconstruct the data distributions of other clients based on shared updates, thus compromising the privacy of the data.

The potential risk of colluding participants also expands beyond the realm of data leakage to include the risk of model poisoning, where colluding entities could manipulate their local updates in a coordinated manner to skew the global model towards their preferred outcomes.

Chapter 4

Adaptivity in Federated Learning: A Review

The landscape of federated learning (FL) has rapidly evolved to address the challenge of device heterogeneity, with several pioneering works laying the groundwork for adaptive and efficient FL frameworks. This chapter delves into key contributions that have significantly advanced the domain of resource-adaptive federated learning.

4.1 All-In-One Neural Composition for Enhanced Resource Adaptivity

Mei et al. [26] tackle federated learning's system heterogeneity challenge through the "All-In-One Neural Composition" mechanism. This approach diverges from conventional model adjustments by enabling the dynamic adaptation of model complexities to fit the computational resources available across different client devices. Utilizing a unified neural basis, it allows for the efficient construction of diverse model complexities, effectively managing the balance between model sophistication and the computational capabilities of client devices. However, FLANC assumes that all clients can support at least the neural basis, and thus its adaptivity is bounded by the size of this pre-defined unified neural basis.

A distinctive feature of this methodology is its leverage of orthogonal regularization to enhance the representational capacity of the neural basis, ensuring a broader encapsulation of knowledge. This is pivotal in federated settings, where the goal is to enrich the collective intelligence of the network through the aggregation of diverse local updates.

Empirically validated across various benchmarks, the All-In-One Neural Composition demonstrates superior performance against existing approaches, showcasing its ability to address the trade-offs inherent in federated learning systems due to device heterogeneity. By fostering a more inclusive federated learning environment, Mei et al.'s work significantly advances the field towards achieving efficient and equitable learning across a heterogeneous array of devices.

4.2 ScaleFL: Resource-Adaptive Federated Learning with Heterogeneous Clients

Ilhan et al. [16] introduce ScaleFL, an innovative framework designed to navigate the complexities of device heterogeneity in Federated Learning (FL). ScaleFL uniquely adapts Deep Neural Networks (DNNs) to the computational constraints of varied clients through two-dimensional model adjustments, optimizing both the width and depth of the model. This optimization, facilitated by the strategic use of early exits, allows for the creation of client-specific models that efficiently balance the extraction of basic and complex features.

A cornerstone of ScaleFL's approach is the implementation of self-distillation techniques during the training phase, where predictions from the final exit act as soft labels to enhance the training of earlier exits. This process not only improves the flow of knowledge across the network but also ensures a more cohesive and effective integration of diverse model updates into the global model. The efficacy of ScaleFL is underscored by its performance enhancements and inference efficiency gains demonstrated across a suite of benchmark datasets in computer vision and natural language processing.

Through its novel methodology, ScaleFL significantly advances the federated learning field by offering a scalable and inclusive solution that accommodates a broad spectrum of device capabilities, thereby fostering a more equitable federated learning ecosystem.

4.3 FjORD: Fair-and-Accurate Federated Learning under Heterogeneous Targets with Ordered Dropout

Horváth et al. [10] present FjORD, a framework innovatively addressing client heterogeneity in Federated Learning (FL) through the Ordered Dropout (OD) mechanism. Unlike traditional dropout methods, OD allows for the efficient creation of nested submodels of varying complexities from a single master model, without the need for retraining. This capability significantly enhances the inclusivity of FL by accommodating the computational diversity of client devices.

Key to FjORD's approach is the integration of self-distillation, where knowledge from more complex models is used to enrich the training of simpler submodels. This technique ensures that devices of all capabilities can meaningfully contribute to the learning process, improving both the performance and fairness of the federated model.

Evaluated across diverse datasets and network architectures, FjORD has shown to outperform baseline methods, demonstrating its ability to adaptively scale models during inference to meet the specific needs of each device. Such adaptability is crucial for real-world FL applications, underscoring FjORD's role in advancing federated learning towards more equitable and efficient deployments across heterogeneous devices.

4.4 HETEROFL: Tailoring Federated Learning to Device Capabilities

HETEROFL, conceptualized by Diao, Tarokh, and Ding (2021) [5], addresses client heterogeneity in Federated Learning (FL) by innovatively adjusting the model's width without altering its depth. This approach enables the inclusion of devices with varying computational resources by cus-

tomizing the width split ratio for each client, thus significantly reducing the computational burden without compromising the model's performance.

Key to HETEROFL's strategy is the use of static Batch Normalization (sBN), which facilitates the training of privacy-sensitive models across devices with different capabilities. This technique is crucial for optimizing performance and ensuring data privacy in constrained environments.

Empirical tests of HETEROFL across diverse datasets and architectures have demonstrated its efficacy, particularly in non-IID data scenarios, showcasing its potential to improve learning efficiency and model accuracy in real-world federated networks. HETEROFL's approach of width-adjustable models presents a scalable solution to the challenges of device diversity in FL, marking a significant step towards a more adaptable and inclusive federated learning framework.

4.5 Federated Optimizations in Heterogeneous Networks

The FedProx algorithm, introduced by Tian Li et al. [21], marks a significant advancement in federated learning by addressing the challenge of device heterogeneity. This method permits variable amounts of local training iterations, effectively allowing each client to participate in the training process according to their computing power. Such adaptability is essential for including a broader array of devices in federated learning networks, especially those with limited computational resources.

FedProx enhances the conventional federated learning approach by introducing a mechanism that accounts for system heterogeneity, making the learning process more inclusive and efficient. However, it operates under the premise that all clients can support the same model architecture, focusing on optimizing the participation level of each device rather than modifying the model itself to fit the device capabilities.

This approach significantly contributes to the federated learning field by offering a practical solution to manage the variances in computational capabilities among participating devices, ensuring that each can contribute to the collective learning effort within their operational constraints.

Chapter 5

Proposal for an Adaptive Federated Learning Tool

5.1 System Overview

Addressing the complexities of device heterogeneity in the rapidly evolving domain of federated learning (FL) necessitates innovative and adaptive solutions. This thesis introduces a sophisticated tool that rises to this challenge. Central to its strategy is a dynamic client selection mechanism, which intelligently assesses and responds to the diverse capabilities of devices based on device-specific metrics.

A pivotal element of our system is the integration and extension of Flower, a flexible and robust framework for federated learning[7]. By extending key methods within Flower, we implement a bespoke client selection logic, allowing for a more nuanced and efficient orchestration of the federated learning process. This tailored approach ensures that the system is not just accommodating the diversity of devices but is actively optimizing the learning process based on their varied capabilities.

The user-configurable nature of the system empowers practitioners to precisely define operational criteria through a configuration file, ensuring that the system aligns with the specific needs of varying federated learning scenarios. This customization extends to setting specific parameters and thresholds that control the participation and contribution of devices in the learning network, making the system highly adaptable to different operational environments.

Integral to the system's functionality is the strategic utilization of Prometheus and cAdvisor, which are instrumental in sourcing both real-time and historical device metrics. These metrics play a crucial role in shaping our sophisticated client selection logic. By integrating these metrics with the user-defined parameters in the configuration file, the system orchestrates an informed and strategic client selection process for each learning iteration. The synergy between these device metrics and user configurations allows for a more responsive and intelligently adaptive approach to managing the diverse capabilities of devices in the federated learning network.

In addition to these capabilities, the system's proficiency is further augmented by integrating with MLflow and Grafana. This combination offers a comprehensive suite for tracking, managing, and visualizing both the machine learning lifecycle and the federated learning process. The use of MLflow and Grafana ensures an optimized and effective federated learning environment by providing detailed insights into the learning process and the performance of the federated network, thereby facilitating continuous improvement and effective decision-making.

Designed with extendability and modular architecture at its core, the system's codebase is

primed for scalability and future enhancements. This foresight allows for the seamless integration of additional features or criteria for client selection, positioning the system as a versatile and evolving solution for the challenge of device heterogeneity in federated learning.

The subsequent sections of this chapter will provide a detailed exploration of the various components and the unique functionalities of our system.

5.2 Integration with Existing Tools and Frameworks

5.2.1 Flower Framework

Introduction and Rationale for Choosing Flower

Flower stands out in the federated learning (FL) landscape, particularly for its adept handling of heterogeneous client environments, a key challenge in FL systems. Unlike other FL frameworks, Flower is uniquely capable of supporting workloads involving clients running on different platforms and using various languages, all within the same workload. This feature is vital in scenarios targeting edge devices, where a client pool might include a diverse array of devices such as smartphones, tablets, and embedded systems [7].

Flower's support for heterogeneous clients is enabled through its communication-agnostic and language-agnostic client-side integration points. This is a distinguishing factor from other frameworks like TensorFlow Federated (TFF) and PySyft, which typically expect a framework-provided client runtime. In contrast, frameworks like FedScale and LEAF primarily focus on Python-based simulations, limiting their applicability in diverse real-world scenarios. Flower's ability to integrate with a multitude of device types and programming languages makes it unparalleled in the federated learning domain for practical, large-scale deployments [7].

Moreover, Flower's design facilitates large-scale and heterogeneous client management, crucial in a landscape where devices vary significantly in terms of computational power, memory capacity, and network connectivity. Its framework-agnostic nature allows for seamless integration with a variety of machine learning frameworks, enhancing its suitability for diverse and evolving FL scenarios. The scalability of Flower, capable of managing experiments with millions of clients, further demonstrates its practical applicability in real-world scenarios [7].

In summary, Flower's unique combination of scalability, flexibility, framework and communication-agnosticism, and its unparalleled ability to handle heterogeneous clients in real-world applications make it an ideal choice for our federated learning system, setting it apart from other available FL frameworks [7].

Flower's Architecture and Core Concepts

Flower's architecture is designed to efficiently manage the complexities of federated learning. Key aspects of its architecture and core concepts include:

- **Server-Client Structure** Central to Flower is a server-client model where the server orchestrates the FL process, and clients, ranging from smartphones to powerful computers, perform model training using their local data.
- **Federated Learning Rounds** In these rounds, selected clients train models on their data and send updates back to the server, where they are aggregated to refine the global model.

- **Strategy** The strategy component is pivotal in the federated learning process within Flower, responsible for critical operations like client selection, model aggregation methods, and model evaluation. Flower comes equipped with ready-made strategies, including the Federated Averaging (FedAvg) algorithm introduced by McMahan et al.[25], which serves as a solid foundation for many FL tasks. In addition to these built-in strategies, Flower's strategy abstraction empowers the implementation of custom strategies, crucial for adapting to diverse FL environments and enhancing client selection logic. This abstraction encompasses several methods that are integral to the federated learning process.

  Within Flower, the `configure_fit` method is primarily responsible for selecting which clients will participate in the current round of federated learning. This selection process is central to the efficacy and efficiency of the federated learning system, as it directly influences the diversity and representativeness of the data used in model training. In our proposed tool, we have extended the `configure_fit` method to further refine this selection process, tailoring it to our specific federated learning needs. The details and implications of this extension will be elaborated upon in the later sections.

  To illustrate how the `configure_fit` method currently operates, we provide a sequence diagram. This diagram offers a detailed visual representation of the method's workflow, demonstrating the interactions between the server and clients and the extended logic for client selection:

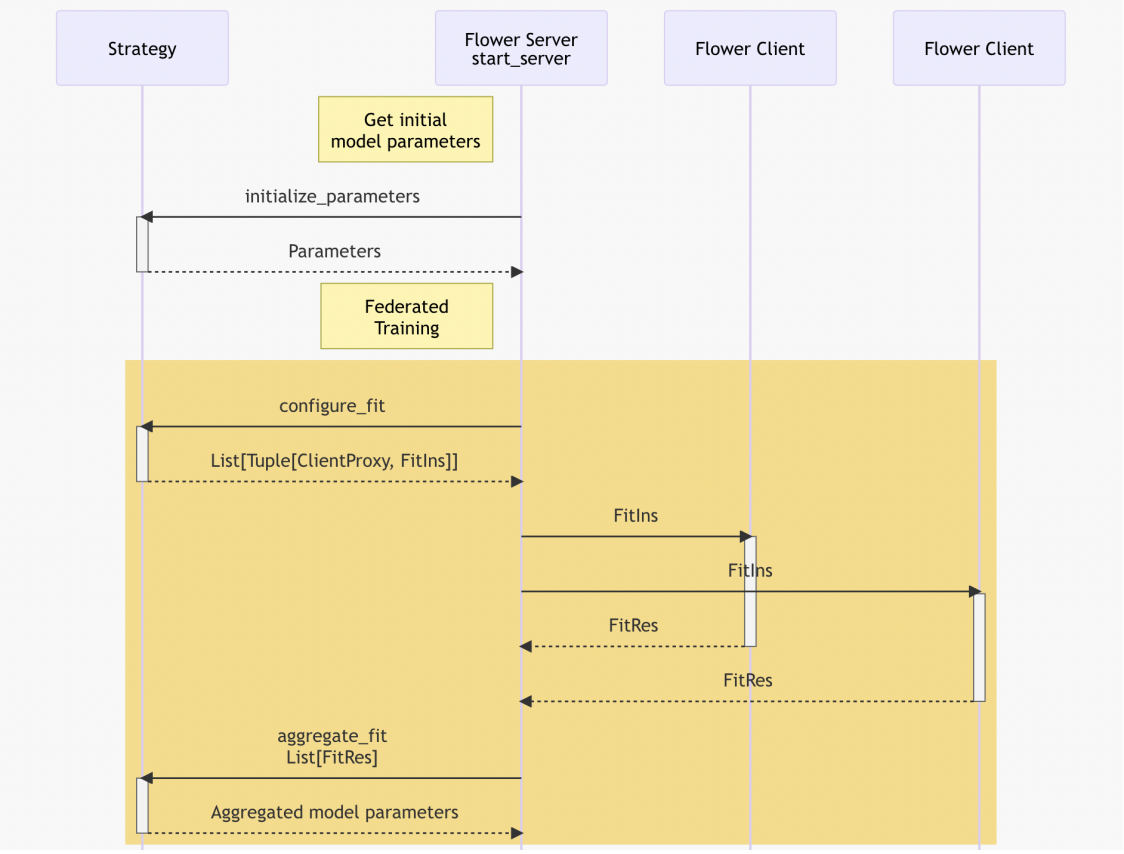

Figure 5.1: Sequence diagram showing the workflow of the `configure_fit` method in Flower, illustrating the client selection process and server-client interactions.[13]

The sequence diagram in Figure 5.1 provides a clear visualization of the `configure_fit` method's operation within the Flower federated learning workflow. The diagram begins with the Strategy module, which initiates the federated training process. It first retrieves the initial model parameters through the `initialize_parameters` call. These parameters are then passed on to the federated training sequence where the `configure_fit` method comes into play.

During the `configure_fit` phase, the Strategy module generates a list of tuples, each containing a ClientProxy and FitIns (fit instructions), which are sent to the selected Flower Clients. Each client processes these instructions, performs local model training, and then returns a FitRes (fit results) object. The server, upon receiving all FitRes objects from the participating clients, passes them to the `aggregate_fit` method. This method aggregates the updates to form the new global model parameters, which then become the starting point for the next round of training. This cyclical process is at the heart of Flower's federated learning approach, ensuring that the global model iteratively improves with inputs from a diverse set of clients.

The architecture diagram depicted in Figure 5.2 illustrates the structure of the Flower federated learning framework. At the highest level sits the Strategy module, which is responsible for overarching decisions in the federated learning process, such as client coordination and global model updates.

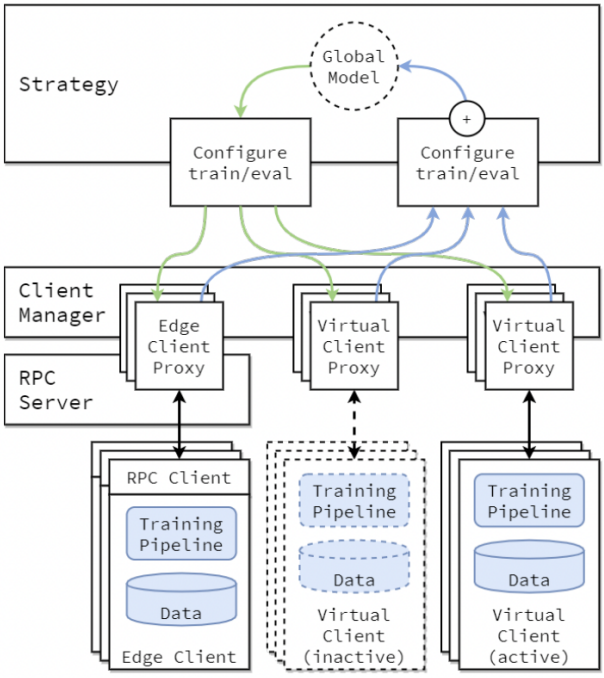

Figure 5.2: The Flower framework's architecture showcases the interaction between the Strategy module, the Client Manager, and the clients themselves, which are categorized into Edge and Virtual Client Proxies.[7]

Below the Strategy module, we see the Client Manager, acting as the intermediary between the Strategy and the clients. It manages the Client Proxies, which represent the connected clients

within the system. The Client Manager utilizes RPC (Remote Procedure Call) to facilitate secure and efficient communication between the server and the edge clients. RPC allows programs on edge devices to request services from the central server program, despite being on separate computers across the network, which is a cornerstone of Flower's federated learning process.

The diagram differentiates between Edge Client Proxies, which communicate directly with Edge Clients via RPC, and Virtual Client Proxies, which represent clients that are not actively participating in the current learning round.

The Edge Client section at the bottom left of the diagram represents actual devices, such as smartphones or embedded systems, which hold the training data and execute the training pipeline. These clients are the workhorses of the federated learning process, performing computations on their local data and communicating results back to the server through their respective proxies.

On the other hand, the Virtual Client, shown at the bottom right of the diagram, stays in an inactive state when not in use, conserving resources. It activates and loads data into memory only when selected for training or evaluation, demonstrating the efficient utilization of computational resources in Flower's design.

This architecture enables Flower to manage a federated learning process with a diverse set of clients, balancing the demands of active and inactive clients while ensuring a robust and scalable learning environment.

5.2.2 Utilizing Prometheus and Grafana for Monitoring

Prometheus, an open-source monitoring solution, plays a crucial role in the proposed advanced federated learning system. It is adept at scraping and storing metrics from a multitude of sources, making it an invaluable tool for tracking and analyzing the performance of federated learning workloads [27].

To visualize Prometheus's role in the system, Figure 5.3 showcases its architecture:

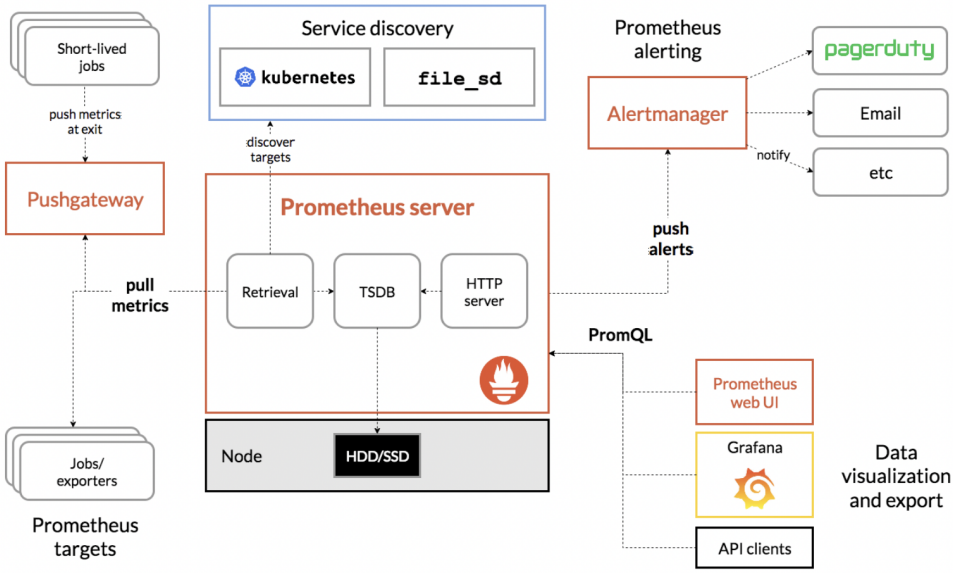

Figure 5.3: Architecture of Prometheus illustrating its ecosystem components and the process of metric collection, storage, and alerting.[27]

In the architecture (Figure 5.3), at the foundation, we have the Prometheus targets, which

could be any instrumented jobs or services that expose metrics. These targets are monitored by Prometheus, which pulls metrics at regular intervals. Prometheus can scrape metrics directly from these jobs or via an intermediary such as the Pushgateway, which is particularly useful for short-lived jobs.

At the core of Prometheus is the server, which consists of several components including the retrieval logic, the time-series database (TSDB), and the HTTP server. The retrieval component is responsible for fetching the metrics, the TSDB is where the metrics are stored, and the HTTP server provides an API for queries.

For alerting, Prometheus includes the Alertmanager, which handles alerts sent by the Prometheus server and takes care of deduplicating, grouping, and routing them to the correct receiver, such as an email, PagerDuty, or other notification systems.

Grafana excels not only in data visualization but also in its ability to create comprehensive dashboards that can be customized to the needs of federated learning monitoring. Its user-friendly interface allows system administrators and data scientists to create complex queries without deep knowledge of the datasources's query language. Grafana's dashboards are dynamic and interactive, offering drill-down features that enable users to pinpoint the exact source of issues in real-time. This makes Grafana an indispensable tool for immediate analysis and response [19].

Furthermore, Grafana supports alerts that can notify the team of any anomalies or threshold breaches in the federated learning process. This proactive alerting mechanism ensures that potential issues can be identified and remedied promptly, minimizing downtime and maintaining the continuity of model training across the distributed network [19].

The synergy between Prometheus and Grafana provides a powerful monitoring solution that enhances visibility into the federated learning process, allowing for proactive management of the system's health and performance.

5.2.3 Leveraging MLflow for Tracking

MLflow emerges as an open-source platform designed to streamline the entire machine learning lifecycle. It encapsulates the intricate processes involved in machine learning development into a cohesive system that enhances productivity, fosters collaboration, and maintains consistency across different stages of model training, evaluation, and deployment. In the context of federated learning, MLflow's tracking capabilities are particularly valuable, offering a systematic approach to record and analyze the distributed training process [1].

The MLflow Tracking Server stands out as a centralized repository that meticulously logs every aspect of the machine learning experiments, from parameters and metrics to models and artifacts. This facilitates a comprehensive understanding of each training session, which is paramount in the collaborative and diverse environment of federated learning. By capturing a detailed snapshot of the training iterations, MLflow enables practitioners to monitor, compare, and iterate upon their models with precision and control [1].

The tracking system within MLflow not only documents the quantitative metrics but also provides qualitative insights, thereby enhancing the decision-making process in the iterative development of federated models [1]. This level of detail and traceability is instrumental in validating the models and ensuring their robustness before deployment.

MLflow's suite of tools is adeptly designed to manage and streamline the machine learning lifecycle. In the domain of federated learning, where collaboration and distributed training are key, MLflow's capabilities shine by providing a centralized and scalable approach to experiment logging. This enables practitioners to record, compare, and analyze models across various devices and data distributions.

The reproducibility of experiments is another crucial advantage offered by MLflow. It meticulously logs all aspects of the machine learning process, from initial data processing to final model deployment [1]. This level of documentation is pivotal in federated learning environments, where numerous iterations and model versions are generated across a network of participants.

Moreover, MLflow's modular design affords data scientists and developers the flexibility to customize their workflows. This adaptability is invaluable in federated learning, where diverse computational resources and data privacy considerations necessitate a flexible approach to model training and evaluation.

By leveraging these and other advantages, MLflow positions itself as a fundamental asset in the federated learning toolkit, enabling efficient management and robust development of distributed machine learning models.

5.3 Adaptive Criteria for Managing Device Diversity

In this section, we investigate a comprehensive set of adaptive standards that have been meticulously devised to enhance the optimization of the federated learning system in the midst of device heterogeneity. These standards play a crucial role in ensuring that participation in the system is both efficient and equitable across various devices, and they can be easily managed through a configuration file. Users are afforded the flexibility to exercise control and establish specific thresholds and rates of adjustment for each standard, thereby tailoring the system's response to the unique capabilities and limitations of each participating device. This customization enables a nuanced approach to federated learning, which permits precise adaptation to the varied computational, memory, and network attributes that exist within the learning network.

5.3.1 Leveraging Sparsification for Low Network Bandwidth

Sparsification is a technique in federated learning that targets the reduction of communication overhead, particularly vital for devices with limited network bandwidth. It does this by compressing the data exchanged during training, focusing on transmitting only the most critical parameters and gradients. This method addresses the challenges of slow network speeds and bandwidth constraints, allowing devices to participate effectively in the learning process with reduced data transmission requirements [6].

When network bandwidth drops below a predetermined threshold, the strategy of sparsification is activated. This technique involves calculating a threshold value, below which a specified percentile of the absolute values of the model's weights fall, as set by the user in the configuration file. For instance, if the user sets the percentile to 50% in the configuration file, it signifies that during sparsification, half of the data elements will be masked, effectively setting them to zero. This approach significantly reduces the amount of data needing communication over the network, focusing on transmitting only the most essential model updates. Such a selective method ensures efficient data transmission while maintaining the integrity and effectiveness of the learning process, even under constrained bandwidth conditions.

The adoption of sparsification highlights the tools's versatility in accommodating a range of network environments. It ensures that devices, irrespective of their bandwidth constraints, can contribute effectively to the training process.

5.3.2 Selective Participation based on Similar Device Resources

Clustering similar devices in federated learning is a technique used to group devices with similar computing and communication performance. By doing this we can accelerate the convergence of the federated learning model, especially in scenarios with heterogeneous clients [35].

The strategic implementation of this criterion involves a nuanced assessment of each device's performance metrics, particularly focusing on memory and CPU utilization. By analyzing these metrics, the system can identify devices that exhibit similar utilization levels. The key is to select devices that neither exceed nor fall significantly short of a defined utilization range. For example, if the set range is between 40% to 60% CPU utilization and a similar bracket for memory usage, devices operating within these parameters are selected for participation in the upcoming training rounds.

This approach ensures a fair representation of diverse devices in training rounds, promoting equality in model contribution. It specifically targets the integration of devices that might have been underrepresented in previous rounds due to their resource capabilities, thereby enriching the model with a broader data spectrum. This tactic not only democratizes the learning process but also enhances the model's generalizability by incorporating diverse insights and data patterns.

5.3.3 Adaptive Data Sampling on Memory Utilization

Data sampling in machine learning pertains to the intricate process of choosing a smaller portion of data from an extensive dataset with the intention of employing it for training or analysis objectives. This technique is widely employed in the realm of machine learning in order to mitigate the computational burden that is associated with the execution of machine learning algorithms when confronted with copious amounts of data [28].

The cornerstone of this strategy is the dynamic adjustment of data sample size, fine-tuned according to the real-time memory utilization of each device. This approach is vital for devices with constrained RAM, as it helps them manage their resources more effectively. For instance, if a device's memory usage reaches a user-set threshold, say 70%, the strategy modifies the data sample size for the next round of training. This modification is determined by a user-specified adjustment factor, which could, for example, reduce the sample size by a particular percentage to accommodate the device's memory capacity.

This method of adapting data sampling based on memory utilization plays a crucial role in enhancing the training efficiency of devices with limited memory. It ensures that these devices maintain active participation in the federated learning process, balancing data volumes with their memory capacities, and contributing to a more diverse and effective learning network.

5.3.4 Learning Rate Adjustment for High CPU Utilization

In the domain of neural network training, the learning rate emerges as a pivotal hyperparameter, dictating the pace at which a model adjusts its weights during the learning process. The choice of learning rate holds significant implications for the training dynamics; a smaller rate leads to gradual weight updates requiring extensive training epochs, while a larger rate promises rapid adjustments at the cost of potential stability issues. The intricacies of selecting an optimal learning rate lie in balancing the trade-offs between training stability, speed, and the ability to converge to an effective solution [4].

This strategy advocates for adjusting the learning rate based on the CPU utilization of devices participating in a federated learning network. Recognizing that high CPU utilization reflects

a device's extensive computational demand, the proposal is to increase the learning rate for such devices. This adjustment aims to accelerate the training process, enabling the model to converge in fewer epochs. By doing so, the strategy seeks to optimize the use of computational resources, especially for devices under heavy load, thereby reducing the overall training time without significantly burdening the device further.

For devices with low CPU utilization, the learning rate can be adjusted more conservatively. Since these devices have more available computational capacity, the risk of negatively impacting their performance with a slightly higher learning rate is reduced. However, the primary focus of this adjustment strategy is on devices with high CPU utilization, where managing the learning rate more aggressively can lead to more efficient training sessions.

Implementing learning rate adjustments in response to CPU utilization highlights the adaptive and inclusive nature of our tool. It allows the system to accommodate a wide range of device capabilities, ensuring that all devices, regardless of their current computational load, contribute effectively to the training process.

5.3.5 Epoch Reduction for Devices with High CPU Utilization

In the context of machine learning, epochs play a crucial role in shaping the model's learning trajectory. Each epoch represents a complete cycle where the model is exposed to the entire dataset, making iterative adjustments to its parameters. The number of epochs directly influences the model's ability to generalize or fit the data, with a higher count offering more opportunities for learning, yet also carrying the risk of overfitting beyond a certain point. In scenarios involving complex tasks like computer vision, deep neural networks (DNNs) benefit significantly from multiple epochs, especially when leveraging advanced techniques such as pre-trained architectures and GPU acceleration [3].

In federated learning environments with devices showcasing high CPU utilization, reducing the number of epochs is a strategic approach to balance computational demand and learning efficacy. This adjustment is made in real-time, considering the current CPU load of the devices and predefined user settings. For example, a device operating at 80% CPU utilization may have its training epochs reduced to alleviate the computational burden. Such a reduction is proportionate and ensures that the device remains an active participant in the learning process without being overwhelmed by the computational requirements.

This method of dynamically adjusting epochs based on CPU utilization exemplifies the flexible and inclusive nature of federated learning. It allows for the seamless integration of devices with diverse processing powers into the learning network, ensuring that high CPU utilization does not impede their ability to contribute meaningfully to the model's development. By adapting to the computational environments of individual devices, this strategy enhances the overall efficiency and effectiveness of the learning process.

5.3.6 Adaptive Batch Size Based on Memory Utilization

Batch size, in the context of machine learning, is a fundamental parameter that defines the number of data samples processed in a single iteration during model training. It's a key factor that influences both the speed and stability of the learning process. In federated learning scenarios, particularly those involving a diverse array of devices, managing batch size becomes crucial due to varying memory capacities. Adaptive batch size is a technique tailored to address this variability, ensuring efficient and effective model training across devices with different memory constraints [24].

The strategy focuses on enhancing the capability of devices with restricted RAM to handle the model training workload. By dynamically regulating the training batch size, it aims to match the memory capacity of each device, thus minimizing the risk of memory-related interruptions and ensuring smoother operation. The adaptive mechanism is triggered when a device's RAM usage approaches or exceeds a user-defined threshold. In such cases, the strategy automatically scales down the batch size to a level that is manageable for the device, effectively balancing the training workload with the available memory resources.

This adaptive batch size strategy is crucial for maintaining the continuity and effectiveness of the federated learning process across a spectrum of devices. It allows for greater participation of memory-limited devices in the network, contributing to a more diverse and representative model training process.

5.3.7 Layer Freezing for Devices with High CPU Utilization

Layer freezing in machine learning refers to the technique of selectively freezing certain layers of a deep neural network (DNN) during the training process. This is based on the observation that the training progress of internal DNN layers differs significantly, with front layers often becoming well-trained much earlier than deep layers. By freezing the converged layers, the backward computation and communication associated with them can be skipped, resulting in significant training speedup without sacrificing accuracy [33]. Layer freezing has been shown to be effective in accelerating training for different types of neural networks, including VGG nets, ResNets, and DenseNets [34].

The adaptive layer freezing method involves calculating a certain percentage of the model's layers to freeze, based on user input. Once identified, the strategy entails systematically halting the update process of these layers during training. The freezing process begins with the early layers of the neural network, as these layers tend to converge quicker than the deeper layers [14]. By preventing updates in these initial layers, the computational demand for each training iteration is substantially reduced, thereby alleviating the strain on devices with high CPU usage.

Upon completion of the intensive training phase, the frozen layers can be reactivated or "unfrozen". This reinstates their ability to update weights during subsequent training sessions. Such dynamic freezing and unfreezing of layers based on real-time CPU utilization ensure that the model adapts to the device's current processing capacity, allowing for efficient utilization of computational resources.

This criterion strikes a balance between maintaining model complexity and adapting to the computational realities of diverse devices, thereby enhancing the effective participation of devices with varied computational capabilities.

5.3.8 Gradient Clipping for Devices with Limited Computational Resources

Gradient clipping is a popular technique used in neural networks to limit the norm of the gradient during training. It is commonly employed to stabilize the training process and address issues such as exploding gradients. However, this technique can also be used to reduce computation load in deep learning applications such as large-scale language modeling [29].

To ensure a uniform and efficient training experience, particularly for devices with limited computational resources, the strategy of gradient clipping is employed. This approach involves independently clipping each component of the gradient vector to a pre-defined maximum size, thus reducing the computational intensity of backpropagation. By preventing large updates that can

be computationally expensive, gradient clipping ensures that these devices can handle the gradient computations without undue stress on their processors.

This constraint on gradient size also reduces the likelihood of numerical instability, which can be a significant concern in lower-capacity devices. As a result, even with reduced processing power, these devices can maintain steady progress in training, contributing efficiently to the model's development.

Through this technique, all devices, irrespective of their hardware capabilities, can contribute effectively and efficiently to the training process, thereby enriching the collaborative learning experience and enhancing the overall performance of the federated model.

5.4 Architecture of the Proposed System

In this section, we present the architecture of the proposed federated learning system, which is designed to cater to a heterogeneous environment of client devices. At the heart of our system lies the Client Selector, a sophisticated component responsible for orchestrating the client selection logic which extends the `configure_fit` method intrinsic to the Flower framework (see Figure 5.4). The Client Selector's decision-making process is informed by a comprehensive set of criteria defined within the configuration file, which is meticulously crafted by the Strategy Auditor. This configuration file encapsulates all the strategic criteria necessary to guide the federated learning process.

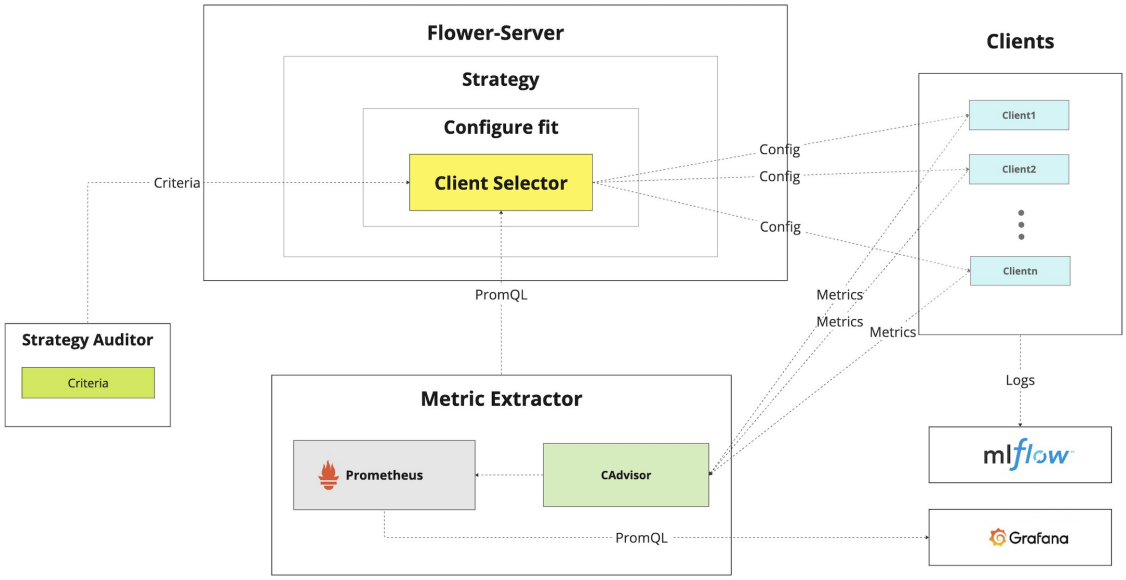

Figure 5.4: The architecture of the proposed tool, highlighting the central role of the Client Selector in coordinating client participation in the learning process.

Further complementing the Client Selector's role is the Metric Extractor, which harnesses the power of PromQL to access and assimilate essential system data from all clients. This data informs

the Client Selector in the tailoring of individualized configuration files that are disseminated to each client for the upcoming training round. These configuration files are instrumental in calibrating each client's training process in accordance with its capabilities and previous performance.

Additionally, our system integrates MLflow for meticulous logging purposes. Each client, through this framework, records various metrics that provide insights into performance, as well as the privacy budget expended in each iteration. This granular logging is vital for monitoring the system's efficacy and ensuring the privacy-preserving aspects of the federated learning process.

As we delve into the subsequent sections, we will explore each component's unique contributions to the architecture, illustrating how they collectively contribute to a federated learning ecosystem that is both resilient and finely tuned to the nuances of device diversity.

5.4.1 Client Selector

The *Client Selector* is a critical component of the federated learning system, designed to dynamically orchestrate the selection and configuration of client devices to optimize the training process. This subsection outlines the operational steps undertaken by the Client Selector:

1. **Client Availability Assessment** The Client Selector identifies all available clients for the specific training round. This step involves assessing the current pool of clients and determining their readiness to participate in the upcoming training cycle.

2. **Criteria Retrieval** Each training round commences with the Client Selector retrieving strategic criteria from the `criteria.yaml` file, crafted by the Strategy Auditor. This file defines each criterion with a unique identifier, thresholds, and adjustment rates for a comprehensive evaluation of client suitability.

3. **Querying Performance Metrics** Working in sync with the Metric Extractor, the Client Selector leverages PromQL to query and retrieve historical performance metrics of clients from the timeframe of the previous training round. This process involves correlating each criterion defined in the `criteria.yaml` file with a specific set of queries. This targeted querying approach allows the Client Selector to accurately assess the operational status of each client, ensuring they align with the strategic selection criteria.

4. **Client Evaluation and Configuration Adaptation** Client Selector engages in a comprehensive assessment of every client, closely examining them based on the established metrics and criteria. In the event that a client fails to meet the standards of a particular criterion, the Selector modifies the associated training parameters in accordance with predetermined adjustment rates. This evaluation not only determines the appropriateness of clients for the present training cycle, but also prompts the generation of personalized configuration files.

5. **Dispatching Fit Configurations** After updating the configuration files, the Client Selector proceeds with dispatching the fit configurations to each selected client. This critical step ensures the communication and application of the personalized training parameters within the clients' training environments. The dispatch of these configurations is a fundamental component of the training cycle, aligning with the operational sequence in the Flower framework's sequence diagram (referenced in Figure 5.1).

This structured approach enables the Client Selector to ensure that the federated learning process is resilient, efficient, and responsive to the varied computational capabilities of the participating devices.

5.4.2 Metric Extractor

The Metric Extractor is an integral component of our architecture, tasked with acquiring and processing performance metrics from client devices. This subsection outlines the functionalities and operations of the Metric Extractor within the system:

1. **Data Collection and Storage** The Metric Extractor capitalizes on Prometheus that collects and stores data in a time series format. Prometheus continuously gathers a vast array of performance metrics from each client device, ensuring a rich dataset that accurately reflects the operational state and capabilities of the clients over time.

2. **Query Generation and Execution** In synergy with the Client Selector, the Metric Extractor plays a pivotal role in formulating and executing queries to extract relevant performance metrics. The Client Selector, equipped with strategic criteria from the `criteria.yaml` file, generates specific PromQL queries tailored to assess each client's performance based on historical data.

3. **Prometheus API Interaction** Upon generation of these queries, the Metric Extractor interfaces with the Prometheus API to retrieve the required data. It sends the PromQL queries to Prometheus, which then processes these queries and returns the corresponding results. This interaction facilitates the acquisition of precise and timely data for each individual client, essential for informed decision-making in the client selection process.

Through these structured operations, the Metric Extractor establishes itself as a crucial link in the architecture, providing the necessary data foundation for optimizing client selection and configuration, thereby contributing to a resilient and adaptive learning network.

5.4.3 Strategy Auditor

The Strategy Auditor plays a pivotal role, particularly in shaping the selection and optimization strategies for client devices. This subsection details the various aspects and operations of the Strategy Auditor:

1. **Criteria Identification and Justification** The Strategy Auditor leverages Grafana dashboards and MLflow logs to identify the bottlenecks in each client's system (such as CPU, memory, and network bandwidth). By analyzing these metrics, the auditor gains insights into which system parameters critically impact the performance and reliability of client devices. This analysis guides the Strategy Auditor in selecting relevant criteria from `criteria.yaml` file, ensuring that the chosen metrics align with the overall objectives of the federated learning system and address specific bottlenecks.

2. **Threshold Setting and Adjustment Factors** The initial thresholds for each criterion in the `criteria.yaml` file can be determined based on a combination of past data, expert input, and specific system requirements. The Strategy Auditor sets these thresholds to define the acceptable performance range for each criterion. Alongside, the auditor defines adjustment factors, which are crucial in fine-tuning client participation in subsequent training rounds. These factors allow for the dynamic adjustment of thresholds based on the evolving conditions and performance of the federated learning network.

Through these activities, the Strategy Auditor significantly enhances the system's efficiency and adaptability. The continual refinement of criteria and thresholds ensures the federated learning network remains responsive and aligned with the diverse capabilities of participating devices.

5.5 Implementation Methodology

5.5.1 Development Technologies

This subsection outlines the core technologies utilized in the implementation. Each technology was selected for its specific strengths and capabilities, contributing significantly to the system's robustness and efficiency.

Docker and Docker Compose

Docker, a widely-adopted containerization platform, is a cornerstone of the development environment for the federated learning tool. At its core, Docker allows the creation of containers, which are lightweight, standalone, and executable packages. These containers encapsulate software, libraries, dependencies, and runtime environments, ensuring applications run consistently across different computing environments.

Containers, as facilitated by Docker, offer several advantages for the project:

- **Isolation** Each container operates independently, providing a segregated environment for individual components of the federated learning system. This isolation reduces conflicts between components and enhances the system's reliability.
- **Portability** Containers can be easily transferred and deployed across various systems, from development to production, ensuring consistency in behavior regardless of the underlying infrastructure.
- **Efficiency** Containers are more resource-efficient than traditional virtual machines, as they share the host system's kernel and do not require a separate operating system for each instance.

Docker Compose, an extension of Docker, is utilized to define and run multi-container Docker applications. The 'docker-compose.yml' file, a key component of the project, specifies the configuration of services, networks, and volumes. This orchestration tool simplifies the process of launching, linking, and managing multiple containers, which is essential for handling the various components of the federated learning system, such as client nodes, server, and monitoring tools.

By leveraging Docker and Docker Compose, the project achieves a harmonized and scalable architecture. This setup allows for streamlined development, testing, and deployment phases, ensuring the federated learning tool operates effectively across diverse environments [11].

Python

The implementation of the tool heavily relies on Python as the primary programming language. Python is renowned for its simplicity, readability, and a vast ecosystem of libraries and tools. The project leverages Python's extensive range of libraries and packages available through the Python Package Index (PyPI), facilitating the integration of advanced functionalities and third-party modules with ease.

All our key technologies are easily accessible via PyPI, ensuring smooth installation and compatibility with the Python environment specified in the Dockerfile. This approach allows for streamlined deployment and execution of the federated learning system across various infrastructures. The utilization of PyPI also simplifies the management of dependencies and version control, making the development process more efficient and reliable.

TensorFlow

TensorFlow, an esteemed open-source machine learning library developed by Google, is integral to our federated learning tool. Known for its versatility and extensive library, TensorFlow supports a wide range of machine learning and deep learning applications [31].

The utilization of TensorFlow in this project is characterized by:

- **Model Building and Training** TensorFlow's comprehensive suite of tools and functions facilitates the construction and training of complex deep learning models. Its flexible structure allows seamless computation across various platforms, including CPUs and GPUs, essential for diverse client devices in federated learning [31].
- **Keras API** Alongside TensorFlow, Keras, a high-level neural networks API, is used for designing and experimenting with deep learning models. Keras simplifies the process of creating complex neural network architectures, making it an ideal choice for rapidly developing and iterating on models in a federated learning context.
- **TensorFlow Privacy** To uphold the privacy-centric nature of federated learning, TensorFlow Privacy, an extension of TensorFlow, is employed. This library offers mechanisms for training machine learning models with differential privacy, a crucial aspect in maintaining user privacy in federated settings [32].

TensorFlow serves as the backbone of our tool, providing robust support for model development, training, and privacy assurance, thereby playing a pivotal role in the tool's functionality and effectiveness.

cAdvisor

cAdvisor (Container Advisor) plays a pivotal role in our federated learning tool, serving as the primary source for monitoring container performance. As a daemon process, it specializes in gathering, processing, aggregating, and exporting comprehensive information about running containers. This functionality is key to understanding and managing the dynamic nature of containers in a federated learning environment.

cAdvisor meticulously tracks a wide array of metrics for each container, such as resource isolation parameters, historical resource usage, network statistics, and more. These metrics are critical for providing deep insights into the behavior and performance of each container within our system. With native support for Docker containers and compatibility with other container types, cAdvisor stands as a versatile and essential tool for our containerized setup.

A notable feature of cAdvisor is its hierarchical container abstraction, allowing for nested and detailed structuring of containers. This design enhances the monitoring capabilities, offering a refined granularity in data analysis and system evaluation.

In our federated learning tool, cAdvisor acts as the foundational element for acquiring source data from each container. By integrating cAdvisor, we ensure a robust mechanism for tracking and evaluating container performance. This, in turn, significantly aids in optimizing resource allocation and enhancing the overall efficiency of our system [15].

Prometheus

Prometheus stands as a cornerstone within the architecture of our federated learning system, leveraging its sophisticated time-series database (TSDB) and the powerful PromQL query language.

This combination enables Prometheus to perform complex data handling and analysis, which is fundamental to the efficacy of our federated learning tool.

- **Time-Series Database (TSDB)** Central to Prometheus's functionality is its time-series database, a specialized storage system optimized for handling time-stamped data sequences. Time-series data is vital in monitoring contexts, where tracking and analyzing changes over time is critical. The TSDB within Prometheus is designed for high write and read throughput, ensuring consistent performance even when dealing with large-scale data.

- **PromQL** PromQL, the Prometheus Query Language, is an integral feature of Prometheus, offering advanced capabilities for querying time-series data. This functional query language enables users to intricately select and aggregate time series data in real-time, which is crucial for the dynamic environment of federated learning. PromQL facilitates comprehensive analysis by allowing the system to derive detailed insights and identify trends from the performance metrics of participating devices. Its versatility extends to presenting results in various formats, including graphical visualizations, tabular data in Prometheus's expression browser, and external systems via the HTTP API. This multi-faceted approach provided by PromQL enhances the system's ability to monitor, analyze, and adapt to the needs of a federated learning network effectively.

Together, the time-series database and PromQL of Prometheus significantly contribute to the system's capability to monitor, analyze, and adaptively respond to the complex dynamics of the federated learning network [27].

5.5.2 Implementation Details

System Set Up and Configurations

The implementation of the system is a multi-stage process that begins with the set up of the Docker environment and configuration files for each service.

Docker Compose Configuration The Docker Compose file orchestrates the various services required for the federated learning system, ensuring each component operates efficiently and in isolation. Notably, all services except for the client service utilize official Docker images, guaranteeing reliability and standardization across the system's infrastructure.

Client Services Each client is encapsulated in its dedicated Docker service. These services are based on custom Docker images that include a Python environment with all the necessary dependencies pre-installed. This setup ensures that each client node can operate independently, with its computational resources and environment isolated from others. The Docker Compose file specifies these services, defining the build context (pointing to the Dockerfile) and any volumes for persistent storage or configuration files that the client might need.

cAdvisor cAdvisor is deployed to collect detailed container metrics, such as CPU, memory usage, and network IO. It runs in privileged mode to have access to all containers' metrics, with specific volumes mounted to read the host's Docker socket and system directories. This allows cAdvisor to provide comprehensive insights into the resource usage of each container.

Prometheus It is configured with a volume that mounts its configuration file, defining the scraping targets and intervals. This setup enables Prometheus to store and query time-series data efficiently, serving as the backbone for system monitoring.

Grafana It is configured to persist its data and dashboards through Docker volumes, ensuring that any customizations or created dashboards remain intact across service restarts. Grafana's dependency on Prometheus is explicitly defined to ensure it starts only after Prometheus is up and running, facilitating seamless data visualization from the get-go.

MLflow Server The MLflow Server service centralizes the tracking of machine learning experiments, including parameters, metrics, and model artifacts. It is set up with volumes for persistent storage, ensuring that experiment data is not lost between restarts.

Dockerfile The Dockerfile provided here serves as the blueprint for building a consistent environment for our client-server services.

Listing 5.1: Dockerfile for Federated Learning System

```
FROM python:3.9-slim-buster
WORKDIR /app
COPY ./ /app
RUN apt-get update && apt-get install -y \
    gcc \
    libgomp1 \
    python3-dev && \
    rm -rf /var/lib/apt/lists/*
RUN pip install --upgrade pip -r requirements.txt
```

It starts with a minimal Python 3.9 base image, sets the working directory to /app, and copies application files into the container. Key dependencies like gcc, libgomp1, and python3-dev are installed, and Python packages listed in requirements.txt are added. This Dockerfile ensures that both server and client components of our federated learning system operate in a uniform environment, promoting compatibility and optimal performance.

Prometheus and Grafana Configuration This section explores the configuration files for Prometheus and Grafana, crucial for monitoring the federated learning system.

Prometheus This file configures Prometheus for metric collection:

Listing 5.2: Prometheus Configuration

```
global:
  scrape_interval:     1s
  evaluation_interval: 1s

rule_files:
scrape_configs:
  - job_name: 'cadvisor'
    scrape_interval: 1s
    metrics_path: '/metrics'
    static_configs:
      - targets: ['cadvisor:8080']
```

```
        labels:
          group: 'cadvisor'
  - job_name: 'server_metrics'
    scrape_interval: 1s
    metrics_path: '/metrics'
    static_configs:
      - targets: ['server:8000']
```

This file specifies two primary components: global configurations and scrape configurations.

- **Global Configuration** These settings apply universally to the Prometheus instance. Here, both the `scrape_interval` and `evaluation_interval` are set to 1 second. This implies that Prometheus will query or "scrape" metrics from its targets every second and evaluate rules at the same frequency.
- **Scrape Configurations** This section defines the specific metrics collection jobs. Each `job_name` represents a set of metrics to scrape.
  - *cAdvisor Job:* This job scrapes metrics from a container advisor service, typically running at the specified endpoint `cadvisor:8080`. These metrics are vital for monitoring and understanding container performance and resource usage.
  - *Server Metrics Job:* Similarly, the server metrics job is configured to scrape data from an endpoint identified as `server:8000`. This could be any server configured to expose Prometheus metrics, and is crucial for gaining insights into server performance.

Grafana's Datasource This file integrates Prometheus as a datasource for Grafana:

Listing 5.3: Prometheus Datasource for Grafana

```
apiVersion: 1

datasources:
- name: Prometheus
  type: prometheus
  access: proxy
  uid: db69454e-e558-479e-b4fc-80db52bf91da
  url: http://host.docker.internal:9090
  isDefault: true
```

The above file provides the necessary identification, connectivity, and configuration details that enable Grafana to seamlessly interact with Prometheus. In essence, they act as the bridge that allows Grafana to access, visualize, and analyze data from Prometheus.

Integration with Flower Framework

Client The client script (`client.py`) is responsible for the local training of models and communication with the FL server. It defines a `Client` class extending the `fl.client.NumPyClient` from the Flower framework, which implements methods like `get_parameters`, `fit`, and `evaluate` to interact with the server. The client retrieves training and evaluation tasks from the server, performs computations, and sends back the results.

Server The server script (`server.py`) is central to orchestrating the federated learning process. It initializes the FL server and manages client interactions. Crucial functions within this script include `configure_fit`, `aggregate_fit`, `configure_evaluate`, and `aggregate_evaluate`. These functions are pivotal in determining how clients are chosen for training and evaluation, how their results are combined, and how the overall evaluation of the federated model is conducted. The strategy, customized for this project, employs the Federated Averaging (FedAvg) algorithm as conceptualized by McMahan et al.[25], adapting it to meet specific system requirements and ensuring efficient and tailored federated learning operations.

Establishing Connections In the federated learning system, the server and client scripts are crucial for establishing connections via the Flower framework. The server initiates a listening endpoint for federated learning communication, while the client connects to this server endpoint. The following code snippets illustrate these processes.

Listing 5.4: Server Connection Initialization

```
fl.server.start_server(
    server_address=SERVER_ADDRESS,
    config=fl.server.ServerConfig(num_rounds=NUM_ROUNDS),
    strategy=STRATEGY
)
```

Listing 5.5: Client Connection Initialization

```
fl.client.start_numpy_client(
    server_address=SERVER_ADDRESS,
    client=CLIENT_INSTANCE
)
```

Query Construction and Analysis

In this section, we delve into the construction and analysis of queries, which are pivotal for extracting and processing data in our environment. These queries play a critical role in gathering device metrics from each client, enabling us to evaluate their performance and determine optimal configurations for the upcoming federated round. It's noteworthy that the metrics utilized in the following queries are efficiently exposed through cAdvisor [9].

Container CPU Usage Percentage The query for calculating the CPU usage percentage of a container over a specified time period is constructed as follows:

Listing 5.6: CPU Utilization Query

```
sum(rate(container_cpu_usage_seconds_total{name="CONTAINER_NAME"}[
   DURATIONs] @{END_TIMESTAMP})) /
sum(container_spec_cpu_quota{name="CONTAINER_NAME"} /
   container_spec_cpu_period{name="CONTAINER_NAME"}) * 100
```

This query computes the average CPU usage of the specified container, normalized by the CPU quota and period, to provide a percentage value. The key metrics used in this query are:

- `container_cpu_usage_seconds_total`: A counter metric representing the cumulative CPU time consumed by the container, measured in seconds.

- `container_spec_cpu_quota`: A gauge metric that specifies the CPU quota for the container. This metric represents the maximum amount of CPU resources that the container is allowed to use, often defined in terms of CPU units or time slices.
- `container_spec_cpu_period`: A gauge metric indicating the CPU period of the container. It defines the time frame for measuring CPU usage, enabling the calculation of CPU resource allocation in relation to the quota. Essentially, it helps in managing and restricting the CPU cycles a container can utilize within a specific period.

Container Memory Usage Percentage The query to determine the memory usage percentage of a container over a defined time interval is outlined below:

Listing 5.7: Memory Utilization Query

```
avg_over_time(container_memory_working_set_bytes{name="CONTAINER_NAME"}[
    DURATIONs] @{END_TIMESTAMP}) /
avg_over_time(container_spec_memory_limit_bytes{name="CONTAINER_NAME"}[
    DURATIONs] @{END_TIMESTAMP}) * 100
```

This query calculates the average memory usage of the specified container, normalized by the container's memory limit, yielding a percentage value. The primary metrics involved in this query are:

- `container_memory_working_set_bytes`: A gauge metric that indicates the current memory usage of the container.
- `container_spec_memory_limit_bytes`: A gauge metric representing the maximum allowable memory usage for the container.

Container Incoming Bandwidth Query The query for determining the incoming bandwidth of a container over a specified time interval is as follows:

Listing 5.8: Incoming Bandwidth Query

```
rate(container_network_receive_bytes_total{name="CONTAINER_NAME",
    interface="NETWORK_INTERFACE"}[DURATIONs] @{END_TIMESTAMP}) / 1024^2
```

This query measures the rate of incoming network traffic to a specific container, represented as bandwidth in megabytes per second. The key metrics and parameters used in this query include:

- `container_network_receive_bytes_total`: A counter metric that tracks the total bytes received by the container.
- `NETWORK_INTERFACE`: The network interface on the container (default is 'eth0').

Container Outgoing Bandwidth Query The query to determine the outgoing bandwidth of a container within a specific time interval is as follows:

Listing 5.9: Outgoing Bandwidth Query

```
rate(container_network_transmit_bytes_total{name="CONTAINER_NAME",
    interface="NETWORK_INTERFACE"}[DURATIONs] @{END_TIMESTAMP}) / 1024^2
```

This query calculates the rate of outgoing network traffic from a given container, expressed in megabytes per second. The key components of this query include:

- `container_network_transmit_bytes_total`: A counter metric that measures the total bytes transmitted by the container.
- `NETWORK_INTERFACE`: Specifies the network interface of the container (default 'eth0').

Each query utilizes the `@` modifier to ensure a consistent and accurate evaluation across the specified time range. The `CONTAINER_NAME` variable denotes the name of the container, `DURATION` specifies the time window in seconds, and `END_TIMESTAMP` is the end timestamp of the query interval.

Client Selector

The Client Selector stands as an intelligent entity within our system, engineered to dynamically orchestrate the inclusion of clients in the training process, based on a suite of real-time performance metrics alongside predefined selection criteria.

This mechanism springs into action with the commencement of each training cycle, where the `configure_fit` method within the strategy module is invoked. This action sets the stage for the Client Selector's in-depth evaluation and filtration process. Through a sophisticated interaction with the Prometheus monitoring system, the Client Selector efficiently evaluates each potential participant against a series of carefully delineated benchmarks.

In preparation for this selection ballet, the system preloads criteria from the `criteria.yaml` configuration, clearly demarcating the non-negotiable (blocking) and negotiable (non-blocking) conditions for client participation. Engaging with the `PrometheusService`, the `ClientSelector` meticulously gathers and parses metrics for each client, utilizing the robust PromQL querying language to extract the necessary data.

Upon fulfilling all blocking conditions, clients are then meticulously appraised against the non-blocking criteria, a step that ensures each participant is configured to contribute optimally to the model's training. It is this rigorous evaluative mechanism rooted in a comprehensive matrix of checks and balances that assures each training round is not only tailored but also runs at peak efficiency, mirroring the Client Selector's discerning nature.

The decision matrix culminates in the judicious finalization of client selection, which may also involve strategic configuration refinements. These bespoke adjustments, informed by the subtle variances in performance metrics, are designed to maximize each client's input into the federated learning process.

The strategic integration and operational influence of the Client Selector on the training protocol are visually encapsulated in the ensuing diagram. This visual representation offers a clear depiction of the Client Selector's methodical workflow, showcasing its pivotal role in our federated learning tool.

For a more detailed overview of the `ClientSelector`'s integration within the federated learning system, an Entity-Relationship Diagram (ERD) is provided. This diagram delves into the complex interactions between the `ClientSelector` and associated components, depicting the systematic flow from criteria definition to client evaluation.

This diagram reinforces how the `ClientSelector`, through a series of methodical processes, leverages the `Criteria` defined in a yaml configuration file. These criteria are mapped to specific Prometheus queries by the `Mappings` module, forming the basis for real-time client performance assessments.

This visualization is particularly instructive in illustrating how new criteria can be seamlessly integrated into the system, enhancing the `ClientSelector`'s functionality and adaptability. It demonstrates the underlying modular architecture, which facilitates the customization and expansion of the client selection criteria and their associated queries and evaluation logic.

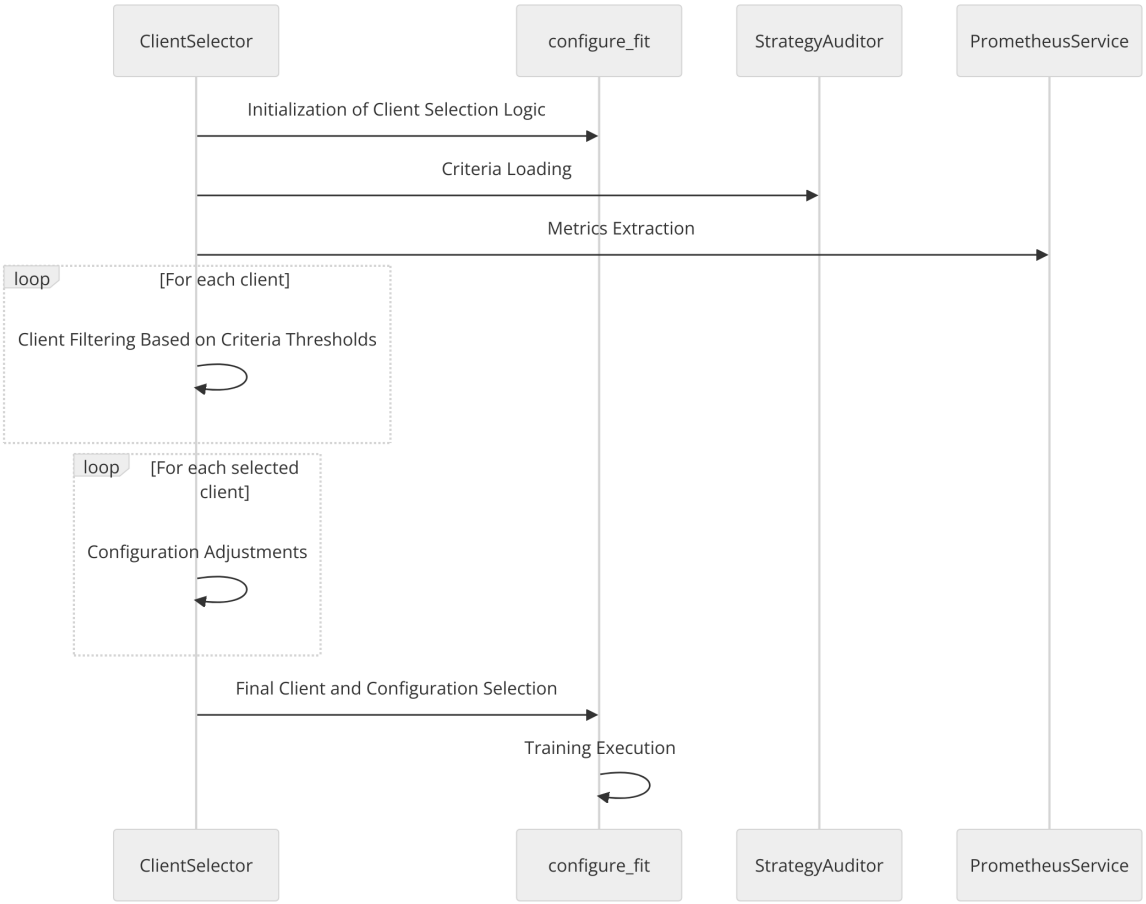

Figure 5.5: The workflow of the Client Selector within the Federated Learning System.

By following the depicted relationships and processes, one can grasp the extensible nature of the system and the straightforward approach to adding or modifying selection criteria and their corresponding query functions. The ERD is shown below.

Metric Extractor

The Metric Extractor is a crucial component within our federated learning system, designed to interface seamlessly with the underlying monitoring infrastructure. Its primary role is to extract and process client's performance metrics, which are essential in informing the Client Selector's decision-making mechanism.

Upon activation, the Metric Extractor systematically scrapes the `cadvisor` for each client's performance data. This process is facilitated by the `PrometheusService`, which is configured to scrape these metrics at regular intervals. The `PrometheusService` then employs a sophisticated set of `PromQL` queries, curated within the `Prometheus Queries` submodule, to extract the necessary data points from the monitored clients.

The extracted metrics are subsequently stored in the Prometheus Time Series Database (TSDB), where they are processed through HTTP queries. These queries are the linchpins that enable real-time, performance-based decision-making. They ensure that the system has access to the most

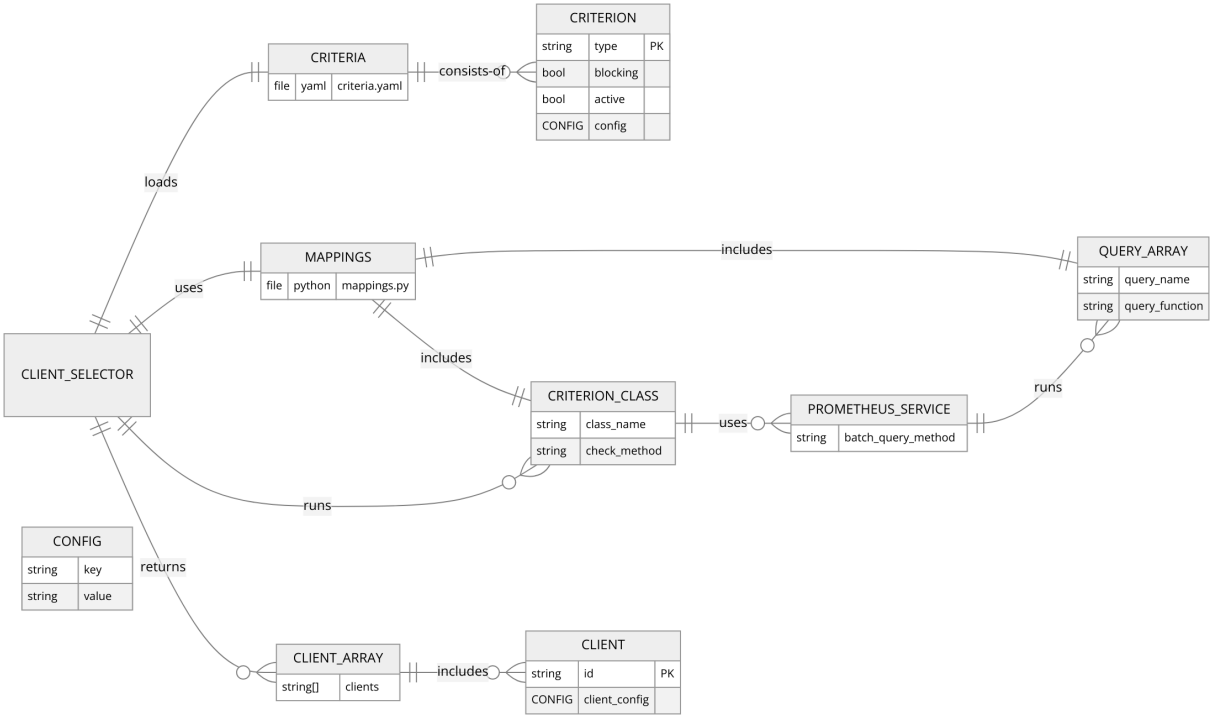

Figure 5.6: Entity-Relationship Diagram showing the Client Selector's interaction with system components.

current and relevant data, thereby allowing the Client Selector to make informed choices about which clients to include in the training round.

The Metric Extractor's operation is not only about data retrieval but also about ensuring data quality and relevance. It performs a critical filtration role, ensuring that only the most pertinent and accurate metrics feed into the subsequent stages of client evaluation and selection.

The following diagram illustrates the Metric Extractor's position within our system, highlighting its interactions with the `PrometheusService`, the `cadvisor`, and the Prometheus TSDB. This diagram underscores the Metric Extractor's pivotal role in bridging the gap between raw performance data and actionable insights for client selection.

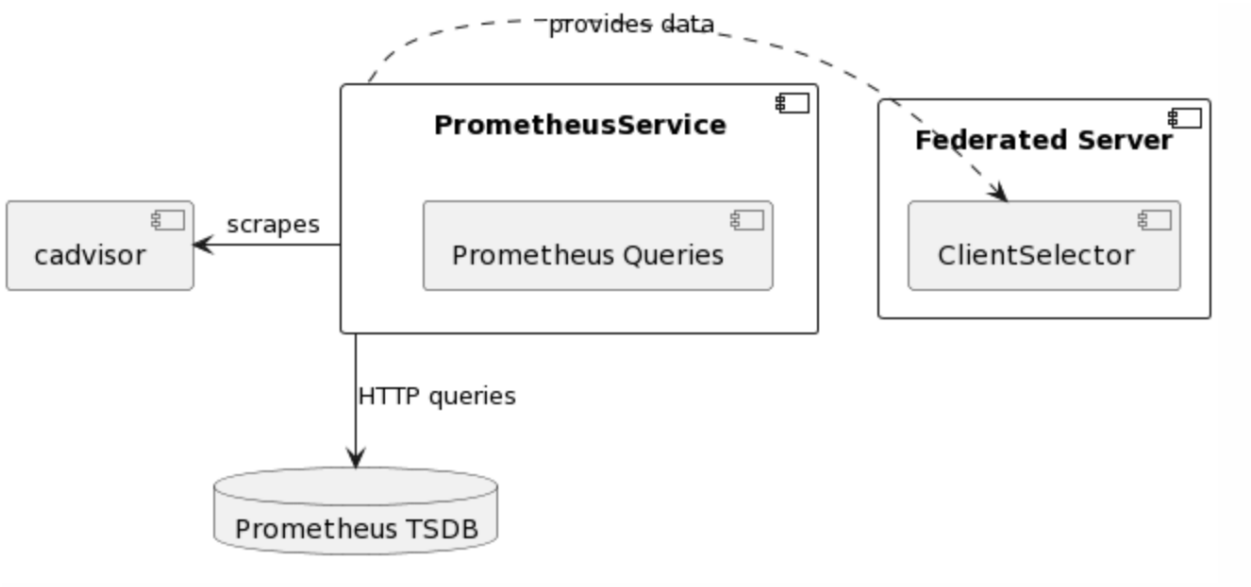

Figure 5.7: The workflow of the Metric Extractor within the Federated Learning System.

Chapter 6

Experiment Results

In this section, we present the results of our experiments designed to evaluate the performance of our tool under two critical scenarios: adapting to Client Capabilities for Optimal Global Accuracy and reducing training time for stragglers. These experiments are pivotal in showcasing the tool's capability to improve global accuracy and efficiency in federated learning environments.

6.1 Enhanced Global Accuracy

In a federated learning scenario where client devices vary significantly in their computational capabilities, not utilizing an adaptive approach may lead to Out-Of-Memory (OOM) errors on straggler devices. This limitation can prevent these clients from contributing their data to the model training process, hindering the achievement of optimal global accuracy. By employing an adaptive mechanism designed to accommodate the varying resources of client devices, we can ensure that all clients contribute to the fullest extent possible, thereby maximizing the overall model accuracy.

6.1.1 Experimental Setup

- **Client Configuration** The experimental framework comprised five clients, strategically diversified in computational and memory capabilities to mirror real-world federated learning conditions. Among them, two clients stood out as high-capacity devices:
  - **Client 1** categorized as powerful, was allocated **4 CPU cores** and **6 GB of memory**.
  - **Client 4**, another robust device, received **2 CPU cores** and **4 GB of memory**.

  The remaining clients, identified as stragglers due to their constrained resources, included:
  - **Clients 2, 3, and 5** each were limited to **1 CPU core** and **1.4 GB of memory**, reflecting the typical computational limitations encountered in diverse federated learning ecosystems. This varied setup, achieved through precise Docker configurations, offered a realistic simulation of the heterogeneity inherent in federated learning networks.
- **Dataset** The CIFAR-10 dataset, consisting of 60,000 32x32 color images in 10 classes, with 6,000 images per class, was used. The dataset was split 80-20 for training and test, respectively. To highlight the importance of utilizing all clients' data, we partitioned the dataset among the five clients. The two powerful clients were allocated images from five classes, and

the remaining three stragglers received images from the other five classes, ensuring that each client's data was crucial for comprehensive learning.

- **Model** We chose MobileNetV2 for its lightweight architecture, depth, and proficiency in classification tasks. Its design, featuring a significant number of layers(105)[2] but requiring fewer computational resources than more extensive networks, makes it an ideal candidate for demonstrating the tool's capability to facilitate training on devices with varied computational powers.

- **Federated Configuration** For this experiment we executed 100 rounds without encountering any round-timeouts and we required a minimum of two clients for both training and evaluation phases. Across 10 experiments—five without and five with our specialized tool—we leveraged the median results to accurately plot the global model's performance.

6.1.2 Results

Client-Specific Parameter Adaptivity Figures 6.1 and 6.2 showcase the parameter adaptivity for a high-capacity client and a straggler, respectively. The adaptivity mechanism effectively adjusts the learning parameters in real-time, enhancing the model's training progress across heterogeneous devices.

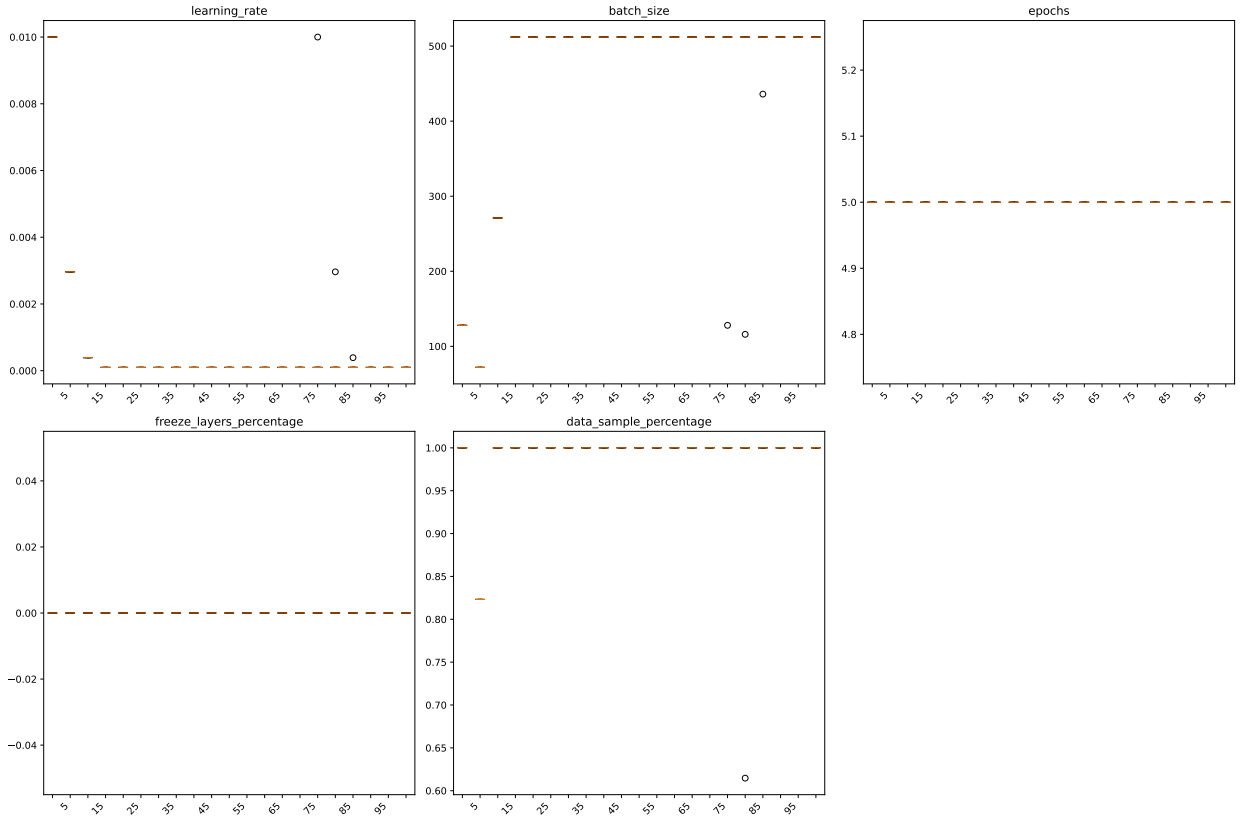

Figure 6.1: Parameter adaptivity for a high-capacity client (Client 1).

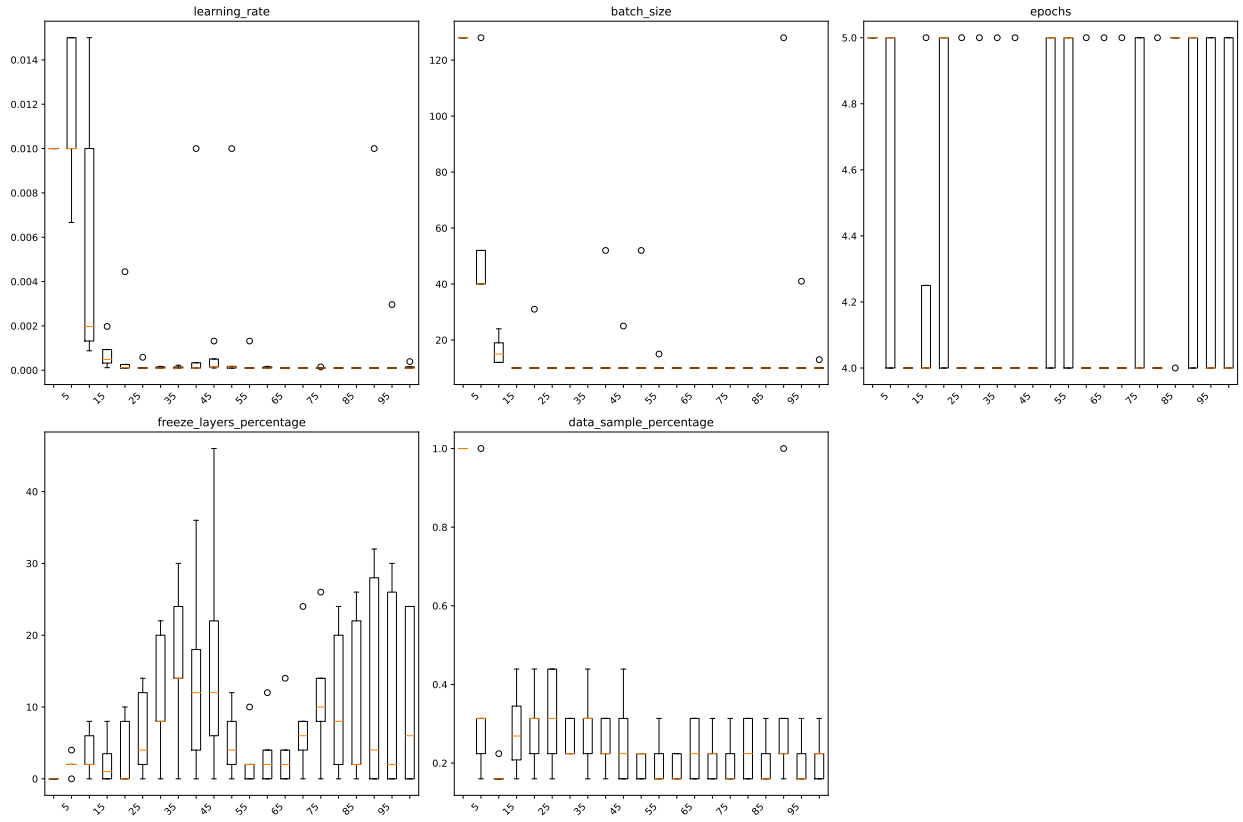

Figure 6.2: Evolution of parameter adaptivity for a straggler (Client 2) across multiple rounds.

For high-capacity clients, such as Client 1, parameter adjustments are minimal due to their ample computational resources. In contrast, stragglers like Client 2 undergo frequent parameter adaptations to mitigate their limited capabilities, ensuring their effective participation in the learning process.

Global Accuracy and Client Dropout Comparison The pivotal advantage of implementing our mechanism is discernibly showcased in the enhancement of global model accuracy and a notable reduction in client dropout rates, as depicted in Figures 6.4 and 6.3. Without our tool, the dropout rate was significantly high due to straggler clients encountering resource constraints, adversely affecting the model's generalizability. However, with our tool, even though complete data utilization from stragglers was constrained, we adeptly optimized each client's contribution, culminating in a marked improvement in global accuracy. This underscores our tool's capacity to not just mitigate dropout rates but also significantly bolster the training inclusivity and global accuracy of the model.

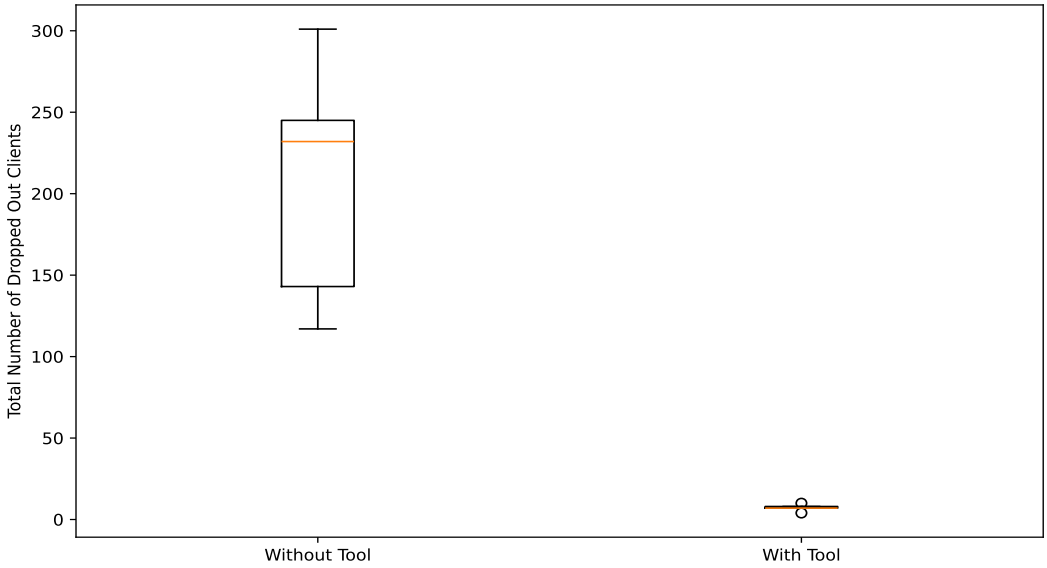

Figure 6.3: Client dropout comparison.

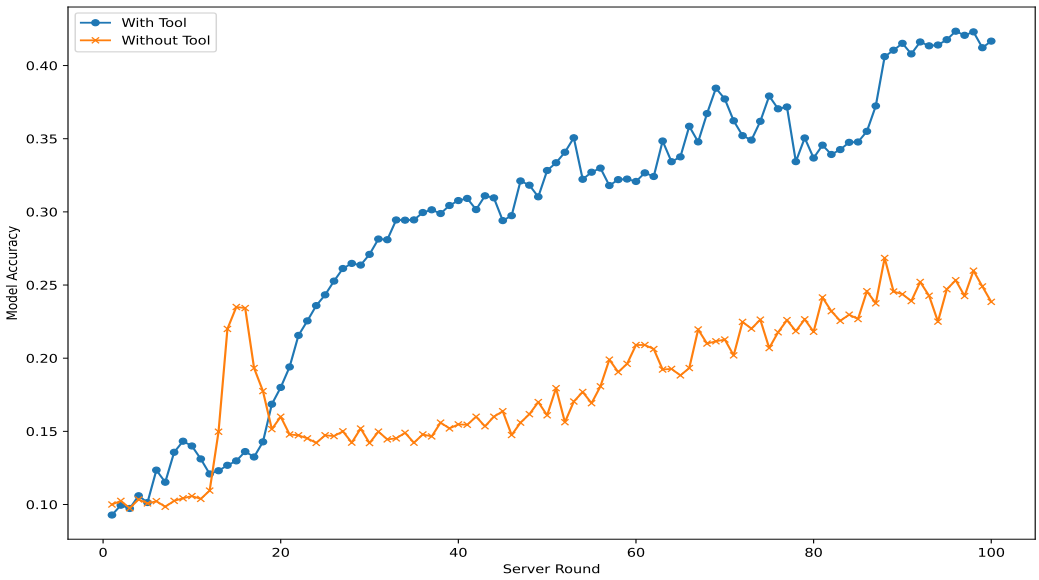

Figure 6.4: Comparison of global model accuracy.

6.2 Reducing Total Training Time

In federated learning, addressing the challenge of stragglers—clients with constrained resources that significantly extend the training duration—is crucial for streamlining the overall learning process. Our experiments demonstrate the impact of our tool in mitigating stragglers' extended training times, thereby reducing the total training duration across the network.

6.2.1 Experimental Setup

- **Client Configuration** The experimental framework comprised four clients, strategically diversified in computational and memory capabilities to mirror real-world federated learning conditions.
  - **Client 1** was equipped with **4 CPU cores** and **6 GB of memory**, making it the most capable device in the experiment.
  - **Client 4** received **2 CPU cores** and **3 GB of memory**, positioning it as another high-capacity client, albeit with slightly lower resources than Client 1.
  - **Clients 2 and 3** were each provided with **1 CPU core** and **2 GB of memory**, considered as the stragglers of the network.

- **Dataset** The dataset was split equally among all four clients, ensuring that each held an identical subset. This equitable distribution aimed to eliminate data quantity and diversity as variables, focusing solely on assessing the tool's effectiveness in computational efficiency and training time reduction.

- **Federated Configuration** The federated learning setup was designed to unfold over 10 rounds, with a keen focus on examining the effects of network stragglers on the learning process's duration. To thoroughly investigate the impact of our optimization tool on training efficiency, we conducted a series of 10 experiments, divided equally between scenarios with the tool enabled and without it. This approach aimed to provide a clear comparison of training dynamics under varying conditions and to determine the tool's capability in alleviating delays caused by slower participants.

6.2.2 Results

Client-Specific Parameter Adaptivity The adaptivity of training parameters for high-capacity clients and stragglers further underscores the tool's efficacy. Figures 6.5 and 6.6 reveal how the tool dynamically adjusted parameters to suit each client's capabilities. High-capacity clients, such as Client 1, experienced no parameter adjustments due to their robust resources. Conversely, stragglers, represented by Client 2, saw more frequent adaptations to ensure their effective participation in the learning process without slowing down the overall training duration.

The resource utilization graphs, detailing CPU usage percentage and average memory usage, lend further credence to the strategic adaptivity of training parameters. For high-capacity clients, the relatively modest resource utilization depicted in Figure 6.7 suggests that substantial adjustments to training parameters are unnecessary. On the other hand, stragglers demonstrate elevated resource utilization, as evidenced in Figure 6.8. This discrepancy underscores the necessity for more agile parameter adaptations to prevent overburdening straggler devices and to sustain an efficient training rhythm across the network.

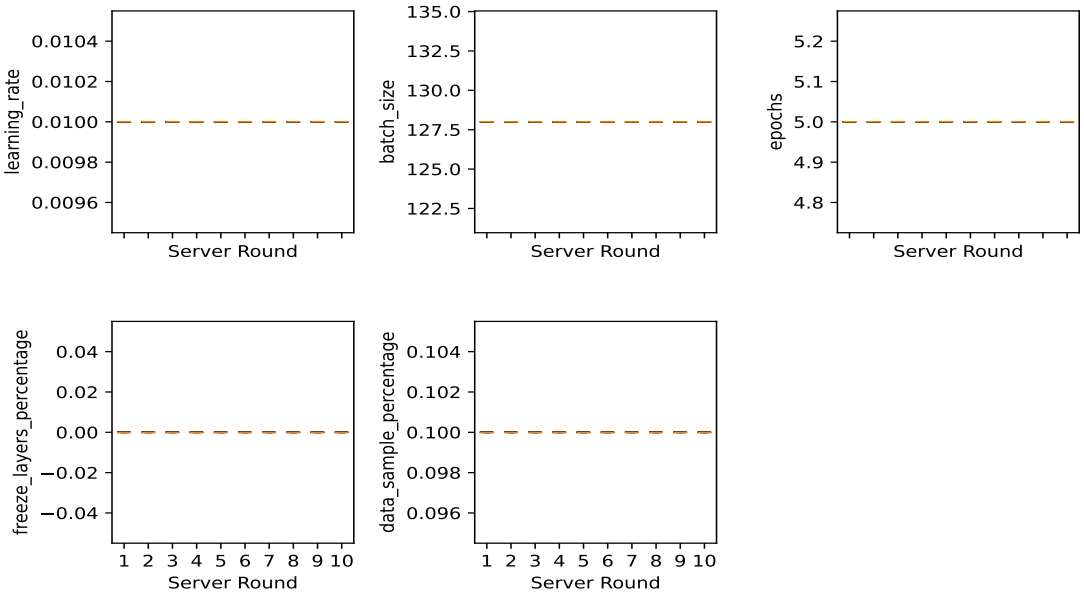

Figure 6.5: Parameter adaptivity observed in a high-capacity client during the training process.

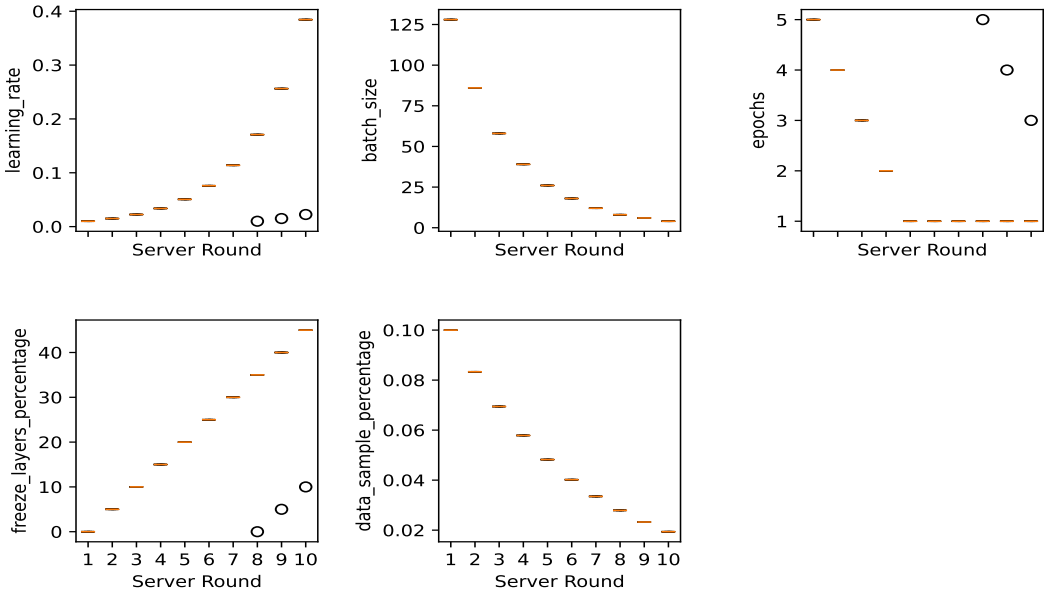

Figure 6.6: Parameter adaptivity in a straggler during the training process.

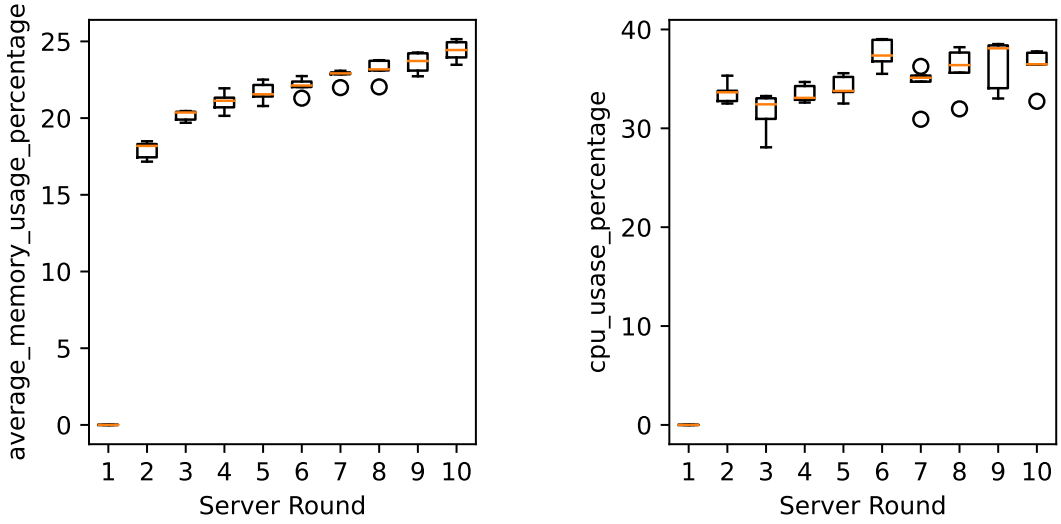

Figure 6.7: System resource utilization for a high-capacity client.

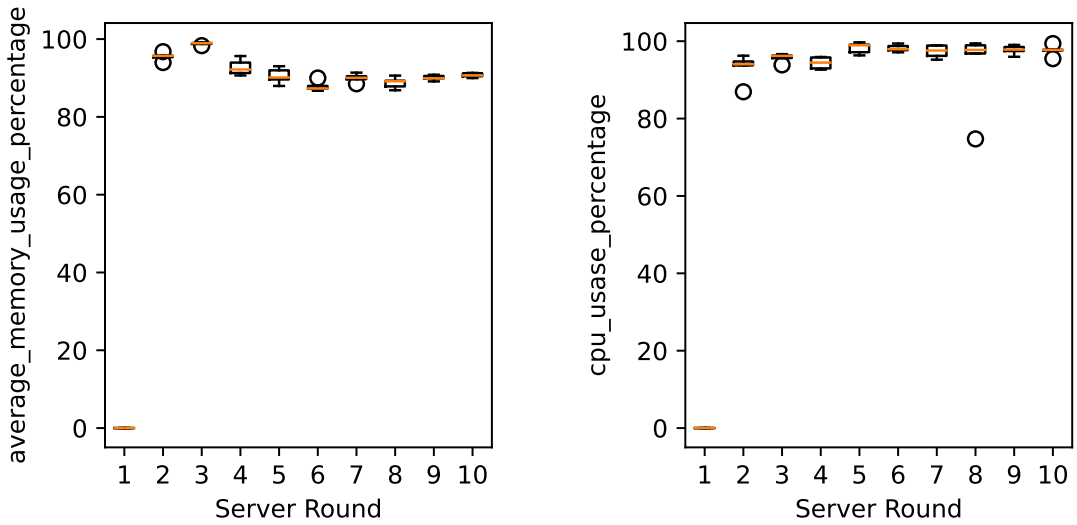

Figure 6.8: System resource utilization for a straggler.

Total Training Time Comparison The analysis as revealed in the 6.9 shows a tangible decrease in total training time, validating the tool's utility in optimizing the involvement of stragglers and thereby streamlining the overall learning process. The experiments consistently showed improved training time efficiency when the tool was employed, as compared to the baseline without

the tool. A box plot crafted from these findings effectively underscores the significant reduction in training durations afforded by the tool, highlighting its critical contribution to making federated learning more scalable and performance-oriented.

Figure 6.9: Comparison of total training times.

Chapter 7

Conclusion

This thesis has presented a comprehensive investigation into the optimization of federated learning environments, focusing on two critical aspects: enhancing global model accuracy through adaptivity to client capabilities and reducing total training time for stragglers. Through a series of meticulously designed experiments, we have demonstrated the effectiveness of our tool in addressing these challenges, which are pivotal for the advancement and practical application of federated learning methodologies.

7.1 Summary of Findings

Our experimental results offer convincing evidence of our tool's capability to significantly improve global accuracy in federated learning setups. By employing an adaptive mechanism that tailors the training process to the computational capabilities of client devices, we were able to mitigate issues such as Out-Of-Memory (OOM) errors on lower-capacity devices, thus ensuring their valuable contributions to the model's learning process. This adaptivity not only enhanced the inclusivity of the federated network but also maximized the overall model accuracy by leveraging the full spectrum of available data across diverse client devices.

Furthermore, our investigations into reducing total training time have highlighted the tool's efficiency in mitigating the impact of stragglers—clients with constrained computational resources that can significantly prolong the training process. By dynamically adjusting training parameters, the tool successfully minimized the delays caused by slower clients, thereby streamlining the overall training duration without compromising on the quality of the model.

7.2 Implications for Federated Learning

Our exploration into enhancing the efficiency and inclusivity of federated learning systems has illuminated a significant potential for further adaptivity within the field. Despite the commendable advancements made by existing works, our findings suggest there remains considerable scope for innovation, particularly in tailoring learning processes to accommodate the diverse computational landscape of client devices. This observation is not to detract from the foundational achievements of prior research but to build upon them, highlighting the evolving nature of federated learning challenges and the continuous need for adaptive solutions.

In contributing to the federated learning community, we aim to underscore the importance of ongoing adaptation and refinement of learning frameworks. Industries and sectors poised to benefit

from federated learning, such as healthcare and finance, could find additional value in approaches that push the boundaries of adaptivity, ensuring that learning outcomes remain robust across a spectrum of operational contexts. Our work, while a step in this direction, reveals the expansive room for exploration and development in making federated learning more adaptable, scalable, and universally applicable.

7.3 Future Work

Our journey into enhancing federated learning systems has uncovered several pivotal areas for future exploration, aimed at pushing the boundaries of adaptivity and efficiency.

Firstly, we plan to significantly expand the adaptivity of our system by incorporating a broader range of criteria to adjust learning parameters. This expansion includes both static and dynamic factors such as CPU capacity, memory availability, and notably, dynamic factors like battery levels and network conditions. This enhancement will enable our system to tailor learning activities more precisely to the fluctuating state of device resources, ensuring that federated learning processes are optimally aligned with each device's capabilities.

Secondly, an advancement in our device management strategy is on the horizon, with a focus on implementing clustering techniques. By organizing devices into clusters based on similar resource profiles, we aim to alleviate the computational load associated with our client selection algorithm. This strategic move is expected to bypass the exhaustive resource usage evaluations for each client in every learning round, streamlining the learning process and reducing overhead.

Moreover, we envision the development of a refined algorithm that evolves the FedAvg framework to incorporate a more nuanced aggregation strategy. This forthcoming algorithm will adjust the weighted average calculation to include various parameters indicative of each client's contribution and effort, such as processed data volume and utilized computational resources. An illustrative parameter could be the number of epochs a client has trained on its local data, shedding light on the depth of its training efforts. By weaving these considerations into the aggregation process, our goal is to cultivate a federated learning environment that more accurately mirrors and values the diverse contributions of participating devices.

These targeted areas of development collectively aim to enhance the operational efficiency and scalability of our federated learning tool. By addressing these specific challenges, we anticipate not only mitigating potential system bottlenecks but also facilitating a smoother and more inclusive federated learning process across an extensive array of devices.

Bibliography

[1] Mlflow tracking, 2023. Accessed: 2023-12-12.

[2] Keras applications, 2024. Accessed: 2024-03-13.

[3] Saahil Afaq and Smitha Rao. Significance of epochs on training a neural network. *International Journal of Scientific and Technology Research*, 2020.

[4] Rashad A. Al-Jawfi. The effect of learning rate on fractal image coding using artificial neural networks. *Fractal and Fractional*, 2022.

[5] Authors. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *Journal Name*, Year.

[6] authors unknown. Complement sparsification: Low-overhead model pruning for federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[7] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, et al. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[9] CAdvisor. Monitoring cadvisor with prometheus, 2024. Accessed: 2024-01-28.

[10] Enmao Diao, Jie Ding, and Vahid Tarokh. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. In *Title of the conference*, Year.

[11] Docker. Docker, 2024. Accessed: 2024-01-18.

[12] M. Donini, F. Rahman, N. Yazdani, L. Cappelli, O. A. Khaled, and E. Haber. Mitigating bias in federated learning. *arXiv preprint arXiv:2012.02447*, 2020.

[13] Flower Labs GmbH. How to implement strategies. https://flower.dev/docs/framework/how-to-implement-strategies.html, 2022. Accessed: 2023-12-11.

[14] Jonathan Frankle. The early phase of neural network training. In *Proceedings of the 2020 International Conference on Learning Representations*, 2020.

[15] Google. cadvisor github repository, 2024. Accessed: 2024-01-18.

[16] Fatih Ilhan, Gong Su, and Ling Liu. Scalefl: Resource-adaptive federated learning with heterogeneous clients. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[17] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[18] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[19] Grafana Labs. Grafana fundamentals, 2023. Accessed: 2023-12-12.

[20] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *arXiv preprint arXiv:1908.07873*, 2019.

[21] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated optimizations in heterogeneous networks. *Title of the journal/conference*, Year.

[22] Y. Liu, T. Chen, Q. Li, Z. Yang, Y. Li, and H. Zhang. A communication efficient collaborative learning framework for distributed features. *arXiv preprint arXiv:1912.11187*, 2019.

[23] Chenyang Lu, Wubin Ma, Rui Wang, Su Deng, and Yahui Wu. Federated learning based on stratified sampling and regularization. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 2022.

[24] Zhenguo Ma, Yang Xu, Hongli Xu, Zeyu Meng, Liusheng Huang, and Yinxing Xue. Adaptive batch size for federated learning in resource-constrained edge computing. *IEEE Transactions on Mobile Computing*, 2023.

[25] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[26] Yiqun Mei, Pengfei Guo, Mo Zhou, and Vishal M. Patel. Resource-adaptive federated learning with all-in-one neural composition. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[27] Prometheus. Introduction to prometheus, 2023. Accessed: 2023-12-12.

[28] Jingyi Qiu. Data selection and machine learning algorithm application under the background of big data. In *Springer Proceedings*, 2021.

[29] A Reisizadeh. Variance-reduced clipping for non-convex optimization, 2023.

[30] F. Sattler, S. Wiedemann, K. R. Müller, and W. Samek. Robust and communication-efficient federated learning from non-iid data. *arXiv preprint arXiv:1903.02891*, 2019.

[31] TensorFlow. Tensorflow learn, 2024. Accessed: 2024-01-18.

[32] TensorFlow. Tensorflow privacy github repository, 2024. Accessed: 2024-01-18.

[33] Yiding Wang. Egeria: Efficient dnn training with knowledge-guided layer freezing. In *Proceedings of the 2022 ACM International Conference on Measurement and Modeling of Computer Systems*, 2022.

[34] Xueli Xiao. Fast deep learning training through intelligently freezing layers. In *Proceedings of the 2019 International Conference on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology*, 2019.

[35] Tong. Yin. Grouped federated learning: A decentralized learning framework with low latency for heterogeneous devices. In *IEEE Proceedings*, 2022.