Characterizing and Optimizing Distributed Machine Learning Systems : Towards a Multi-Objective Approach

Yasmine Djebrouni

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Informatique
Unité de recherche : Laboratoire d'Informatique de Grenoble

Caractérisation et Optimisation des Systèmes d'Apprentissage Machine Distribué : Vers une Approche Multi-Objectif

Characterizing and Optimizing Distributed Machine Learning Systems: Towards a Multi-Objective Approach

Présentée par :

Yasmine DJEBROUNI

Direction de thèse :

Vania MARANGOZOVA MAITRESSE DE CONFERENCES HDR, UNIVERSITE GRENOBLE ALPES	Directrice de thèse
Sara BOUCHENAK PROFESSEURE DES UNIVERSITES, INSA Lyon	Co-directrice de thèse

Rapporteurs :

SEBASTIEN MONNET
PROFESSEUR DES UNIVERSITES, POLYTECH ANNECY - CHAMBERY
GAËL THOMAS
DIRECTEUR DE RECHERCHE, CENTRE INRIA DE SACLAY

Thèse soutenue publiquement le **20 février 2024**, devant le jury composé de :

DENIS TRYSTRAM, PROFESSEUR DES UNIVERSITES, GRENOBLE INP	Président
VANIA MARANGOZOVA, MAITRESSE DE CONFERENCES HDR, UNIVERSITE GRENOBLE ALPES	Directrice de thèse
SARA BOUCHENAK, PROFESSEURE DES UNIVERSITES, INSA LYON	Co-directrice de thèse
SEBASTIEN MONNET, PROFESSEUR DES UNIVERSITES, POLYTECH ANNECY - CHAMBERY	Rapporteur
GAËL THOMAS, DIRECTEUR DE RECHERCHE, CENTRE INRIA DE SACLAY	Rapporteur
ALEXANDRE BENOIT, PROFESSEUR DES UNIVERSITES, POLYTECH ANNECY - CHAMBERY	Examinateur

To my family, with love.

Remerciements

Je tiens à exprimer ma sincère gratitude envers ma directrice de thèse, Vania Marangozova, pour avoir partagé avec moi son expertise en recherche académique et systèmes distribués, ainsi que pour m'avoir guidée avec ses critiques pertinentes lors de la rédaction de ce mémoire. Je salue son regard critique mais aussi son ouverture d'esprit. Merci également, Vania, pour m'avoir soutenue durant ces trois années, tant sur le plan personnel que professionnel.

Mes remerciements distingués s'adressent également à ma co-directrice, Sara Bouchenak, pour avoir partagé avec moi sa maîtrise de la recherche académique et des systèmes distribués. Je salue sa disponibilité, sa motivation et ses critiques pertinentes, des éléments clés qui ont enrichi mon expérience. Merci, Sara, j'ai appris tant de choses de toi.

Je souhaite également exprimer ma vive gratitude envers Pr. Gaël Thomas, Pr. Sébastien Monnet, Pr. Alexandre Benoit et Pr. Denis Trystram d'avoir accepté de faire partie du jury de ma thèse.

Je tiens à témoigner ma reconnaissance et toute ma gratitude envers ma famille pour toute leur affection, leur soutien constant et leurs encouragements. Merci mon cher père et ma chère mère, merci mes chères sœurs. Merci à tous mes amis proches, qui m'ont soutenue et inspirée durant ces trois années.

Un grand merci à tous les membres des laboratoires LIG et LIRIS, et à l'école doctroale MSTII de Grenoble. En particulier, merci à l'équipe ERODS, pour l'agréable ambiance de travail et les nombreux bons moments passés ensemble. Je garderai longtemps un souvenir ému de mon passage parmi ERODS qui m'a tant enrichi.

Je tiens à exprimer mes remerciements à Georges Da Costa, Lynda Ferraguig, Nawel Benarba, Ousmane Touat, Isabelly Rocha, Pasquale De Rosa, Valerio Schiavoni, Angela Bonifati et Pascal Felber. Leur expertise, leurs conseils et leur soutien ont joué un rôle significatif dans l'avancement de mes travaux.

Merci infiniment à tous !

Grenoble, 20 Novembre 2023 Y. D.

Abstract

The past decade has witnessed a significant rise in the utilization of Machine Learning (ML) across various domains. This is attributed to the design of powerful learning techniques and advancements in hardware, enabling the development of sophisticated ML systems. However, the exponential growth of ML workloads from the utilization of massive datasets has outpaced ML systems' capabilities. This has led to the emergence of Distributed Machine Learning (DML), involving the execution of ML algorithms on distributed platforms. DML presents numerous challenges, including configuration complexity and models' fairness concerns.

In the first part of this thesis, we address the configuration challenge in DML systems deployed in data centers. We conduct extensive experiments to collect DML workload traces and analyze their performance under different configurations, shedding light on the impact of tuning strategies. We show that the multi-level parameter tuning (i.e., hyper-parameters and platform parameters jointly tuned) improves model quality and training time, while also optimizing resource costs.

In the second part, we focus on Federated Learning (FL), a contemporary DML paradigm designed for privacy-preserving collaborative learning across distributed nodes. We address the significant challenge of bias and unfairness in FL models outcomes. To tackle this issue, we propose the ASTRAL framework for bias mitigation in FL models, demonstrating its effectiveness in mitigating bias while maintaining accuracy.

Keywords: Distributed Machine Learning, Workload characterization, Federated Learning, Bias, Fairness

Résumé

La dernière décennie a vu une augmentation significative de l'utilisation de l'apprentissage machine (ML) dans divers domaines. Cette évolution est attribuée aux nouvelles techniques puissantes d'apprentissage et aux progrès du matériel, qui ont permis le développement de systèmes d'apprentissage machine avancés. Cependant, la croissance exponentielle des traitements de l'apprentissage machine due à l'utilisation d'énormes ensembles de données a dépassé les capacités des systèmes ML. Cela a conduit à l'émergence de l'apprentissage machine distribué (DML), qui implique l'exécution d'algorithmes de ML sur des plateformes distribuées. Le DML présente plusieurs défis, notamment la complexité de configuration et des problèmes d'équité des modèles.

Dans la première partie de cette thèse, nous abordons le défi de la configuration des systèmes DML déployés dans les centres de données. Nous menons des expériences approfondies pour collecter des traces de traitements DML et analyser leur performance sous l'impact de différentes stratégies de configuration. Nous montrons que la configuration conjointe des hyperparamètres et des paramètres de la plateforme améliore la qualité du modèle et le temps d'entrainement, tout en optimisant les coûts.

Dans la deuxième partie, nous nous concentrons sur l'apprentissage fédéré (FL), un paradigme DML contemporain conçu pour l'apprentissage collaboratif préservant la confidentialité. Nous nous attaquons à un défi important de l'apprentissage fédéré, à savoir le biais dans les résultats des modèles. Pour résoudre ce problème, nous proposons la plateforme ASTRAL pour la mitigation du biais dans les modèles FL. Nous montrons son efficacité dans l'atténuation des biais tout en maintenant la précision.

Mots clefs : Apprentissage Automatique Distribué, Caractérisation des Workloads, Apprentissage Fédéré, Biais, Équité

Table of Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network.

BSP Bulk Synchronous Parallel Systems.

CNN Convolutional Neural Network.

DI Discrimination Index.

DL Deep Learning.

DML Distributed Machine Learning.

DMLP Distributed Machine Learning Platform.

DNN Deep Neural Network.

EOD Equal Opportunity Disparity.

FL Federated Learning.

ML Machine Learning.

RNN Recurrent Neural Network.

SGD Stochastic Gradient Descent.

SPD Statistical Parity Difference.

TPU Tensor Processing Unit.

1 Introduction

1.1 Context and Problem Statement

The past decade has witnessed a significant increase in the utilization of Machine Learning (ML) due to the emergence and the prevalence of powerful learning techniques [74, 162, 77]. Furthermore, the evolution of hardware has played a crucial role in enabling the design and development of powerful accelerators, which have greatly contributed to the enhancement of ML systems [196, 99, 65].

ML entails the extraction of latent patterns from input data to construct models that facilitate the generalization of learned patterns to new data. These models are used in order to make predictions and guide decision-making processes. Today, ML is extensively applied across numerous domains. In healthcare, it is used for accurate disease diagnosis [219]. In the finance sector, ML plays a crucial role in credit scoring [203]. E-commerce platforms leverage ML to deliver personalized recommendations tailored to individual users [245]. ML also finds its way into everyday applications, enabling tasks such as speech recognition [6], language translation [121], sentiment analysis [12], and facial recognition [24].

In order to enhance the quality of learned models and enable ML solutions for predictive tasks in complex applications, a substantial volume of data may be required [212]. However, the exponential growth of ML workloads resulting from the utilization of massive datasets has outpaced the growth of computing power of hardware. Consequently, designers and developers have turned to novel strategies to enable computationally intensive workloads. Among these strategies, the distribution of ML workloads across multiple nodes and the transition from non-distributed ML systems to distributed ML systems have gained prominence, and have lead to the emergence of Distributed Machine Learning (DML).

DML involves the execution of ML algorithms on multiple nodes forming platforms known as DML Platforms (DMLPs). DML leverages distributed computing architectures to enable parallel processing and efficient utilization of computational resources for ML tasks. DML typically allows exploiting massive amounts of input data called training data. This data can

already exist in a central site such as a data center, or be pooled to a central site from various locations. The popularity of DML has increased with the emergence of libraries such as Spark MLlib [139] and TensorFlow [1], enabling users to effortlessly deploy their ML algorithms on already existing distributed computing platforms like Spark [238] and Hadoop [218]. With the widespread adoption of cloud infrastructures, DML platforms started being offered as a service [58]. This service empowers clients to train their own models by leveraging distributed infrastructure and platforms provided by service providers.

Despite their advantages, DML platforms introduce a significantly more complex execution environment compared to a non-distributed environment deployed on a single machine. This complexity stems from various new aspects that come into play and need to be taken into account when utilizing DML including workload parallelization, computations distribution, node communication, results synchronization, and aggregation of intermediate and final results. Each of these factors contributes to the overall complexity of the execution environment and significantly impacts the performance of the platform. Consequently, it is crucial to implement and configure each one of these aspects effectively, according to the specific requirements of the target workload, making the DML configuration challenging. Indeed, platforms like Spark offer more than 180 configuration parameters [90]. Furthermore, for developers and data scientists who are unfamiliar with distributed environments, identifying potential bottlenecks or determining the optimal configurations for DML workloads is a difficult task due to to the opaque relationship between different configuration parameters and the execution of DML workloads.

Other critical challenges of DML include the communication overhead arising from the transmission of a substantial volume of geographically distributed data and concerns regarding data privacy. As a response to these challenges, a novel ML paradigm known as Federated Learning has emerged [135]. FL is a DML paradigm designed to operate in a collaborative environment where data is already distributed across multiple computing nodes, usually geographically distributed, and is not transmitted through the network, preserving its privacy and improving communication efficiency. Instead of sharing data, the participants in FL, known also as clients, share their learned models. However, the inevitable presence of statistical and hardware heterogeneity among the clients poses significant challenges in these systems. This heterogeneity involves uneven data distribution among different clients at the network's edge, as well as variations in computing and communication resources. FL thus rises issues like unequal client participation, and the potential for biased ML models favoring certain clients due to under-representation of specific clients.

In this thesis, we address two distinct research challenges. Firstly, we investigate the configuration issue in the context of a DML system deployed in a data center environment. In this system, a massive volume of data is independent and identically distributed among computing nodes of a data center that collaborate to construct a ML model. We use workload characterization to assess DML system performance and study configuration impacts. We perform extensive experiments with popular DML workloads, analyzing how different settings

affect performance, advancing DML system optimization knowledge.

Afterwards, we shift our attention to the more contemporary DML system, federated learning (FL) [135], where heterogeneous parties in terms of data and hardware collaborate to create an ML model. In this context, we identify a pressing concern, namely, the pervasive issue of bias and unfairness in FL models' outcomes. We provide an overview of the current state of bias mitigation in FL and identify several limitations and challenges associated with existing research efforts. Consequently, we propose, implement, and extensively evaluate a bias mitigation framework tailored specifically for FL.

1.2 Research Objectives and Contributions

In the following, we present our research objectives and contributions.

1.2.1 Characterizing Distributed Machine Learning Workloads

Our Objectives. The optimization of DML systems performance presents several challenges that need to be addressed to ensure their efficient performance and resource utilization. Some of the key challenges include managing configuration complexity. Our first objective is to address this challenge. To do so, it is crucial to conduct a comprehensive study focused on characterizing the configuration requirements of DML workloads. Indeed, workload characterization plays a pivotal role in understanding the performance characteristics of DML systems by quantifying and analyzing performance metrics of their workloads such as model quality and execution time under various platform configurations. This field of study utilizes experimental approaches that involve collecting and analyzing measurements generated by the targeted infrastructures during their operational phase. By leveraging advances in workload characterization, researchers and practitioners can gain insights into the performance of DML workloads, identify potential bottlenecks, and make informed decisions to enhance the overall performance and efficiency of DML systems.

Our Contributions. Our initial contributions are in the field of workload characterization of DML and are presented in the following points:

- We collect traces of DML workloads from extensive experiments conducted on two popular distributed ML and DL libraries, MLlib [139] and BigDL [38], running on a Spark cluster. In total, 13 widely used ML and Deep Learning (DL) algorithms were applied to 6 different real-world datasets. The collected traces amount to a total of 16 GB and include application-level performance metrics as well as platform and system-level metrics. These traces are publicly accessible at *https://github.com/DMLCharacterization/DMLCharacterization/*.

- We perform a detailed analysis of the statistical distributions of our DML workloads and discuss their common patterns and characteristics that we observed.
- We conduct in-depth experiments with different configuration strategies and compare their effectiveness in terms of model quality, training time, inference throughput, and resource costs. These strategies include tuning only the hyperparameters of the learning algorithms, tuning only the lower-level platform parameters, and finally, jointly tuning both the hyperparameters and platform parameters.
- We derive key insights, including the main aspects characterizing distributed ML/DL workloads and the counter-intuitive behavior of workloads involving high-dimensional data.

1.2.2 Mitigating Bias in Federated Learning

Our Objectives. To address data privacy issues in DML, the DML community has proposed federated learning (FL) [135]. In FL, data remains with the clients. Only information defining the locally learned model, *i.e.*, model parameters, is exchanged with the server or the clients during the learning process. Since data remains with the clients, FL ensures data privacy and reduced communication costs. However, despite its advantages, the FL approach introduces several challenges such as the heterogeneity of hardware resources and the heterogeneity of data among different clients [244], leading to the exacerbation of bias in FL models [155]. Bias in ML is a phenomenon that occurs when ML systems produce unfair models due to the use of unbalanced, incomplete, or flawed datasets, models, and training procedures. Biases can have serious consequences for some demographic groups, including lower service quality, reduced revenue, or even illegal actions. In FL, additional challenges are introduced in front of bias mitigation compared to the classical ML run on a single node without privacy constraints. Firstly, due to privacy constraints, FL is characterised by the lack of visibility of the training data, limiting the ability to perform bias analysis to identify biased data, and mitigate potential biases as in classical ML. Indeed, classical ML bias mitigation techniques require access to data. Moreover, FL involves training ML models on distributed data sources, which are often heterogeneous in terms of data distribution, data quality, and hardware resources. This heterogeneity can lead to variations in the contribution and participation of the different clients. Thus even when the data don't contain biases, the FL inherent heterogeneity may introduce potential biases in the training process, mainly in the clients' models aggregation and in the clients' selection components of FL. Thus, FL must contend with the challenges of heterogeneity to be unbiased.

Our contribution. We propose ASTRAL, a novel framework for mitigating bias in FL. The goal of ASTRAL is to keep FL models bias below a specified threshold while maintaining high model accuracy. To achieve this, we propose an FL aggregation method that adjusts the contributions of FL clients' local models based on their impact on the global model's bias and accuracy.

Unlike existing approaches, ASTRAL has the capability to handle bias with regard to multiple demographic groups existing in data and supports bias mitigation for different bias metrics. To evaluate the effectiveness of ASTRAL, we conduct extensive experiments using five widely used datasets and various FL settings. We compare ASTRAL against three other FL bias mitigation techniques, examining model bias, accuracy, scalability, and robustness to client heterogeneity. The results demonstrate that ASTRAL outperforms existing approaches in mitigating bias while maintaining high accuracy. Our main contributions can be summarized as follows:

- We design and implement the ASTRAL framework, which effectively ensures a bias limit while preserving model accuracy in FL.
- We ensure that ASTRAL accommodates the presence of multiple demographic groups and supports various bias metrics, making it adaptable to different practical scenarios, outperforming state-of-art techniques.
- We conduct a thorough evaluation of ASTRAL using seven datasets and diverse FL settings, providing comparisons with three alternative bias mitigation techniques. The evaluation shows the scalability and effectiveness of ASTRAL, independently of data heterogeneity, size and the number of participating clients.
- We provide the ASTRAL software prototype and the datasets used in our experiments, accessible at *https://github.com/FL-Bias/ASTRAL*.

1.3 Summary of Scientific Production

In the following, we list the publications, communications, software, and datasets resulting from this thesis.

1.3.1 Publications and Communications

- Yasmine Djebrouni, Nawel Benarba, Ousmane Touat, Sara Bouchenak, Angela Bonifati, Pasquale Derosa, Pascal Felber, Vania Marangozova, and Valerio Schiavoni. 2023. Bias Mitigation in Federated Learning for Edge Computing. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies. 7, 4, Article 157, 35 pages.
- Yasmine Djebrouni, Isabelly Rocha, Sara Bouchenak, Lydia Chen, Pascal Felber, Vania Marangozova, and Valerio Schiavoni. 2023. Characterizing Distributed Machine Learning Workloads on Apache Spark. In Proceedings of the 24th International Middleware Conference (Middleware '23). Association for Computing Machinery, New York, NY, USA, 151–164.
- Yasmine Djebrouni. 2022. Towards Bias Mitigation in Federated Learning. In the 16th EuroSys Doctoral Workshop.
- Lynda Ferraguig, Yasmine Djebrouni, Sara Bouchenak, and Vania Marangozova. 2021. Survey of Bias Mitigation in Federated Learning. In Conférence Francophone d'Informatique en Parallélisme, Architecture et Système (ComPAS'2021).

- Yasmine Djebrouni, Isabelly Rocha, Sara Bouchenak, Lydia Chen, Pascal Felber, Vania Marangozova-Martin, and Valerio Schiavoni. 2021. Characterizing Distributed Machine Learning and Deep Learning Workloads. In Conférence Francophone d'Informatique en Parallélisme, Architecture et Système (ComPAS'2021).

1.3.2 Developed Software and Published Dataset

- Software:
  - LACAN *https://github.com/DMLCharacterization/LACAN/*
  - ASTRAL *https://github.com/FL-Bias/ASTRAL*
- Dataset: DISTMLBENCH *https://github.com/DMLCharacterization/DMLCharacterization/*

1.4 Thesis Roadmap

This thesis is structured as follows. In Chapter 2, we present the necessary background on DML. Then, in Chapter 3, we introduce our contribution to DML workloads characterization by providing background and literature review on DML characterization. Afterwards, in Chapter 4, we present our DML characterization methodology. In Chapter 5 we present our collected traces and the traces analysis. Chapter 6 introduces our contribution to bias mitigation in FL, we provide there generalities on FL and bias, discuss existing works in bias mitigation in FL, and motivate our work. Then, in Chapter 7, we present ASTRAL, the bias mitigation framework that we have proposed. We present a detailed evaluation of ASTRAL and assess its effectiveness in Chapter 8. Finally, Chapter 9 concludes this thesis and discusses our research perspectives.

2 Background on Distributed Machine Learning

In this chapter, we present generalities on Distributed Machine Learning (DML). Firstly, we provide a background on machine learning (ML), its components, and its pipeline. Afterwards, we discuss the motivation behind the widespread adoption of Distributed Machine Learning (DML). We present various methodological approaches designed to enable the execution of DML. We also discuss several optimizations implemented within these systems, which aim to address the distinctive challenges posed by DML.

2.1 Machine Learning

2.1.1 Overview of Machine Learning

Computer science has empowered humans to program machines, providing them with the means to give precise instructions to machines to accomplish specific tasks. This has resulted in the automation of numerous processes across various domains. For instance, in the field of finances, machines automation has played a significant role in data management and mining [21]. Similarly, in education, a wide range of programs have been developed to implement teaching mechanisms and support learners in achieving pedagogical objectives [202], while the industrial sector has witnessed the prevalence of several software solutions that assist humans in managing various tasks [151]. These advancements in technology have undeniably improved the quality of human lives.

In the mid-20th century, researchers began exploring new techniques that enable machines to learn and make autonomous decisions, without relying on explicit programming and detailed instructions. This marked the advent of ML. One of the early pioneers in this field was Frank Rosenblatt, who along with his team, constructed a machine known as the Perceptron, capable of recognizing letters of the alphabet from images [66]. This breakthrough in computer science opened doors for increased automation and reduced human effort in various domains.

ML is the set of methods within the field of computer science enabling machines to acquire

decision-making capabilities. The term "Machine Learning" itself signifies the ability of machines to learn how to make decisions without being literally programmed [57]. Specifically, ML algorithms learn patterns and relationships from data, and build a decision model that serves as a representation of the learned patterns and relationships. This model is used by machines to generalize the acquired knowledge, enabling them to make predictions and take decisions when faced with new data in future scenarios.

ML methods typically leverage existing data and ground truth collected from the real world. In addition, ML methods exploit mathematical formulations of reward or penalty, as well as statistical and optimization approaches to search for the best model with regard to the formulated reward or penalty, and to the data at the disposal of the machine. The process of learning a ML model is referred to as *ML training*. The data used during that process is called the *training data*. The process of using the learned model is typically referred to as *ML test* or *ML inference*, while the data used here is called the *test* or *inference* data.

Data

Data in ML refers to the input observations or records, issued from the real word. It embeds real world patterns and relationships and is used to build ML models. Data can come in various forms and can be categorized as structured or unstructured. Structured data is organized in a tabular format with rows and columns. Each row represents a data record, and each column represents a specific property or feature of that record. Examples of structured data include data in spreadsheets, databases, or CSV files. On the other hand, unstructured data does not follow a specific format or organization. It can include text documents, images, audio files, or any other form of data that does not have a predefined structure. To train an ML model, the data is typically divided into two subsets: training data and testing data. The training data is used to train the model, allowing it to learn patterns and relationships from it. The testing data is used to evaluate the performance of the trained model on unseen data, and thus assess its accuracy and generalization capacity.

Ground Truth

The ground truth in ML represents the desired pattern that an ML algorithm aims to discover and learn [112]. It is the truth that the models' outcomes should be consistent with, and the reference against which the performance and accuracy of the models are assessed. The ground truth can take various forms, such as labels that classify data records into classes, or numerical values that measure a phenomenon and that the ML method should learn to predict.

Sometimes, depending on the ML use case, the ground truth is available and is fed to the algorithm along to the training data, to allow it to learn a model that predicts outcomes consistent with the ground truth. Other times, it is not available. The algorithm should deduce the patterns by itself.

Machine Learning Model

A model in ML is a mathematical representation that is learned from the training data to make predictions or decisions. It captures the patterns and relationships in the provided data. Generally, a model is defined by a parametric function that maps the training data features to the ground truth. The model parameters are typically real numbers that are learned or adjusted during the training process. Different types of functions can be used to define a model. They can range from simple functions like a regression line [200], which represents a linear relationship between the input feature and the ground truth, to complex functions like artificial neural networks [108], which consist of interconnected layers of artificial neurons that process data (see §2.1.4). However, it's worth noting that not all models are defined solely by parameters and mathematical functions. Some models are based on decision rules. These models take the input features of a data record and pass them through a set of decision rules to make a prediction or decision. Decision trees are an example of such models [177]. In addition, some models, like k-nearest neighbors (KNN) [170], are defined by distance functions. KNN outputs decisions for new instances based on their proximity or similarity to the labeled instances in the training data, using a distance metric such as the euclidean distance.

2.1.2 Machine Learning Pipeline

A ML pipeline typically consists of three main phases. First is the training phase, where the ML model is learned. Afterwards, the model proceeds to the testing phase where its performance is evaluated. Finally, the model is deployed and put into practical use.

Training

ML training plays a fundamental role in a ML pipeline, as it involves the process of applying ML algorithms to build the ML model. This means finding the optimal model that accurately fits the given training data and the potentially associated ground truth. To find such model, a decision penalty known as the *loss function* is formulated and optimized during the training phase. It quantifies the disparity between the predicted outputs generated by the model and the actual outcomes provided in the ground truth.

More formally, let $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \subset \mathscr{X} \times \mathscr{Y}$, be the training data, where $x_i \in \mathscr{X}$ represents the feature vector of the i^{th} record, $y_i \in \mathscr{Y}$ represents the ground truth value for that record, and $n \in \mathbb{N}$ is the records number. We denote the ML model with a function f_θ, where $\theta \in \mathbb{R}^d$ represents the d-dimensional model parameters' vector. f_θ takes as input a feature vector from $\mathscr{X}$ and returns a value from $\mathscr{Y}$. We denote the loss function by $\mathscr{L}$, it takes as input the training data, the ground truth, and the values predicted by f_θ, and returns the overall error of f_θ predictions. The goal of the training phase is to find the optimal values of θ that minimize $\mathscr{L}$.

There exists multiple functions for $\mathscr{L}$, depending on the problem at hand. For instance,

in regression tasks, where the goal is to predict continuous values, common loss functions include the mean squared error (MSE) [41] or mean absolute error (MAE) [20]. The MSE measures the average squared difference between predicted and the ground truth, while the MAE calculates the average absolute difference.

In classification tasks, where the objective is to assign data points to predefined classes, different loss functions are used. For example, for binary classification, when there are two classes in the ground truth, typically referred to as positive and negative, the cross-entropy loss [132] (also known as log loss) is commonly employed. It quantifies the difference between the predicted probability of the positive class, and the actual ground truth label.

In addition to regression and classification tasks, ML training techniques are also applicable to clustering tasks. Clustering is the process of grouping similar data points together based on their intrinsic characteristics or patterns. Unlike regression and classification, clustering does not involve the ground truth. Formally, the training data in clustering contains only features vectors, *i.e.*, $D = \{x_1, x_2, \ldots, x_n\} \subset \mathscr{X}$. A clustering algorithm thus doesn't aim to learn the ground truth. Instead, it aims to discover inherent similarities between the data records, and then to groups similar data records in clusters. One commonly used loss function for clustering techniques is the sum of squared distances between data points and the centers of their assigned clusters [126]. It measures how close are data points to their assigned clusters.

To minimize the loss function and identify the best-fitting model for the training data, various algorithms are employed. regardless of whether the model is a parametric function with fixed parameters or a set of decision rules that accurately represent the relationships in the training data. One approach is to apply optimization techniques like gradient descent [184]. Different training algorithms are presented in §2.1.3.

Test and Deployment

Once the model is trained, it goes through the test phase. Here the model f_θ is applied to unseen data known as the test data. The outputs of the model is then used to assess its performance. Various metrics are used to measure the model's performance depending on its type.

In regression tasks, metrics such as mean squared error (MSE) [41], and mean absolute error (MAE) [20] are used. In addition, there is the and R-squared [141] metric that measures the proportion of variance in the ground truth explained by the model.

In binary classification tasks, several common metrics are used to evaluate the performance of the classification model. These metrics include accuracy, precision, recall, and F1-score [173]. Accuracy calculates the proportion of correctly classified instances, while precision measures the proportion of true positive predictions out of the predicted positive instances. Recall measures the proportion of true positive predictions out of the actual positive instances, and the F1-score combines precision and recall to provide a balanced measure of performance.

Clustering models are evaluated using metrics such as the Silhouette Coefficient [191] and the Sum of Squared Errors (SSE) [153]. The Silhouette Coefficient measures the separation of clusters, providing a value between -1 and 1, with higher values indicating better-defined and well-separated clusters. SSE measures the overall intra-cluster compactness by summing the squared distances between each data point and its assigned cluster centroid.

Once validated, the model is deployed on machines. Deployment involves integrating the model into the target system or application, making it available for real-time predictions or decision-making tasks.

2.1.3 Types of Machine Learning Training Algorithms

An ML algorithm, also called ML method or ML technique, is a specific computational procedure or set of rules that machines use to build ML models from data and ground truth. It is executed in the training phase and usually involves the optimization of an objective function leading to the definition of the ML model. There exist several approaches to train models in ML. In the following, we present the most known approaches.

Gradient Descent Based Algorithms

Widely used in ML, these methods rely on the gradient descent optimization algorithm [184] to optimize the loss function and determine the best parameters that fit the training data. The gradient descent process involves iteratively updating the parameters using the algorithm presented in Algorithm 1. Precisely, in each iteration of gradient descent, the gradients of the loss function of the training data with respect to each parameter are computed (see Line 3 in Algortihm 1). These gradients denoted by $\{\Delta\theta_i\}_{i=1}^{i=d}$ indicate the direction of the steepest descent through the loss function values towards the loss function minimum. Once the gradients are computed, the parameters are adjusted by subtracting a portion of the gradients from their current values (see Line 5 in 1). This update allows the parameters to move closer to the optimal values that minimize the loss function. The portion to be subtracted from the weights is computed by multiplying the gradient per a factor γ called the learning rate. Here the learning rate controls the magnitude of the update, and thus the speed of the convergence. The convergence of the optimization process occurs when the gradients become sufficiently small, indicating that further updates to the parameters would make little difference. This means that the algorithm has found a locally or globally optimal solution. Several ML algorithms are based on gradient descent, including linear regression, logistic Regression [111], support Vector Machines (SVM) [130], and artificial neural networks.

In gradient descent based optimization approaches, the computation of gradients can be tailored to different strategies. Gradients are essentially the derivatives of the loss function with respect to each model parameter. The choice of how these gradients are computed characterizes the three main variants of gradient descent. In Stochastic Gradient Descent

Algorithm 1: The general gradient descent algorithm; different choices of the learning rate γ, number of iterations T, and the loss function for $\mathscr{L}(\theta)$ may lead to different models.

Input: Initial parameters θ, training data $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, number of iterations T, learning rate γ
Output: Final parameters θ
1. **for** $t = 1$ **to** T
2. Estimate $\nabla\mathscr{L}(\theta)$ using D
3. For each component θ_i of θ, compute $\Delta\theta_i = \frac{\partial\mathscr{L}(\theta)}{\partial\theta_i}$
4. For each component θ_i of θ, compute: $\theta_i := \theta_i - \gamma\Delta\theta_i$
5. **return** θ

(SGD), Gradients are computed using only one randomly selected data point in each iteration [17]. This introduces stochasticity and faster updates, making it computationally efficient, particularly for large datasets. In Batch Gradient Descent [81], gradients are computed using the entire training dataset in each iteration. This provides a more accurate estimate of the direction for parameter updates but can be computationally expensive, especially for large datasets. In Mini-Batch Gradient Descent [81], gradients are computed using a randomly sampled subset or "mini-batch" of the data in each iteration. This strikes a balance between the efficiency of SGD and the stability of batch gradient descent, allowing for quicker updates that are less computationally intensive than the full dataset.

Decision Rule Based Algorithms

Decision rule-based algorithms involve learning a set of rules from the training data to make decisions. In these algorithms, rules are formulated as conditions based on feature values, serving as criteria for classifying or predicting outcomes. Each rule represents a specific combination of feature values that guides the decision-making process. For instance, in a decision tree algorithm, a rule could be expressed as follows: "IF age ≥ 30 AND nb_weeks_worked ≥ 48, THEN classify as 'high income'" [201]. These rules are learned during the training phase by recursively partitioning the data based on the feature values of the data points, resulting in a hierarchical structure for decision making. The partitioning process aims to create data partitions that maximize the separation or purity of data points belonging to different classes [201]. The quality of a split is evaluated using metrics such as Information Gain and Gini Impurity, commonly employed in decision tree algorithms [201].

Ensemble Methods

Ensemble methods involve combining multiple models to create a single, more powerful model. There are two common ways to combine models: boosting [68] and bagging [18]. Boosting is a sequential approach where multiple models are trained iteratively. Each subsequent model focuses on the instances that were poorly predicted by the previous models.

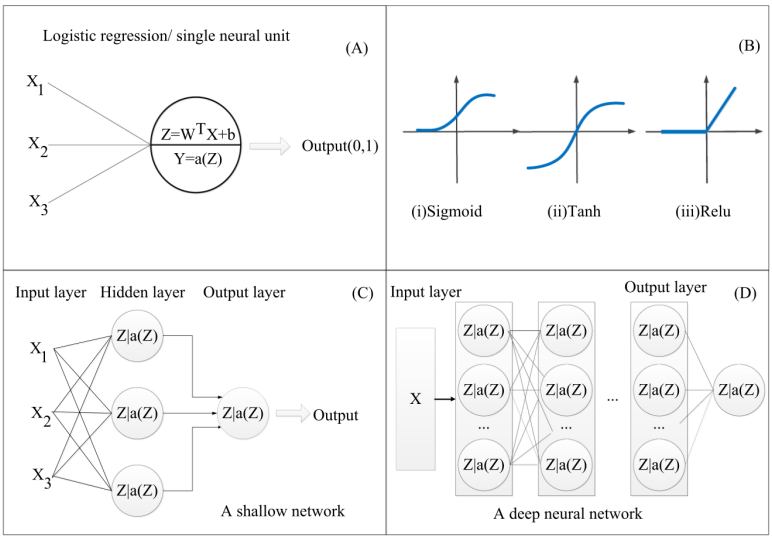


Figure 2.1: Foundations of Deep Learning: (A) One neuron neural network (equivalent to the Logistic Regression model), (B) Visualization of various activation functions, (C) Single hidden layer shallow network, and (D) Multi-layer neural network with over two hidden layers [50].

By emphasizing the difficult instances, boosting aims to improve the overall performance. The final model is a weighted combination of these individual models, where each model's contribution is determined by its performance. Bagging, short for bootstrap aggregating, involves training multiple models on different subsets of the training data. Each model is trained independently, and the predictions from all models are combined using averaging or voting to make the final prediction. By creating diverse models on different subsets, bagging helps reduce the model variance and improve generalization performance. Both boosting and bagging methods leverage the collective knowledge of multiple models to enhance the overall prediction capability. Examples of these algorithms are ensemble methods, such as Random Forest [13] (combining decision trees through bagging) and Gradient Boosting Decision Tree [103] (combining decision trees through boosting).

Instance Based Algorithms

In contrast to other approaches, Instance-based algorithms do not explicitly learn models but instead memorize the training data. During the prediction phase, these algorithms calculate the proximity of the input data to the stored training instances using a distance function. They then provide predictions based on the labels or values of the nearest records in the training data. One example of an instance-based algorithm is the K-nearest neighbors (KNN) algorithm.

2.1.4 Deep Learning

Deep Learning (DL) is a subfield of ML that uses Artificial Neural Networks (ANN) [230] to construct powerful models capable of learning from complex data. ANNs are complex mathematical functions represented by interconnected nodes and edges that looks like neural networks, hence the name. An ANN is defined by its structure, including the number of neurons, links between neurons, and layers. More elements define an ANN. Each neuron in an ANN applies a specific function, known as an activation function [188], to the input data. The role of activation functions is to generate non-linear relationships between the input and the output. There are many activation functions (some of them presented in Figure 2.1(B)). The connections between neurons are represented by edges, which define the flow of information through the network (see Figure 2.1(D) and Figure 2.1(C)).

There are several types of ANN, such as Fully Connected Networks (FCNs), where each neuron is connected to every neuron in the preceding and succeeding layers, facilitating comprehensive information processing. Convolutional Neural Networks (CNNs) [80] specialize in capturing spatial patterns in data, particularly images, by utilizing convolution operations. Recurrent Neural Networks (RNNs) [137] are designed for sequential data analysis, incorporating loops to capture temporal dependencies.

During training, the parameters of an ANN, including the weights assigned to edges and biases assigned to neurons, need to be learned. Optimization techniques like gradient descent are commonly employed to update and adjust these parameters to minimize the difference between the predicted and actual outputs.

The complexity, flexibility, and structured nature of artificial neural networks enable them to learn complex relationships and extract meaningful features from data, making them a powerful tool in the field of ML, for both supervised and unsupervised learning.

2.2 Scaling Machine Learning Techniques

Nowadays, both large and small organizations leverage the power of massive amounts of data collected from various sources such as social media accounts [10], health records [48], and computer logs [113], to extract valuable insights that can be applied to their strategies.

ML presents an optimal approach to unlock the potential inherent in Big Data. First, it is a data-driven technology. The injection of larger volumes of data into ML systems enriches the knowledge base and amplifies the system's learning capacity. Thus, it enables the extraction of valuable information from extensive and diverse data sources. Moreover, ML possesses the capability to adapt to complex datasets that are ubiquitous in our era such as text, images, or audio data and that are characterized by their high number of features, the presence of non-linear relationships between variables, and the presence of temporal dependencies. The adaptability of ML to such data is rooted in the inherent flexibility and learning capacity of ML

algorithms. By adjusting the structure and the parameters of models, and by increasing their complexity when necessary, they can effectively capture and represent the complex patterns and relationships present in complex data.

However, in specific instances where ML is applied to Big Data, the process of ML can become excessively time-consuming due to the intensiveness and complexity of the computations. To address this challenge, designers resorted to increasing the scalability of ML training techniques to large and complex data through two main strategies: scaling-up and scaling-out. The test phase, in comparison, is typically less computationally demanding. Therefore, optimization efforts focus on either the training, or test phase of complex models deployed on constrained devices like real-time prediction models in autonomous vehicles. On the following, we present techniques used to scale ML training techniques.

2.2.1 Scaling Up

This approach augments the resources of the node running ML training, leading to improved performance and scalability. Enhancing computation resources by allocating additional powerful processing units is widely explored in ML literature due to its advantages. It allows running faster computations and parallelizing the workload on the same machine, significantly reducing the overall execution time.

GPUs (Graphics Processing Units) have become the predominant choice for accelerating ML techniques [196]. Early work by Steinkraus et al. [196] used GPUs, specifically the AT1 Radeon X800 and NVlDlA GeForce 6800 Ultra, to implement a two fully-connected layers network. They achieved a threefold speedup compared to CPU usage. Their work inspired the utilization of GPUs for more complex models like Multilayer Perceptrons (MLPs) [71] and Convolutional Neural Networks (CNNs) [80]. Indeed, modern GPU nodes have made MLPs more affordable and suitable for large datasets [32]. Moreover, running convolution operations that are inherently parallelizable on GPUs has significantly benefited CNNs performance during training and inference [172, 227].

In contrast to GPUs, FPGAs (Field-Programmable Gate Arrays) [102] provide adaptable hardware configurations and frequently offer superior performance per watt for crucial DL subroutines, including sliding-windows computation [65]. For these subroutines, FPGA provided significantly faster performance with speedups up to 11× and 57× compared to GPUs and CPU, except for small inputs sizes [65]. Nevertheless, the programming of FPGAs necessitates specialized hardware expertise, thereby restricting their utilization primarily to researchers and application scientists who possess knowledge and proficiency in this domain [110].

TPUs (Tensor Processing Units) [228] are recent processing units developped by Google, primarily designed for DL inference and training [99]. TPUs are used in Google Cloud data centers to handle massive amounts of data. Google claims that TPUs are, on average, 15-30 times faster than contemporary GPUs or CPUs [99, 187].

Numerous benchmarking studies have been conducted to evaluate and compare the performance of these different accelerators for ML tasks. Authors of [187] and [65] indicate that accelerated hardware including CPUs, GPUs, FPGAs, and TPUs, provide significant and sometimes comparable performance improvements for neural network training and inference. The choice of specialized hardware depends on the specific data, model and its requirements. Additionally, benchmarking studies must be continuously updated and frequently run due to the rapid evolution and changes in deep learning models [187].

2.2.2 Distributed Machine Learning (Scaling Out)

In this approach, an ML workload is distributed across multiple machines or workers that collaborate in order to run ML in a parallel way. This approach has given rise to a new research discipline known as Distributed Machine Learning.

In their work [212], the authors highlight several reasons for choosing scaling out ML (DML) over scaling up, or even combining both strategies instead of solely relying on scaling up. First, DML allows for horizontal scalability by adding more nodes to the system. This increases I/O bandwidth enabling the handling of larger datasets compared to a single specialized hardware node. Then, DML across multiple nodes can be more cost-effective than investing in expensive specialized hardware. Finally, DML across multiple nodes enhances fault tolerance. If one node fails or experiences issues, the overall system can still function by offloading the workload to the remaining nodes. In the following, we shed light on DML and the different strategies used to enable it.

2.3 Distributed Machine Learning

Distributed Machine Learning, also known as DML, refers to multi-node ML or DL algorithms and systems that are designed to improve performance and scale to more training data and bigger models [225].

There exist several strategies, known as DML paradigms, that allow distributing training processes of ML models across multiple machines or computing nodes. Among these paradigms, parallelism paradigms play a pivotal role. They focus on how the workload is divided and how data is partitioned across several computation nodes. Thus, they address the challenge of effectively leveraging distributed resources to handle the computational demands of large-scale datasets and complex models which is the core of DML.

Architecture and synchronization paradigms complement parallelism paradigms by addressing broader aspects of DML. Architecture refers to the underlying structure or nodes' connections pattern that governs communication and coordination between distributed nodes, impacting aspects such as fault tolerance, scalability, and communication efficiency.

On the other hand, synchronization paradigms deal with the crucial aspect of ensuring consis-

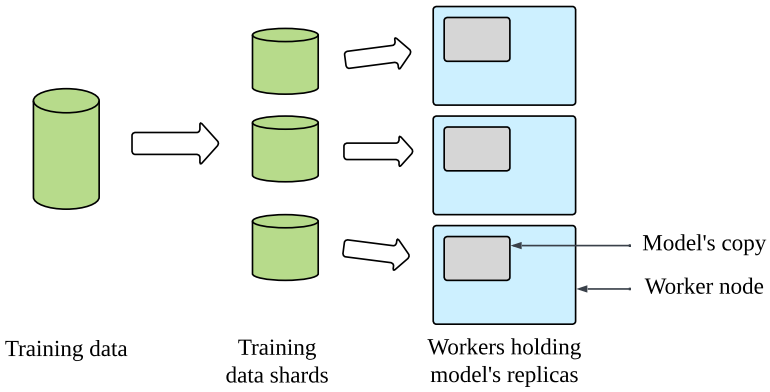


Figure 2.2: Data parallelism: The training data is divided into smaller subsets called shards, and each shard is assigned to a specific worker node holding a copy of the model.

tent updates and coordination between distributed workers during training. Synchronization plays a pivotal role in achieving convergence and maintaining consistency across the distributed system. Together, the three paradigms form the foundational pillars for developing efficient and scalable DML systems.

2.3.1 Parallelism Paradigms

There exist mainly two scalability dimensions that motivated parallelism paradigms: The data scale and the model scale. These two scales have motivated data and model parallelism. Other parallelism paradigms exist, they leverage the combination of both data and model parallelisms.

Data Parallelism

With datasets reaching unprecedented sizes, it became increasingly difficult to accommodate them and perform model training efficiently on a single computing node. This challenge has motivated for the emergence of the data parallelism paradigm [224]. In data parallelism, multiple computing nodes work together in a coordinated manner. Each node has its own copy of the ML model. The training data is divided into smaller subsets called shards, and each shard is assigned to a specific computing node. Each node trains its own model copy using its assigned data shard as presented in Figure 2.2.

During the training process, each node generates partial updates for the model based on its local training data. These partial updates represent the changes that should be made to the model to improve its performance. In particular, in the prevalent gradient descent-based algorithms, the partial updates computed by a node are equal to the gradients computed by that node using its local loss function. Once a training iteration is complete, the partial updates from all the nodes are aggregated. Typically, a central node receives the partial updates

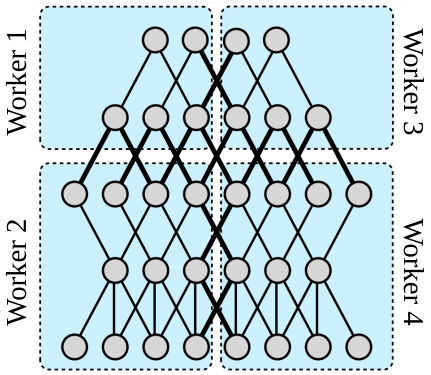


Figure 2.3: An example of model parallelism showing a five layer deep neural network partitioned across four workers (blue rectangles). Nodes with edges that cross partition boundaries (thick lines) will need to have their state transmitted between workers [43].

from each node and computes the overall update that needs to be applied to the model. The aggregation of partial updates is performed through summation [224].

It is important to note that for this summation to be valid, the data should be distributed among the nodes independently and identically [224], which is typically assumed in DML systems implementing data parallelism. In other words, each data shard should be representative of the overall dataset, and there should be no overlap or dependency between the shards. If the data distribution is not independent and identical, the summation of updates would not accurately reflect the update that could have been computed on the entire training data.

Once the central node computes the overall update, it updates its own copy of the model and distributes the updated model to all the nodes. The updated model is then used for the next iteration of training, and the process repeats until the model converges to a satisfactory level of performance.

One drawback of data parallelism is the increased communication overhead required to synchronize the updates among the devices. As the number of devices increases, the amount of communication and synchronization can become a bottleneck, impacting training efficiency. Additionally, data parallelism may encounter challenges when dealing with imbalanced datasets, as the workload may not be evenly distributed across devices, leading to slower convergence or sub-optimal results. These challenges motivated many works that have proposed solutions for efficient communication and synchronization as we will present §2.3.2 and §2.3.3.

Model Parallelism

In certain scenarios, ML models can become excessively large, especially in the case of DL models that contain billions of parameters. Consequently, these models may surpass the

memory capacity of a single computing node. To address this issue, the concept of model parallelism has been introduced.

In model parallelism, the model is distributed across multiple computing nodes as presented in Figure 2.3. The training data is fed to the workers responsible for the initial stage of the model training. For example, for DL models, this stage consists in the computations done by the first layers of the model (done by Workers 1 and 3 in Figure 2.3). When the initial stage of training is performed, its results are propagated to subsequent workers responsible for subsequent training stages, until the entire model is trained. Model parallelism faces a significant challenge in determining how to divide the model into partitions and assign them to workers [133]. Such decisions can be made by human experts based on heuristics and intuition regarding which parts of the model should reside on specific nodes [166]. However, more sophisticated methods exist. In [143, 142], researchers introduced a reinforcement learning-based method that uses a predictive model to determine which subsets of operations within a TensorFlow graph should be executed on available workers. Initially, the model partitions are subjected to permutations between the workers, measuring performance metrics such as execution time (*e.g.*, for a training iteration) and using them as reward signals. If performance improves, the permutation is retained, and additional permutations are performed until performance convergence is achieved.

Model parallelism is not without limitations. Interdependencies between a model's partitions lead to high communication overhead between workers which leads to congestion and synchronization delays. Consequently, increasing the level of model parallelism does not necessarily result in accelerated training [142]. Another potential drawback to consider is that several workers may be idle simultaneously, especially when it's not their turn to handle the training stage.

Hybrid Parallelism

To overcome the limitations of data parallelism and model parallelism, researchers have adopted a hybrid approach that combines both techniques. Pipeline parallelism is a hybrid approach proposed by [154] and [91]. In pipeline parallelism, the training data is divided into micro-batches, and the model is partitioned by assigning specific stages to individual workers. Each worker is responsible for loading a distinct portion of the model for training. It then computes outputs for a set of micro-batches, propagates these outputs to the next set of workers as soon as they become available, and continues computing outputs for the remaining micro-batches. The calculations overlap between two stages residing on different workers when the data from the first stage is ready for processing by the second stage, and the first stage is ready to process a new micro-batch of data. Consequently, pipeline parallelism can be viewed as a combination of data parallelism, where data is processed concurrently across multiple layers, and model parallelism, where the model is divided among the workers.

Other hybrid approaches exist. For convolutional neural networks (CNNs), The author of [107]

proposed to leverage data parallelism for convolutional layers, while model parallelism is applied to fully-connected layers. The idea is that model parallelism is efficient when there is high computation per neuron activity, which is the case in fully-connected layers [107]. In contrast, data parallelism is effective when there is high computation per weight, as seen in fully convolutional layers [107].

Another example of a hybrid parallelism approach is Hypar [194], which mixes data and model parallelism. For model parallelism, Hypar searches for the best layer-wise parallelism configuration to minimize communication during training using dynamic programming. Its evaluation demonstrates performance gains and energy efficiency improvements compared to default data parallelism, outperforming other hybrid-parallelism methods as well [107].

Other works explore and implement hybrid parallelism techniques. For instance, [4] proposes a method combining model and data parallelism, along to a Genetic Algorithm-Based Heuristic Resources Allocation (GABRA) mechanism for optimal distribution of partitions on available GPUs to optimize training time. Another study [52] introduces a library for implementing hybrid-parallel 2D CNNs, while [163] extends this work to support 3D data.

2.3.2 System Architecture Paradigms

During data-parallel training, workers' updates are brought together at each iteration to combine their knowledge. Thus, an essential consideration in DML systems is effective collaboration among the workers. This involves determining the communication channels and establishing clear communication patterns between the workers. Deciding which workers communicate with each other and how they exchange updates is crucial for ensuring efficient collaboration during training.

Parameter Server Architecture

The parameter server architecture [115] is a distributed framework that consists of a central node known as the server or parameter server, connected to multiple worker nodes. In this architecture, the parameter server holds a copy of the model parameters that are accessible to all workers. Across the training iterations, the workers send their updates computed on their respective data shards to the server. The server incorporates these updates into the model and communicates the updated model back to the workers as presented in Figure 2.4. The parameter server paradigm is widely used in DML systems and has been implemented in popular platforms such as TensorFlow, Spark, and PMLS.

However, this architecture faces certain challenges. One significant concern is its vulnerability to failures. If the server experiences a breakdown, the entire system collapses, leading to disruptions in training. Additionally, the parameter server architecture may encounter potential bottlenecks at the server level due to the communication overhead involved in handling updates from all the workers.

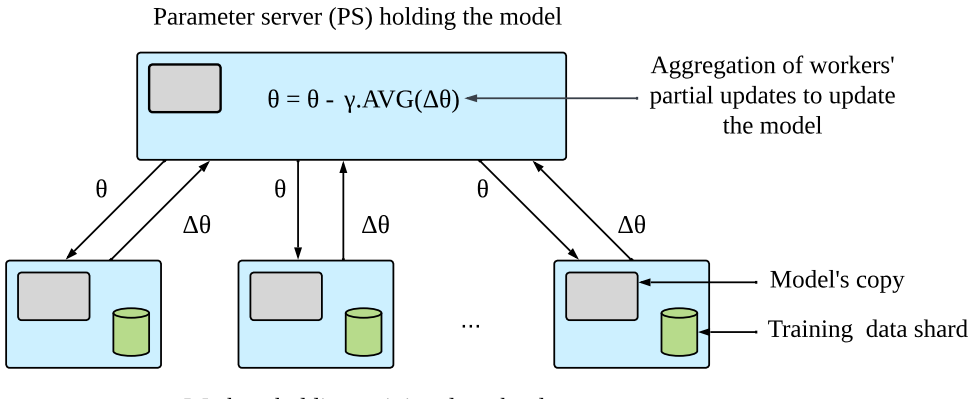


Figure 2.4: Parameter server architecure: the parameter server updates the model by applying on it the average of the worker's partial updates.

Peer-to-peer (P2P)

To mitigate the fault tolerance issues associated with the parameter server architecture, an alternative approach called the peer-to-peer architecture has been proposed. In this architecture, workers establish direct communication among themselves, eliminating the dependency on a parameter server node. Within the P2P architecture, all the workers are fully connected to each others. Each worker receives updates from each of the other workers and performs an *all-reduce* operation to aggregate the received updates. Then, each incorporates the computed update into their respective models. However, it's important to note that this architecture introduces a notable challenge in the form of significant communication overhead. In fact, the communication cost in a network with n workers scales with $\mathcal{O}(n^2)$. As the network expands, the communication demands between workers can become excessively burdensome.

In their paper [183], the authors present BrainTorrent, a DML framework designed specifically for medical applications, operating without a central server in a peer-to-peer fashion. The learning process follows an iterative approach. In each iteration, a worker initiates a connection request to all other workers in order to compare their model versions. Only those workers with more recent model versions send their updates to the initiating worker. Subsequently, an aggregated model is created by combining all the received models, which is further fine-tuned using the local data of the worker who initiated the connection request.

Ring All-Reduce

To address the weak fault tolerance of the parameter server architecture and reduce the communication overhead observed in peer-to-peer (P2P) architectures, an alternative solution known as the ring-based peer-to-peer architecture, or ring-allreduce, has been proposed.

In the ring-based architecture, workers are connected in a circular ring topology rather than a

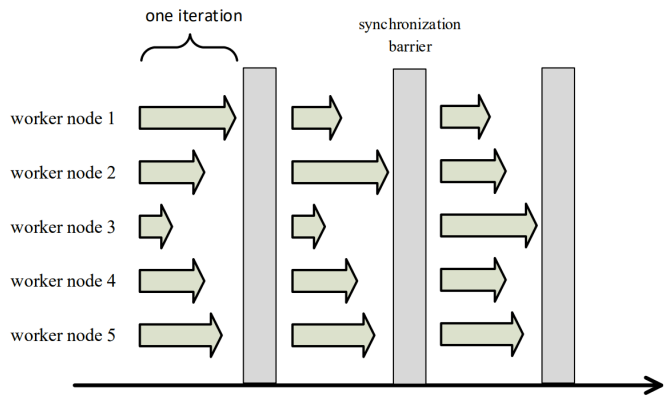


Figure 2.5: The Bulk Synchronous Parallel synchronization model [242].

fully connected network. Each worker propagates its computed updates to its neighboring worker in the ring topology. Upon receiving an update, the neighbor worker performs a summation operation with its own update. This process continues through several rounds of communication between pairs of workers, allowing the last worker in the ring to accumulate an aggregation of updates from all other workers. Subsequently, this worker communicates its aggregated update to the other nodes in a pairwise manner.

Baidu was among the pioneers in proposing the use of ring-allreduce for data parallel deep learning training [78]. Uber's Horovod [186] replaced Baidu ring-allreduce implementation with NVIDIA's library "NCCL" [93] to implement the ring-allreduce communication pattern. However, it is worth noting that this architecture may introduce potential delays as workers have to wait for sequential communication with each other to compute the overall updates.

2.3.3 Synchronization Paradigms

For data-parallel training, a synchronization paradigm answers the question: When exactly should the communication between the workers happen. It ensures that workers operate on the same version of the model. Additionally, such paradigm may consider the tolerance for workers to use slightly outdated parameter versions. This will allow handling the trade-off between the the training quality and the training speed. We present in the following the different synchronization paradigms in DML.

Synchronous Learning

Synchronous learning follows the Bulk Synchronous Parallel synchronization model (BSP) [208, 76] presented in Figure 2.5. In synchronous training, the updates of the model are synchronized among workers after each iteration, typically involving the processing of a batch of data. A single iteration of this paradigm includes three distinct stages. Firstly, in the computation phase, the different workers execute computations on the model derived from the preceding

iteration using their data shards. Subsequently, in the communication phase, workers communicate the results of their computations to each other, leveraging the system architecture. Finally, in the synchronization phase, workers synchronize their progress using a barrier mechanism, ensuring that no worker proceeds to the next iteration until all workers have completed the communication step [31].

Synchronous training is widely adopted in popular open-source deep learning frameworks like TensorFlow [1], and MXNet [26]. One of the primary advantages of this approach is that, when coupled with data parallelism, it ensures the convergence of parallel training towards the optimal model that would have been achieved when training on all the gathered data without parallelization. However, a limitation of this approach is that faster workers must wait at synchronization barriers until slower workers complete their computations, which can introduce delays and hinder overall progress [31]. This is known as the straggler problem. It is common in heterogeneous environments, where the participating nodes vary in terms of data distribution and computational power, leading to variation in the processing speed.

An aggressive synchronization scheme called A-BSP was proposed by Wang et al. [215] to tackle data and computational power heterogeneity. A-BSP employs a strategy where, upon completion of its workload, the fastest worker retrieves the current updates from the remaining workers who have partially processed their input data. Thus, this approach implements aggressive synchronization that forces slow workers to synchronize. Additionally, unprocessed data is given priority in subsequent iterations to ensure algorithm convergence. The effectiveness of this approach has been demonstrated in frameworks such as Spark and PMLS (Petuum) [224].

Asynchronous Learning

In asynchronous training, workers have the autonomy to update their models independently, without waiting for each other's updates. This approach grants them utmost flexibility in their training process and effectively eliminates any concerns regarding slower workers. The authors of the paper [178] introduce a strategy for implementing asynchronous learning in non-distributed multi-processor environments, which they named HOGWILD!. In HOGWILD!, parallel processors have equal access to shared memory and can individually modify components of the model, all without the need for memory locks. Although this lock-free scheme might initially seem prone to failure, as workers could overwrite each other's progress, the authors demonstrate that as long as processors only modify small portions of the model, memory overwrites are infrequent and introduce minimal computational errors when they do occur [178]. The authors provide both theoretical and experimental evidence of nearly linear speedup with the number of processors in typical use cases. The HOGWILD! scheme has been successfully applied to neural network training [45]. DOGWILD! [160] is a distributed implementation of HOGWILD!.

Other existing asynchronous training algorithms primarily focus on enhancing training per-

formance through an asynchronous communication strategy. They adopt a simple update method for each worker's local model, *i.e.*, the model trained on the worker. In this method, the local model of a worker is entirely replaced with the global model computed by all the nodes during the training process. Consequently, these existing algorithms do not take into account the heterogeneity among workers when updating their local models, leading to delayed model convergence. The authors of [104] aim to accelerate model convergence. To achieve this, they propose a novel update rule for the local models in asynchronous distributed training: after applying the computed local update to their respective locals models, workers further apply an additional update where the local model is replaced by a weighted average between the local model and the global model. This approach reduces the disparity between workers' local models and the global model, facilitating faster convergence.

Asynchronous training is a well-established technique, and numerous implementations are available in popular frameworks such as TensorFlow, MXNet, CNTK [185], and PyTorch [167].

Bounded asynchronous Learning

To alleviate the detrimental effects of asynchronous learning on convergence speed, several protocols have been proposed to impose constraints on asynchrony. These protocols, known as Bounded Asynchronous Learning, introduce constraints such as the maximum number of iterations or the allowed number of unsynchronized workers. One notable approach is Asynchronous ADMM [241], which employs a "partial" synchronization barrier. Instead of requiring full synchronization among all workers in each iteration, only a subset of workers need to synchronize. This means that updates from slower workers are incorporated into the global model less frequently compared to faster workers. To ensure sufficient freshness of updates, a bounded delay condition is enforced. Specifically, every worker's update must be taken into account at least once every τ iterations, where τ is a parameter ≥ 1 defined by the user.

Another method to bound asynchronicity is by allowing workers to train with outdated parameters while restricting the degree of staleness. By setting a limit, such as a specific number of iterations, workers can make independent progress, mitigating the impact of stragglers and improving efficiency. It is worth noting that ML algorithms exhibit intrinsic self-correction behavior due to their iterative-convergent nature. Small computation errors incurred during training are gradually mitigated by subsequent iterative improvements [176]. Hence, a minor staleness in parameters only minimally affects the overall convergence. The Stale Synchronous Parallel (SSP) model, introduced by Cipar et al. [31], offers a framework for delayed updates, allowing for a time lag between a worker updating parameters and other workers observing the effects of those updates. The delay is specified in terms of the number of iterations.

Building upon SSP, Dai et al. [39] proposed Eager SSP (ESSP). ESSP *eagerly* propagates the updates of workers in contrast to SSP where updates are retrieved by a worker only when its state becomes significantly outdated. Te authors show ESSP achieves faster convergence

compared to SSP both theoretically and empirically. ESSP has been implemented in the PMLS system [224].

To address heterogeneity among workers, Jiang et al. [96] suggest incorporating dynamic learning rates alongside SSP. Based on a worker's speed, its learning rate is adjusted to mitigate the impact of stale updates on global model compared to fresh updates. Notably, Li [19] mentioned in a GitHub discussion that SSP was not implemented in MXNet due to minimal observed delays resulting from uniform GPU-intensive operations, rendering the benefits of SSP relatively insignificant.

2.4 Privacy-Preserving Distributed Machine Learning

In many practical use cases of DML, data transfer plays a pivotal role. There are several reasons for this. Firstly, in some cases, data is inherently distributed across multiple sites due to its origin from diverse sources, such as various organizations or wearable devices. To leverage this data effectively, it becomes necessary to pool it into a central location. There it can undergo crucial preprocessing steps like cleaning and analysis, which are essential for ensuring data quality and preparing it for training on DML platforms. Furthermore, data transfer enables better resource optimization. By transferring data between nodes, DML frameworks like Spark can distribute the training workload (data and computation) across multiple resources according to the availability of the latter. This dynamic allocation of tasks and data helps optimize the overall training process.

However, data transfer rises concerns about privacy leakage. Data privacy is crucial to protect individuals' personal information from unauthorized access. It ensures that sensitive data such as social security numbers, financial records, and health information remains secure, mitigating the risks of identity theft, fraud, and other malicious activities against individuals [217]. Data privacy also builds trust between individuals and organizations. Since data privacy is important in today's world, several laws and regulations exist to ensure it is respected. They mainly outline guidelines for the collection, storage, and sharing of data with third parties. The most widely discussed data privacy laws include GDPR [86] and CCPA [79]. The increasing awareness of privacy requires a DML system to take privacy-preserving into consideration. Privacy-preserving DML system generally protects some or all of the following information [211]: *i)* Input training data, *ii)* Output predicted labels, *iii)* Model information, including parameters, structure, and loss function, *iv)* Identifiable information, such as which site a record comes from.

Federated learning [135] is a recent DML paradigm introduced by Google that enables data owners to collaborate on training ML models while preserving data privacy. Instead of sharing raw data, participants train models on their local data and exchange only the models parameters in an iterative process. At each iteration, the clients' models are averaged to get one shared model. FL offers notable advantages, but also presents several challenges, particularly in the areas of data privacy, training performance, and model fairness.

2.5 Summary

This chapter provides an overview of DML. We have explored the core concepts of both ML and DML. We have delved into the motivations behind the adoption of DML. We presented diverse approaches designed to enable DML. Additionally, we presented several optimizations implemented in these systems as responses to the unique challenges presented by DML.

In the following, we will focus on two major challenges associated with DML systems, namely DML configuration complexity and bias issues in FL. We will present these two challenges, discuss related work addressing them, and propose our contributions.

Part I

Characterization of Distributed Machine Learning Workloads

Overview

Several well-supported distributed machine learning (DML) libraries and platforms exist (*e.g.*, MLlib [139], BigDL [38], TensorFlow [1], *etc.*), contributing to the prevalence of DML. Such libraries allow users to easily run learning methods on distributed clusters dedicated to big data processing. In order to optimize the performance of DML services in terms of model training time, model quality and high inference throughput, existing DML services using DML libraries and platforms are carefully tuned. However, tuning DML is a difficult task as the complexity of the underlying distributed platforms may be overwhelming. Uninitiated users, *i.e.*, data scientists, are more familiar with tuning parameters related to the algorithmic approach of the learning method (*i.e.*, hyper-parameters), lacking an in-depth understanding of the trade-offs and challenges to parameterize DML platforms. System administrators focus on tuning distributed platforms without considering high-level hyper-parameters, unaware of the implications of the platform on the quality of the learning models.

Characterizing DML workloads is essential for optimizing performance and have gained a lot of attention from researchers. The available characterization works provide valuable insights into specific aspects of DML workloads, however they often lack a comprehensive view of the entire tuning landscape. Indeed, existing characterization work generally highlight the impact of hyper-parameters on model quality or the influence of platform parameters on execution times, but they do not always bridge the gap between these two crucial dimensions of tuning. In this part we shed light on the joint tuning of platform- and hyper-parameters, in order to provide guidance to both data scientists and systems administrators in their workload optimization. We first start by providing background on workload characterization and DML configuration. Then, we review existing literature on the issue and motivate our work. Afterwards, we present our contribution including a comprehensive study of joint configuration impact, DML trace collection and analysis results, and key observations and recommendations for DML operators and data scientists.

3 Background and Related Work

In this chapter, we start by introducing Distributed Machine Learning (DML) workload characterization. We then delve into the Apache Spark platform that we use in our study, exploring its configuration challenges. Subsequently, we present a comprehensive review of existing DML workload characterization studies conducted on Spark and other popular platforms. We highlight the limitations inherent in these studies, and we illustrate these limitations through several use cases empirically. Finally, we briefly introduce our proposed approach to address the identified research problem.

3.1 Background on Workload Characterization in Distributed Machine Learning

The characterization of workloads represents an important step in the design of computer systems including DML systems. It involves developing a deep understanding of the properties of workloads and how they execute, thus providing a better understanding of resource utilization and guiding performance optimization at both hardware and software levels.

In the literature, there are no commonly accepted definitions for the term "workload". However, some authors have mentioned this term in their work and provided their definitions. In [190], the authors define a workload as a task performed by an entity during a given period that requires access to a combination of resources, including the processor, storage, disk input/output, and network bandwidth. Authors in [168] define a workload as a set of workflows, which are sets of tasks. It has been observed that in the literature, although different definitions exist, workloads primarily refer to different sets of tasks performed by an entity.

A workload can be broken down into basic components (*e.g.*, tasks) for which characteristics can be observed. The term "characteristics of workloads" refers to the specific attributes and patterns of the tasks or jobs that a computer system or network is expected to handle. These attributes can dynamically change during the execution or can remain static. Different workloads have different characteristics, and the choice of the best platform to execute a given

workload depends on the nature and characteristics of the workload.

The characterization of workloads involves mainly quantifying and analyzing workloads characteristics. It relies on experimental methods to collect and analyze measurements from infrastructures during their operation and has two main stages: measurement and analysis. Measurements should be specific to the components of interest (*e.g.*, the whole workload, or its tasks) and capture their static and dynamic properties. Various monitoring tools like command-line utilities or profiling tools (*e.g.*, PCM [169]) are used to collect these measurements, but careful attention is needed to manage data volume and avoid impacting workload execution. Sampling of collected data may be necessary to facilitate analysis. Once measurements are collected, data analysis explores workload attributes and characteristics to create models. These models summarize workload properties and allow for the generation of synthetic workloads for performance evaluation. Analysis techniques include statistical analysis, graphical analysis, multivariate analysis (clustering), and modeling data distribution.

3.1.1 Background on Spark-Based Distributed Machine Learning

In our work, we focus on DML workloads. We use Spark as our DML platform. Spark [236] is a distributed computing platform that was originally designed for processing large-scale data. Over time, Spark has evolved to support DML as well, thanks to the introduction of its Machine Learning (ML) libraries, namely MLlib and Spark ML [55]. One of the key features of Spark is its ability to leverage data parallelism (see Chapter 2 §2.3.1) and its dataflow computation model, where computations are represented as a directed graph in which nodes are computations and data flow along the edges. Furthermore, in Spark, intermediate and output results of jobs can be stored in memory, which is known as Memory Computing. This approach greatly enhances the efficiency of data processing, particularly for iterative ML algorithms. Spark represents data transformations as a directed acyclic graph (DAG) composed of multiple data-parallel operators. Each vertex of a Spark DAG represents an RDD (Resilient Distributed Dataset) [237], which is Spark's fundamental data structure, and each edge represents an RDD transformation. An edge directs from earlier to later stages in the RDD transformation. The DAG is submitted to the platform's scheduler to define the transformation stages and manage the corresponding computation tasks. In Spark, ML methods exist as distributed and parallel algorithms provided by libraries like MLlib [139] and BigDL [38]. The most appropriate method for the domain of interest is to be chosen and applied out-of-the-box.

A Spark based system consists of a driver process running inside a central node (known as the master node), along with multiple executor processes running on several nodes (known as worker nodes). The driver takes on the responsibility of scheduling the executors and maintaining the global model while the executors perform the computation on their data partitions. The training process unfolds through several iterations involving each a series of steps. As a first step, the driver broadcasts the current model to all the executors. Secondly, each executor computes updates for the global model using its partitioned data. Thirdly,

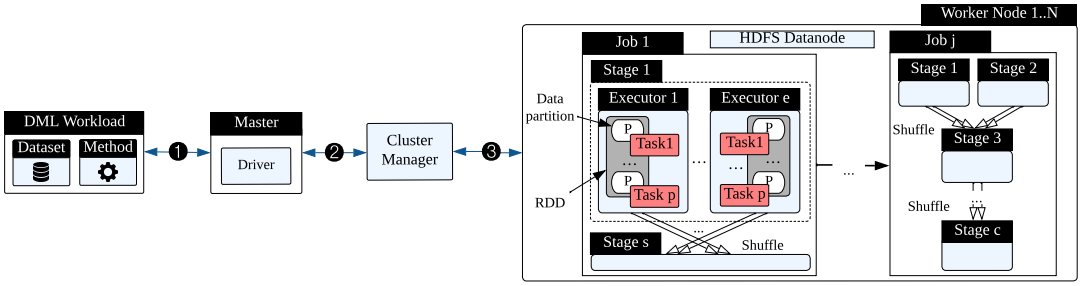


Figure 3.1: Infrastructure of a typical Spark based DML system deployment.

executors send the computed updates back to the driver. Lastly, the driver collects and aggregates the updates collected from the executors, employing them to edit the model in a similar protocol to the parameter server architecture (see Chapter 2 §2.3.2). Once the model is trained, it is cached on each machine and can serve inference requests. Spark manages the distributed training described above by generating a sequence of jobs from the DML workloads. Jobs consist of a set of stages running several parallel and independent computation tasks. Tasks are scheduled and processed by executors. For inference, the workflow is similar. The deployment of a pre-trained model across the cluster is managed by a single Spark job through its broadcast mechanism [223].

Figure 3.1 illustrates a typical Spark based system architecture. It includes a physical cluster composed of a master node and worker nodes, a distributed computing platform like Spark, a distributed data storage system (HDFS [192]) and DML libraries (MLlib [139] and BigDL [38]).

3.1.2 Background on Spark Configuration

Workload characterization serves as the foundation for identifying performance bottlenecks or areas where the system configuration may not effectively meet the workload's demands, thus guiding configuration tuning. Configuration tuning is the process of adjusting system parameters to address these bottlenecks and align the system's configuration with the workload's requirements. DML environments come with many configuration parameters, classified in two categories.

Platform parameters. The list of tunable parameters in distributed platforms is extensive. Hadoop [204] and Spark [7] support more than 50 and 180 configurable platform parameters, respectively. These parameters usually fall in one of the following types: *(i)* memory-related, *(ii)* data representation, *(iii)* scheduling, *(iv)* parallelization, and *(v)* data distribution. The two main aspects of platform parameters considered in most distributed computing systems for learning deal with parallelization degree and memory management. In Spark, one sets the number of executors an application should use, as well as the number of cores to assign

to each of these executors. For memory, Spark lets users define resources not only for the executors but also for the drivers, as well as other memory related parameters, *e.g.*, the ratio between the Java heap and Spark's memory cache.

Model hyper-parameters. These impact directly the training process while being external to the model. Examples include the maximum number of iterations for the convergence of the model [229, 248, 73], the maximum depth of decision trees for decision trees based ML algorithms, or the number of trees in random forests.

3.2 Related Work on Workload Characterization in Distributed Machine Learning

In the following, we present studies related to DML characterization in the context of Spark and then general characterization solutions. Table 3.1 summarizes the related work.

3.2.1 Spark-specific DML Characterization Studies

While there are numerous studies on the characterization of Spark workloads, only few tackle the performance of ML workloads. Te state of the art in DML workload characterization for Spark involves a nuanced analysis of ML algorithm-specific behaviors and understanding of the specific patterns if any. Works focus also on resource requirements of different workloads representative of real world workloads, and challenges associated with running ML workloads on the Spark platform such us Spark parameters tuning. Research in this area aims to provide benchmarks and insights that enable practitioners to optimize resource utilization and improve the efficiency of ML tasks on the Spark platform.

SparkBench [114] is a benchmarking suite, where three ML methods are considered as part of a larger set of applications. It characterizes the workloads in terms of data access patterns, job execution time and system resources consumption. SparkBench supports synthetic datasets and briefly explores the variation of one Spark parameter (the number of cores per Spark executor). Using SparkBench, the authors observed that memory is intensively used across all workloads. Moreover ML workloads in particular are CPU intensive. Finally, the authors observed that the the parallelism degree of workloads impacts the execution time significantly.

In [136], McSherry et al. introduced COST (Configuration that Outperforms Single-Threaded configuration) as a new metric for evaluating DML systems and platforms. This metric describes the configuration under which a distributed solution surpasses a competent single-threaded implementation. Motivated by this example, [15] considers efficient single-threaded implementations of supervised machine learning algorithms and, on the other hand, explores distributed solutions. Through experiments conducted on Spark, it demonstrated that two nodes (8 cores) constitute sufficient hardware congestion to outperform a competent single-threaded implementation on both tested platforms.

Authors in [159] investigate ML based tuning approaches for Spark. they propose a framework that can automatically search and identify the set of recommended Spark platform settings that may improve performance significantly compared to the default settings. Specifically, for a dozen of Spark settings that may affect performance identified by state-of-the-art, authors first use Latin hypercube design strategy to identify a set of configurations to benchmark the system and collect training data. The considered parameters are related to memory management, prallelization degree, and shuffle operations. Next, authors train performance prediction models using collected data in order to use them to predict the most effective configuration. To evaluate their framework, the authors used nine different applications among which ML methods as KMeans, Support Vector Machines, Matrix Factorization, and Decision Trees. The evaluation shows that the framework can improve runtime significantly and the improvement ranges between 22.8% to 40.0% depending on applications.

In [150] the authors analyze ML performance on Spark and compare it to Hadoop [218] in terms of runtime, and memory, network and CPU consumption. They vary the number of Spark nodes and the dataset size and observe the effect on performance. The work focuses only on the K-Nearest Neighbours (KNN) [35] method over a single tabular dataset. The results show that the runtime of the KNN algorithm implemented on Spark is 4 to 4.5 times faster than Hadoop. Spark's ability to hold data in memory makes it suitable for iterative algorithms. This saves I/O (Input/Output) operations of intermediate results, which is a large part of the time wasted on the Hadoop. Moreover, Hadoop uses more resources, including CPU and network due to its longer execution time and disk access. On the other hand, the memory usage in Hadoop is less than Spark due to Spark's in memory computations.

3.2.2 General DML Workload Characterization Studies

Other DML characterization works consider different DML platforms. Works such as [243] and [15], compare DML platforms by running selected workloads in a specific configuration and collecting metrics such as execution time, memory usage, CPU utilization, and network usage. [243] examines the architectural design of some platforms (Spark, TensorFlow, PMLS, and MXNET) and its impact on DML workloads. All these platforms follow the dataflow computation model. They find that for complex ML tasks, especially for training deep neural networks, the basic dataflow model of Spark fails due to its lack of support for mutable state (in-place update) and the iterative nature of these workloads. On the other hand, advanced dataflow systems like MXNET and TensorFlow and parameter server systems like PMLS enable cyclic computation graphs with mutable states which result in better performance. A similar finding regarding basic dataflow systems was reported by [15]. The authors stated that while able to adapt robustly to increasing dataset sizes, basic dataflow systems such as Apache Spark or Apache Flink without mutable states face inefficient execution of learning algorithms on high-dimensional data.

In [29], the authors highlights a gap in the literature concerning CPU-based DNN training,

especially using Horovod with TensorFlow and PyTorch. The research aims to address specific challenges, including the impact of the number of CPU cores on DNN training performance, the influence of batch size on different CPU architectures, the comparison between single-process (SP) per node and multiple-process (MP) per node configurations and which of them better exploits all the available CPU cores, the interplay between CPU clock speed and core count with DNN architecture, and the comparison of CPU and GPU performance for various models and frameworks. The proposal suggests employing a multi-process (MP) approach for single-node training, showcasing superior performance compared to a single-process (SP) approach. The optimal process per node (ppn) configuration for SP depends on the available CPU cores, with specific configurations offering the best performance for Intel and AMD processors in TensorFlow. PyTorch, however, performs optimally with a ppn configuration equal to the number of cores. The proposed MP approach outperforms SP on a single CPU, providing up to 1.35× and 1.47× better performance for TensorFlow in ResNet-152 and Inception-v4, respectively. TensorFlow demonstrates better CPU performance compared to PyTorch, while PyTorch outperforms on GPUs. TensorFlow scales well up to 4 nodes, and the CPU architecture Skylake shows up to 2.35× better performance than K80s, with V100 surpassing Skylake by up to 3.32×. For PyTorch on Skylake, tuning HOROVOD CYCLE TIME provides up to 1.25× better performance for ResNet-50.

The work presented in [214] focuses on tracing the execution of DML workloads using TensorFlow on top of Alibaba's Platform of Artificial Intelligence (PAI), a machine learning-as-a-service platform. The authors characterize various resource requirements and identify performance bottlenecks in the thousands of daily training jobs submitted to PAI, considering diverse computing, communication, and I/O constraints. The analysis reveals that I/O time is non-negligible, especially for single-node training workloads, and may become a potential performance bottleneck for distributed workloads after optimizing gradient communication. The authors establish simple analytical performance models based on key workload features to expose fundamental performance bottlenecks. These models estimate potential performance gains under different software architectures and hardware configurations, focusing on system architecture choices (PS or AllReduce), the benefits of high-speed multi-GPU interconnects like NVLink, and how performance bottlenecks may shift with different configurations. Case studies demonstrate that the estimated performance using analytical methods closely aligns with actual measurements. The authors explore optimization techniques, including mixed-precision training, operation fusion, and changes to system architectures, providing observations and implications for improving practical deep learning training workloads.

In [94], the author address the challenges of managing multi-tenant clusters used for training DL workloads with GPUs. The authors introduce Philly, a service in Microsoft designed for resource scheduling and cluster management for training jobs. They conduct a comprehensive two-month workload characterization, analyzing around 100,000 jobs from hundreds of users. The study explores the impact of factors such as gang scheduling, locality requirements, and failures on cluster utilization. The analysis highlights the influence of waiting for locality constraints on queuing delays and discusses how locality-aware scheduling affects GPU

utilization for distributed training jobs. The paper identifies reasons for job failures, with programming errors being a predominant factor. Based on their findings, the authors provide three guidelines to enhance the next generation of cluster schedulers for deep neural network (DNN) workloads, emphasizing the trade-off between queueing delay and adherence to locality constraints, the isolation of jobs on dedicated servers to minimize interference, and early detection and prevention of failures.

In [216], authors explore the challenges of running DML in large GPU clusters, focusing on the Alibaba PAI. The authors present an extensive two-month workload trace collected from a production cluster with 6,742 GPUs, covering a diverse range of ML algorithms and frameworks submitted by over 1,300 users. The study addresses challenges such as low utilization caused by fractional GPU uses, long queueing delays for short-running task instances, and difficulties in scheduling high-GPU tasks. Solutions include GPU sharing for low-GPU workloads, predicting task durations for improved scheduling, and employing a reserving-and-packing policy for hard-to-schedule high-GPU tasks. The paper also discusses open challenges, including load imbalance in heterogeneous machines and potential bottlenecks on CPUs during data processing and simulation tasks. The insights derived from the analysis aim to inspire further research in optimizing ML workload scheduling and GPU cluster management, providing valuable considerations for system optimization opportunities.

3.2.3 General Distributed ML and DL Benchmarking Solutions

In response to the rapid evolution of DML, numerous recent initiatives have emerged with the goal of establishing generic ML benchmarks suitable for a wide array of software and hardware platforms. One such notable work is MLBench [146], designed specifically for supervised ML benchmarking. MLBench is a public and reproducible collection of reference implementations and benchmark suite for DML algorithms, frameworks and systems. It offers compatibility with popular frameworks such as TensorFlow and PyTorch. While MLBench emphasizes various performance metrics, it does not include detailed execution traces.

TBD [247] addresses the increasing importance of efficiently training deep neural network (DNN) models in various domains, highlighting that existing evaluations often focus on inference and image classification, neglecting the diversity of DNN applications and models. The authors propose a comprehensive benchmarking suite covering six major application domains and eight state-of-the-art models. The benchmark takes into account multiple frameworks, and datasets and places a particular focus on metrics related to training throughput (number of data records processed in a unit of time), CPU/GPU utilization, and memory usage. The benchmark suite is complemented by an analysis toolchain designed for end-to-end performance analysis, including memory profiling tools for major DNN frameworks. The paper addresses three main challenges: the significant differences between training and inference, the need for workload diversity in benchmarks, and the identification of critical bottlenecks in hardware resources during training.

DLBench [59] on benchmarking DL frameworks by considering three main dimensions: the impact of computational environments (CPU, GPU), different types of datasets, and various DL architectures. The study compares six popular DL frameworks (TensorFlow, MXNet, PyTorch, Theano, Chainer, and Keras) evaluating their performance in terms of training time, accuracy, convergence, CPU and memory usage on both CPU and GPU environments. Additionally, it assesses the impact of different DL architectures (CNN, Faster R-CNN, LSTM) on performance and system resource consumption using diverse datasets. The paper provides a holistic approach to benchmarking DL frameworks and makes its source code and detailed experiment results accessible for repeatability.

3.2.4 Discussion

The current state-of-the-art in DML workload characterization exhibits several limitations. Regarding trace publication, to the best of our knowledge, the only related works that have released their traces are Philly [94], DLBench [59], Weng et al.[216], and Yang et al.[226]. However, these traces primarily focus on job descriptions and CPU/GPU/memory consumption.

Published works often lack a comprehensive coverage of classical ML and DL methods, and provide an incomplete view of the tuning landscape. Notably, some works cover various ML libraries parameters but lack a detailed exploration of distributed platforms parameters and vice-versa, neglecting potential interactions between the two groups of parameters. As far as we know, PipeTune [182] is the only work that profiles DML workload executions to jointly tune hyperparameters and infrastructure parameters. Nevertheless, it concentrates on deep learning methods provided by the DL library of Spark, namely BigDL, and doesn't uncover the black-box relationships between tuning the parameters of the platform and the library and the workloads.

Finally, the exploration of the impact of platform parameters on the model quality in diverse DML workloads is limited. [165] observed, in a single specific case of image segmentation, a direct impact of Spark parallelization parameters on model quality. However, as far as we know, no other work has explored this phenomenon more in-depth.

The work we present in the following is the first study considering both classical machine learning and deep learning and analyzing the impact of jointly configuring hyper- and platform parameters. We also explore the impact of platform parameters on the model quality. Our DML traces are released to the community.

3.3 Problem Statement and Propositions

In this section, we motivate our work and present our contributions.

Table 3.1: Overview of related work

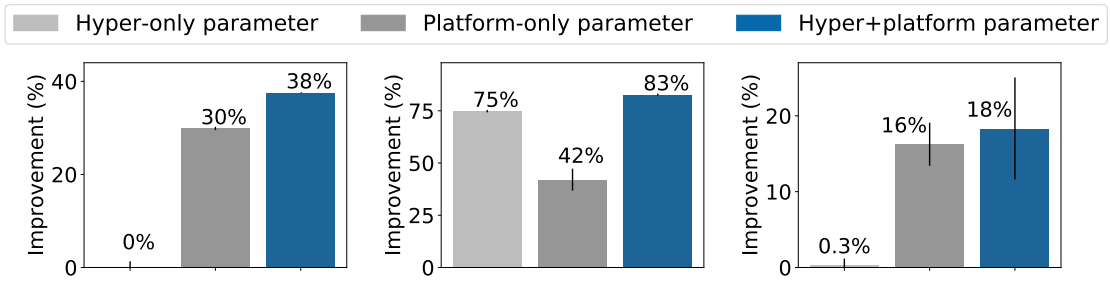
	Machine learning	Deep learning	Published traces	Training	Inference	Hyper-parameters	Platform parameters	Distributed platform	ML library
SparkBench [114]	✓	✗	✗	✓	✗	✗	✓	Spark	MLLib
Mostafaeipour et al. [150]	✓	✗	✗	✓	✗	✓	✗	Spark, Hadoop	Ad hoc implementation
Nguyen et al. [159]	✓	✗	✗	✓	✓	✗	✓	Spark	MLLib
PipeTune [182]	✗	✓	✗	✓	✓	✓	✓	Spark	BigDL
MLBench [146]	✓	✗	✗	✓	✓	✗	✓	Google Kubernetes	TensorFlow, PyTorch
TBD [247]	✗	✓	✗	✓	✓	✓	✓	TensorFlow, MXNet, CNTK	
DLBench [59]	✗	✓	✓	✓	✓	✗	✗	TensorFlow, MXNet, PyTorch, Theano, Chainer, Keras	
MLPerf [147]	✗	✓	✗	✓	✓	✓	✗	diverse	Ad hoc implementation
Philly. [94]	✗	✓	✓	✓	✗	✗	✓	Philly	Ad hoc implementation
Mengdi et al. [214]	✗	✓	✗	✓	✗	✗	✗	PAI	TensorFlow
Weng et al. [216]	✗	✓	✓	✓	✓	✗	✗	PAI	TensorFlow, PyTorch, AliGraph
Yang et al. [226]	✗	✓	✓	✓	✓	✗	✗	CloudBrain-I	TensorFlow, PyTorch
Our work	✓	✓	✓	✓	✓	✓	✓	Spark	MLLib, BigDL

3.3.1 Problem Statement

DML environments are challenging to configure as they come with many configuration parameters. Several tools exist to fine-tune the hyper-parameters of learning methods [64, 34, 97] and the configuration parameters of the underlying distributed computing platforms [158, 85]. These two tasks, usually conducted separately, require different expertise: data scientists for the former, system administrators for the latter. We claim that DML workloads would benefit from more advanced tools for joint configuration of hyper- and platform parameters, leading to better performance optimization.

To illustrate the issue, in the following we consider three examples of distributed learning workloads. For each one of them, we set a performance objective (*e.g.*, model accuracy, model training time or inference throughput), comparing the following configuration strategies: *(i)* tuning only hyper-parameters of the learning method as done by the data scientist, *(ii)* tuning only platform parameters as done by the system administrator and *(iii)* jointly tuning hyper-parameters and platform parameters in a coordinated approach between the data scientist and system administrator. In the rest of the chapter, strategies *(i)* and *(ii)* are referred to as single-level tuning strategies, because each of them only handles the configuration tuning of parameters related to one level of the DML system, either the application (ML methods) level, or the platform (DML platform) level. On the other hand, strategy *(iii)* is referred to as a multi-level tuning strategy as it involves tuning parameters of different levels of execution. We show to which extent these strategies improve the baseline system with its default configuration. For each case, we show the configurations yielding the best performance for the considered objective. The error bars represent the 95% confidence intervals of the represented values, and are almost equal to zero due to low variation in data. Default Spark configuration values are shown in Chapter 4 §4.2. Further details (*i.e.*, tuning strategy, experimental setup, DML learning methods, datasets, *etc.*) are given in Chapter 4.

Figure 3.2(a) shows the model quality of the Gradient-Boosted Tree method on the DDF dataset. The model quality is measured by the R^2 coefficient as it is used for regression models. In the best-case scenario, the modeled and observed values are identical, resulting in a R^2 of 100%. Tuning hyper-parameters (*i.e.*, depth of the decision tree, number of iterations) has no effects on the model quality compared to the baseline. Tuning platform parameters provides



(a) Model R^2 (b) Training time (c) Inference tput.

Figure 3.2: Comparing different configuration strategies

an improvement of +30%. Surprisingly, for this workload, platform parameters have much higher impact on model quality than hyper-parameters. This contradicts the general belief that model quality should be improved through hyper-parameter tuning as usually done by data scientists. Furthermore, combining both levels of configuration of platform parameters and hyper-parameters improves the model quality by +38%.

Figure 3.2(b) illustrates the case of the workload running MLP (Multi-Layer Perceptron) on the DDF (Driveface) dataset, atop the Spark platform. We compare the improvements of the model training time for the three configuration strategies against the baseline. Tuning hyper-parameters (*i.e.*, number of iterations and the convergence threshold) improves the training time by +75%. Intuitively, decreasing their values implies less computations and, thus, decreases the training time. Tuning the platform parameters achieves up to +42% improvement. Indeed, increasing the parallelization degree (*i.e.*, number of cores or Spark executors) accelerates the computation. Combining both levels of configuration leads to the best improvement, reducing the training time by +83%.

Figure 3.2(c) considers the inference throughput of a workload that runs K-means on the Higgs dataset. We see that the hyper-parameters, including the number of iterations and convergence tolerance, improve inference throughput by 0.3%. Instead, platform tuning results in a better improvements (+16%). The reduction of the execution time due to parallelization translates into an improved inference throughput. Nonetheless, a coordinated configuration of hyper-parameters and platform parameters, achieves an improvement of +18%, better than the single-level approach. These use cases highlight the potential benefits of multi-level tuning for DML workloads with regard to several aspects including model quality, training time, and inference throughout.

3.3.2 Our Contributions

To clarify the interplay between platform parameters and hyper-parameters, and provide guidance to both data scientists and system administrators, we carry out an extensive workload characterization. We rely on multiple DML workloads composed of 13 ML algorithms and 6 ML datasets, considering both classical ML and deep learning algorithms. We compare

different configuration strategies against several criteria, including model quality, training time, inference throughput, and resource costs. We consider the performance metrics in isolation before studying the trade-offs. Our study makes the following contributions:

- We perform the first comprehensive study of the joint impact of both hyper-parameters and platform parameters on different types of DML workloads. We consider three strategies of tuning: only hyper-parameters, only lower-level platform parameters, and jointly tuning both. We study the performance metrics in isolation and their trade-offs. Noteworthy, we are the first to consider the DML general case and report that multi-level parameter configuration (*i.e.*, hyper-parameters and platform parameters jointly tuned) improves model quality and training time, while also optimizing resource costs.
- We collect and release (see [49]) traces from our extensive experiments leveraging two popular DML libraries (MLlib [139], BigDL [38]) atop two Spark clusters [236]. Our statistical analysis can help the future development of modeling and simulation tools of DML workloads, as well as tools for synthetic DML trace generation.
- We derive several observations and key takeaways concerning the characteristics of DML workloads. We also provide a number of recommendations for both DML service operators and data scientists. We release our deployment code to facilitate reproducibility of our results.

3.4 Summary

In the realm of DML, workload characterization plays a significant role. It provides crucial insights into the behavior and resource utilization patterns of DML tasks, aiding system designers, researchers, and practitioners in optimizing performance and resource allocation. However, while several DML workload characterization works exist, they often exhibit limitations in terms of trace publication, coverage of classical ML and DL methods, and exploration of the whole and complex tuning landscape of DML environments.

This chapter presented background and related work on DML workload characterization and identified an important open research problem. In the upcoming chapter, we will delve into our first contribution, a workload characterization study on Spark, building upon the limitations identified in the current state-of-the-art. Leveraging the released traces from our work, we aim to provide a comprehensive view of tuning classical ML and DL methods in a distributed setting. Through empirical analysis and experimentation, we will study the relationships between hyperparameters, platform parameters, and the behavior of popular DML workloads. This exploration will contribute valuable knowledge to the field, addressing gaps in understanding DML and paving the way for more effective DML systems.

4 Characterization Methodology

In the following, we present our DML workload characterization methodology conducted on Spark platform. We first describe the DML workloads we consider (§4.1) and the configuration parameters (§4.2). We also detail the exploration methodology of the configuration parameters space to understand their impact on the workloads' performance. Then, we present the performance metrics of interest (§4.3) and the hardware setup on which the characterization study is run (§4.4).

4.1 DML Workloads

We consider a workload running on a DML system as a tuple consisting of a *dataset* and a *learning method*.

ML Datasets. We consider six commonly used and publicly available datasets [156, 221, 44, 46, 213, 11] shown in Table 8.1 (the CC columns in Tables 8.1 and 4.2 indicate the color codes used later in the graphs). We selected datasets that vary in terms of content type (*e.g.*, text, images), number of records, number of features, and total size, in order to collect heterogeneous traces. We use random split to define the training and inference sets. 80% of each dataset is used for model training, the remaining 20% for inference.

ML Methods. We test 13 state-of-art ML methods commonly used by data scientists including 9 MLlib's methods and 4 BigDL's methods (see Table 4.2). The MLlib methods implement clustering, classification or regression and are based on different learning methods such as means, decision trees, and linear regression. BigDL provides deep neural networks (DL), we consider several methods with the following architectures: a CNN consisting of 9 layers, a GRU with 7 layers, a LENET5 (a specific CNN for the MNIST dataset) with 5 layers and a LSTM with 7 layers.

In the rest of the chapter, the names of the workloads are composed of the name of the dataset followed by the name of the learning method. For example, for the `DDF` dataset and the

Table 4.1: Learning datasets

CC	Dataset	Description	#Records	#Features	Size
	DDF (Drive-face)	Images sequences of subjects while driving [46].	606	6 400	19.9 MB
	DGS (Drift)	Measurements from 16 chemical sensors utilized in a discrimination task of 6 gases [213].	13 910	129	40.3 MB
	DHG (Higgs)	A collection of kinematic measures to detect signal processes which produce Higgs bosons [11].	11 000 000	28	7.5 GB
	DN (News20)	Messages collected from 20 different newsgroups [156].	118 845	2	68.7 MB
	DFM (fashion MNIST)	Images of fashion articles, associated with labels from 10 classes [221].	700 000	784	54.9 MB
	DM (MNIST)	Handwritten digit images for ML research [44].	70 000	784	52.4 MB

Table 4.2: Learning methods

Library	Category	ML Method	CC
MLlib	Clustering	KM (K-Means)	
		BKM (Bisecting K-Means)	
		GMM (Gaussian Mixture Model)	
	Classification	DT (Decision Tree)	
		MLP (Multilayer Perceptron)	
		BLR (Binomial Logistic Regression)	
	Regression	LR (Linear Regression)	
		RFR (Random Forest Regressor)	
		GBT (Gradient-Boosted Tree)	
BigDL	Classification	CNN (Convolutional Neural Network)	
		GRU (Gated Recurrent Unit)	
		LENET5 (Convolutional Neural Network)	
		LSTM (Long Short-Term Memory)	

clustering methods `KM`, `BKM`, and `GMM`, we have the `DDF-KM`, `DDF-BKM` and `DDF-GMM` workloads.

4.2 Parameter Settings

We deploy each workload (*i.e.*, dataset and ML method) on the corresponding DML environment (*i.e.*, MLlib or BigDL on a Spark cluster). For each deployed workload, we investigate the performance variation under the tuning of the learning method hyper-parameters and Spark platform parameters. We consider during the tuning default values of hyper-parameters and Spark platform parameters, and variations for hyper-parameters and variations for different

Spark parameters. Each experiment is replicated three times, which is enough due to low variations in data (see the confidence intervals in our plots).

Hyper-parameters. Based on previous results [67, 179, 175], we choose hyper-parameters with high impact on performance in terms of execution time and accuracy. These include: the maximum depth of tree-based algorithms, the maximum number of iterations to reach model convergence, and the learning rate and batch size of deep neural networks. Table 4.4 provides further details about the hyper-parameters and their values.

Platform Parameters. We leverage existing studies [131, 7] that evaluate which Spark parameters affect performance most. These include scheduling, data transfers, data storage and representation, parallelization and memory management. Details about the used configuration parameters and their values are in Table 4.3.

In most cases, ML experts optimize hyper-parameters while system administrators configure the underlying distributed computing platforms. However, advanced tools that combine hyper-parameter optimization with platform configurations could potentially increase the performance of DML workloads. To investigate this, we consider and evaluate three tuning strategies, *(i)* tuning only hyper-parameters of the learning method as done by the data scientist, *(ii)* tuning only platform parameters as done by a system administrator and *(iii)* jointly tuning hyper-parameters and platform parameters. In §5.2 we compare these three strategies.

The adopted tuning process depends on the strategy objective. Strategy *(i)* (respectively *(ii)*) assesses the impact of individual hyper-parameters (respectively platform-parameters). As such, we varied one parameter at a time while keeping the others at their default Spark values. In strategy *(iii)*, we study potential interdependencies between platform parameters and hyper-parameters. To achieve this, we vary the platform parameters and the hyper-parameters together, akin to grid-search tuning [120]. In practice, the exploration process may be helped by tools like Pipetune [182] which integrates system parameters as hyper-parameters and thus varies them together. The process may be sped up by automation tools for parallel exploration of parameter combinations. A collaboration between data scientists and system administrators is necessary to select relevant parameters and their values' range.

4.3 Characterization Metrics

Application-level Metrics. Application-level metrics capture different aspects related to DML applications, including the execution time and the quality of the underlying model. We report the *training time* and normalize it per thousand records. We report the inference execution time in the form of *inference throughput, i.e.*, the number of requests processed per second (reqs/s). The quality of a classification model is typically measured through *training accuracy* [180], to measure how well the trained model fits the training data. Some methods such as clustering algorithms (*e.g.*, K-Means) use the *silhouette* metric to evaluate the similarity

Table 4.3: Spark platform parameters

Configuration aspect	Spark platform parameters	Description	Values
Parallel computing	EXEC_NUM (executor.instances)	Number of executors	Single-level tuning: (One executor per node), MLlib: 1, 2, 3, 4, 5, 6, 7, 8, 12, 48, 72, 96; BigDL: 1, 2, 4 Multi-level tuning: (One executor per node), 2, 4, 6, 8
	EXEC_COR (executor.cores)	Number of cores per executor	Single-level tuning: (4 for Cluster1, 8 for Cluster2), MLlib: 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16; BigDL: 1, 2, 4, 8 Multi-level tuning: (All node cores), 1, 2, 4
Memory management	EXEC_MEM (executor.memory)	Amount of memory per executor	(1 GB), MLlib: 5 GB, 10 GB, 15 GB, 20 GB, 25 GB, 30 GB., 50 GB., 70 GB., 100 GB.; BigDL: 4 GB, 8 GB, 16 GB, 24 GB, 32 GB
	MAX_SIZ_INF (reducer.maxSizeInFlight)	Maximum size of map outputs to fetch simultaneously from reduce tasks	12 MB, (48 MB), 72 MB, 128 MB, 256 MB, 512 MB
	PD_BUFS (shuffle.io.preferDirectBufs)	Must use off-heap buffers to reduce garbage collection during data transfer	(true), false
	STR_MEM (storage.memoryFraction)	Fraction of Java heap to use for Spark's memory cache	10%, 20%, 40%, (60%), 80%
Data compression	COMP_CODEC (io.compression.codec)	Codec to compress internal data such as RDD partitions, shuffle outputs, *etc.*	snappy, lz4
	RDD_COMP (rdd.compress)	Must compress serialized RDD partitions	true, (false)
	SHF_SPL_COMP (shuffle.spill.compress)	Must compress data spilled during shuffles	(true), false
Scheduling	LOC_WAIT (locality.wait)	How long to wait to launch a data-local task on a less-local node	10 ms, 100 ms, 500 ms, 1 s, (3 s), 10 s
Serialization	SER (serializer)	Data serialization mechanism	(Java), Kryo
Shuffle	SHF_COMPR (shuffle.compress)	Must compress map output files	true, false
	SFL_BUF (shuffle.file.buffer)	Size of in-memory buffer of shuffle file output stream	8 KB, (32 KB), 64 KB, 128 KB, 256 KB, 512 KB

Table 4.4: Hyper-parameters

Hyper-parameters	Description	Values	MLlib / BigDL method
maxIter	Maximum number of iterations to achieve convergence.	5, 10, 15, 20, 50, 100	BLR, MLP, BKM, KM, GMM, GBT, LR, CNN, GRU, LSTM, LENET
maxBins	Number of classes for the discretization of continuous variables	4, 16, 32, 48	DT, GBT, RFR
maxDepth	Maximum depth of decision trees before convergence	5, 10, 15, 20	DT, RFR, GBT
tol	Convergence threshold for stopping the algorithm	0.000001, 0.0001, 0.01, 0.1	KM, GMM, MLP, BLR, LR
numTrees	The number of decision trees built	10, 20, 50, 100, 150	RFR
stepSize	Model learning rate	0.003, 0.03, 0.3	GBT, MLP, CNN, GRU, LSTM, LENET
blockSize	Batch size	32, 128, 256, 512, 1024	MLP, CNN, GRU, LSTM, LENET

of an object to its own cluster's objects. Other metrics used with regression algorithms include *R-squared (R^2), root mean squared error (RMSE)* and *mean absolute error (MAE)* [129].

Platform-level Metrics. We use *SparkMeasure* [195] to collect metrics from the Spark cluster. The reported metrics include: *(1)* task duration (ms), *i.e.*, the total time to perform the task;

Table 4.5: Examples of collected metrics

Application-level metrics	**Description**
Training accuracy	Percentage of predicted values that match the actual values in the training dataset
Inference throughput	Number of records inferred by unit of time
Silhouette	Evaluation metric for clustering methods. It measures how similar an object is to its own cluster
R2 Score	Proportion of the variance in the dependent variable that is predictable from the independent variable(s)
RMSE (root mean square error)	The square root of MSE
F1 score	Harmonic mean between precision and recall
Platform-level metrics	**Description**
Task duration	Total elapsed time
Task deserialization time	Elapsed time spent to deserialize this task
Garbage collection time	Total JVM garbage collection time
Result serialization time	Elapsed time spent serializing the task result
Shuffle wait time	Time that tasks spent blocked waiting for shuffle data to be read from remote machines
Records read	Total number of records read
Infrastructure-level metrics	**Description**
CPU usage	Percentage of used CPU.
Memory usage	Percentage of memory utilization
Network traffic	Amount of bytes read from and written to the network.
Energy consumption	Trapezoidal integral of power measurements collected per second.

(2) task deserialization time (ms), *i.e.*, the time spent to deserialize a given task; *(3)* shuffle time (ms), *i.e.*, the time spent to transfer data between tasks and stages; and *(4)* JVM garbage collection time (ms).

Infrastructure-level Metrics. From each cluster node, we collect CPU usage (from `/proc/stat`), the memory usage (from `/proc/meminfo`), the network traffic (using `/proc/net/netstat` and filtering sent/received bytes), and the energy consumption. We collect power measurements every second via the processor counter monitor API [169].

4.4 Experimental Setup

We use Spark 2.4.0 as distributed computing platform and HDFS 2.7.7 as distributed file system. The used DML library is MLlib (v2.4.0) [139] or BigDL (v2.4.0) [38]. We conduct our experiments on two clusters as described below.

Cluster 1. Cluster 1 is a 4-nodes cluster equipped with a quad-socket Intel E3-1275 CPU processor, 8 cores per CPU, 64 GiB of RAM, 480 GB SSD drives, on a switched 1 Gbps Ethernet LAN, running Ubuntu Linux 16.04.1 LTS.

Cluster 2. To consider more hardware diversity and run bigger workloads, we also deploy a 24-nodes cluster, dual-socket 8 core Intel Xeon E5-2630 CPU, 128 GB RAM, 600 GB HDD, 2× 10 Gbps Ethernet, running Debian GNU/Linux 9.7.

Why DML on CPUs Still Matters.

In our work, we leverage general purpose CPUs to execute DML workloads. While the use of

specialized hardware, *e.g.*, GPUs and TPUs, may lead to remarkable performance improvements for DML, CPU-based clusters remain a preferred choice in many cases. Indeed, CPUs come with established performance in terms of models' quality, execution time and inference throughput. Recent studies [144, 117, 161, 125, 22] indicate that CPUs remain competitive and may even surpass GPUs in terms of performance, making CPU clusters a cost-effective solution for a broad range of application domains. [161] and [125], for example, put forward CPUs' mature development ecosystem, recent hardware advancements for ML treatments and latency benefits. They successfully scale DNN inference and respectively obtain 3,94x and 1,6x speedups for convolution computations and DNN model compression. [144] and [117] show that CPUs are better suited for applications that need more memory, have irregular memory accesses and varying degrees of parallelism. Examples of such applications are multi-dimensional convolutions and digital pathology.

Another major advantage of CPUs compared to dedicated HW accelerators is the lower cost in terms of both initial investment and maintenance. Using general-purpose CPUs in a data center or using cloud service is usually less expensive and more prevalent than using specialized hardware [9]. CPUs are characterized by their ease of access, their portability, and their scalability.

Finally, CPUs remain the frugal choice for mobile devices and at the edge, as they typically consume less power than GPUs [145, 144].

4.5 Summary

In this chapter, we presented a detailed DML workload characterization methodology. The chapter begins by providing an overview of the DML workloads considered, along with a discussion of the relevant configuration parameters. The exploration methodology employed to understand the impact of these configuration parameters on workload performance is then outlined. Additionally, the chapter introduces the performance metrics of interest. The following chapter presents the results of the execution of the characterization methodology, describing the collected traces and offering insights derived from them.

5 Experimental Results

In this chapter, we present the results and findings obtained from our DML workload characterization study conducted on the Spark platform. The chapter begins by offering insights into the identified patterns, trends, and performance nuances observed during the analysis of the collected traces. Afterwards, we delve into the intricate details of the experimental outcomes, shedding light on how various configuration parameters and tuning strategies impact the performance of DML workloads. The chapter also discusses unexpected observations encountered during the traces analysis.

5.1 Traces Overview and Analysis

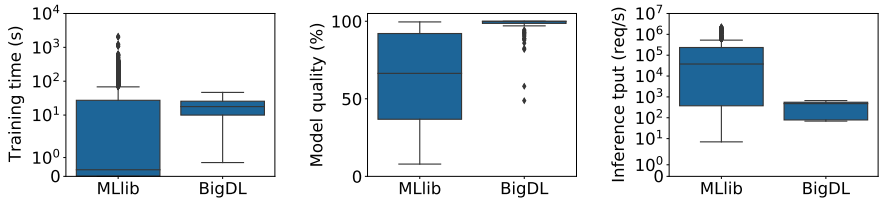
First, we suggest possible uses of the traces collected during our experiments. Then, we present their statistical features.

5.1.1 Potential usages

Synthetic Trace Generation. Using our traces' statistical profiles, *e.g.*, the distribution of the number of tasks and their duration or the distribution of CPU and memory usage, one can generate artificial traces that mimic real-world behaviors. Such traces can be used for resource utilisation analysis, performance analysis, troubleshooting, etc. Thus one can reason about DML systems, without the need for real experimentations. We refer to [25, 106, 70] for tools and techniques for synthetic trace generation from real traces.

Table 5.1: Collected DML workload traces

Trace description	#Records	#Features	Size
Infrastructure-level traces	43,346,092	11	5.3 GiB
Platform-level traces	41,481,857	42	10.8 GiB
Application-level traces	12,934	18	9.6 MiB
Total	**84,840,883**	**Up to 42**	**16.2 GiB**



(a) Training time (b) Model quality (c) Inference tput.

Figure 5.1: Distribution of application-level metrics

Workload Modeling and Simulation. Our traces can be used to build abstract models that simulate DML workloads. These models capture workload patterns, that can be used to predict performance, characterize DML workloads, optimize existing DML platforms, and even design new platforms embedding the workloads' specificities. The analysis and the modeling of workloads have already been considered in different application domains including data centers, the cloud and the web [69, 127, 171].

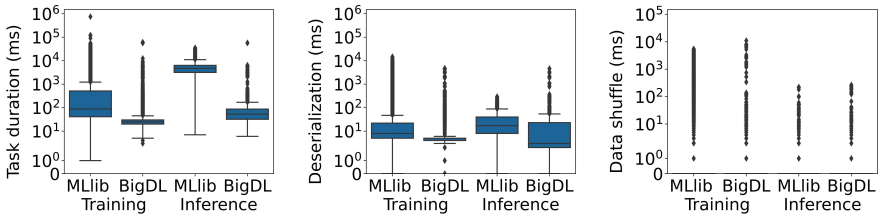
5.1.2 Statistical Analysis

We have harvested metrics at the application-, platform- and the infrastructure levels. Our traces have been collected on both clusters 1 and 2 (see §4.4). All in all, they consist of 16.2 GiB of data with more than 80 millions records (see Table 5.1). The traces are available on a public archive for the research community [49]. We employ box-and-whiskers plots to report the statistical distribution of our metrics. The values are grouped by the ML library used in the experiments (MLlib or BigDL) and by the learning phase (training or inference).

Application-level Traces. Figure 5.1 reports the statistical distribution for three of the collected application-level metrics, *i.e.*, training time, accuracy and inference throughput. Figure 5.1(a) shows normalized training times for a training set of 1,000 records. In our experiments: 50% of the MLlib cases have very short training times ($\leq$ 0.4 s) and 25% have training times between 25 s and 30 min. BigDL training times span between 4 s and 47 s. MLlib exhibits high variation while BigDL is more stable. The greater dataset and learning methods heterogeneity in our MLlib workloads compared to BigDL explains these behaviours.

Figure 5.1(b) gives the models quality metrics: accuracy for classification models, R^2 for regression models, and silhouette for clustering models. The median model quality for MLlib workloads is 66.4%, and up to 99.5%. For BigDL, it is between 98% and 100% for 75% of the workloads. BigDL methods are deep learning methods that have the ability to discover hidden patterns in the training data, leading to more representative models of the training data compared to MLlib classical ML methods.

Finally, Figure 5.1(c) shows that the median inference throughput of our BigDL workloads (476 reqs/s) is less than that of MLlib workloads (around 37,747 requests/s). Indeed, classical



(a) Task duration (b) Deserialization time (c) Data shuffle

Figure 5.2: Distribution of platform-level metrics

ML inference is cheaper and more time-efficient than DL-based inference. In DL, the learned models are more complex than classical ML [89]. The cost gets proportional to the learned DL network complexity.

Platform-level Traces. Figure 5.2 represents the distribution of 3 Spark metrics: *(a)* task duration *(b)* task serialization duration, which is the necessary time for loading tasks by executors, and *(c)* data shuffling duration, which corresponds to data transfers time. Results include both the training and inference phases. As shown in Figure 5.2(a), short tasks (in the 1–100 ms range) are very common with BigDL with up to 79% of the tasks. Less common for MLlib, they are up to 50% for training, and even less for inference where 75% of tasks last at least 3,000 seconds. Longer tasks are more frequent in the inference phase for both MLlib and BigDL. Inference tasks that last more than 100 ms represent 21% for BigDL and up to 98% for MLlib. In training, such tasks represent only 3% for BigDL and 47% for MLlib.

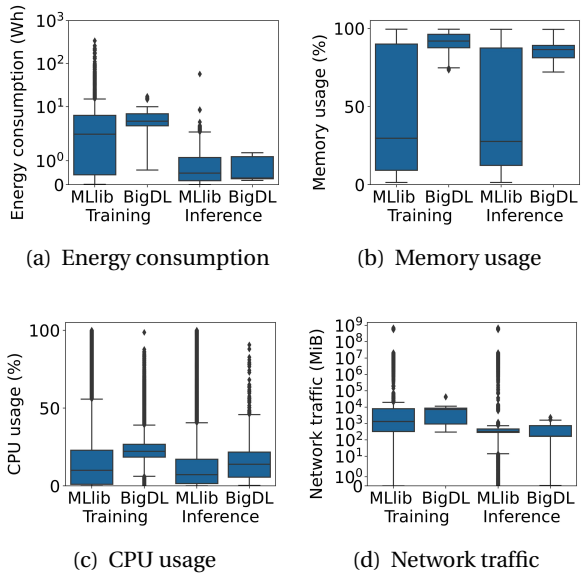
Task deserialization (Figure 5.2(b)) is very fast (*i.e.*, ≤ 10 ms) in 75% of BigDL training tasks, and in 25% to 50% for the other tasks. It is longer (10 ms-10,000 ms) in 75% of MLlib inference tasks but at maximum 50% of the other tasks.

Data shuffling (Figure 5.2(c)) is negligible for 75% of all tasks. However, it reaches up to 10,000 ms for training tasks and up to 100 ms for inference tasks, *i.e.*, it can be equal to the duration of the whole task (Figure 5.2(a)).

Infrastructure-level Traces. Figure 5.4 reports the infrastructure measurements of energy consumption, network traffic, CPU and memory usage. Figure 5.4(a) indicates that up to 25% of MLlib workloads consume very little energy *i.e.*, ≤ than 0.4 Wh. BigDL inference executions consume at least 0.2 Wh and BigDL training at least 0.6 Wh. However, BigDL workloads consume at most 17 Wh whereas the consumption reaches up to 340 Wh for MLlib. This is directly related to the longer MLlib executions.

Regarding memory usage, Figure 5.4(b) shows all our BigDL workloads to be memory-intensive, with at least 70% of memory usage. Also, our workloads are memory-bound and not CPU-bound as CPU usage does not exceed 30% in 75% of all measurements (Figure 5.4(c)).

Finally, Figure 5.4(d) shows collected measurements of network traffic. We observe that



(a) Energy consumption (b) Memory usage

(c) CPU usage (d) Network traffic

Figure 5.3: Distribution of infrastructure-level metrics (CPU usage, memory usage, network usage and energy consumption)

inference for both MLlib and BigDL workloads involves the lowest network traffic: 75% of inference executions consume at most 800 MiB. In contrast, in 50% of MLlib and BigDL training network traffic exceeds 1 GiB. This is due to the per-iteration exchanges needed to build the model.

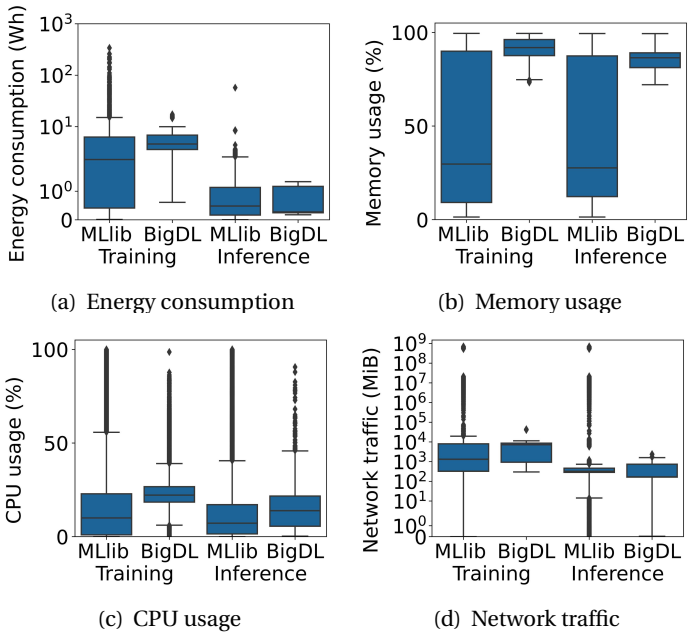
5.2 Characterizing DML Workloads

To characterize the sensitivity of DML workloads to different configuration parameters and strategies, we study the effect of varying only platform parameters (§5.2.1) and of varying only hyper-parameters (§5.2.2). We then compare single-level *vs.* multi-level parameter configuration strategies (§5.2.3) and analyze unexpected behavior with some distributed learning workloads (§5.2.4). These strategies and performance metrics are first considered in isolation, then their trade-offs are discussed in §5.2.5 which tackles multi-level and multi-objective tuning in the context of AI-as-a-Service.

These results concern both Cluster 1 and Cluster 2 (see §4.4). Due to space limitations, §5.2.3, §5.2.4 and §5.2.5 focus on a subset of workloads.

5.2.1 Tuning Platform Parameters

We characterize the impact of platform parameters on training time and inference throughput by varying each parameter individually, for each DML workload. We measure the relative performance variations obtained while tuning each platform parameter. We consider that pa-



(a) Energy consumption (b) Memory usage

(c) CPU usage (d) Network traffic

Figure 5.4: Distribution of infrastructure-level metrics (CPU usage, memory usage, network usage and energy consumption)

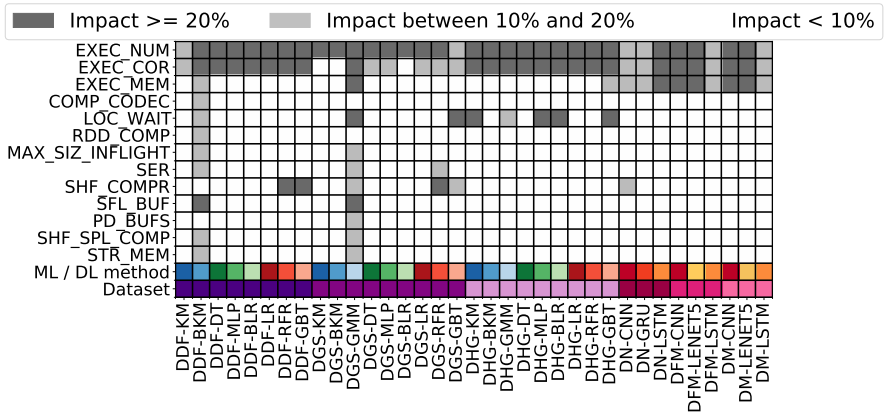
rameters have high impact on performance if their corresponding variation ≥ 20%. Parameters which incur variations in the 10%-20% range have medium impact. Finally, if the variations are ≤ 10%, we consider those low impact. We present the results in the heat map representations of Figures 5.5(a) and 5.5(b), where we use a 3-color scheme (shades of grey), from dark grey (high impact) to white (low impact). The considered platform parameters (vertical axis) are introduced in §3.1 (additional details of the parameters values are in Table 4.3).

For the majority of the workloads, the number of Spark executors (`EXEC_NUM`) and the number of cores per executor (`EXEC_COR`) play key roles, as they control the parallelization, impacting the performance particularly while training.

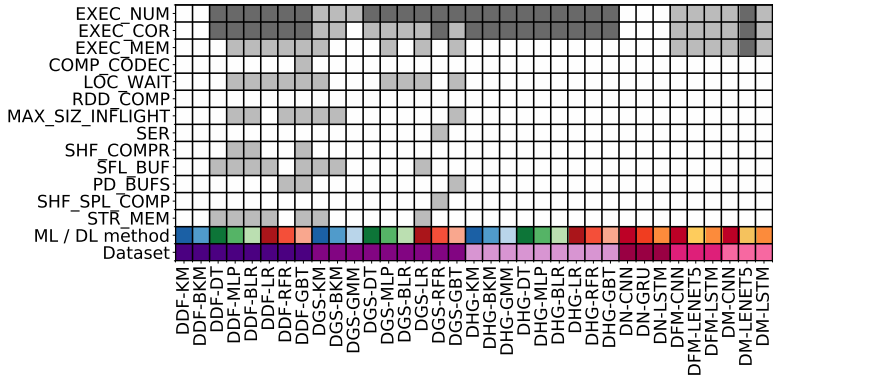
In addition, the default behavior of Spark with regard to these parameters is to create as many executors as there are nodes, and to exploit all the available cores on a node. However, our experiments have demonstrated that this configuration is not optimal for several workloads, such as those involving DDF. Therefore, it is advisable to perform tuning in order to determine the most suitable degree of parallelization for each specific workload. Our first observation concerning DML workloads is thus the following.

Observation 1 : *DML workloads' performance is significantly impacted by parallelization.*

Figure 5.5(a) shows that for ensemble learning methods (*e.g.*, random forests and gradient-boosted trees) run on datasets with many features (from 100 and beyond) the training phase is highly impacted by the `SHF_COMPR` parameter.



(a) Training time



(b) Inference throughput

Figure 5.5: Impact of platform parameter on performance (dark-grey for high: > 20%, light-grey for medium: 10% ≤ 20%, white for low: < 10%)

Examples of high impact of `SHF_COMPR` include `DDF-RFR`, `DDF-GBT`, `DGS-RFR`. `DGS-GBT` exhibits an impact close to the high impact threshold (19%). These workloads trigger frequent shuffle operations. According to our experiments, keeping the default Spark's shuffle compression behavior, *i.e.*, `SHF_COMPR` set to `True`, generally reduces the size of data during shuffle, causing less network traffic and therefore faster training time.

Observation 2: *Training ensemble learning methods on datasets with large number of features significantly benefits from shuffle data size reductions.*

Our large datasets highlight that `LOC_WAIT` (*i.e.*, the timeout after which a data-local task is launched on a distant node) can significantly affect the training time, in particular for jobs that deal with large amounts of data. Examples are workloads with the Higgs dataset ((DHG) and four methods for clustering and classification (`DHG-KM`, `DHG-MLP`, `DHG-BLR` and `DHG-GBT`).

For workloads such as `DHG-BLR` and `DHG-GBT`, increasing this parameter from 10 ms to 1 s or 3 s (which is the default value, as presented in Table 4.3) allows to wait more for a task to be

run on the same node where its data is before sending it to another node. This prevents huge data transfers and shuffles that consume time and bandwidth, and thus significantly impacts efficiency.

Observation 3 : *Training with large datasets is significantly impacted by task re-scheduling.*

Figure 5.5(b) shows that in addition to parallelization parameters, the `MAX_SIZ_INFLIGHT`, `SFL_BUF` and `STR_MEM` parameters also impact the performance. `MAX_SIZ_INFLIGHT` controls the maximum size of shuffled data and thus affects network communication. Setting it too low may lead to network bottlenecks, while setting it too high may increase memory usage and resource contention, indirectly impacting inference throughput. For example, in our experiments, the highest tested value for `MAX_SIZ_INFLIGHT`, *i.e.*, 512 MB, yielded the best performance for `DDF-MLP` while it yielded the worst performance for `DGS-BKM` (all tested values are presented in Table 4.3). The default value of this parameter (*i.e.*, 48 MB) was not the best value for most workloads. Thus, tuning is necessary to find the best value of this parameter and this for each workload. `SFL_BUF` defines the buffer size for reading shuffled data. A smaller buffer increases disk I/O, negatively affecting inference throughput, while a larger buffer may increase memory usage to affect cluster performance. `STR_MEM` manages heap space for caching and shuffle data. Setting it too high may limit resources for inference, while setting it too low can impact query performance and inference throughput. Similarly to `MAX_SIZ_INFLIGHT`, both `SFL_BUF` and `STR_MEM` need tuning to find the best value for each workload.

5.2.2 Tuning Hyper-parameters

We investigate the impact of hyper-parameters on our models' quality considering accuracy for classification tasks, R^2 coefficient for regression tasks and silhouette score for clustering tasks. We vary individually several hyper-parameters, such as the number of iterations, the number of classes for discretization of continuous variables and the depth of decision trees (see the full list in Table 4.4).

In Figure 5.6 we distinguish: *(i)* high-impact hyper-parameters for variations beyond 5%, *(ii)* medium-impact for variations in the 5%-1% range, and *(iii)* low-impact parameters, for variations ≤ 1%. Note that empty (*i.e.*, white) cells indicate that the corresponding hyper-parameters have low impact or are irrelevant for the target learning methods. We observe that the *maxIter* hyper-parameter, *i.e.*, the parameter giving the maximum number of iterations for a given learning method, has high impact on the quality of several methods like `BKM` (*e.g.*, `DDF-BKM`, `DGS-BKM` and `DHG-BKM` workloads) and `MLP` (*e.g.*, `DGS-MLP` and `DHG-MLP` workloads). Further, *maxDepth* and *maxBins* are also impactful hyper-parameters for decision-tree (`DT`) methods.

For BigDL workloads, the number of epochs lightly affect the methods accuracy. Instead, the *stepSize* and *batchSize* hyper-parameters affect several BigDL workloads.

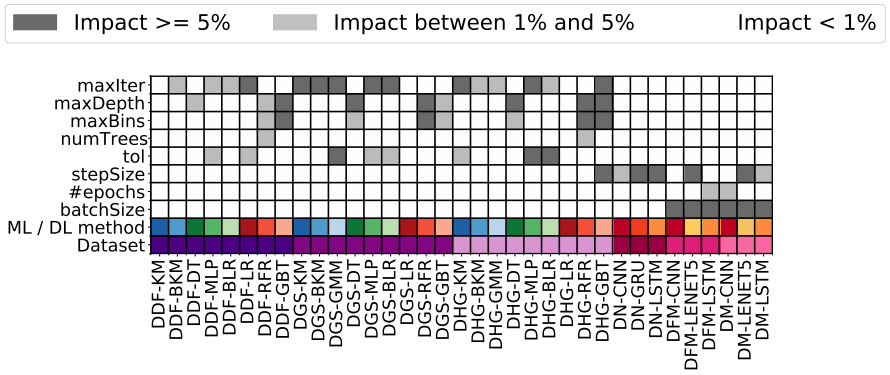


Figure 5.6: Impact of hyper-parameter on performance (dark-grey for high: > 5%, light-grey for medium: 1% ≤ 5%, white for low: < 1%)

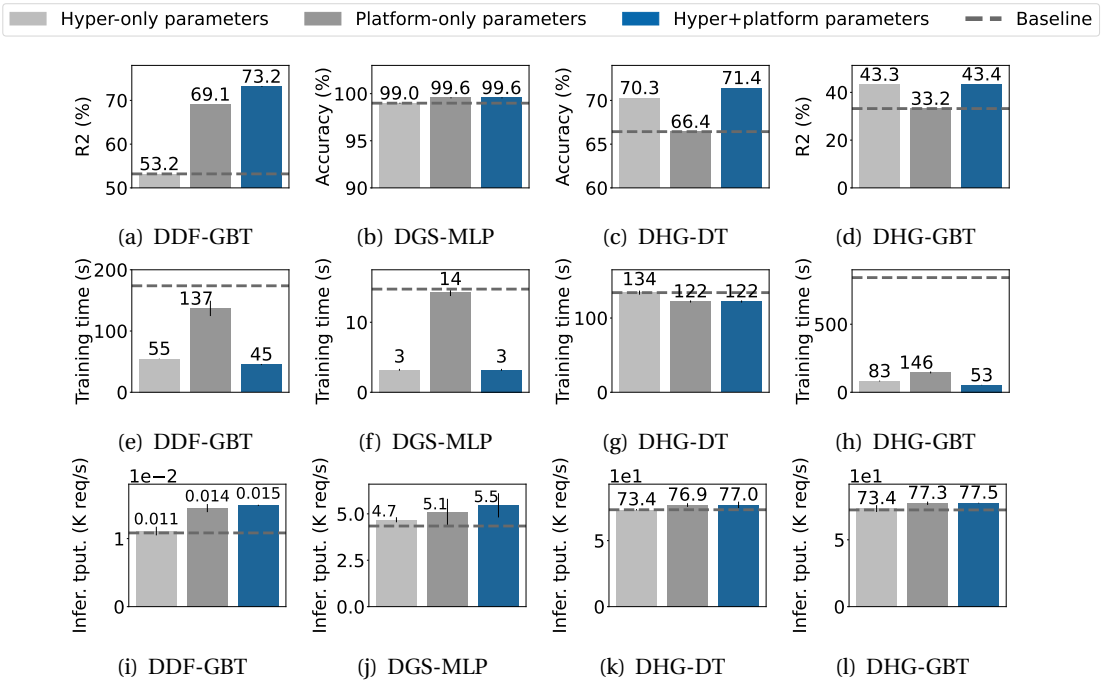


Figure 5.7: Single-level vs. multi-level configuration on cluster 1 (1^{st} row model quality, 2^{nd} row training time, 3^{d} row inference throughput).

5.2.3 Single-level *vs.* Multi-level Configuration

To compare the impact of the single-level *vs.* the multi-level configuration on training time, inference throughput, and model quality, we consider the following three configuration strategies: *(1)* tuning hyper-only parameters, *(2)* tuning platform-only parameters, and finally *(3)* jointly tuning hyper-parameters and platform parameters. We consider high impact parameters as identified in §5.2.1 and §5.2.2. Hyper-only parameters include the maximum number of iterations, the tolerated convergence threshold, and the tree depth for tree-based algorithms. Platform-only parameters mainly include the parameters for controlling parallelization, *i.e.*, the number of executors per worker node, and the number of cores per executor. When tuning

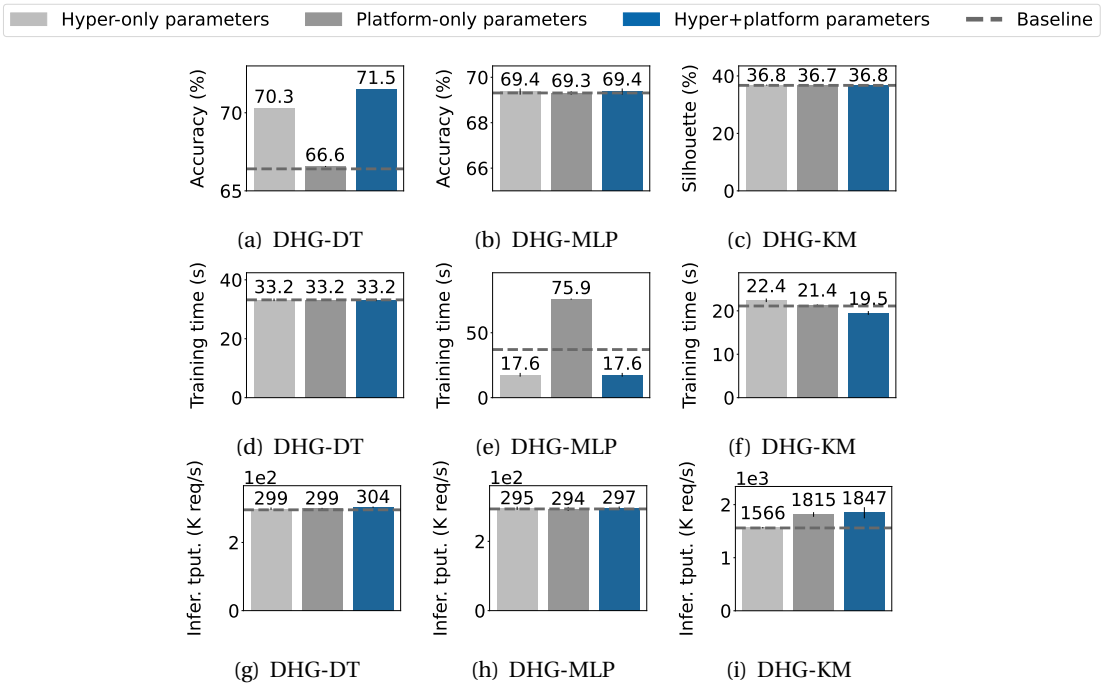


Figure 5.8: Single-level vs. multi-level configuration on cluster 2 (1^{st} row model quality, 2^{nd} row training time, 3^{d} row inference throughput)

jointly hyper-parameters and platform parameters, the previously considered parameters are varied together.

Figure 5.7 gives the best raw values for model quality, training times and inference throughputs obtained with the three configuration strategies, for different workloads run on Cluster 1. The performance of the default Spark configuration is represented by a dashed line. Default configuration values are shown in Table 4.3. The error bars represent the 95% confidence intervals. The low metric variation for some cases makes the error bar too narrow to be visible. In Figure 5.7(a)-(d) and Figure 5.7(i)-(l), we see that, for the same workload, the multi-level configuration strategy achieves the best strategy for model quality and inference performance. Indeed, it improves the other approaches up to 37% for R-squared score and up to 45% for inference throughput. In Figure 5.7(e) we observe that the multi-level configuration strategy is the best configuration strategy for the training time of `DDF-GBT`. It reduces training time by 71% compared to the platform-only approach and by 27% compared to the hyper-only approach. Multi-level configuration achieves the best improvements also for the training time of `DHG-GBT` (Figure 5.7(h)), as well as for the inference throughput of `DGS-MLP` (Figure 5.7(j), `DHG-DT` (Figure 5.7(k)) and `DHG-GBT` (Figure 5.7(l)). The multi-level configuration consistently outperforms Spark's default configuration.

For the workloads shown in this experiment, the multi-level configuration strategy achieves the best model quality (*e.g.*, in Figure 5.7(a), (b), (c), (d)). Surprisingly, Figure 5.7(a) and Figure 5.7(b) reveal that for the concerned workloads the platform-only approach is better than

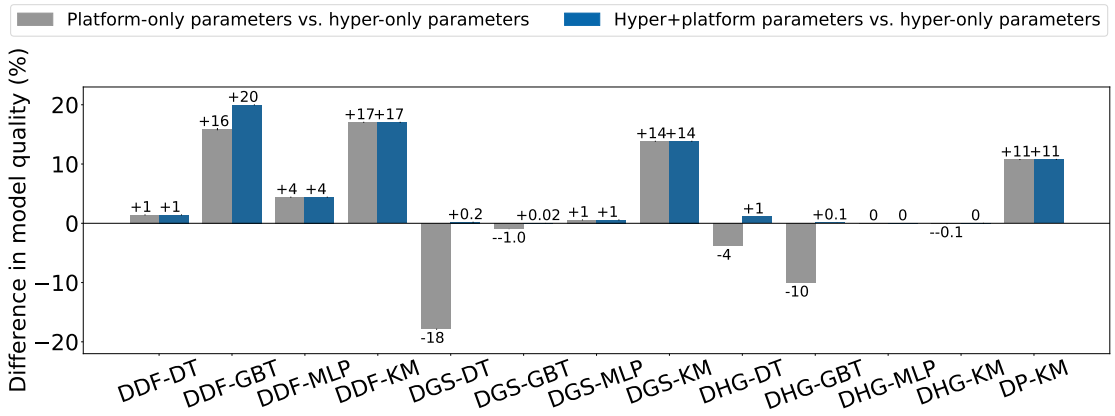


Figure 5.9: Comparison with hyper-parameters only

the hyper-only approach. For `DDF-GBT` the improvement goes up to 28% while for `DGS-MLP` it is up to 0.6%. This is unexpected since platform parameters are not supposed to impact model quality. The issue is further explored in §5.2.4.

We observe the same phenomena on Cluster 2, as shown in Figure 5.8. For the shown workloads, `DHG-KM`, `DHG-MLP` and `DHG-DT`, the multi-level strategy provides the best results for training time and inference throughput. As for model quality, it achieves the same or better performance compared to both the hyper-only and the platform-only approach.

In summary, our results demonstrate that the joint configuration of platform parameters and model hyper-parameters consistently outperform other configuration strategies. It is better than tuning platform-only parameters, tuning hyper-only parameters, or the default Spark configuration. Specifically, the multi-level tuning approach achieved better results than tuning platform-only parameters for 86%, 100%, and 86% of the workloads in terms of training time, inference throughput, and model quality, respectively. It also outperformed tuning hyper-only parameters for 57%, 100%, and 71% of the workloads, in terms of training time, inference throughput, and model quality, respectively. Finally, the multi-level tuning approach yielded better performance compared to the default Spark configuration for all presented workloads and metrics.

Observation 4: *The multi-level configuration strategy leads to the exploration of new configurations that outperform single-level configuration strategies.*

5.2.4 Unexpected Impact of Platform Parameters

The unexpected behavior observed in Figure 5.7(a) shows higher model quality due to the tuning of platform parallelization parameters, namely number of cores and number of executors. To study more in-depth this behavior, in Figure 5.9 we compare the platform-only and the joint configuration strategies to the hyper-only strategy, taken as a baseline. When

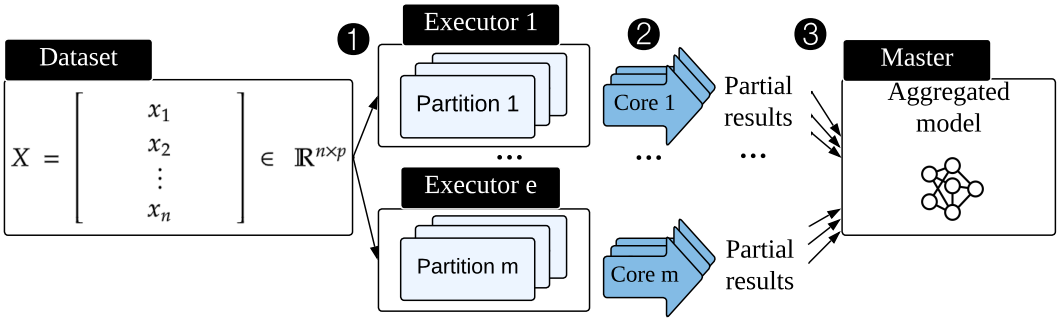


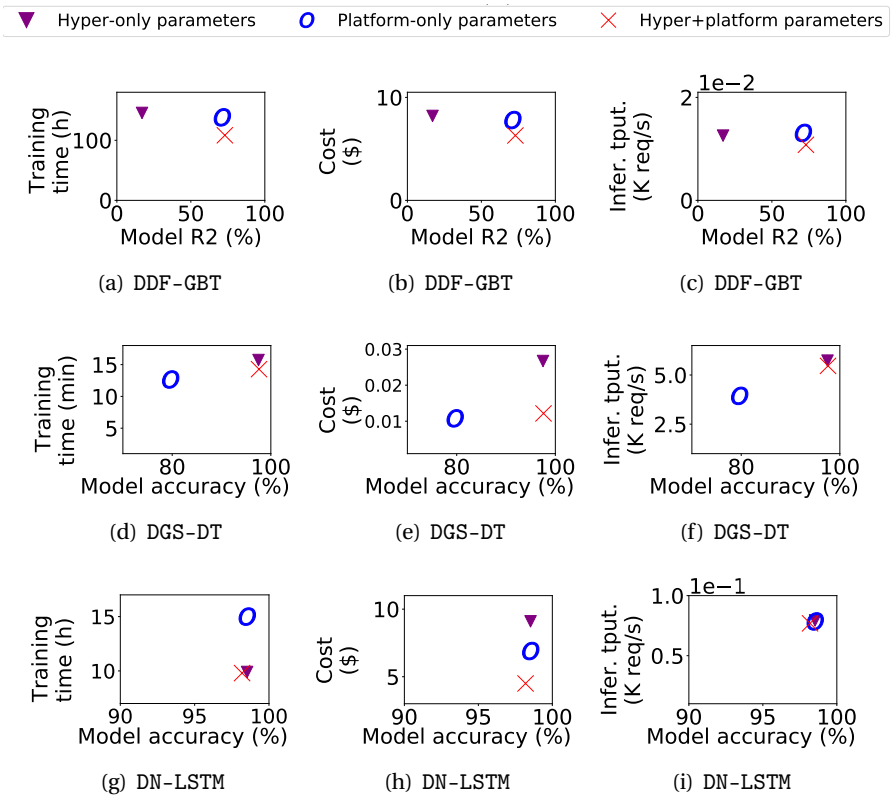
Figure 5.10: Aggregation of partial models during DML job

the comparison between platform-only and hyper-only parameters is positive, configuring platform parameters does improve model quality. When the value is negative, we have the standard case of model quality conditioned by hyper-parameters.

Surprisingly, platform parameters have a strong impact for three of our datasets: DGS, DDF and DP. The latter (DP [40]) is a high-dimensional dataset with 5,409 features, added to our benchmark on Cluster 2 (see Chapter 4 §4.4). From Figure 5.9, we observe that DDF is the most affected. Interestingly, it is the dataset with the highest number of features (see Table 8.1). At the opposite, DHG, the dataset with the lowest number of features, is never impacted by the platform parameters and its model quality benefits almost exclusively from the hyper-parameter configuration. This brings the following counter-intuitive observation.

In Spark, parallelization can impact model accuracy in unexpected ways when dealing with high-dimensional data. We attribute this phenomenon to Spark's data partitioning and task parallelization strategies. Indeed, the number of cores in Spark determines how many partitions can be processed in parallel. Each core processes one data partition at a time and the intermediate results are combined at the end of each training iteration (see Figure 5.10). During the training, ML methods' error, which represents the objective function to be optimized by the ML process, is computed locally within each partition. In the case of high-dimensional data, these partitions exhibit higher dimensionality and a smaller number of instances compared to other datasets. Consequently, the local computation of the error across different partitions can lead to the identification of distinct local optima, varying in quality depending on the partitioning strategy, specifically the number and content of the partitions. As a consequence, the degree of parallelization influences the partitioning and, subsequently, the quality of the optima. Interestingly, similar observations were made in a few prior studies. The same phenomenon was reported in [5] with the Genetic Algorithm and in [165] with the SLIC algorithm when applied to high-dimensional data.

Observation 5 : *One must take into account platform parameters as they may directly benefit model quality.*



(a) DDF-GBT (b) DDF-GBT (c) DDF-GBT

(d) DGS-DT (e) DGS-DT (f) DGS-DT

(g) DN-LSTM (h) DN-LSTM (i) DN-LSTM

Figure 5.11: Impact of different configuration strategies. Quality *vs.* training time in 1^{st} column, quality *vs.* cost in 2^{nd} column, and quality *vs.* inference throughput in 3^{rd} column.

5.2.5 Multi-level and Multi-objective Analysis

In this section we consider DML workloads in the context of AI-as-a-Service. We consider the problem of a service operator who needs to train on a dataset of 2M records and searches for a configuration fulfilling the following threefold objective: *(i)* provide the best accuracy, *(ii)* minimize training time, and *(iii)* result in a minimal training cost. We apply the previously presented strategies (platform-only parameters, hyper-only parameters and hyper+platform parameters) to the AI-as-a-Service configuration and compare their impact on the actual service performance and cost. We consider an AI-as-a-Service as if deployed in Amazon EC2 (N. Virginia) [9]. Following AWS pricing scheme, we consider computing instances similar to our experimental setup, *i.e.*, 8 vCPU with 64 GB of memory, billed $0.46/hour. Outbound data transfers are charged $0.05 per GB.

The first row of Figure 5.11 illustrates the case of DDF-GBT. It compares the three configuration strategies in terms of model quality, training time, inference throughput and training cost. We consider the *maxDepth* and *maxIter* hyper-parameters, and the numbers of executors and cores as platform parameters. The goal is to achieve at least 70% of model R^2, with the shortest possible training, and the cheapest possible training costs. In Figure 5.11(a) we observe that the joint configuration strategy leads to an average model training time that is 22% faster than

the platform-only strategy and 26% faster than the hyper-only one. It also has the best R^2, training cost and inference throughput trade-off as shown in Figure 5.11(b) and Figure 5.11(c).

The second row of Figure 5.11 considers the same analysis but in the case of `DGS-DT`. We consider *maxDepth* and *maxBins* hyper-parameters and the same (executors and cores) platform parameters. We can see that, compared to the other two strategies, the platform-only parameters strategy has shorter training time but much lower accuracy. On the other hand, the hyper-only parameter and the multi-level configurations have similar accuracies, training times and inference throughputs, but the latter optimizes resources costs up to 54% (see Figure 5.11(e)). We can therefore state the following.

Observation 6: *Multi-level configuration may not improve single-level configuration's performance but may lower costs.*

The third row of Figure 5.11 focuses on the BigDL workload `DN-LSTM`. We keep the same platform parameters (number of executors and cores) and use the number of epochs and the batch size as hyper-parameters. What we observe is similar to our first case: the strategy that configures both hyper- and platform parameters achieves the best trade-off between accuracy, costs and training time. In this case, the strategy succeeds in dividing the costs per two. Given the experiments with both MLlib and BigDL, we observe the following.

Observation 7: *The combination of hyper- and platform parameter configurations may be beneficial across several perspectives (*e.g.*, model quality, training time and costs), to both MLlib and BigDL workloads.*

5.3 Key Takeaways

From our detailed analysis and observations, we derive high-level takeaways and recommendations for both DML service operators and data scientists.

(1) **For many DML workloads, the joint configuration of platform parameters and model hyper-parameters provides better performance and allows cost savings.** Our use-cases show that if the tuning of model hyper-parameters and platform parameters are conducted separately, respectively by the data scientist and the system operator, one might observe contradicting and negative side-effects, reducing the overall performance and increasing the execution cost of DML workloads.

Data scientists and DML service operators should jointly configure model hyper-parameters and the underlying platform parameters to obtain greater benefits in terms of training time, model quality, inference throughput and resources costs.

(2) **Surprisingly, platform-level parameters may have higher impact on model quality than hyper-parameters.** Model accuracy is usually impacted by model hyper-parameters

and not by the underlying distributed computing platform parameters. Counter-intuitively, we observe the contrary in the cases of workloads with high-dimensional datasets where parallelization impacts model accuracy, which happens as a result to Spark's parallelization approach.

Data scientists should include platform parameters while tuning the accuracy of high-dimensional models.

(3) **Most DML workloads benefit significantly from the tuning of parallelization.** We show that the performance of most DML workloads in terms of training time and inference throughput is highly impacted by the number of ressources (processes and cores) used for parallelization.

DML service operators should always tune `EXEC_COR` and `EXEC_NUM` in Spark.

(4) **Workloads involving ensemble learning methods and datasets with large number of features benefit from shuffle data size reduction.** We show that the training time of ensemble learning methods executed on datasets with large number of features is negatively impacted by important shuffle data compression.

DML service operators and data scientists should enable shuffle data compression (`SHF_COMPR`) for ensemble learning methods executed on datasets with large number of features.

(5) **DML workloads involving large datasets are significantly impacted by task rescheduling.** Workloads that deal with large amounts of data would benefit from tuning when tasks on local data are to be launched on remote nodes.

DML service operators should tune the `LOC_WAIT` parameter when dealing with datasets with a large number of records.

5.4 Summary

We conducted a DML workload characterization on diverse and heterogeneous workloads from 13 widely used learning methods, with 6 real-world datasets. Our extensive execution traces [49] amount for a total of 16.2 GiB and over 80 million records. We provided an analysis of the statistical distributions of the traces, showing their main characteristics. Our extensive experiments showed the importance of a multi-level configuration strategy for jointly tuning DML hyper-parameters and lower-level platform parameters. Unexpectedly, we observed for some distributed learning workloads that configuring platform parameters has higher impact on the model quality than tuning hyper-parameters.

We derived several interesting observations to characterize DML workloads. We then drew out key takeaways and provided a number of recommendations for DML service operators and data scientists. We claim that DML workloads need more advanced tools that guide the joint configuration of hyper- and platform parameters as this approach may result in better performance. We hope that our observations inspire the development of such tools.

Finally, we publicly release our collected traces to help researchers and practitioners in future DML studies, such as building realistic modeling and simulation tools of DML workloads, or building tools for synthetic DML trace generation.

The results of our experimental characterization will be useful for both data science developers and system administrators architects, and will motivate future research and collaborations to propose new DML tuning tools allowing for optimal mapping between DML workloads' computations and the underlying platform execution models.

As future perspectives, we will characterize other DML platforms than Spark to further enrich the understanding of DML workloads in diverse environments.

In the following chapter, we shift our attention to another challenge of DML systems, namely the issue of bias in federated learning.

Acknowledgement

I would like to express my sincere gratitude to Isabelly Rocha, Sara Bouchenak, Vania Marangozova, Valerio Schiavoni, Lydia Y. Chen, and Pascal Felber for their valuable contributions to the DML characterization study presented here. Their expertise, guidance, and support have been instrumental in advancing this project.

Bias Mitigation in Federated Learning Part II

Overview

Federated learning (FL) [135], an emerging Distributed Machine Learning (DML) paradigm, is gaining attention as a promising approach to intelligent services. It allows multiple data owners referred to as clients to collaboratively build a Machine Learning (ML) model by sharing models learned from their data while keeping data private at their premises. This approach offers a number of advantages over traditional DML. In the latter, data is typically aggregated from various locations to a central data center where training is performed in a distributed manner using the data center nodes. FL thus result in increased privacy and reduced data transfers compared to DML. In particular, FL has great promise for improving the capabilities of distributed learning systems in critical domains where data is sensitive, such as disease diagnosis [27], smart governance [164], and smart security [95].

Despite the advantages of FL, it is important to acknowledge and address its challenges. One key concern in FL is the potential for model bias and unfairness. Bias or unfairness are present in a system whose decisions are prejudicial against users based on their sensitive attributes. Sensitive attributes capture sensitive and demographic information that is protected by law and may include, among others, race, religion, gender, and age [14, 60]. Instances of unfair ML algorithms in the real world are abundant. For example in [231, 193], the authors show that health sensors like oximeters consistently misclassify individuals of color as they have mostly been tested on white populations. Similarly, voice recognition models may exhibit higher accuracy for men than for women due to the differences in vocal characteristics between genders. Women's voices are generally higher in pitch and thinner, making it more challenging for accelerometers to capture their voice features compared to men [199].

In FL, bias can be caused by clients that inadvertently train biased models because of their biased local data and then propagate bias to the global aggregated model. Bias in FL can also result from other sources. For exemple, FL participating clients may indeed exhibit variability not only in their local data distributions but also in connectivity and resource availability. However, as this variability may reflect the demographic and socio-economic profiles of clients, client exclusion may reduce model representativeness and increase the likelihood of bias [231]. The potential harm of biased AI and ML techniques ignited several legislative actions, *e.g.*, the *National AI Initiative Act* in the US [206] and the *EU AI Act* [205], aiming at promoting fairness, accountability, and transparency in AI systems. Bias represents a widely recognized concern that has been extensively explored in ML [138, 28]. However, bias mitigation techniques used to achieve fairness in classical (*i.e.* centralized) ML are incompatible with the privacy constraints of FL.

In this part, we address the bias mitigation problem in FL. We first provide background on FL and bias, and review FL bias mitigation literature. Then we propose ASTRAL, a bias mitigation framework that constrains bias below a given threshold, while maintaining FL model accuracy as high as possible. We carry extensive empirical evaluations to validate our proposal and compare it to other FL bias mitigation techniques.

6 Background and Related Work

In this chapter, our attention shifts towards the critical domain of bias mitigation within the specefic DML paradigm: Federated Learning (FL). We first establish a comprehensive understanding of the foundational concepts and challenges associated with bias in FL systems. We provide a background on FL systems, bias in the FL context, and the relevant bias metrics. Subsequently, we explore the current research landscape and diverse methodologies aimed at mitigating bias in FL systems. This involves an exploration of existing works, their challenges, and lays the groundwork for our contributions. As we conclude the chapter, the problem statement section outlines the specific challenges and complexities intrinsic to addressing bias within FL systems.

6.1 Background on Federated Learning

Federated learning (FL) is a distributed learning paradigm that allows data owners (or clients) to collaboratively train an ML model without exchanging their data. In FL systems, data is usually distributed among clients in a non-IID manner (non-identically and independently distributed) [87], which is one of the main aspects that distinguish it from classical DML systems. The local data of client c_k is denoted U_k and contains n_k samples of data. U_k is referred to as the *local distribution* of client k. The union of all clients' data is denoted $\mathcal{U}$ and is referred to as the *global distribution*. We use $X = (X_1, X_2, .., X_d)$ to denote the features, where d is the features number. X takes on values in $\mathcal{X}$. We use Y to refer to the class label variable that represents the classification decision for an instance x of X and takes on values in $\mathcal{Y}$.

The objective of the FL system is to learn $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$, where f_θ denotes a classification model learned over the union of the clients' data, and θ is a d-dimensional vector of parameters that define f_θ. θ contains the real values representing the parameters. f_θ predicts for an instance $(x, y) \in \mathcal{X} \times \mathcal{Y}$ an approximation $\hat{y}$ of the label y as : $\hat{y} = f_\theta(x)$. To learn the objective model f_θ, the clients collaboratively search for the set of parameters θ that minimizes the system classification loss L, where L is a measure of the overall prediction error of f_θ and is defined as the weighted average of the classification loss of all participating clients. The objective of the

system can be formulated as follow :

$$\arg\min_{\theta} L(\theta) = \sum_{k=1}^{M} w_k L_k(\theta) \tag{6.1}$$

Where $M \in [1, N]$ is the number of participating clients. $\{w_k\}_{k=1}^{k=M}$ denotes the aggregation weights defined by the aggregation method adopted by the system, such that $w_k \leq 1, \forall k \in [1, M]$, and $\sum_{k=1}^{M} w_k = 1$. $L_k(\theta)$ denotes the local loss at client k and is defined by the averaged loss over the data instances of D_k, i.e., $L_k(\theta) = \frac{1}{n_k} \sum_{(x,y)\in D_k} l(x, y, \theta)$. l is a loss function defined = that measures the prediction error for an instance $(x, y) \in \mathscr{X} \times \mathscr{Y}$ using a classifier defined by $\theta \in \Theta$. One key example of l is the cross-entropy loss (see Chapter 2 §2.1.3).

In FL, when data is distributed in independant and identical fashion (i.i.d), and all FL clients participate in the process, the learning task as described is equivalent to learning an ML model that minimizes the expected value of a loss function using the dataset D such that

$$D = \mathscr{U} = \bigcup_{k=1}^{k=N} D_k$$

.

In order to optimize the loss defined in Equation 6.1, the FL server and clients follow the FL protocol presented in the following. Assuming the FL protocol is composed of a total of T communication rounds. Given a client selection policy Ψ and an aggregation policy Γ, at each communication round $t \in [1, T]$ the FL system performs the following procedures :

- **Participant selection:** The server selects a subset of M participants among the N existing clients in the FL system using the selection policy Ψ. Let $P = \{c_1, .., c_M\}$ denotes the set of the selected clients, such that $M \leq N$.
- **Model initialization:** The server sends the global model θ^{t-1} learned in the previous round to the participants P. In the special case of $t = 1$, the server either sends a randomly initialized model or a pre-trained model θ^0.
- **The local update process:** Each client from P trains the received global model θ^{t-1} on its own private data D_k to learn θ_k^t. The learning is done by minimizing the local loss function L_k, the learned parameters $\{\theta_k^t\}_{k=1}^{k=M}$ are sent to the server.
- **The global aggregate process :** The classical FL aggregation method for combining clients' models is FedAvg [135]. It aggregates the clients' parameters to obtain the parameters of the global model. It uses Equation 6.2 according to which the weight of a client's parameters is proportional to the quantity of its data. The output of the FL system is a classification model that predicts the class label value for an unlabeled

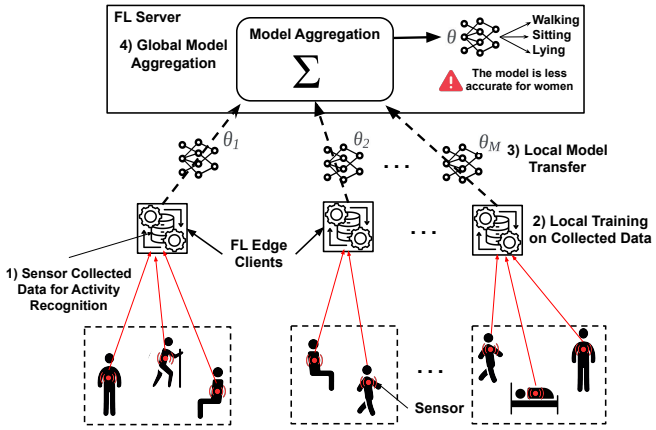


Figure 6.1: Example of bias in FL systems

data instance $\overline{x} \in \mathscr{X}$. The model's prediction for $\overline{x}$ is referred to as $\hat{y}$. The quality of the model's predictions is usually measured with metrics such as model accuracy, precision or recall (see Chapter 2 §2.1.3).

$$\theta^t = \sum_{k=1}^{k=N} \frac{n_k}{|\mathscr{U}|} . \theta_k^t \tag{6.2}$$

6.2 Background on Bias

Bias in ML is defined as the presence of prejudice or favoritism in a FL model outcome that disproportionately affects certain individuals based on their sensitive attributes such as race or gender [138]. Such bias leads to unfair outcomes and exacerbates existing social inequalities. It can result in a range of negative consequences, such as denying opportunities, resources, or services to certain individuals or groups. ML model bias with respect to sensitive attributes has been addressed in several studies in the emerging field of ethical ML [63, 56, 101]. These studies consider different bias concepts including statistical parity difference [56], equal opportunity difference [82] and discrimination index [240]. While these concepts differ somewhat, they all quantify bias by examining if and how the outcomes of the models depend on sensitive attributes. In the literature, these concepts are grouped under the term of *group fairness* [56]. Sensitive attributes are defined by law and include, among many others, race, age, gender, health situation, *etc.* [60, 14]. A sensitive attribute, denoted S_j, typically has two possible values that we denote by v_1^j and v_2^j. It separates data into two demographic groups. Usually, we refer to these groups as privileged and unprivileged groups. In fairness literature, the *privileged group* is a group that has historically been at a systemic advantage [128], while the *unprivileged group* has been disadvantaged. For example, in the United States, race has been a significant factor in determining access to resources and opportunities. White individuals have had more advantages than individuals from other racial groups, and are thus identified

as the privileged group. However, it is important to note that the definition of privileged and unprivileged groups may vary depending on the context and the specific task at hand.

Bias in ML arises from issues like unbalanced, incomplete, or flawed datasets, leading to unfair models. In FL, bias can originate from biased local data on clients. Client variability in FL, including data distribution and resources, coupled with unequal FL selection and aggregation can also contribute to bias. For example, let us consider the FL use case of smart healthcare in which several hospitals use FL for activity recognition [157], as illustrated in Figure 6.1. A central server aggregates into a global model the hospitals' local models that are trained using sensor data collected from their local patients. An imbalance in the collected data can result in a biased model against underrepresented groups.

6.3 Background on Bias Metrics

To quantify bias, a variety of metrics have been proposed [56, 82]. These metrics operate under the assumption that the sensitive attribute is binary (we explore in section 8.1 the way we binarize sensitive attributes in our experiments). We denote the number of sensitive attributes as sa, and the model outcome variable, which we consider binary for simplicity, as $\hat{Y} \in \{0,1\}$. The value 1 is referred to as the positive outcome, while 0 is referred to as the negative outcome. In the following, we briefly detail the different bias metrics considered in our work. In practice, the choice of which metric to use depends on the specific context, the type of discrimination that needs to be addressed, and the desired outcome of the system.

Statistical Parity Difference (SPD) is a classical bias metric for binary classifiers used to measure the demographic parity fairness notion [56]. It measures the difference between the positive outcome rates for the two groups defined by the considered sensitive attribute. A model is fair and ensures demographic parity if its SPD is equal to zero. If the SPD is different than zero, it indicates that the model is more likely to assign positive predictions to one particular group. Given a dataset $\mathscr{U}$ and a sensitive attribute S_j, SPD_{S_j} is defined in Equation 6.3, with $Pr_{\mathscr{U}}$ being the probability distribution of $\mathscr{U}$. SPD is used in systems interested only in the prediction outcome. For example, consider the use case of a mobile recruiting application [16]. Measuring the SPD with respect to attributes such as race or gender on the output of the hiring system can indicate whether the model is hiring candidates of certain demographic groups more than the others.

$$SPD_{S_j} = Pr_{\mathscr{U}}(\hat{Y} = 1 | S_j = v_1^j) - Pr_{\mathscr{U}}(\hat{Y} = 1 | S_j = v_2^j) \tag{6.3}$$

Equal opportunity Difference (EOD) is used to reflect the fairness notion of equality of opportunity [82]. Unlike SPD, EOD imposes constraints on the *correct* or *true* positive predictions [82]. Formally, EOD measures the difference between the correct positive predictions

for the two groups defined by S_j, as presented in Equation 6.4. It is more prone to be used in systems where not only the outcome but also the quality of prediction matter, such as identifying dangerous individuals from cameras' footage [8].

$$EOD_{S_j} = Pr_{\mathscr{U}}(\hat{Y} = 1 | S_j = v_1^j, Y = 1) - Pr_{\mathscr{U}}(\hat{Y} = 1 | S_j = v_2^j, Y = 1) \tag{6.4}$$

Discrimination Index (DI) quantifies the variation of the F1 score between the two groups defined by a sensitive attribute S_j [240]. The F1 score is a statistical measure that evaluates the quality of a binary classification model, and is equal to the harmonic mean of the recall and the precision measures. Formally, DI for a model θ with regard to the sensitive attribute S_j is calculated as in Equation 6.5, where $F1_{S_j=v_i^j}(\theta)$ denotes the F1 score of the model θ computed using only data records from the group with $S_j = v_i^j$. DI is more prone to be used in systems where the prediction quality is of utmost concern such as healthcare applications [3]

$$DI_{S_j} = F1_{S_j=v_2^j}(\theta) - F1_{S_j=v_1^j}(\theta) \tag{6.5}$$

6.4 Related Work on Bias Mitigation in Federated Learning

Bias is a well-known problem studied in ML [138, 28]. In FL, there are different categories of bias problems depending on the considered FL fairness notion. The first category of problems deals with (i) collaborative fairness which aims to reward clients with high contribution to the FL process with better models in terms of performance compared to low-contributing clients. The second category deals with (ii) performance fairness which aims to achieve a performant FL model for the different clients with regard to their local distributions. The third category deals with (iii) group fairness, where the aim is to prevent any discrimination in the FL model against particular data groups based on their sensitive attributes.

6.4.1 Collaborative Fairness in FL

Several works study the first type of bias problems [124, 232]. In [124], the authors introduce CFFL. In CFFL, collaborative fairness is quantified via the correlation coefficient between participant contributions (test accuracies of clients' models which characterize their individual learning capabilities on their own local datasets) and participant rewards (test accuracies of final models received by the participants). CFFL evaluates and iteratively updates the contributions of participants, ensuring that each participant receives models with performance commensurate with their contributions.

In [232], the authors propose the FLI payoff-sharing scheme, which incentives FL data owners

to contribute high-quality data to the FL system. Data owners who have contributed a large set of high-quality get rewarded by a higher share of revenues generated by the FL system. Experimental evaluations of FLI show that it achieves the highest expected revenue for a data owner compared to other existing payoff-sharing schemes.

6.4.2 Performance Fairness in FL

Other works focus on addressing the second category of bias problems, specifically those related to performance fairness. For instance, AFL [148] aims to ensure that the global model is optimized for any target distribution. To achieve this, AFL optimizes the model for a distribution formed by any mixture of the clients' distributions. This is accomplished by minimizing the worst convex loss combination of clients' losses. In doing so, the framework ensures a high-performing model for the worst-case scenario, and consequently, a high-performing model for clients in general.

In a different approach, the authors of [246] tackle the complexities of cross-device federated learning, with a specific focus on wearable devices. They pinpoint challenges stemming from network communication instability causing biased client selection. The authors categorize clients into three groups: (1) over-represented, (2) under-represented, and (3) never-represented, referring to those selected too frequently, too infrequently, and never or rarely selected, respectively, due to their network capabilities. Their primary emphasis is on addressing bias resulting from unfair client selection in the third group. Clients in this category, whose data patterns differ significantly from those of well-connected clients selected most frequently, encounter diminished model performance on their distribution due to skewed learning. To mitigate this issue, the authors raise the network capacity threshold, enabling equitable client selection irrespective of networking conditions. Additionally, they introduce a mechanism to identify the most crucial updates from the FL clients.

Additionally, [118] introduces FedCHAR, a personalized FL system. FedCHAR enhances both accuracy and fairness in model performance by leveraging the inherently similar relationships between FL clients. The distinguishing features of FedCHAR include dynamic clustering and adaptability to the inclusion of new users or changes in dataset composition, making it well-suited for realistic FL-based scenarios.

6.4.3 Group Fairness in FL

In our study, we focus on the third type of bias issue, specifically group fairness. Various techniques exist for mitigating group fairness biases in Federated Learning (FL), encompassing both client-side [2] and server-side methods [101, 234, 51, 235, 233]. Client-side methods typically involve adjusting local loss with bias penalties that quantify discrimination against demographic groups. These techniques may also rely on reweighing client data records based on sensitive attributes. However, global bias mitigation is not guaranteed, given the heteroge-

neous data distribution among FL clients [61]. For instance, a sensitive group considered a minority in the global distribution (e.g., African-American) might be a majority for a specific client (e.g., credit data in a black-majority city such as Detroit), so reweighing a clients' majority group to locally mitigate bias in that client might not only fail to mitigate global bias but potentially exacerbate it.

On the other hand, several works follow a server-side approach for FL bias mitigation [53, 30, 240, 239, 61, 36]. In this approach, local debiasing methods are done but in coordination between clients and are orchestrated and oriented by a server. Other server-side methods are completely server-sided and do not require any local debiasing. Generally, in the process of applying server-side bias mitigation, the clients may need to exchange additional information with the server about their data constitution or local model metrics.

AgnosticFair [53] considers the SPD metric. It applies a minimax gaming optimization between the clients and the server to minimize the global loss under a bias constraint that imposes a threshold on the bias. However, since SPD is not a convex function, AgnosticFair considers a convex approximate for it. Thus, the bias constraint does not ensure that the threshold is respected by the objective bias metric itself. AgnosticFair assumes the presence of a single sensitive attribute in data [53] and requires exchanges of local bias information at each FL learning round.

Another proposal is FCFL [36], which takes into account the SPD and EOD metrics. It frames the problem as a multi-objective constrained optimization task and resolves it, guaranteeing that the global FL model adheres to the bias constraint on each client's distribution while maintaining consistent accuracy across clients. The primary objective is to ensure group fairness at each client's level rather than focusing on the global distribution. In FCFL, bias and loss information of clients are exchanged in each FL round. Similar to AgnosticFair, it addresses only bias related to a single sensitive attribute.

FairFL [240] explores various metrics of group fairness, employing a different approach with client selection based on their individual performance. At each FL round, FairFL dynamically chooses the optimal set of FL clients that collectively optimize as much as possible accuracy and bias simultaneously. The global bias and accuracy in FairFL are estimated through the average of local accuracies and bias measurements provided by individual clients at each round. While FairFL accounts for multiple sensitive attributes, it fails to effectively reducing bias for each of the existing groups [240].

FairFed [61] adjusts the weights of clients for the server global aggregation with the aim of minimizing SPD and EOD as much as possible. Specifically, each client conducts local debiasing on its individual dataset. Then, to enhance the effectiveness of local debiasing globally, clients assess the fairness of the global model on their respective datasets in each FL round. They subsequently collaborate with the server to collectively adjust the model aggregation weights. These weights are determined based on the disparity between the global fairness measurement (calculated on the entire dataset) and the local fairness measurement

Table 6.1: Summary of state-of-the-art bias mitigation methods related to group fairness

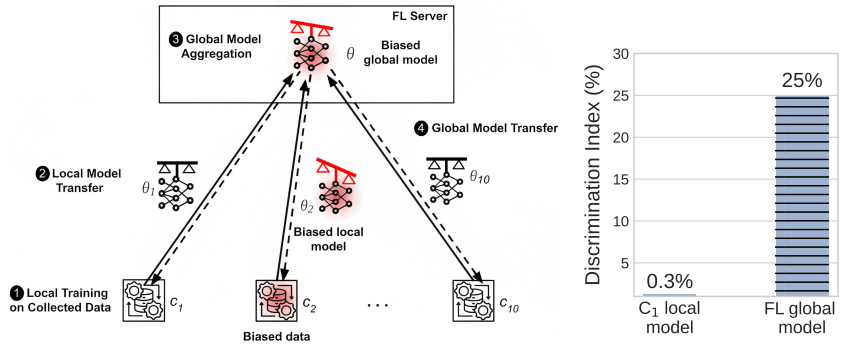
System	Handling multiple sensitive attributes	Ensuring a bias threshold	Generic approach
Abay and al. [2]	✗	✗	✓
AgnosticFair [53]	✗	✗	✗
FCFL [36]	✗	✓	✗
FairFL [240]	✓	✗	✓
FairFed [61]	✗	✗	✓
GIFAIR-FL [233]	✓	✗	✗
q-FFL [116]	✓	✗	✗
ASTRAL	✓	✓	✓

at each client. The adjustment process favors clients whose local fairness measures align with the global fairness measure.

Finally, there exist works that explicitly address group fairness by addressing performance fairness for groups of FL clients clustered according to their sensitive attributes. Specifically, q-FFL [116] considers an FL setting where each client belongs to a demographic group, thus clients can be clustered into groups based on their sensitive attributes. Their solution focus on accuracy group fairness. It encourages a more equitable distribution of accuracy across FL clients by reweighing the aggregate loss and assigning higher weights to clients with higher losses. Their approach results in improving performance for all FL clients and thus for the demographic groups they belong to. GIFAIR-FL [233] considers a similar setting and focuses also on accuracy group fairness. It offers an optimization-based approach to address group fairness with regard to model accuracy among groups of clients. This is achieved through the application of regularization methods that penalize variations in model accuracy between clients' groups during the training leading to uniform accuracy between demographic groups. A particularity of q-FFL and GIFAIR-FL is that they reduce discrimination between demographic groups under the assumption that each FL client can be clearly assigned to a demographic group. However, if this assumption is not met, the used approach does not guarantee group fairness. In this thesis, we refrain from making any assumptions regarding a client's affiliation with a specific demographic group. Specifically, clients can hold data records belonging to one demographic group or to different ones.

6.4.4 Discussion

In summary, most existing bias mitigation methods typically address bias for a single sensitive attribute at a time. They don't provide a mechanism to ensure a predefined bias value on the objective metric. Instead, they aim to minimize the bias objective metric as much as possible, often relying on approximations, which may be insufficient. In addition, they lack generality as they are tailored to support specific bias metrics. This limits their applicability across diverse domains with varying bias guarantee requirements. Table 6.1 presents a comparison of various bias mitigation methods, including our proposed solution. To the best of our knowledge, ASTRAL stands out as the only FL bias mitigation approach capable of handling



(a) FL system architecture illustrating the propagation of bias among FL clients

(b) Model bias with regard to gender for the unbiased client's model *vs.* the FL global model

Figure 6.2: Bias propagation in FL

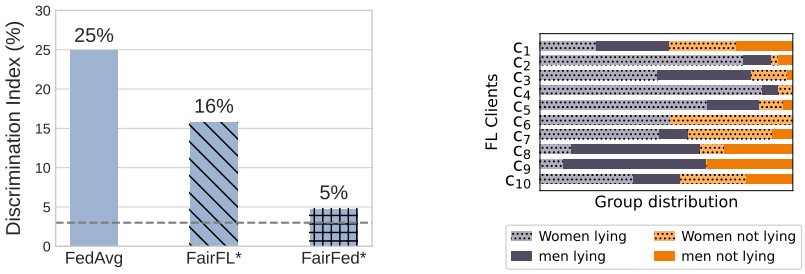
multiple sensitive attributes. It ensures that the model bias remains below an adjustable threshold while maximizing accuracy, and it accommodates any bias metric.

6.5 Problem Statement and Propositions

6.5.1 Why Bias Matters in Federated Learning

In FL, the presence of biased data at an FL client may lead to the propagation of bias to all the FL clients, including the "fair" ones, thus rendering the whole system biased. Indeed, clients that have unbiased local data, that allow them to train fair models, can "inherit" a biased global FL model, due to the biased FL clients, as shown in Figure 6.2(a). This rises a problem, since classical FL protocols do not allow selecting clients based on their data due to FL privacy constraints. To illustrate these phenomena, we implement an activity recognition FL scenario that uses motion data collected from wearable sensors from several patients[I]. Here, we consider gender as sensitive attribute and DI as the bias metric. We compare the DI of the following two models. The first model is trained using classical centralized ML on a FL client denoted c_1, with a balanced data subset of men and women. The second model is trained collaboratively by c_1 and other FL clients holding the remaining data using the classical FL protocol FedAvg. We present the corresponding bias values in Figure 6.2(b). We can observe that FL induces a much higher bias on the resulting model (right bar). Indeed, the unbiased FL client c_1 inherited bias from the other FL clients.

[I]These experiments use the ARS dataset (see section 8.1), executed on our experimental environment (see section 8.2).



(a) FL model bias with regard to gender for the activity recognition FL task: without bias mitigation *vs.* state-of-the-art bias mitigation

(b) Non-IID client data distribution of the activity recognition FL task

Figure 6.3: Existing FL bias mitigation methods are not able to achieve the defined threshold

6.5.2 On the Importance of Maintaining Bias Below a Threshold

Minimizing the gap between groups without establishing a specific threshold may lead to two potential outcomes: either a model with reduced bias but still exhibiting more bias than desired, or a model with no bias but compromised accuracy. The latter occurs because, in the pursuit of mitigating bias to the greatest extent, the model may excessively compensate for certain groups without considering the underlying ground truth. Moreover, fixing a threshold in a bias mitigation system ensures that the system is consistent with legal requirements. Indeed, in many real-world scenarios, quotas are defined to ensure that certain groups are not unfairly disadvantaged. For example in the United States, their exist a rule known as the 80% rule, or the four-fifths rule, that is imposed on companies in their hiring process to fight against hiring discrimination [92]. It ensures that the quotient of the hiring rate of the unprivileged group on the privileged group's is no less than 80%.

6.5.3 Existing FL Bias Mitigation Methods Are Not Able to Limit Bias Below a Threshold

To investigate the effectiveness of FL bias mitigation methods, we consider the same activity recognition FL scenario as above, initially without any bias mitigation, and subsequently, using two state-of-the-art bias mitigation techniques. We evaluate the bias (DI) of the learned FL model with regard to gender and report the results in Figure 6.3(a). It reveals that the FL model created without any bias mitigation methods has a DI of 25%, surpassing the threshold 3%. We then apply two bias mitigation techniques: FairFL* and FairFed* (described in details in section 8.3). FCFL (see section 8.3), another bias mitigation technique cannot be applied since it does not support bias mitigation with regard to DI. We can see from Figure 6.3(a) that none of these methods succeeds in reducing bias below the typical 3% bias threshold within the setting shown in Figure 6.3(b). This illustrates the fact that bias mitigation represents a real challenge in FL.

Table 6.2: Bias mitigation with regard to single sensitive attribute age – Adult dataset

Systems	SPD_{age}(%)	SPD_{race}(%)	SPD_{gender}(%)
FedAvg	13	4	13
FCFL	1	9	19

Table 6.3: Bias mitigation with regard to single sensitive attribute race – MEPS dataset

Systems	SPD_{race}(%)	SPD_{gender}(%)
FedAvg	5	8
FCFL	2	10

6.5.4 Why Not Handling Multiple Sensitive Attributes May Induce More Bias

FL applications typically use data that contains multiple attributes that can be considered sensitive. For example, data collected from mobile applications may include information on an individual's gender, race, and age, while data collected from sensors and other ubiquitous devices may include attributes revealing the region and socio-economic information, among others. Therefore, data in general is characterized by the presence of several sensitive attributes. Although several works have been conducted to mitigate bias in FL, most of them focus on mitigating bias with regard to only one sensitive attribute at a time. This may be problematic, as optimizing a model to be unbiased with regard to one sensitive attribute does not guarantee bias mitigation for other sensitive attributes. Even worse, mitigating bias with regard to one sensitive attribute may exacerbate bias with regard to the other sensitive attributes. We have observed this phenomenon when applying the state-of-the-art bias mitigation method FCFL on several datasets considering the SPD metric. FCFL handles single attribute bias mitigation only. Table 6.2 and Table 6.3 show bias measurements collected from the FL model resultant from this method and obtained using respectively the dataset Adult that contains three sensitive attributes, and MEPS that contains two sensitive attributes. We report the model bias for the two datasets with regard to one sensitive attribute that was considered during the mitigation (bias column at left), and also bias with regard to the other sensitive attributes that exist in the dataset (bias columns at right). As we can see, although FCFL mitigates model bias with regard to age for the Adult dataset, and with regard to race for the MEPS, it exacerbates the model bias with regard to the remaining sensitive attributes. Thus it is important to design bias mitigation techniques that handle bias mitigation for all the sensitive attributes that exist in data.

6.5.5 Our Propositions

To address the bias mitigation problem in FL, we propose ASTRAL, a bias mitigation framework that constrains bias below a given threshold, while maintaining FL model accuracy as high as possible. ASTRAL proposes a novel and self-corrective FL aggregation method, which reweighs FL clients' local models contributions based on their impact on the global FL model's bias and accuracy. In contrast to its competitors, ASTRAL allows dealing with multiple sensitive attributes at a time, and supports bias mitigation with regard to any given bias metric. To showcase ASTRAL's results in terms of model bias, accuracy, scalability, and robustness to clients heterogeneity, we conduct an extensive evaluation across seven widely used datasets. Our findings confirm that ASTRAL outperforms existing FL bias mitigation approaches.

6.6 Summary

In this chapter, we focused on bias mitigation within FL environments. We provided background on FL systems, bias phenomena in FL, and relevant bias metrics with real-world examples. Subsequently, we surveyed existing works for bias mitigation in FL pointing out their limitations. Among the limitations, we emphasized the critical importance of maintaining bias below defined thresholds for legal compliance and ethical considerations and explored the necessity for handling multiple sensitive attributes concurrently, shedding light on potential pitfalls associated with mitigating bias for individual attributes. The problem statement section articulates these specific challenges in addressing bias within FL systems.

The subsequent chapter introduces ASTRAL, our bias mitigation framework for FL. ASTRAL aims to constrain bias below a specified threshold while maximizing FL model accuracy. It employs a novel and self-corrective FL aggregation method, reweighing FL clients' local model contributions based on their impact on the global FL model's bias and accuracy. Distinguishing itself from competitors, ASTRAL handles multiple sensitive attributes simultaneously and supports bias mitigation across various metrics.

7 The ASTRAL System

In this chapter we present ASTRAL, a FL framework for bias mitigation. First, we define formally the problem of bias mitigation in FL in section 7.1. Then, we present an overview of ASTRAL to solve the formulated problem in section 7.2. We describe in detail the design principles of our solution in section 7.3. Finally, in section 7.4, we provide analytical insights of ASTRAL.

7.1 Problem Formulation

The problem of bias mitigation in FL consists in designing an FL system that produces FL models meeting a predetermined bias threshold, and this for all the sensitive attributes existing in data simultaneously. As respecting bias constraints should not come at the expense of accuracy, the accuracy of the resulting FL model should be kept as high as possible. The target distribution in FL systems is the global distribution $\mathscr{U}$, thus the bias and the accuracy objective must be ensured for the global distribution $\mathscr{U}$ [135].

More formally, the defined problem of FL bias mitigation can be formulated as a constrained optimization problem as follows. Given an FL model with parameters θ, the accuracy $A(\theta)$ of the model should be maximized (Equation 7.1), while the model bias $\beta_{S_j}(\theta)$ with regard to each sensitive attribute S_j does not exceed a predefined threshold ϵ (Equation 7.2). Here β is used to denote any bias metric (SPD , EOD , DI , etc. (See section 6.2)).

$$\max_{\theta \in \mathbb{R}^d} A(\theta) \tag{7.1}$$

$$\text{s.t } |\beta_{S_j}(\theta)| \leq \epsilon, \forall S_j \in \{S_1 \dots S_{sa}\} \tag{7.2}$$

The global model parameters θ can be expressed as a linear combination of the local clients' parameters $\theta_1, \theta_2 \dots \theta_N$, and this according to the aggregation formula in Equation 6.2. If $\theta_k \in \mathbb{R}^d$ are the local parameters for client c_k, and $w_k \in \mathbb{R}$ the weight assigned to this client for the aggregation, the global model parameters produced in a FL round are computed according

to Equation 7.3.

$$\theta = \sum_{k=1}^{k=N} w_k.\theta_k \tag{7.3}$$

To simplify Equation 7.3, we represent clients' aggregation weights as the vector $\overline{w} = (w_1 \dots w_N)$, and clients' parameters as the $N \times d$ matrix Θ (Equation 7.4). In Θ, row k contains the d parameters for client c_k, d being the number of parameters of the learning model.

$$\Theta = \begin{pmatrix} \theta_1^1 & \dots & \theta_1^d \\ \dots & \dots & \dots \\ \theta_N^1 & \dots & \theta_N^d \end{pmatrix} \tag{7.4}$$

Thus, we have $\theta = \overline{w}.\Theta$. Therefore, the problem formulated by Equation 7.1 and Equation 7.2 can be expressed using the clients' weights $\overline{w}$ and thus formulated as follows:

$$\max_{\overline{w} \in \mathbb{R}^N} A(\overline{w}.\Theta) \tag{7.5}$$

$$\text{s.t } \beta_{S_j}(\overline{w}.\Theta)| \leq \epsilon, \forall S_j \in \{S_1 \dots S_{sa}\} \tag{7.6}$$

In order to solve the constrained maximization problem formulated by Equation 7.5 and Equation 7.6, our solution finds the best real number aggregation weights $w_1, w_2 \dots w_N$.

Bias metrics in Equation 7.6 are known to be non-convex functions [220]. As a result, the considered problem is also non-convex. Thus, the considered problem is computationally intractable using exact methods, *i.e.* it can not be solved efficiently in polynomial time [72]. Furthermore, the exploration of the search space becomes intractable when considering trivial optimization methods like grid search. Indeed, as our search space is potentially infinite ($\overline{w} \in \mathbb{R}^N$), we have to resort to more sophisticated methods to more efficiently explore the search space. Thus, to find the best weights vector $\overline{w}$, we apply a black-box optimization method, which does not require a specific type (convexity, continuity, *etc.*) of the studied function [181]. The black-box optimization relies on the exploration of different candidates of aggregation weights through a metaheuristic-based algorithm. The algorithm guides efficient and non-exhaustive exploration of the search space and outputs the best weights that optimize the objectives at the end of the learning.

7.2 Overview of ASTRAL

Figure 7.1 presents an overview of the architecture of ASTRAL, which intervenes at each FL round at the FL server-side after receiving the client's parameters and before the aggregation step. Specifically, ASTRAL applies the pipeline described in Figure 7.2 to calculate the weights used for aggregating the FL client model updates in a way that ensures the aggregated global model bias remains below a fixed threshold while maximizing accuracy.

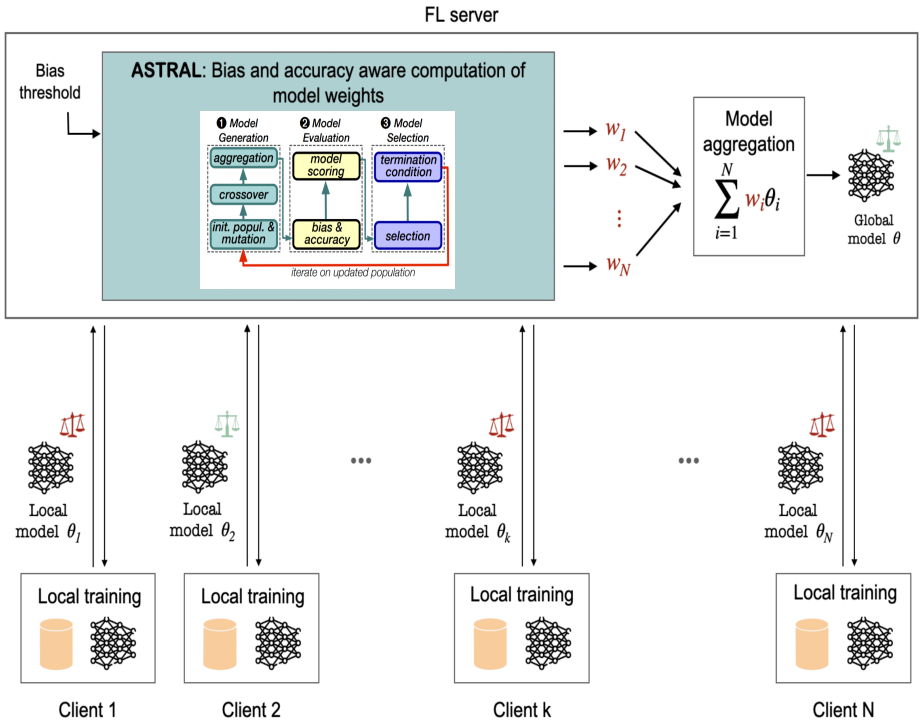


Figure 7.1: Overview of ASTRAL architecture

ASTRAL's pipeline at the server level consists of an iterative optimization process with three essential steps to explore diverse aggregation weights and select the best among them with regard to our objectives, as shown in Figure 7.2. Initially, the server receives the local models of FL clients and the bias threshold as input and launches the ASTRAL's pipeline. The first step in each iteration of the pipeline involves Model Generation (Figure 7.2-❶) which uses the Differential Evolution algorithm (DE) to construct a population of potential aggregation weights and corresponding global models [197]. The second step, *i.e.*, Model Evaluation (Figure 7.2-❷), targets the assessment of the accuracy and the bias of each global model created in the first step and computes upon those two measurements each global model's score. Finally, Model Selection (Figure 7.2-❸) uses the scores computed in Figure 7.2-❷ to select the best global models to keep in the population. The process iterates and loops back to Figure 7.2-❶ with the updated population, until reaching a predefined termination condition. It then outputs the final model which is the model that has the best score of all the population

with regard to the objectives of the constrained optimization problem (see section 7.1). Once the iterative process stops, the current FL round is terminated, and the selected model will be sent to the FL clients at the beginning of the next round.

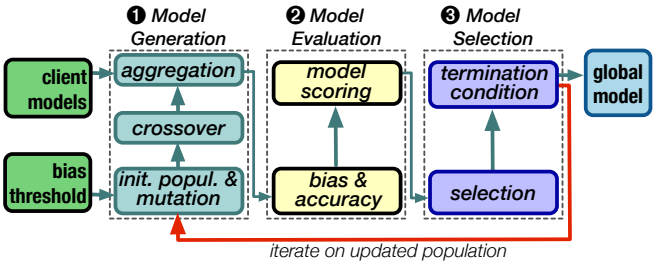


Figure 7.2: The ASTRAL pipeline at the FL server-side

7.3 ASTRAL Design Principles

Here, we describe in details the steps of the ASTRAL pipeline, presented in Algorithm 2. The pipeline starts by the random initialization of a first population of potential candidates for aggregation weights $\overline{w}$. ASTRAL iteratively modifies this population through the three introduced processes: Model generation (lines 7-11 of algorithm 2), Model evaluation (line 12 of algorithm 2) and Model selection (lines 13-16 of algorithm 2). At the end of the pipeline, the best aggregation weight is selected and applied on the clients models to output the optimized global model.

7.3.1 Model Generation

During model generation, current aggregation weights candidates are modified to create new candidate solutions called offspring candidate solutions. Precisely, if P is the current population of candidate solutions, then this module creates for each $\overline{w_i} \in P$ an offspring individual $T_{\overline{w_i}}$ using the *mutation* and *crossover* operators from the DE algorithm, as follows. First, the mutation operator selects randomly two members of the population e_1 and e_2. Their difference, scaled by a factor F, is used to mutate the best member of the population b, as follows: $m_{\overline{w_i}} = b + F * (e_1 - e_2)$ (line 8). Then, the crossover operator creates the offspring $T_{\overline{w_i}}$ by randomly selecting a component w_k from $\overline{w_i}$'s components, *i.e.*, $\{w_1 \dots w_N\}$, and replacing it with the corresponding component from the mutant vector $m_{\overline{w_i}}$ (line 9). The mutation and crossover operators introduce randomness and maintains diversity of populations. Finally, the weight candidate solutions and their offsprings are used to compute global aggregated models (lines 10-11).

Algorithm 2: ASTRAL bias mitigation pipeline at server level

Inputs : Θ: models' parameters of FL clients
ϵ: Global FL model bias threshold

Output : $\theta_{w_{best}}$: Global FL model's parameters meeting ϵ bias threshold with the highest model accuracy

1 Initialise population P with random vectors of weights of FL clients' updates
2 Initialise *max_iter* to the maximum #iterations of Differential Evolution
3 Initialise *nb_iter* to 0, P_{new} as empty, and *termination_condition* to False
4 **do**
5 **foreach** *vector of weights of FL clients' updates* $\overline{w_i} \in P$ **do**
 `// Model Generation`
6 Select randomly e_1 and e_2 from P
7 Select the best individual b from P
8 Create mutant vector $m_{\overline{w_i}}$ using e_1, e_2, b
9 Create offspring vector $T_{\overline{w_i}}$ using DE crossover operator between $\overline{w_i}$ and $m_{\overline{w_i}}$
10 Create model $\theta_{\overline{w_i}}$ by applying weigths $\overline{w_i}$ on clients' model parameters Θ
11 Create model $\theta_{T_{\overline{w_i}}}$ by applying weigths $T_{\overline{w_i}}$ on clients' model parameters Θ
 `// Model Evaluation`
12 Compute score of $\theta_{\overline{w_i}}$ and score of $\theta_{T_{\overline{w_i}}}$ with regard to ϵ bias threshold and high accuracy, and store them in $Score(\theta_{\overline{w_i}})$ and $Score(\theta_{T_{\overline{w_i}}})$ respectively
 `// Model Selection`
13 **if** $Score(\theta_{T_{\overline{w_i}}}) > Score(\theta_{\overline{w_i}})$ **then**
14 Add $T_{\overline{w_i}}$ to P_{new}
15 **else**
16 Add $\overline{w_i}$ to P_{new}
17
18 *nb_iter*++
19 Replace the content of P by the content of P_{new}
20 **if** *all elements of P have similar scores, or if* nb_iter = max_iter **then**
21 Set *termination_condition* to True
22 **while** *termination_condition* $\neq$ *True*;
23 Find among elements of P the vector of weights w_{best} with the highest score
24 Compute new global FL model $\theta_{w_{best}}$ by applying w_{best} on Θ
25 **return** $\theta_{w_{best}}$

7.3.2 Model Evaluation

This step evaluates the accuracy and bias of the models generated in the Model Generation part, and assigns a score to each using a fitness function (line 12). To define an appropriate fitness function for our problem, we follow the known approach of black-box constrained optimization where the fitness function is composed of two main terms: the objective function, and a global penalty term [134, 98]. The objective function term represents the goal of the problem and measures how well a solution performs in achieving this goal, which in our case

is model's accuracy. The global penalty term is to be subtracted from the objective function in the fitness function formula. Its role is to take into account all the constraints imposed by the problem by penalizing solutions that do not respect the constraints, which in our case are the bias constraints. The global penalty term is the sum of penalties of the different constraints of the problem. A penalty is usually proportional to the deviation from the constraint and takes various forms, such as squared penalty and absolute penalty. Squared penalty gives more penalty to solutions that deviate more from the constraint. When the deviation from the constraint is below 1, the squared penalty gives this deviation even smaller weight. In our case, the deviation from the bias constraints is generally below 1. We apply on our constraints the absolute penalty that does not exacerbate or underestimate deviations from the constraints. Thus, the fitness function of ASTRAL is defined as follows:

$$Score(\theta_{\overline{w}}, \epsilon) = A(\theta_{\overline{w}}) - \sum_{\left|\beta_{S_j}(\theta_{\overline{w}})\right| > \epsilon} \left|\beta_{S_j}(\theta_{\overline{w}})\right| \tag{7.7}$$

Where $Score(\theta_{\overline{w}}, \epsilon)$ is the fitness of a solution candidate $\overline{w}$, with ϵ being the predefined bias threshold, $\theta_{\overline{w}} = \overline{w}.\Theta$ being the corresponding global model to $\overline{w}$, $A(\theta_{\overline{w}})$ its accuracy and $\beta_{S_j}(\theta_{\overline{w}})$ the model bias with regard to the sensitive attribute $S_j \in \{S_1 \dots S_{sa}\}$. The global penalty term is computed by summing bias penalties of all the sensitive attributes S_j whose bias exceeds the predefined bias threshold ϵ. This fitness function is then used to evaluate the fitness scores of candidate models during the model evaluation step. The computation of these fitness scores and global metrics in practice is described afterwards in the dedicated paragraph.

7.3.3 Model Selection

After the models' evaluation, the system uses the *selection* operator to compare the parent $\overline{w}$ with its offspring $T_{\overline{w}}$ and only keeps the best in terms of model fitness in the new population called P_{new} (line 13-16). After iterating through all current elements in P, the latter is replaced with the newly updated population P_{new}. Finally, the system has a termination condition to determine whether it should finish ASTRAL execution for the current round or send back the updated population for a new iteration (lines 20-21). This termination condition can be whether the number of iterations done through ASTRAL has reached a maximum value denoted max_iter, or if all elements of P have similar scores. If the termination condition is respected, then ASTRAL identifies the best individual w_{best}, which is the aggregation weight that results in the model with the best score (line 21). It then computes the associated aggregated model as the new global model $\theta_{w_{best}}$ (line 24) for the current round. The new global model is sent back to the clients to use at the beginning of the next FL round.

The described ASTRAL's pipeline is executed at each FL round. It may occur that for some rounds, the final population does not contain solutions that conform to the bias constraints. In that case, the selection operator selects solutions that do not respect the bias constraints. To overcome this issue, we rely on the exploration capabilities of ASTRAL, which we explain in the

next section and which showed high effectiveness in finding high quality solutions during the evaluation process (see Chapter 8). Another efficient strategy we employ is increasing when necessary the size of the populations to be explored in order to explore better the search space. Nevertheless, our empirical analysis shows that such scenario is infrequent. Our approach demonstrated its ability in generating high-quality feasible solutions, in the FL early rounds (see Chapter 8). At the end of the FL rounds, ASTRAL outputs the best global model among the rounds' global models in terms of bias and accuracy objectives.

7.3.4 How FL Server Calculates Global Metrics

A pivotal factor in mitigating bias within the ASTRAL framework is the server's ability to compute the global model accuracy and bias metrics in each iteration of ASTRAL's pipeline (See Equation 7.7). The server uses the exchanged model updates and a proxy dataset at its possession to get the required metrics, obviating the need of sharing additional sensitive information. Using proxy dataset has found application in previous contexts such as [122] and [23]. In these works the proxy dataset is used for knowledge distillation enabling the transfer of knowledge from individual client models to the global model. Likewise, FedLAW uses a globally consistent proxy dataset and the gradient descent algorithm to optimize FL aggregation [119]. Within ASTRAL, the use of this proxy dataset gives a strategic advantage in mitigating bias. In Chapter 8 §8.12, we show the effectiveness of our proposed method, even when the proxy dataset diverges from the training data distribution.

To evaluate global metrics on the server-side without requiring a proxy dataset, alternative methods exist. One method involves cryptographic techniques such as secure aggregation [61], where global metrics are computed by aggregating values from participating FL clients at the server without revealing individual values. However, this approach introduces a computational overhead. Another approach, similar to [198], involves considering a subset of the clients as proxy data holders and computing the required metrics from these clients, while not including them in the training. Additionally, in the absence of proxy data, artificial data can be generated using existing privacy-preserving data generation methods in FL [222].

7.4 Analytical Insights

7.4.1 Maximizing FL Model Accuracy

ASTRAL finds the solution that maximizes accuracy by integrating the accuracy in the fitness function. This allows to assign high fitness scores to high accuracy solutions leading to their systematic selection as the best solution candidates by our pipeline. Furthermore, ASTRAL is able to ensure globally optimal solutions that maximize accuracy as much as possible by optimizing the exploration of the search space. First, ASTRAL leverages mutation, crossover, and selection operations, which are critical in forming sequences of populations and generating both diverse and good quality solutions. These operations are applied iteratively at the

server level, enabling ASTRAL to explore the search space more effectively. Then, ASTRAL runs through several FL rounds, which further enhances exploration, as explained in the following. At the start of each round, the pipeline of ASTRAL sends the global model to the clients for training. The clients subsequently submit their models (locally trained) to ASTRAL at the end of the round. Each client has its own local model, which represents its unique knowledge learned from its local data. The pipeline of ASTRAL then explores new populations of aggregation weights to apply to these local models. The exploration of new aggregation weights leads to the exploration of new global models. The explored global models of each FL round are formed via a linear combination of the local models' parameters using the aggregation weights candidates. Formally, at each round, ASTRAL explores a potentially new subspace of the global model search space, equivalent to the span generated from the algebraic basis formed by the local models of the clients of the round. The span generated by the algebraic basis formed by the local models of the clients for a round t is formally defined by the following equation:

$$Span_t(\theta_1^t, \theta_2^t, \ldots, \theta_N^t) = \{w_1.\theta_1^t + w_2.\theta_2^t + \cdots + w_N.\theta_N^t | w_1, w_2, \ldots, w_N \in \mathbb{R}\} \tag{7.8}$$

where $\theta_1^t, \theta_2^t, \ldots, \theta_N^t$ are the local models of the clients at round t. By combining the local models of clients in different FL rounds, ASTRAL can leverage the collective knowledge and learning from various training perspectives, depending on the FL round. The exploration of different spans generated by the local models as we progress through multiple FL rounds enhances the exploration of the search space.

7.4.2 Ensuring FL Model Bias Below a Threshold

In ASTRAL, the bias constraints are enforced by including them in the penalty term of the fitness function defined in Equation 7.7. Specifically, the penalty assigned to a model given a bias objective is computed as the sum of all the left terms of bias inequality constraints that are not respected, as follows:

$$penalty(\theta_{\overline{w}}, \epsilon) = \sum_{S_j \in [S_1, S_{sa}]} max(0, \beta_{S_j}(\theta_{\overline{w}}) - \epsilon) \tag{7.9}$$

This penalty ensures that models that exceed the bias threshold are penalized proportionally to the magnitude of their bias, while the models that do not exceed the bias threshold are not penalized and thus are the ones selected as potential solutions. The exploration capacities of ASTRAL described above have also crucial role at ensuring the bias objective. By leveraging the DE operators, and by exploring different spans generated by the local models of different clients in each FL round, ASTRAL effectively explores diverse subspaces of the global model search space. This allows to discover a diverse array of solutions that meet the bias constraints.

7.4.3 Generic Approach to Handle Several Bias Metrics

ASTRAL's generic framework can accommodate a wide range of bias metrics. Its black-box optimization approach does not require any specific conditions on the function being optimized, or the constraints, such as continuity or derivability. Moreover, leveraging black-box optimization offers another advantage to ASTRAL compared to existing methods. ASTRAL stands out as a bias mitigation solution by successfully optimizing the objective bias constraint itself without relying on approximations or surrogate functions. For instance, AgnosticFair [53] adopts an exact optimization technique and thus resort to approximate the statistical parity difference metric with its convex proxy, the decision boundary fairness metric [234]. However, this approximation fails to provide guarantees regarding the mitigation of statistical parity difference [234, 220].

7.4.4 Algorithm Complexity Analysis

The algorithm complexity of the proposed solution outlined in algorithm 2 depends on the algorithm complexity of ASTRAL's pipeline steps. The pipeline functions over a maximum of max_{iter} iterations. In each iteration, the aggregation weights population of size $|P|$ is browsed, and the model generation, evaluation and selection operations are applied. The model generation process (lines 6-11) comprises the selection of the best solution candidate, which cost depends on $|P|$. Then the mutation and crossover operations are applied, with costs proportional to number of the participating clients k that represents the dimension of the aggregation weights. The model generation process also includes the cost of creating the global models corresponding to the mutant and crossover vectors, dependant on $k \cdot d$, where d is the number of parameters of the learning model. The model evaluation phase (line 12) cost depends on c, the cost of applying the fitness function on a candidate solution. The fitness function computations costs depend on how the FL server calculates global metrics (See section 7.3). Finally, during the model selection phase (lines 13-16), the population is updated based on the evaluation scores, introducing a constant cost. When the stopping criteria is met, in the worst case attaining the predefined maximum iteration count max_{iter}, the weights with the highest scores are selected, and used to create the new global FL model (lines 23-24). This operation entails a cost depending on $|P| + (k \cdot d)$. Thus, the overall complexity is formulated as follows:

$$C = \mathcal{O}(max_{iter} \cdot |P| \cdot (|P| + k + k \cdot d + c) + |P| + k \cdot d) \tag{7.10}$$

As presented in previous studies [75], population size does not exceed 50 for most engineering problems. Thus, model generation operations are less intensive than the model evaluation and $|P|$ is lower than c. The algorithm complexity of the proposed method is expressed

as Equation 7.11.

$$\mathcal{O}(max_{iter} \cdot |P| \cdot (k \cdot d + c)) \tag{7.11}$$

Taking this complexity into consideration helps to determine the various factors impacting the computational cost of ASTRAL. Reducing both of population size and the maximum iteration count and adopting low-cost techniques to get models' metrics reduces significantly ASTRAL's cost, and allows it to scale efficiently to complex models and data, and high number of clients. The calibration of these parameters depends on the unique requirements of the given scenario and the characteristics of the used data and models.

7.5 Summary

In this chapter, we delved into the design of ASTRAL, exploring its architecture and components. We justified design decisions, demonstrating how ASTRAL tackles challenges in existing FL bias mitigation works. In the upcoming chapter, we evaluate ASTRAL's performance through empirical analysis and experiments, demonstrating its practicality and effectiveness in real-world scenarios.

8 Experimental Evaluation

In this chapter, we assess the effectiveness of our bias mitigation proposal with three popular bias metrics, seven widely used datasets and various experimental setups. Specifically, we seek to answer the following questions:

- How efficient is ASTRAL in reaching bias constraint compared to other techniques? (in §8.4)
- How efficient is ASTRAL in maximizing accuracy while reaching bias constraint compared to other techniques? (in §8.5)
- In what ways might the mitigation of a single sensitive attribute result in more bias? (in §8.6)
- How well does ASTRAL meet different fairness objectives, from the least stringent to the most stringent? (in §8.7)
- How scalable is ASTRAL with regard to the data size? (in §8.8)
- How scalable is ASTRAL with regard to the number of clients? (in §8.9)
- How scalable is ASTRAL with regard to different FL clients participation ratios? (in §8.10)
- How robust is ASTRAL with regard to data heterogeneity? (in §8.11)
- What is the impact of the data distribution of the proxy dataset on ASTRAL's performance? (in §8.12)
- How stable is ASTRAL's bias mitigation? (in §8.13)
- What is the cost of ASTRAL in terms of bias, accuracy, and communication efficiency trade-offs? (in §8.14)

We first describe the datatsets, implementation details and experimental setup, and the bias mitigation techniques against which we compare. Then, we present the empirical evaluation of ASTRAL.

Table 8.1: Characteristics of the real-world datasets used in our evaluation. We include: topic, the geographical location (Loc.), the total number of raw and cleaned records, the number of available attributes (#Attr.), the sensitive attributes and the target class.

Dataset	Topic	Loc.	#Records (cleaned)	#Attr.	Sensitive Attributes	Other Attributes	Target Class
ARS	Healthcare	ASTL	75,128 (75,128)	9	*gender*	acceleration on axis x,y,z,signal strength	activity
MobiAct	Healthcare	GR	16,756,325 (1,852,861)	16	*age*, *gender*	activity, roll, pitch, acc_x	activity
CelebA	Emotion recognition	CN	202,599 (202,599)	3074	*age*, *gender*	Images' pixels values	emotion
KDD	Finance	US	299,285 (272,507)	41	*age*, *gender*, *race*	education level, work, category, native country	income
DC	Finance	NL	60,420 (60,420)	12	*age*, *gender*	citizenship, birth country, education level	job
Adult	Finance	US	48,842 (48,842)	15	*age*, *gender*, *race*	education level, hours, worked/week, native country	income
MEPS	Healthcare	US	35,428 (33,401)	134	*gender*, *race*	occupation, US region, income	health

8.1 Datasets and Models

We use seven real-world datasets, chosen specifically for their well-known bias issues. Table 8.1 summarizes their characteristics.

ARS. The ARS ubiquitous dataset is used for activity recognition of healthy older people using a batteryless wearable sensor dataset [189]. It has been collected in two clinical rooms equipped with four and three RFID reader antennas respectively. Fourteen volunteers aged between 66 and 86 years were trialed. Each wore a wearable sensor (W2-ISP) at the sternum level, and undertook a series of scripted activities including (*i*) lying on a bed, (*ii*) sitting on a bed, (*iii*) sitting on a chair, and (*iv*) ambulating. The classification task is to predict whether a person is lying or not for ambulatory monitoring. The only sensitive attribute present in this dataset is gender. To accomplish the learning task of ARS, we use a linear support vector machine model. In the case of FCFL which does not support SVM models, we employ logistic regression.

MobiAct. The MobiAct dataset has been designed for smartphone-based human activity recognition [210]. It contains data collected from accelerometers, gyroscopes, and orientation sensors of smartphones. It captures various daily activities performed by 66 participants with more than 3200 trials. Our primary objective is to identify individuals who are standing and those who are not. The sensitive attributes are gender and age. To accomplish the learning

task, we employ a 3-layer Multilayer Perceptron (MLP) model [152].

CelebA. CelebA is a public dataset containing over 200,000 facial images of 10,177 different celebrities [123]. Each image is annotated with 40 attribute labels including gender and age as sensitive attributes. As learning task we consider emotion recognition, more precisely smile detection. We use a deep learning model composed of the ResNet18 convolutional neural network [83] and a fully connected neural network. ResNet18 is used to process the dataset images' pixels while the other network is used to process ResNet18's outcomes and the sensitive attributes of the images. The whole model outputs the classification result.

KDD. The KDD dataset contains weighted census data extracted from the 1994 and 1995 population surveys conducted in the US [54]. It includes 299,285 records with 41 attributes including age, education level, race, gender, *etc.* The prediction task is to determine whether an individual's income exceeds $50K or not. To perform this task we use a logistic regression model. Sensitive attributes are *age*, *gender* (men, women), and *race*. We binarized the *race* attribute into two groups, considering as *privileged* the Whites, Asians and Pacific Islanders, and *unprivileged* all others. Similarly, we binarized the *age* attribute by considering *privileged* all people aged between 30 and 60, and *unprivileged* all the others.

DC. The Dutch Census dataset collects census information in the Netherlands in 2001 [109]. It includes 60,420 records with 12 attributes. The prediction task is to classify a person's occupation as having high or low prestige, the model used for this task is logistic regression. Sensitive attributes are *age* and *gender*. We binarized the *age* by considering *privileged* those whose *age* is below 50, and *unprivileged* the others.

Adult. The Adult dataset contains 48,842 records and a total of 15 attributes [105], extracted from the 1994 US Census database. It includes individuals' information such as age, education level, race, gender, *etc.* The prediction task is the same as KDD, considering the same sensitive attributes. The used model here is logistic regression as well. We applied the same binarization process to the *race* and *age* attributes of Adult as in KDD.

MEPS. The MEPS dataset originates from the Medical Expenditure Panel Survey in the US [33]. It consists of 35,428 records with 134 attributes. The task is to predict whether an individual has used medical facilities more than 10 times. To do so, we apply a logistic regression model. Sensitive attributes are *gender* and *race*. We binarized the *race* attribute, considering Non-Hispanic Whites as *privileged*, and the others as *unprivileged*.

For all the datasets, the binarization of non-binary sensitive attributes is done through a preprocessing step. The binarization is necessary since bias metrics are defined for binary sensitive attributes only. It involves clustering the sensitive attribute values into two groups based on the positive outcome ratio computed for each group using the ground truth. The group with higher ratios is classified as privileged, while the other group is unprivileged. The value of the sensitive attribute is then replaced by the obtained binary representation.

Additionally, in each group of experiments performed with the same dataset, we initialize the training with the same initial model. It provides a fair ground for experimentation and comparison with baselines.

8.2 Implementation and Experimental Setup

ASTRAL is implemented in Python (v3.9) and PyTorch (v1.10). We use the SciPy implementation of Differential Evolution [47]. By default, we set in our experimentations the initial population size to 15, we set 8 parallel workers, and keep the other method parameters equal their default values. We run our experiments on a testbed platform with the following characteristics: 2× Intel Xeon E5-2650 v4 (14 cores), 128 GB of RAM and 2× NVIDIA GTX 1080 Ti. We simulate a network with a download speed of 20 Mbps and an upload speed of 5 Mbps. Datasets are split into three distinct subsets, namely training set (80%), test set (10%), and proxy dataset (10%) to compute global metrics (section 7.3). To emulate FL clients and run the FL process, the training set is distributed according to the FL settings of each experiment. In our experiments, data is distributed among FL clients in a non-IID setting using Dirichlet function with regard to a particular sensitive attribute for each dataset [88]. This means that each FL client receives a different distribution of data based on the considered sensitive attribute. For each value v^j of the sensitive attribute S_j, we sample $p_{v^j} \sim Dir(\alpha)$ with α parameter, and allocate a portion $p_{v^j,k}$ of the data points with $S_j = v^j$ to client k. The heterogeneity of the distributions across clients is controlled via α parameter. When $\alpha \to \infty$, the distributions are IID, whereas smaller values of α lead to more heterogeneous distributions. In our experiments, we generate clients data distributions with $\alpha = 0.02$. The FL number of clients varies depending on the dataset, with respectively 4 clients for MEPS and CelebA, 5 for KDD, 10 clients for both ARS, DC and Adult. For MobiAct, we use a subset of the whole dataset (12 clients) that exhibits bias since the dataset as a whole, with a total of 66 clients, does not demonstrate bias. Logistic regression models are applied to Adult, MEPS, KDD and DC datasets. For Adult and MEPS datasets, we use a batch size of 128 and a learning rate of 0.01, while for KDD, we use a model with the same batch size and a learning rate of 0.001. In the case of DC, we use a batch size of 256 and a learning rate of 0.0005. For ARS dataset, we use a batch size of 1024 and a learning rate of 0.001 with SVM model. For MobiAct, we use a Multilayer perceptron with a batch size of 256 and a learning rate of 0.01. For CelebA, we use a ResNet18 based model with a batch size of 1024 and a dynamic learning rate initialized to 0.01. In several bias mitigation works [53, 36], the typical bias threshold is considered to be 3%. Thus, we consider this threshold per default in our study. Further details of the implementation and experimental setup can be found in ASTRAL's repository, where we provide the software prototype, the used datasets, and the configuration settings: *https://github.com/FL-Bias/ASTRAL*

8.3 Baselines

We compare ASTRAL against the following FL methods.

FCFL [36]. This state-of-the-art FL bias mitigation method formulates a per-client bias mitigation problem, where the objective is to guarantee a bias constraint on each client local distribution while maintaining similar model accuracy across all clients. This is done through gradient-based optimization. We use the original FCFL implementation [37], based on PyTorch 1.6 and Python 3.7.

FairFed*. As far as we know, the software prototype of FairFed [61] is not publicly available. We implement the algorithm presented in FairFed, and adapt it to reweigh client's updates based on their bias and the predefined bias threshold. More precisely, given a sensitive attribute S_j and a bias metric β, in order to mitigate bias in FairFed*, a factor is applied to the classical FL weight w_k from Equation 6.2, depending on the amount of bias $\beta_S(\theta_k)$ of client k. Formally, we compute the weights w'_k as $w'_k = w_k.g(\beta_S(\theta_k))$, where:

$$g(\beta_S(\theta_k)) = e^{-b(-\log(1-|\beta_{S_j}(\theta_k)|)^a} \tag{8.1}$$

where a and b are two constant parameters. We tune them in each experiment to obtain the best model with regard to accuracy and bias objectives. The function g is inspired by the Prelec reweighing function [174]. Thus, FairFed* favors clients with lower model bias.

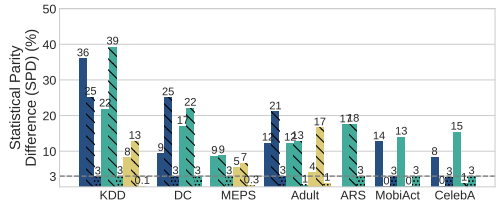
FairFL*. FairFL proposes a client selection policy that relies on a reinforcement learning algorithm and selects participants depending on their contribution to the final model's bias [240]. Since the original software prototype of FairFL is not publicly available, we implemented a simple heuristic, referred to as FairFL*, that selects at each round clients whose local models' bias is below the given threshold ϵ. More formally, w_k in Equation 6.2 is set to 0 for every θ_k for which there exists a sensitive attribute $S_j \in \{S_1 \dots S_{sa}\}$ such that $\beta_{S_j}(\theta_k) > \epsilon$.

FedAvg [135]. We also consider the case of FedAvg, where no bias mitigation is applied, and where default FL model aggregation and client selection are used (see Chapter 6 §6.1). This allows us to compare the FL model accuracy obtained in ASTRAL with the case where no bias mitigation is applied.

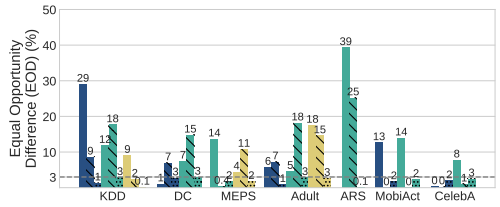
We note that among the mentioned techniques, only FairFL* supports the existence of multiple sensitive attributes in data. Instead, FCFL and FairFed* can only handle single sensitive attribute bias mitigation. Additionally, it is not a trivial task to adapt the two latter methods to handle multiple sensitive attributes bias mitigation.

8.4 Comparison of ASTRAL against Existing FL Bias Mitigation Mechanisms

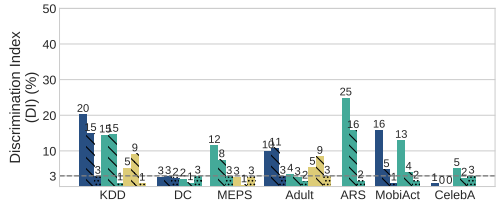
We start by evaluating how ASTRAL addresses bias when considering several sensitive attributes simultaneously. We consider multiple sensitive attributes depending on the dataset columns as follows: for KDD and Adult, we consider gender, race, and age; for DC, MobiAct and CelebA,



(a) Statistical Parity Difference



(b) Equal Opportunity Difference

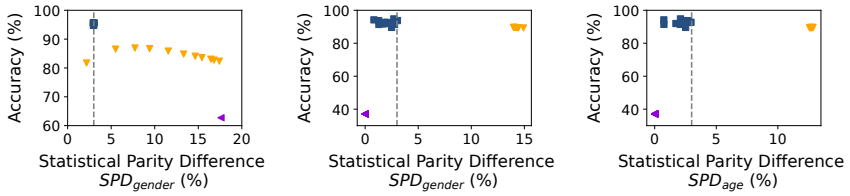


(c) Discrimination Index

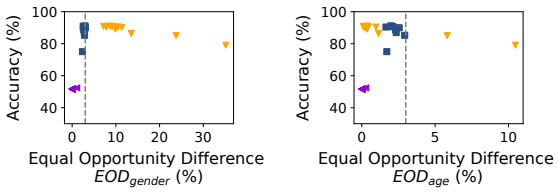
Figure 8.1: Bias mitigation with regard to multiple sensitive attributes using SPD, EOD and DI metrics – KDD, DC, MEPS, Adult, ARS, CelebeA and MobiAct datasets – With respectively 5, 10, 4, 10, 10, 4, and 12 FL clients and a bias threshold of 3%.

sensitive attributes are gender and age; for MEPS sensitive attributes are gender and race. The ARS dataset contains only one sensitive attribute (gender), but we still include the results of applying ASTRAL on it. To assess its effectiveness, we compare ASTRAL against FedAvg and FairFL*. We only use FairFL* because it is the only method that can handle multiple sensitive attributes, as mentioned in section 8.3. In all experiments, we fixed a bias constraint of 3%, and our evaluation includes SPD, EOD, and DI metrics. We report the model's bias results in Figure 8.1. We observe that ASTRAL matches the bias constraint for all the multiple sensitive attributes scenarios with the different datasets and bias metrics. FedAvg and FairFL* do not satisfy the bias constraint. We explain this as follows. First, FedAvg does not mitigate bias by design, so it yields biased models. Then, for FairFL*, the method takes a highly pessimistic approach. In most of the experiments, it does not select any client model for aggregation from the first round because their local models are already biased; so it just outputs the initial model. Exceptional cases are when FairFL* is applied on DC considering DI metric, on MobiAct considering SPD and EOD, and on CelebA considering the different bias metrics. In the latter cases, FairFL* is able to select clients that meet the bias constraint, resulting in an unbiased

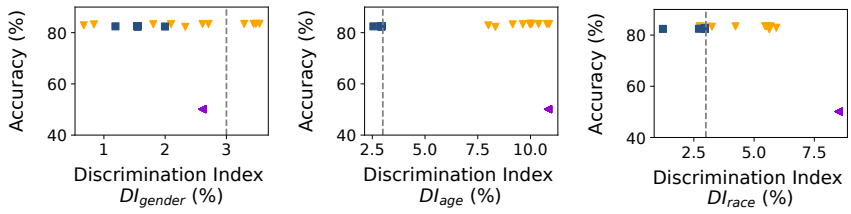
model. However, since FairFL* allows less clients to join the training, it achieves low accuracies compared to ASTRAL. For example, for DC when considering DI, FairFL* achieves 81% while ASTRAL achieves 83%. More bias and accuracy trade-offs results are shown in section 8.5.



(a) ARS – Bias *vs.* Accuracy (b) MobiAct – Bias_{gender} *vs.* Accuracy (c) MobiAct – Bias_{age} *vs.* Accuracy



(d) CelebA – Bias_{gender} *vs.* Accuracy (e) CelebA – Bias_{age} *vs.* Accuracy



(f) Adult – Bias_{gender} *vs.* Accuracy (g) Adult – Bias_{age} *vs.* Accuracy (h) Adult – Bias_{race} *vs.* Accuracy

Figure 8.2: Trade-off between bias and model accuracy

8.5 Trade-off Between Bias and Model Accuracy

In the following, we evaluate the effectiveness of ASTRAL in terms of bias and accuracy trade-offs. To do so, we report measurements of bias with regard to multiple sensitive attributes along with measurements of accuracy. We consider the baseline system without bias mitigation (*i.e.*, FedAvg), FairFL*, and ASTRAL. The measurements are collected for each method and dataset at 10 representative rounds. The representative rounds are the percentiles 10^{th}, 20^{th} ... until the 100^{th} of the FL rounds' ordered values. In addition to these rounds, we include the performance of each method at the round where it achieves its highest accuracy while maintaining the bias constraint. Due to space limitations, we present the results considering

different bias metrics for ARS (Figure 8.2(a)), MobiAct (Figure 8.2(b) and Figure 8.2(c)), CelebA (Figure 8.2(d) and Figure 8.2(e)), and Adult (Figure 8.2(f), Figure 8.2(g), and Figure 8.2(h)). The bias threshold is fixed to 3%.

The reported measurements indicate that ASTRAL consistently maintains the bias objective for the different sensitive attributes of the considered datasets across the considered rounds. Simultaneously, ASTRAL achieves higher model accuracies compared to the other bias mitigation technique FairFL* and this for all the datasets, we refer this to FairFL*'s clients selection approach that results in low accuracies. ASTRAL is even able to reach higher accuracy compared to the baseline which does not mitigate bias by design in the case of ARS and MobiAct. This result is explained by the effectiveness of ASTRAL's algorithm in selecting the best aggregation weights that maximize the system accuracy, allowing to reach the best trade-offs of bias and accuracy in our experiments.

8.6 Mitigating Single Sensitive Attribute Bias May Induce a Higher Bias With Regard to the Other Sensitive Attributes

To measure how bias mitigation on a single sensitive attribute affects the model bias on other sensitive attributes within the same dataset, we compare ASTRAL with the two other state of the art methods, that handle a single sensitive attribute, namely FCFL and FairFed*. We fix a bias constraint of 3%, and we consider one single sensitive attribute for each dataset: *age* for KDD and Adult, *race* for MEPS and *gender* for DC, ARS and MobiAct. Table 8.2(a), Table 8.2(b) and Table 8.2(c) show the results respectively for a set of experiments done on the SPD, EOD and DI bias metrics. The second column from left for each table reports bias measurements on the sensitive attribute considered during bias mitigation, while the two last columns report bias measurements on the remaining sensitive attributes.

We observe that focusing solely on mitigating bias with respect to one sensitive attribute may have unwanted consequences with regard to the others. For instance, the EOD of FairFed* for KDD dataset on the age sensitive attribute, shown in Table 8.2(c), is equal to 3%, showing that the model bias on the age sensitive attribute is mitigated. However we see that for the same scenario, the EOD on the race sensitive attribute increased when comparing FedAvg and FairFed*, from 12% to 21%. So, mitigating the bias towards one sensitive attribute does not guarantee mitigating the bias for the other sensitive attributes, and worse, it may degrades them. The same phenomenon is observed for SPD and DI with both FCFL and FairFed* for Adult, MEPS, KDD and MobiAct as shown in Table 8.2(a), Table 8.2(b), and Table 8.2(c). On the contrary, ASTRAL is able to mitigate bias with regard to all sensitive attributes simultaneously.

Table 8.2: Exacerbation of bias when mitigating considering a single sensitive attribute, with SPD, EOD, and DI – the first bias column from left represents the bias with regard to the sensitive attribute considered by the mitigation mechanism

(a) Mitigating bias with regard to SPD

Adult			
System	**SPD$_{age}$(%)**	**SPD$_{race}$(%)**	**SPD$_{gender}$(%)**
FedAvg	13	4	13
FCFL	1	9	19
ASTRAL	3	1	3
MEPS			
System	**SPD$_{race}$(%)**	**SPD$_{gender}$(%)**	
FedAvg	5	8	
FCFL	2	10	
ASTRAL	0	3	
MobiAct			
System	**SPD$_{gender}$(%)**	**SPD$_{age}$(%)**	
FedAvg	13	14	
FairFed*	3	21	
ASTRAL	3	3	

(b) Mitigating bias with regard to DI

KDD			
System	**DI$_{age}$(%)**	**DI$_{race}$(%)**	**DI$_{gender}$(%)**
FedAvg	20	5	14
FairFed*	0	22	8
ASTRAL	3	1	1
Adult			
System	**DI$_{age}$(%)**	**DI$_{race}$(%)**	**DI$_{gender}$(%)**
FedAvg	10	6	4
FairFed*	0	13	13
ASTRAL	3	3	2

(c) Mitigating bias with regard to EOD

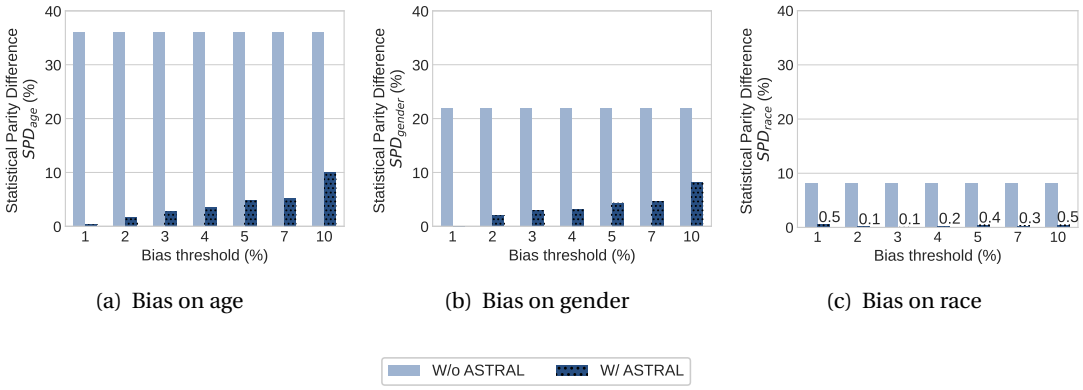
KDD			
System	**EOD$_{age}$(%)**	**EOD$_{gender}$(%)**	**EOD$_{race}$(%)**
FedAvg	29	9	12
FairFed*	3	2	21
ASTRAL	1	3	0
DC			
System	**EOD$_{gender}$(%)**	**EOD$_{age}$(%)**	
FedAvg	8	4	
FairFed*	3	6	
ASTRAL	3	3	
Adult			
System	**EOD$_{age}$(%)**	**EOD$_{gender}$(%)**	**EOD$_{race}$(%)**
FedAvg	8	8	17
FairFed*	0	10	20
ASTRAL	1	3	3
MEPS			
System	**EOD$_{race}$(%)**	**EOD$_{gender}$(%)**	
FedAvg	6	14	
FairFed*	2	16	
ASTRAL	2	2	

8.7 Evaluation of Bias Mitigation under Different Bias Constraints

To determine if ASTRAL is able to enforce different stringent bias constraints on the FL model, we run our solution under more or less strict bias constraints, using KDD dataset with age, gender and race as sensitive attributes, and considering the SPD metric. The results are presented in Figure 8.3. It shows the global model's bias with regard to the three sensitive attributes, measured in different setups with different bias limits imposed. We observe that ASTRAL is able to successfully provide FL a model that satisfies the objective for all the sensitive attributes, even when hardening the bias constraint.

8.8 FL System Scalability with Regard to the Data Size

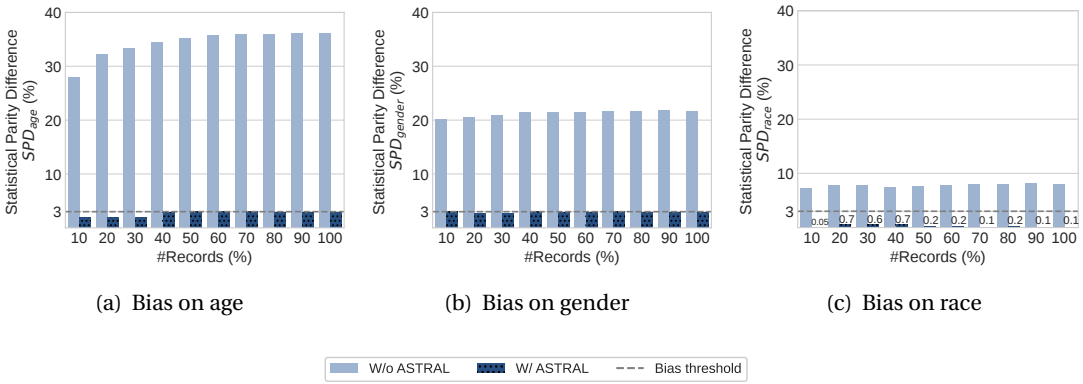
We analyze how ASTRAL performs when varying the size of the training data. For this set of scalability experiments, we consider the SPD as the bias metric. We also consider the KDD dataset, split in several partitions of data, each containing a different percentage of data records of the whole dataset. We include the data from each partition in the ones with a larger



(a) Bias on age (b) Bias on gender (c) Bias on race

Figure 8.3: Effectiveness of bias mitigation under different bias thresholds

number of records. Namely, the 10% partition is included in the 20% partition, and so on, until reaching the 100% partition, which contains the whole KDD train set. Figure 8.4 reports bias measurements without bias mitigation (*i.e.*, FedAvg) and with ASTRAL, and shows how ASTRAL performs as a function of the partition size for KDD. We observe that ASTRAL's trend is extremely stable for this dataset. ASTRAL maintains the bias metric to 3%, ensuring the bias objective as data size increases, and this for the three sensitive attributes age, gender and race, as shown by Figure 8.4(a), Figure 8.4(b), and Figure 8.4(c). In summary, ASTRAL's bias mitigation is not affected by the data scale.



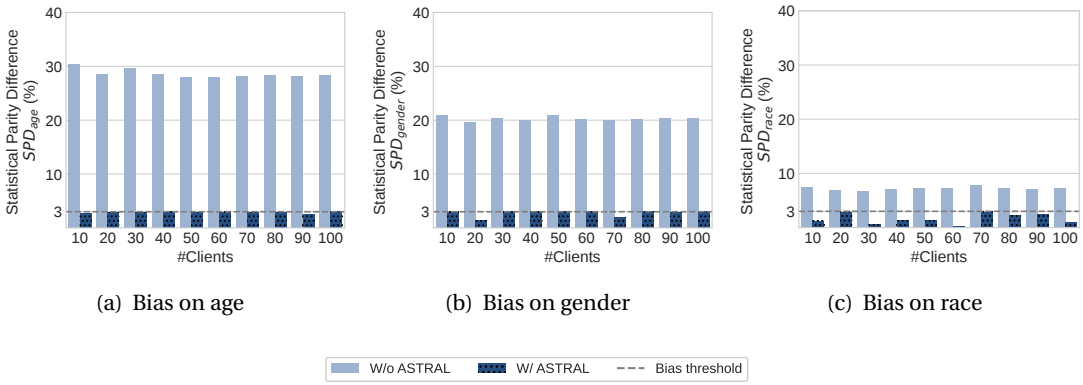
(a) Bias on age (b) Bias on gender (c) Bias on race

Figure 8.4: ASTRAL's scalability with regard to the data size scale

8.9 FL System Scalability with Regard to the Number of Clients

We analyze how ASTRAL performs with the number of FL client. We consider the KDD dataset, where we configure FL setups ranging from 10 clients to 100 clients. We include the data from a specific client setup in the data of the other FL setup with a more significant number of clients. Namely, the 10-client setup data is included in the 20-client setup data, and so on,

until reaching the 100-client setup, which contains the whole KDD train set. For each client setup, we used the same data distribution, experimental setup, and hyper-parameters for ASTRAL and we set the bias threshold to be 3%. We report how ASTRAL performs with the number of clients compared to FedAvg baseline in Figure 8.5, showing SPD on age, gender and race sensitive attributes. We observe that for every number of clients, ASTRAL succeeds at making the global model respect the bias threshold.



(a) Bias on age (b) Bias on gender (c) Bias on race

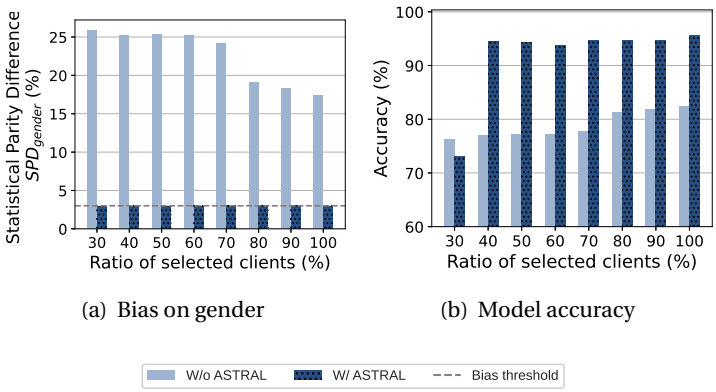
Figure 8.5: ASTRAL's scalability with regard to the number of clients scale

8.10 FL System Scalability with Regard to Client Selection Ratios

In addition to evaluating ASTRAL's performance when varying the number of FL clients, we assess ASTRAL's robustness under different FL client participation ratios. We gradually increase the ratio of selected clients participating in a FL round, from 30% to 100% out of 10 FL clients, using ARS dataset. The corresponding results for model accuracy and model bias measured through SPD are presented in Figure 8.6. The results show that ASTRAL consistently ensures that the global model adheres to the bias threshold while maximizing model accuracy with different client selection ratios. However, we observe that when 3 FL clients are selected, the model accuracy decreases compared to FedAvg. This can be explained by the fact that having only 3 FL participating clients reduces the aggregation weights search space for ASTRAL and, thus, impacts accuracy.

8.11 Robustness of Bias Mitigation with Respect to Clients' Data Heterogeneity

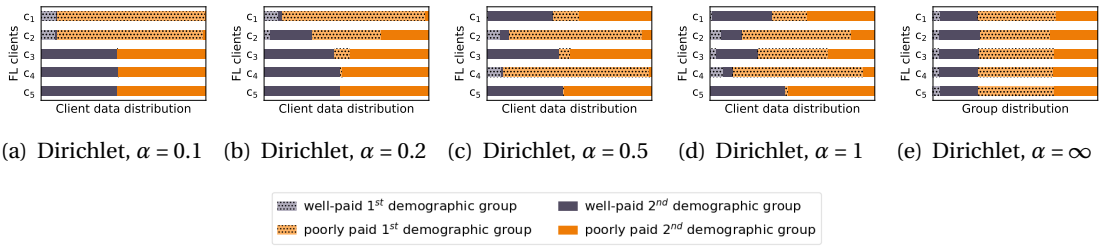
To evaluate a FL solution, a comprehensive benchmark must investigate its behavior in practical FL settings, in the presence of data heterogeneity. In this section, we evaluate how ASTRAL performs under different client's data heterogeneity degrees. To achieve this, we use KDD dataset. We generate several FL setups using Dirichlet process. We vary the α parameter of the Dirichlet process to control how identical the clients are in each setup. Smaller α values



(a) Bias on gender (b) Model accuracy

Figure 8.6: ASTRAL's scalability with regard to client selection ratios

result in a more heterogeneous data distribution among clients, while larger values lead to more similar and uniform clients as shown in Figure 8.7. We evaluate ASTRAL under different levels of heterogeneity, with $\alpha \in [0.1, 0.2, 0.5, 1]$ and $\alpha \to \infty$. We consider for bias mitigation the three sensitive attributes age, gender, and race, and the SPD metric.



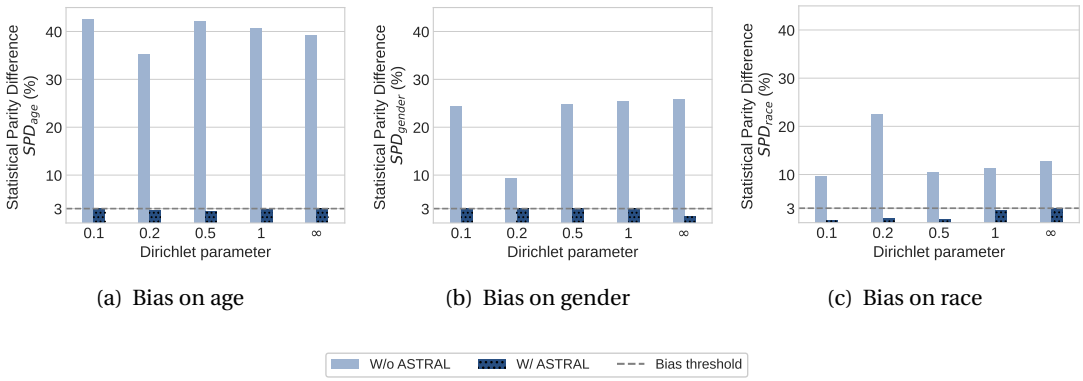
(a) Dirichlet, $\alpha = 0.1$ (b) Dirichlet, $\alpha = 0.2$ (c) Dirichlet, $\alpha = 0.5$ (d) Dirichlet, $\alpha = 1$ (e) Dirichlet, $\alpha = \infty$

Figure 8.7: Different non-IID client data distributions – More heterogeneous distribution from left to right – KDD dataset with 5 FL clients

Figure 8.8 shows the SPD values of both FedAvg and ASTRAL for the considered Dirichlet parameters, for age, gender, and race. ASTRAL consistently satisfies the 3% bias constraint at different levels of heterogeneity. These findings demonstrate that ASTRAL remains unaffected by varying data heterogeneity levels and effectively addresses bias concerns.

8.12 Impact of Data Distribution of Proxy Dataset on Bias Mitigation

In order to investigate the impact of the proxy dataset's data distribution on the proposed method, we evaluate the performance of ASTRAL with three distinct proxy datasets. These include one dataset that is representative of the training distribution, and two others deviating from the training distribution. We quantify the dissimilarity between the proxy datasets and the training data using the 1-Wasserstein distance [209], a proven metric for comparing data



(a) Bias on age (b) Bias on gender (c) Bias on race

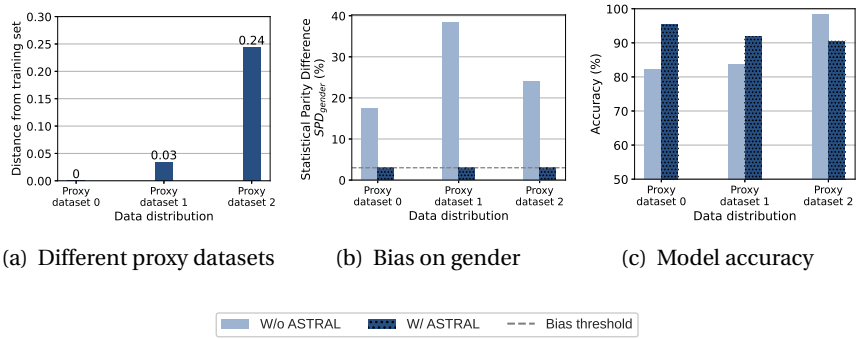
Figure 8.8: Effectiveness of bias mitigation under different levels of clients' data heterogeneity

distributions [207, 62, 84]. It explores dissimilarity by optimally mapping one distribution to another. The higher the distance value, the further apart the two distributions are. The range of the distance is influenced by the inherent characteristics of the data.

We show the evaluation results for ARS dataset in Figure 8.9, where we present the Wasserstein distances between the considered proxy datasets distributions and the training data distribution, the model bias (SPD) for each considered case, and the model accuracy. We observe that ASTRAL is able to successfully mitigate bias while achieving high accuracy for the different considered cases. Furthermore, we observe that when the proxy dataset's distribution is representative of the training data, ASTRAL achieves the highest accuracy as it can leverage fully the knowledge gained from the local models trained on the training distribution to construct the global model. As the proxy dataset diverges more from the training data distribution, the accuracy of the model obtained by ASTRAL declines. This is because it becomes more challenging to leverage the knowledge gained from one distribution (the training distribution) to produce a high-quality overall model with regard to a different distribution. This problem, that we consider orthogonal to our work, is known in ML literature as *data shift* [149]. There are several strategies and techniques that can be employed to mitigate its impact such as data augmentation, data monitoring, regularization techniques, *etc.* [149].

8.13 Stability of Bias Mitigation

In this section, we analyze the stability of ASTRAL at ensuring the bias threshold constraint through FL rounds. For that, we perform bias mitigation using ASTRAL applied on the SPD metric on all datasets. To be able to compare ASTRAL's stability to all the state of art methods at once, including FairFed* and FCFL, we report results from experiments considering one sensitive attribute for each dataset: *gender* for DC, ARS, CelebA and MobiAct, *age* for KDD and Adult and *race* for MEPS. The results for ARS, MobiAct, CelebA, KDD, DC, Adult and MEPS are reported respectively in Figure 8.10(a), Figure 8.10(b), Figure 8.10(c), Figure 8.10(d), Figure 8.10(e), Figure 8.10(g), and Figure 8.10(f). We can observe that for all the datasets, ASTRAL



(a) Different proxy datasets (b) Bias on gender (c) Model accuracy

Figure 8.9: Impact of different data distributions of proxy datasets on bias mitigation

maintains the bias under the threshold constraint for all the FL rounds, meaning that ASTRAL's pipeline is able to find feasible solutions that ensures the bias constraint starting from the first round. FairFL* fails as its restrictive policy makes it ignore all clients, except in MobiAct, CelebA and Adult where it selects some clients, but resulting in a global model with less accuracy. We explained this observation in section 8.4. The FairFed* policy reduces bias for the different datasets but only ensures the threshold for ARS, MobiAct and temporary for DC and CelebA. Moreover finding the proper aggregation weights that reduce bias in this method is a slow process compared to ASTRAL. As for FCFL, it is not compatible with the use of ResNet applied to CelebA dataset and MLP used with MobiAct. Consequently, the results are unavailable for this method on these particular datasets. It succeeds in guaranteeing the bias constraint only for MEPS and Adult. We observe on MEPS that in the first dozens rounds, FCFL struggles to keep the bias constraint in check and only stabilizes after 38 rounds. We observe a similar behavior on the Adult dataset for FCFL, only reaching stability in bias constraint respect after 300 rounds. In general, FCFL needs more FL rounds to stabilize. This is because FCFL relies on a gradient-based algorithm that modifies the global model after each local iteration at the clients' level, trying to ensure the optimization objective in the long run. For KDD and DC, FCFL is not able to reach the predefined threshold. For ARS, FCFL is not able to stabilize while maintaining the bias objective. As the authors mentioned in their work [36], FCFL's objective is to ensure the bias threshold for each client at its local level. This implies that, when data among clients is heterogeneous, and the global distribution is not represented by the individual clients distributions, than even if FCFL mitigates bias locally it does not mitigate it with regard to the global distribution. Meanwhile, ASTRAL manages to directly reach and stabilizes bias constraint within the first FL round on all datasets.

8.14 Cost Analysis

In order to assess how practical a given bias mitigation mechanism is, we evaluate its cost in terms of the number of FL rounds necessary to meet a bias constraint and maximize accuracy, as well as in terms of the amount of data exchanged over the network by the FL bias

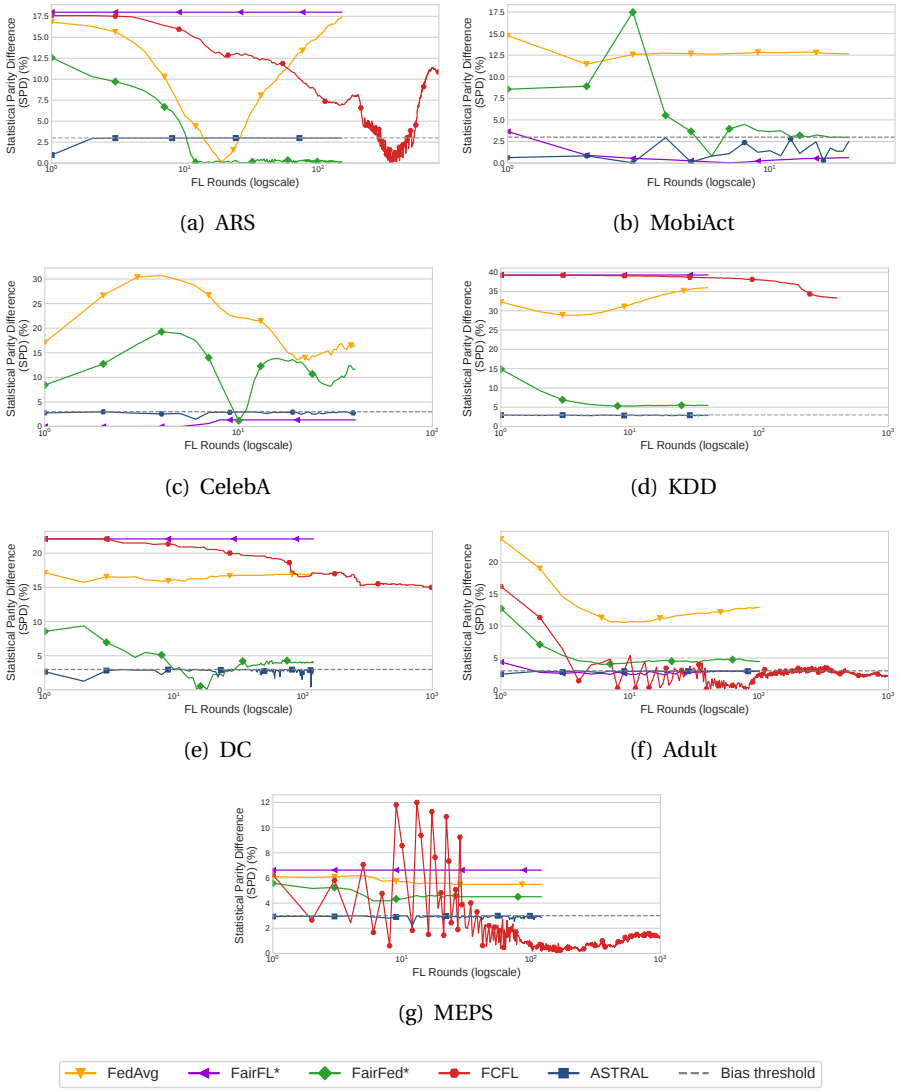


Figure 8.10: Stability of bias mitigation

mitigation protocol and its total execution time. We evaluate the different bias mitigation mechanisms (FairFL*, FairFed*, FCFL, ASTRAL) constraining SPD below 3%, on all the datasets KDD, DC, Adult, MEPS, ARS, CelebA, and MobiAct. In order to be able to consider FCFL in this comparative evaluation, and since the former only handles a single sensitive attribute, we evaluate the cost of all systems with bias mitigation for a single sensitive attribute. As mentioned previously, we have not been able to conduct experiments with FCFL for MobiAct and CelebA due to its lack of support to ResNet and MLP models.

We show the results in Table 8.3. The column 100^{th} of Table 8.3 presents the measurements collected when the bias mitigation methods reach their highest accuracy while maintaining bias under the predefined threshold. The column 95^{th} (respectively 90^{th}) presents the mea-

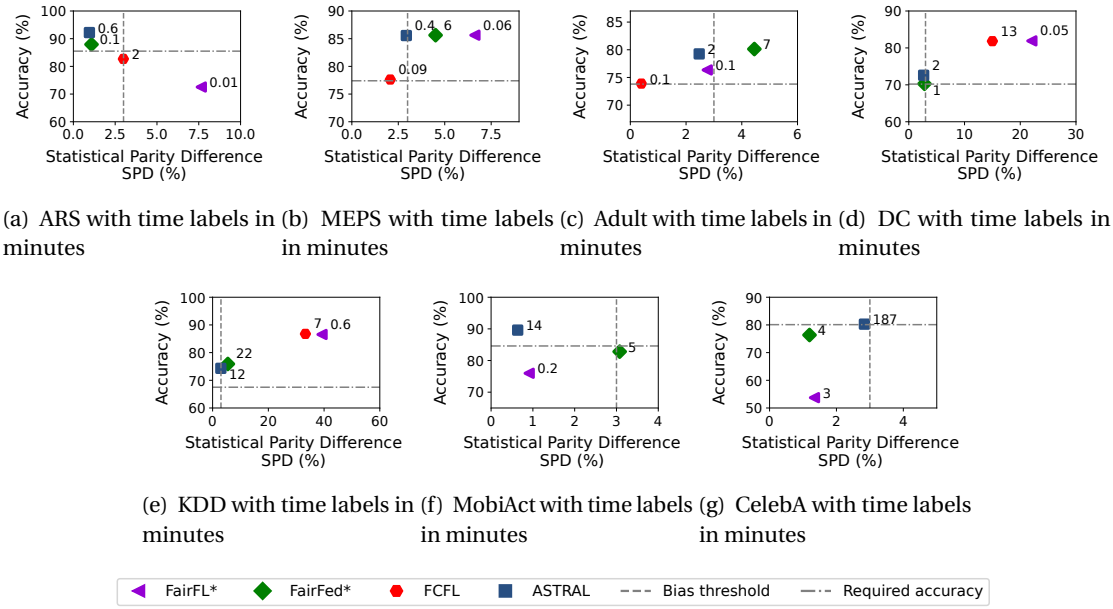
Table 8.3: Cost of bias mitigation. We show results for the max (100^{th}), the 95^{th} and the 90^{th} percentile of accuracy

		100^{th}			95^{th}			90^{th}		
Dataset	**System**	**FL Round**	**Exchanged data (KB)**	**Accuracy**	**FL Round**	**Exchanged data (KB)**	**Accuracy**	**FL Round**	**Exchanged data (KB)**	**Accuracy**
ARS	FairFL*	✗	✗	✗	✗	✗	✗	✗	✗	✗
ARS	FairFed*	48	38	93	16	13	90	11	9	88
ARS	FCFL	**519**	**649**	**83**	✗	✗	✗	✗	✗	✗
ARS	ASTRAL	**26**	**20**	**95**	**1**	**0.8**	**92**	**1**	**0.8**	**92**
MEPS	FairFL*	✗	✗	✗	✗	✗	✗	✗	✗	✗
MEPS	FairFed*	✗	✗	✗	✗	✗	✗	✗	✗	✗
MEPS	FCFL	**12,636**	**80,357**	**86**	**79**	**502**	**81**	**50**	**318**	**78**
MEPS	ASTRAL	**1**	**4**	**86**	**1**	**4**	**86**	**1**	**4**	**86**
Adult	FairFL*	31	245	**79**	5	39	**78**	2	16	76
Adult	FairFed*	✗	✗	✗	✗	✗	✗	✗	✗	✗
Adult	FCFL	**7,691**	**91,631**	**82**	**111**	**3,534**	78	**74**	**882**	**74**
Adult	ASTRAL	**13**	**103**	80	**1**	**8**	**79**	**1**	**8**	**79**
DC	FairFL*	✗	✗	✗	✗	✗	✗	✗	✗	✗
DC	FairFed*	**27**	**211**	**71**	✗	✗	✗	**25**	**195**	**70**
DC	FCFL	✗	✗	✗	✗	✗	✗	✗	✗	✗
DC	ASTRAL	**116**	**906**	**78**	**2**	**16**	**76**	**1**	**8**	**73**
KDD	FairFL*	✗	✗	✗	✗	✗	✗	✗	✗	✗
KDD	FairFed*	✗	✗	✗	✗	✗	✗	✗	✗	✗
KDD	FCFL	✗	✗	✗	✗	✗	✗	✗	✗	✗
KDD	ASTRAL	**6**	**93**	**75**	**1**	**15**	**74**	**1**	**15**	**74**
MobiAct	FairFL*	**2**	**590**	**76**	✗	✗	✗	✗	✗	✗
MobiAct	FairFed*	✗	✗	✗	✗	✗	✗	✗	✗	✗
MobiAct	FCFL	✗	✗	✗	✗	✗	✗	✗	✗	✗
MobiAct	ASTRAL	**5**	**1,475**	**94**	**1**	**295**	**89**	**1**	**295**	**89**
CelebA	FairFL*	**8**	**3 GB**	**54**	✗	✗	✗	✗	✗	✗
CelebA	FairFed*	10	3 GB	76	✗	✗	✗	✗	✗	✗
CelebA	FCFL	✗	✗	✗	✗	✗	✗	✗	✗	✗
CelebA	ASTRAL	**33**	**11 GB**	**89**	**10**	**3 GB**	**85**	7	2 GB	**80**

surements collected when a method achieves a required accuracy defined as 95% (respectively 90%) of the maximum accuracy achieved for the dataset. For example, the maximum accuracy achieved for the dataset DC is 78%, by ASTRAL. the column 95^{th} presents the measurements of all methods applied on DC when they achieve 95% of the maximum accuracy, *i.e.*, 74%. The presence of a cross mark indicates that a method is not able to mitigate bias while achieving the required accuracy.

Overall, FairFL* is able to meet the bias constraint only with Adult, FairFed* meets it for ARS and for DC, FCFL meets it with Adult and MEPS, and ARS, and ASTRAL achieves the bias objective with all the datasets. For KDD, ASTRAL reaches the bias objective while scoring its highest accuracy at FL round 6, and scores more than 95% of its maximum accuracy starting at FL round 1. The amount of exchanged data is between 15 KB and 93 KB. In contrast, FairFL* and FCFL are not able to reach the bias objective for that dataset. For the DC dataset, ASTRAL takes 116 rounds to reach its highest accuracy (78%), and only 2 rounds to reach the accuracy (76%). In general, it ensures higher accuracy compared to FairFed*. For Adult, ASTRAL reaches its maximum accuracy while fulfilling the bias objective at FL round 13, with 103 KB of exchanged data. This is far better than FairFL* and FCFL which score their maximum accuracy only after respectively 31 and 7,691 FL rounds, and after respectively 245 KB and 91,631 KB of exchanged data. If FCFL is able to achieve the bias objective at earlier rounds, it is at the expense of a lower accuracy (74%). The trend is even stronger with MEPS for which ASTRAL scores its maximum accuracy at FL round 1, while FCFL scores the same maximum accuracy after its $12,636^{\text{th}}$ FL round. ASTRAL needs 4 KB of exchanged data, while FCFL uses

80,357 KB. As explained in section 8.13, FCFL takes typically more FL rounds to stabilize because it requires from the clients to communicate with the server after each local iteration at the client's level, via a gradient-based algorithm. In other terms, one FL round in FCFL comprises necessary one local epoch of training. For the ARS dataset, we observe that FairFL* does not achieve the bias constraint due to the selection process which fails from the first round. FairFed* reaches the bias constraint starting from round 11, and achieves the best accuracy while respecting the bias constraint at round 48. FCFL reaches the bias constraint only after more than hundred rounds, and achieves the best accuracy while respecting the bias constraint only at round 519. ASTRAL is able to ensure the bias objective directly from the first round with an accuracy of 92%. Moreover, it achieves the best accuracy of 95% compared to the other methods while respecting the bias threshold at round 26. For both MobiAct and CelebA, ASTRAL maintains the bias objective while achieving the highest accuracies compared to FairFL* and FairFed*. Moreover, ASTRAL outperforms FairFL* and FairFed* in terms of accuracy starting from early rounds, 1^{st} round for MobiAct and 7^{th} for CelebA. In summary, these experiments show that ASTRAL is a practical FL bias mitigation solution, and is more efficient than state-of-the-art solutions in terms of maximizing accuracy, communication rounds, and amount of exchanged data.



(a) ARS with time labels in minutes (b) MEPS with time labels in minutes (c) Adult with time labels in minutes (d) DC with time labels in minutes

(e) KDD with time labels in minutes (f) MobiAct with time labels in minutes (g) CelebA with time labels in minutes

Figure 8.11: Trade-off between execution time, accuracy and bias

In the following we are interested in analyzing the trade-offs between accuracy, bias, and execution time for the considered bias mitigation methods. The trade-offs measurements are presented for all the datasets in Figure 8.11. Each sub-figure presents the accuracy and bias of the different bias mitigation methods for a given dataset, at the round where the methods attain 90% of the highest observed accuracy (see column 90^{th} of Table 8.3). Each sub-figure also presents, through labels, the total execution time to achieve the required objective of a

method including the clients' training time, the models' transmission time, and the necessary time for models' aggregation at the server level.

We can observe from Figure 8.11 that ASTRAL is the only method that consistently occupies the upper-left square of the graphs, limited by the required accuracy threshold at the bottom and the predefined bias threshold on the right. This shows that ASTRAL consistently attains the highest accuracies while ensuring bias mitigation.

In terms of execution time, ASTRAL's performance ranges from just a few seconds to around 12 minutes, except for the case of CelebA, where ASTRAL's execution time stands out. In this particular scenario, ASTRAL's overall execution time is relatively higher compared to that of the other datasets. This difference can be attributed to the overhead involved in training a complex model like ResNet on the large image dataset of CelebA. Moreover, specifically for the CelebA case, we noted that ASTRAL's algorithm requires additional time to identify the optimal aggregation weight. This phenomenon is attributed to the characteristics of the model and data employed in this particular use case. Nevertheless, as mentioned in our earlier discussions, ASTRAL's execution time can be optimized through the fine-tuning of its hyperparameters. Furthermore, making use of parallelization, such as employing multi-core machines at the server level, greatly benefits the execution time of the Differential Evolution algorithm [42].

8.15 Summary

We presented ASTRAL, a novel FL aggregation method designed to mitigate bias for FL systems. ASTRAL finds the best combination of FL clients' local model parameters in order to obtain a global FL model ensuring that bias remains below a given threshold, while keeping model accuracy as high as possible. ASTRAL allows for a per-use-case bias threshold configuration, a practical approach that can be easily deployed in diverse FL applications. ASTRAL handles the presence of multiple sensitive attributes in data and mitigates bias for all of them simultaneously. It also supports by design bias mitigation for any bias metric, adapting bias mitigation to diverse practical use cases. Through our extensive empirical study of ASTRAL with seven real-world datasets, we assessed its performance in terms of bias and accuracy objectives, scalability, and robustness against data heterogeneity. The results show that ASTRAL's aggregation approach that provides the best possible aggregation weights with regard to our objectives leads to effective bias mitigation and model accuracy. We intend to further extend this work as follows. We plan to extend multi-objective approach of ASTRAL to consider additional elements on top of bias and accuracy, including privacy, robustness, and efficiency [100]. As the use of FL gains more traction across several applications, it becomes apparent that mitigating bias and optimizing accuracy may not be sufficient. Privacy must also be taken into account to protect sensitive information. Similarly, robustness is essential for ensuring that FL models are not susceptible to attacks [140]. Incorporating these additional aspects into the multi-objective approach results in effective and reliable models that can be trusted to be

leveraged for FL systems.

Acknowledgement

I would like to express my sincere gratitude to Nawel Benarba, Ousmane Touat, Pasquale De Rosa, Sara Bouchenak, Vania Marangozova, Valerio Schiavoni, Angela Bonifati, and Pascal Felber for their contributions to the design and development of ASTRAL. Their expertise, advice, and support have played a significant role in the progress of this project.

9 Conclusion and Research Perspectives

In the following, we present the conclusion and the research perspectives of this thesis.

9.1 Conclusion

This thesis has addressed two major issues encountered in DML systems, yielding two significant contributions. Our first contribution involves a characterization study of DML workloads aiming at the optimization of DML models quality, execution time and cost. Our second contribution introduces ASTRAL, a bias mitigation mechanism designed for FL.

Specifically, in the initial part of this thesis, we delved into configuration challenges within DML systems with the objective of improving the understanding of DML execution and enhancing DML performance. We have conducted an extensive workload characterization using the Spark platform and popular ML methods and datasets on a distributed cluster, while varying configuration parameters. We have analyzed the impact of different configuration parameters and tuning strategies on the performance of DML workloads in terms of execution time, model quality, and cost. We have provided observations on DML workloads behavior and recommendations to both system administrators and data scientists on optimizing the accuracy, execution time, and cost of their DML workloads. The experiments traces are publicly available for the benefit of researchers and practitioners. This work has been accepted for publication at Middleware 2023.

In the second part of this thesis, we have focused on fairness issues within a specific DML framework: Federated Learning. We have introduced ASTRAL, a novel bias mitigation framework that aims at achieving fairness in FL models outcomes while maintaining models' accuracy as high as possible. ASTRAL employs black-box optimization techniques to accomplish its objectives. It allows for a per-use-case bias threshold configuration and handles the presence of multiple sensitive attributes by mitigating bias for all of them simultaneously. ASTRAL also supports by design bias mitigation for multiple bias metric, adapting bias mitigation to diverse practical use cases. We have provided a publicly available software prototype of ASTRAL for the

benefit of researchers and practitioners. This work has been accepted in the 2023' proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT).

9.2 Research Perspectives

We outline in the following research directions stemming from our work.

9.2.1 Characterizing DML on Different Platforms and Designing Tools for DML Multi-Objective Performance Optimization

Our current study mainly investigates DML workloads using the Spark platform. An important improvement would be to expand our research to other platforms with different processing and programming models, *e.g.*, TensorFlow [1]. This broader view will help us compare DML platforms, and assist DML practitioners in choosing the right platform for their workload.

A more long-term research direction for our first contribution involves the development of novel and efficient tools capable of fine-tuning DML. In order to ensure the optimal system configurations, such tools should consider the different existing parameters that impact the different aspects of performance such as model quality, execution time, and resources consumption, while taking into account the trade-offs and interactions between parameters. Novel DML tuning tools should also enable users to simultaneously achieve multiple objectives related to the metrics of interest. To address these challenges, first steps involve building mathematical models or simulations that capture the relationships between configuration parameters and multiple performance metrics, and investigating optimization techniques that efficiently explore interactions between configuration parameters.

9.2.2 Improving the Efficiency and Generalization of ASTRAL

Our current implementation of ASTRAL relies on the black box-optimization algorithm of differential evolution. This algorithm requires a relatively large number of evaluations of the objective function, which can increase the computational cost, especially for complex or computationally expensive objective functions. The efficiency of ASTRAL could benefit from a comparison among existing black-box optimization algorithms in terms of their solutions quality and their computational cost. In addition, while we have evaluated ASTRAL's performance using group fairness metrics, we should also assess its performance with other fairness notions metrics, namely performance fairness and collaborative fairness. Moreover, even though ASTRAL does not need any data from FL clients, it doesn't protect the clients against malicious attacks that could expose their sensitive information from the model parameters. Thus, we plan to extend the multi-objective approach of ASTRAL to consider additional objectives along with bias and accuracy, including ensuring privacy and robustness against malicious attacks.

9.2.3 Identifying and Addressing Sources of Bias in Learning Systems

Looking ahead, broader and long-term research prospects include identifying more sources of bias in FL in particular and in ML in general, and creating strategies to reduce bias in the different steps of the learning process. ASTRAL currently helps to mitigate bias in FL resulting from biased clients' data and uneven clients' participation, but there are other parts of the learning process that may induce bias to the overall system.

Researchers need to investigate how the datasets used to tune the system or to moderate the training process are gathered and annotated, and whether they introduce bias unintentionally. Indeed, if these datasets are not sampled to be representative of the target distribution then the model performance metrics collected on these datasets will yield an inaccurate and biased idea about the real model performance. Developing ways to create representative datasets is challenging especially under privacy constraints but remains necessary.

Furthermore, the selection of fairness metrics is a decision that requires careful consideration to avoid introducing bias into the system. Indeed, optimizing one metric may negatively impact another. Thus, it is essential to establish and analyze relations and trade-offs between various fairness metrics to clarify the implications of adopting one fairness notion rather than another. Research efforts should additionally focus on the feasibility of optimizing multiple fairness notions simultaneously. Such analysis will lead to the formulation of clear guidelines for choosing the most suitable fairness metrics, taking into account the specific characteristics of the learning use case, data, models, and the perspective of the stakeholders.

Similarly, the choice of the sensitive attributes is important. Bias is often linked to sensitive attributes like race, gender, or age. However, we should explore all potential sensitive attributes and handle these attributes carefully to ensure fairness in the FL process for all demographic groups.

Decisions about the fairness metrics, sensitive attributes, and other aspects like training moderation are generally not automatic and are made by humans. Thus the makeup of teams involved in the process can affect fairness. Examining the diversity and the workflow within these teams and understanding how they impact the fairness of decision-making and ML systems outcomes is another research path. Here, collaborations between ML/FL practitioners, researchers, and human and societal sciences researchers can provide insights to establish guidelines and best practices for ML/FL teams.

Finally, in all ML pipelines data can change over time, causing data shifts that can introduce new forms of bias. Developing methods to continuously watch and evaluate the learning system to detect and correct such shifts is essential for maintaining fairness during the learning process.

Bibliography

[1] Martín Abadi et al. "TensorFlow: A System for Large-scale Machine Learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. Osdi'16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: http://dl.acm.org/citation.cfm?id=3026877.3026899.

[2] Annie Abay et al. *Mitigating Bias in Federated Learning*. 2020. arXiv: 2012.02447.

[3] Simeon Okechukwu Ajakwe et al. "Real-Time Monitoring of COVID-19 Vaccination Compliance: A Ubiquitous IT Convergence Approach". In: *2021 International Conference on Information and Communication Technology Convergence (ICTC)*. Jeju Island, Korea: IEEE, 2021, pp. 440–445.

[4] Samson B. Akintoye et al. "A Hybrid Parallelization Approach for Distributed and Scalable Deep Learning". In: *IEEE Access* 10 (2022), pp. 77950–77961. DOI: 10.1109/ACCESS.2022.3193690.

[5] Laila Alterkawi and Matteo Migliavacca. "Parallelism and Partitioning in Large-Scale GAs Using Spark". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 736–744. ISBN: 9781450361118. DOI: 10.1145/3321707.3321775. URL: https://doi.org/10.1145/3321707.3321775.

[6] Dario Amodei et al. "Deep speech 2: End-to-end speech recognition in english and mandarin". In: *International conference on machine learning*. PMLR. 2016, pp. 173–182.

[7] Apache Spark. *Spark Configuration*. https://spark.apache.org/docs/2.4.3/configuration.html. 2021. (Visited on 02/01/2021).

[8] P Apoorva et al. "Automated Criminal Identification by Face Recognition Using Open Computer Vision Classifiers". In: *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*. Erode, Tamil Nadu, India: IEEE, 2019, pp. 775–778. DOI: 10.1109/ICCMC.2019.8819850.

[9] AWS. *Amazon EC2 On-Demand Pricing*. Last accessed: Oct 24, 2023.

[10] TK Balaji, Chandra Sekhara Rao Annavarapu, and Annushree Bablani. "Machine Learning Algorithms for Social Media Analysis: A Survey". In: *Computer Science Review* 40 (2021), p. 100395.

[11] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. “Searching for Exotic Particles in High-energy Physics with Deep Learning”. In: *Nature Communications* 5.C (July 2014).

[12] Muhammet Sinan Basarslan, Fatih Kayaalp, et al. “Sentiment Analysis with Machine Learning Methods on Social Media”. In: (2020).

[13] Gérard Biau and Erwan Scornet. “A Random Forest Guided Tour”. In: *Test* 25 (2016), pp. 197–227.

[14] *Blueprint for an AI Bill of Rights*. OSTP. 2022. URL: https://www.whitehouse.gov/wp-content/uploads/2022/10/Blueprint-for-an-AI-Bill-of-Rights.pdf.

[15] Christoph Boden et al. “Benchmarking Data Flow Systems for Scalable Machine Learning”. In: *Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*. BeyondMR’17. Chicago, IL, USA: Association for Computing Machinery, 2017. ISBN: 9781450350198. DOI: 10.1145/3070607.3070612. URL: https://doi.org/10.1145/3070607.3070612.

[16] Stephan Böhm and Susanne J. Niklas. “Mobile Recruiting: Insights from a Survey among German HR Managers”. In: *Proceedings of the 50th Annual Conference on Computers and People Research*. SIGMIS-CPR ’12. Milwaukee, Wisconsin, USA: Association for Computing Machinery, 2012, 117–122. ISBN: 9781450311106. DOI: 10.1145/2214091.2214124. URL: https://doi.org/10.1145/2214091.2214124.

[17] Léon Bottou et al. “Stochastic Gradient Learning in Neural Networks”. In: *Proceedings of Neuro-Nımes* 91.8 (1991), p. 12.

[18] Leo Breiman. “Bagging Predictors”. In: *Machine learning* 24 (1996), pp. 123–140.

[19] Butterluo. *Does MXNET Support Stale Synchronous Parallel (aka. SSP)*. https://github.com/apache/incubator-mxnet/issues/841. Last access Nov 2023. 2015.

[20] Tianfeng Chai and Roland R Draxler. “Root Mean Square Error (RMSE) or Mean Absolute Error (MAE)”. In: *Geoscientific model development discussions* 7.1 (2014), pp. 1525–1534.

[21] Badrish Chandramouli et al. “Data Stream Management Systems for Computational Finance”. In: *Computer* 43.12 (2010), pp. 45–52.

[22] Beidi Chen et al. “SLIDE : In Defense of Smart Algorithms over Hardware Acceleration for Large-Scale Deep Learning Systems”. In: *Proc. of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. Ed. by Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze. mlsys.org, 2020.

[23] Hong-You Chen and Wei-Lun Chao. “FedBE: Making Bayesian Model Ensemble Applicable to Federated Learning”. In: *9th International Conference on Learning Representations, ICLR 2021, May 3-7, 2021*. Virtual Event, Austria: OpenReview.net, 2021.

[24] Jiachen Chen and W Kenneth Jenkins. “Facial Recognition With PCA and Machine Learning Methods”. In: *2017 IEEE 60th international Midwest symposium on circuits and systems (MWSCAS)*. IEEE. 2017, pp. 973–976.

[25] Jian Chen and Russell M. Clapp. "Astro: Auto-Generation of Synthetic Traces Using Scaling Pattern Recognition for MPI Workloads". In: *IEEE Transactions on Parallel and Distributed Systems* 28.8 (2017), pp. 2159–2171. DOI: 10.1109/TPDS.2017.2649518.

[26] Tianqi Chen et al. "Mxnet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems". In: *arXiv preprint arXiv:1512.01274* (2015).

[27] Yiqiang Chen et al. "FedHealth: A Federated Transfer Learning Framework for Wearable Healthcare". In: *IEEE Intelligent Systems* 35.4 (2020), pp. 83–93. DOI: 10.1109/MIS.2020.2988604.

[28] Zhengyu Chen, Teng Xiao, and Kun Kuang. "BA-GNN: On Learning Bias-Aware Graph Neural Network". In: *38th IEEE International Conference on Data Engineering, (ICDE 2022)*. Kuala Lumpur, Malaysia: IEEE, 2022, pp. 3012–3024.

[29] Steven W. D. Chien et al. "Characterizing Deep-Learning I/O Workloads in TensorFlow". In: *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems (PDSW-DISCS)* (2018). DOI: 10.1109/pdsw-discs.2018.00011. URL: http://dx.doi.org/10.1109/PDSW-DISCS.2018.00011.

[30] Lingyang Chu et al. *FedFair: Training Fair Models In Cross-Silo Federated Learning*. 2021. arXiv: 2109.05662.

[31] James Cipar et al. "Solving the straggler problem with bounded staleness". In: *14th Workshop on Hot Topics in Operating Systems (HotOS XIV)*. 2013.

[32] Dan Claudiu Cireşan et al. "Deep Big Multilayer Perceptrons for Digit Recognition". In: *Neural Networks: Tricks of the Trade: Second Edition* (2012), pp. 581–598.

[33] Steven Cohen. "Design Strategies and Innovations in the Medical Expenditure Panel Survey". In: *Medical care* 41 (Aug. 2003), pp. III5–III12.

[34] Corinna Cortes et al. "AdaNet: Adaptive Structural Learning of Artificial Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. 2017, pp. 874–883.

[35] Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27.

[36] Sen Cui et al. "Addressing Algorithmic Disparity and Performance Inconsistency in Federated Learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 26091–26102.

[37] Sen Cui et al. *Repository of FCFL Software for Bias Mitigation in Federated Learning*. Institute for Artificial Intelligence, Tsinghua University THUAI). 2021. URL: https://github.com/cuis15/FCFL.

[38] Jason Jinquan Dai et al. "Bigdl: A Distributed Deep Learning Framework for Big Data". In: *Proceedings of the ACM Symposium on Cloud Computing*. 2019, pp. 50–60.

[39] Wei Dai et al. “High-Performance Distributed ML at Scale Through Parameter Server Consistency Models”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2015.

[40] Samuel Danziger et al. “Predicting Positive p53 Cancer Rescue Regions Using Most Informative Positive (MIP) Active Learning”. In: *PLoS computational biology* 5 (Sept. 2009), e1000498. DOI: 10.1371/journal.pcbi.1000498.

[41] Kalyan Das, Jiming Jiang, and JNK Rao. “Mean Squared Error of Empirical Predictor”. In: (2004).

[42] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. “Differential Evolution: A Survey of the State-of-the-Art”. In: *IEEE Transactions on Evolutionary Computation* 15.1 (2011), pp. 4–31. DOI: 10.1109/TEVC.2010.2059031.

[43] Jeffrey Dean et al. “Large Scale Distributed Deep Networks”. In: *Advances in neural information processing systems* 25 (2012).

[44] Li Deng. “The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142. DOI: 10.1109/MSP.2012.2211477.

[45] Valentin Deyringer et al. “Parallelization of Neural Network Training for NLP with Hogwild!” In: *The Prague Bulletin of Mathematical Linguistics* 109.1 (2017), p. 29.

[46] Katerine Diaz-Chito, Aura Hernández-Sabaté, and Antonio M. López. “A Reduced Feature Set for Driver Head Pose Estimation”. In: *Appl. Soft Comput.* 45.C (Aug. 2016), pp. 98–107. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2016.04.027.

[47] *Differential Evolution - SciPy implementation*. SCIPY. 2023. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html.

[48] Ivo D Dinov. “Volume and value of big healthcare data”. In: *Journal of medical statistics and informatics* 4 (2016).

[49] *DML Workload Characterization Git Repository*. https://github.com/DMLCharacterization/DMLCharacterization/. May 2023.

[50] Shi Dong, Ping Wang, and Khushnood Abbas. “A Survey on Deep Learning and Its Applications”. In: *Computer Science Review* 40 (2021), p. 100379.

[51] Michele Donini et al. “Empirical Risk Minimization Under Fairness Constraints”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NeurIPS’18. Montréal, Canada: Neural Information Processing Systems, 2018, pp. 2796–2806.

[52] Nikoli Dryden et al. *Improving Strong-Scaling of CNN Training by Exploiting Finer-Grained Parallelism*. 2019. arXiv: 1903.06681 [cs.DC].

[53] Wei Du et al. “Fairness-Aware Agnostic Federated Learning”. In: *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. Virtual Event: SIAM, 2021, pp. 181–189.

[54] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[55] Rajdeep Dua, Manpreet Singh Ghotra, and Nick Pentreath. *Machine Learning with Spark*. Packt Publishing Ltd, 2017.

[56] Cynthia Dwork et al. "Fairness Through Awareness". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS '12. Cambridge, Massachusetts: Association for Computing Machinery, 2012, pp. 214–226. ISBN: 9781450311151.

[57] Issam El Naqa and Martin J Murphy. *What Is Machine Learning?* Springer, 2015.

[58] Peter Elger and Eóin Shanaghy. *AI As a Service: Serverless Machine Learning with AWS*. Manning Publications, 2020.

[59] Radwa Elshawi et al. "DLBench: a comprehensive experimental evaluation of deep learning frameworks". In: *Cluster Computing* (Feb. 2021). ISSN: 1573-7543. DOI: 10.1007/s10586-021-03240-4. URL: https://doi.org/10.1007/s10586-021-03240-4.

[60] *Equal Credit Opportunity Act*. Departments and Agencies of the Federal Government. 2017. URL: https://www.ecfr.gov/current/title-12/chapter-II/subchapter-A/part-202/section-202.2.

[61] Yahya H. Ezzeldin et al. "FairFed: Enabling Group Fairness in Federated Learning". In: *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, February 7-14, 2023*. Ed. by Brian Williams, Yiling Chen, and Jennifer Neville. Washington, DC, USA: AAAI Press, 2023, pp. 7494–7502.

[62] Wei Fan et al. "Fair Graph Auto-Encoder for Unbiased Graph Representations with Wasserstein Distance". In: *IEEE International Conference on Data Mining, ICDM 2021, December 7-10, 2021*. Ed. by James Bailey et al. Auckland, New Zealand: IEEE, 2021, pp. 1054–1059.

[63] Michael Feldman et al. "Certifying and Removing Disparate Impact". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '15. Sydney, NSW, Australia: Association for Computing Machinery, 2015, pp. 259–268. ISBN: 9781450336642.

[64] Matthias Feurer et al. "Auto-Sklearn 2.0: The Next Generation". In: *CoRR* abs/2007.04074 (2020). arXiv: 2007.04074. URL: https://arxiv.org/abs/2007.04074.

[65] Jeremy Fowers et al. "A Performance and Energy Comparison of FPGAs, GPUs, and Multicores for Sliding-Window Applications". In: *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. 2012, pp. 47–56.

[66] Alexander L. Fradkov. "Early History of Machine Learning". In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 1385–1390. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.1888. URL: https://www.sciencedirect.com/science/article/pii/S2405896320325027.

[67] Pasi Fränti and Sami Sieranoja. "How Much Can K-Means Be Improved by Using Better Initialization and Repeats?" In: *Pattern Recognition* 93 (2019), pp. 95–112.

[68] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[69] Paulo Gabriel and Rodrigo Mello. "Modelling distributed computing workloads to support the study of scheduling decisions". In: *International Journal of Computational Science and Engineering* 11 (Jan. 2015), pp. 155–166. DOI: 10.1504/IJCSE.2015.071879.

[70] Hugo E. S. Galindo et al. "WGCap: A Synthetic Trace Generation Tool for Capacity Planning of Virtual Server Environments". In: *2010 IEEE International Conference on Systems, Man and Cybernetics*. 2010, pp. 2094–2101. DOI: 10.1109/ICSMC.2010.5641705.

[71] Matt W Gardner and SR Dorling. "Artificial Neural Networks (The Multilayer Perceptron) - A Review of Applications in The Atmospheric Sciences". In: *Atmospheric environment* 32.14-15 (1998), pp. 2627–2636.

[72] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Hardness*. 1979.

[73] Jean Gaudart, Bernard Giusiano, and Laetitia Huiart. "Comparison of the performance of multi-layer perceptron and linear regression for epidemiological data". In: *Computational Statistics and Data Analysis* 44 (2004), pp. 547 –570. URL: https://hal.archives-ouvertes.fr/hal-01755553.

[74] Jonas Gehring et al. *Convolutional Sequence to Sequence Learning*. 2017. arXiv: 1705.03122 [cs.CL].

[75] Manolis Georgioudakis and Vagelis Plevris. "A Comparative Study of Differential Evolution Variants in Constrained Structural Optimization". In: *Frontiers in Built Environment* 6 (2020), p. 102.

[76] Alexandros V Gerbessiotis and Leslie G Valiant. "Direct Bulk-Synchronous Parallel Algorithms". In: *Journal of Parallel and Distributed Computing* 22.2 (1994), pp. 251–267.

[77] Marjan Ghazvininejad et al. *A Knowledge-Grounded Neural Conversation Model*. 2018. arXiv: 1702.01932 [cs.CL].

[78] Andrew Gibiansky. "Bringing HPC Techniques to Deep Learning". In: *Baidu Research, Tech. Rep.* (2017).

[79] Eric Goldman. "An Introduction to the California Consumer Privacy Act (CCPA)". In: *Santa Clara Univ. Legal Studies Research Paper* (2020).

[80] Jiuxiang Gu et al. "Recent Advances in Convolutional Neural Networks". In: *Pattern recognition* 77 (2018), pp. 354–377.

[81] Saad Hikmat Haji and Adnan Mohsin Abdulazeez. "Comparison of Optimization Techniques Based on Gradient Descent Algorithm: A Review". In: *PalArch's Journal of Archaeology of Egypt/Egyptology* 18.4 (2021), pp. 2715–2743.

[82] Moritz Hardt, Eric Price, and Nati Srebro. “Equality of Opportunity in Supervised Learning”. In: *Advances in Neural Information Processing Systems* 29 (2016), pp. 3315–3323.

[83] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, June 27-30, 2016*. Las Vegas, NV, USA: IEEE Computer Society, 2016, pp. 770–778.

[84] Xing He, Changgen Peng, Weijie Tan, et al. “Fast and Accurate Deep Leakage from Gradients Based on Wasserstein Distance”. In: *International Journal of Intelligent Systems* 2023 (2023).

[85] Herodotos Herodotou, Yuxing Chen, and Jiaheng Lu. “A Survey on Automatic Parameter Tuning for Big Data Processing Systems”. In: *ACM Comput. Surv.* 53.2 (Apr. 2020). ISSN: 0360-0300. DOI: 10.1145/3381027. URL: https://doi.org/10.1145/3381027.

[86] Chris Jay Hoofnagle, Bart Van Der Sloot, and Frederik Zuiderveen Borgesius. “The European Union General Data Protection Regulation: What It Is and What It Means”. In: *Information & Communications Technology Law* 28.1 (2019), pp. 65–98.

[87] Kevin Hsieh et al. “The Non-IID Data Quagmire of Decentralized Machine Learning”. In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*. Vol. 119. Proceedings of Machine Learning Research. Virtual Event: PMLR, 2020, pp. 4387–4398.

[88] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. *Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification*. 2019. arXiv: 1909.06335.

[89] Xia Hu et al. “Model Complexity of Deep Learning: A Survey”. In: *Knowledge and Information Systems* 63 (2021), pp. 2585–2619.

[90] Xu Huang, Hong Zhang, and Xiaomeng Zhai. “A Novel Reinforcement Learning Approach for Spark Configuration Parameter Optimization”. In: *Sensors* 22.15 (2022), p. 5930.

[91] Yanping Huang et al. “Gpipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism”. In: *Advances in neural information processing systems* 32 (2019).

[92] Legal Information Institute. *29 CFR 1607.4 - Information on Impact*. 1978. URL: https://www.law.cornell.edu/cfr/text/29/1607.4.

[93] Sylvain Jeaugey. “NCCL 2.0”. In: *GPU Technology Conference (GTC)*. Vol. 2. 2017.

[94] Myeongjae Jeon et al. “Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads”. In: *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*. USENIX ATC ’19. Renton, WA, USA: USENIX Association, 2019, 947–960. ISBN: 9781939133038.

[95] Ji Chu Jiang et al. “Federated Learning in Smart City Sensing: Challenges and Opportunities”. In: *Sensors* 20.21 (2020), p. 6230.

[96] Jiawei Jiang et al. “Heterogeneity-Aware Distributed Parameter Servers”. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. SIGMOD ’17. Chicago, Illinois, USA: Association for Computing Machinery, 2017, 463–478. ISBN: 9781450341974. DOI: 10.1145/3035918.3035933. URL: https://doi.org/10.1145/3035918.3035933.

[97] Haifeng Jin, Qingquan Song, and Xia Hu. “Auto-Keras: An Efficient Neural Architecture Search System”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 1946–1956.

[98] Yaochu Jin. “A Comprehensive Survey of Fitness Approximation in Evolutionary Computation”. In: *Soft computing* 9.1 (2005), pp. 3–12.

[99] Norman P. Jouppi et al. “In-Datacenter Performance Analysis of a Tensor Processing Unit”. In: *SIGARCH Comput. Archit. News* 45.2 (2017), 1–12. ISSN: 0163-5964. DOI: 10.1145/3140659.3080246. URL: https://doi.org/10.1145/3140659.3080246.

[100] Peter Kairouz et al. “Advances and Open Problems in Federated Learning”. In: *Found. Trends Mach. Learn.* 14.1-2 (2021), pp. 1–210.

[101] Toshihiro Kamishima et al. “Fairness-Aware Classifier With Prejudice Remover Regularizer”. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Peter A. Flach, Tijl De Bie, and Nello Cristianini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 35–50.

[102] Kaan Kara et al. “FPGA-Accelerated Dense Linear Machine Learning: A Precision-Convergence Trade-Off”. In: *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2017, pp. 160–167.

[103] Guolin Ke et al. “Lightgbm: A Highly Efficient Gradient Boosting Decision Tree”. In: *Advances in neural information processing systems* 30 (2017).

[104] Yunyong Ko and Sang-Wook Kim. “SHAT: A Novel Asynchronous Training Algorithm That Provides Fast Model Convergence in Distributed Deep Learning”. In: *Applied Sciences* 12.1 (2021), p. 292.

[105] Ron Kohavi. “Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid”. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD’96. Portland, Oregon: AAAI Press, 1996, pp. 202–207.

[106] Furkan Koltuk and Ece Güran Schmidt. “A Novel Method for the Synthetic Generation of Non-I.I.D Workloads for Cloud Data Centers”. In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. 2020, pp. 1–6. DOI: 10.1109/ISCC50000.2020.9219577.

[107] Alex Krizhevsky. “One Weird Trick for Parallelizing Convolutional Neural Networks”. In: *arXiv preprint arXiv:1404.5997* (2014).

[108] Anders Krogh. “What Are Artificial Neural Networks?” In: *Nature biotechnology* 26.2 (2008), pp. 195–197.

[109] Paul Van der Laan. “The 2001 Census in The Netherlands”. In: *Conference the Census of Population*. Washington, USA: U.S. Department of Commerce, 2000.

[110] Griffin Lacey, Graham W. Taylor, and Shawki Areibi. *Deep Learning on FPGAs: Past, Present, and Future*. 2016. arXiv: 1602.04283 [cs.DC].

[111] Michael P LaValley. "Logistic Regression". In: *Circulation* 117.18 (2008), pp. 2395–2399.

[112] Yves Lemoigne and Alessandra Caner. *Molecular Imaging: Computer Reconstruction and Practice*. Springer Science & Business Media, 2008.

[113] Mouad Lemoudden and Bouabid El Ouahidi. "Managing Cloud-Generated Logs Using Big Data Technologies". In: *2015 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. 2015, pp. 1–7. DOI: 10.1109/WINCOM.2015.7381334.

[114] Min Li et al. "SparkBench: A Comprehensive Benchmarking Suite for in Memory Data Analytic Platform Spark". In: *Proceedings of the 12th ACM International Conference on Computing Frontiers*. CF '15. Ischia, Italy: Association for Computing Machinery, 2015. ISBN: 9781450333580. DOI: 10.1145/2742854.2747283. URL: https://doi.org/10.1145/2742854.2747283.

[115] Mu Li et al. "Parameter Server for Distributed Machine Learning". In: *Big learning NIPS workshop*. Vol. 6. 2. 2013.

[116] Tian Li et al. *Fair Resource Allocation in Federated Learning*. 2019. arXiv: 1905.10497.

[117] Weizhe Li, Mike Mikailov, and Weijie Chen. "Scaling the Inference of Digital Pathology Deep Learning Models using CPU-based High-Performance Computing". In: *IEEE Transactions on Artificial Intelligence* (2023), pp. 1–15. DOI: 10.1109/TAI.2023.3246032.

[118] Youpeng Li, Xuyu Wang, and Lingling An. "Hierarchical Clustering-Based Personalized Federated Learning for Robust and Fair Human Activity Recognition". In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7.1 (2023).

[119] Zexi Li et al. *Revisiting Weighted Aggregation in Federated Learning with Neural Networks*. 2023. arXiv: 2302.10911.

[120] Petro Liashchynskyi and Pavlo Liashchynskyi. "Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS". In: *arXiv preprint arXiv:1912.06059* (2019).

[121] Lin Lin et al. "Automatic Translation of Spoken English Based on Improved Machine Learning Algorithm". In: *Journal of Intelligent & Fuzzy Systems* 40.2 (2021), pp. 2385–2395.

[122] Tao Lin et al. "Ensemble Distillation for Robust Model Fusion in Federated Learning". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020*. Ed. by Hugo Larochelle et al. Vol. 33. NeurIPS'20. Virtual: Neural Information Processing Systems, 2020, pp. 2351–2363.

[123] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, December 7-13, 2015*. Santiago, Chile: IEEE Computer Society, 2015, pp. 3730–3738.

[124] Lingjuan Lyu et al. “Collaborative Fairness in Federated Learning”. In: *Federated Learning - Privacy and Incentive*. Ed. by Qiang Yang, Lixin Fan, and Han Yu. Vol. 12500. Lecture Notes in Computer Science. Springer, 2020, pp. 189–204.

[125] Xiu Ma et al. “Accelerating Deep Neural Network Filter Pruning with Mask-Aware Convolutional Computations on Modern CPUs”. In: *Neurocomputing* 505 (2022), pp. 375–387. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2022.07.006. URL: https://www.sciencedirect.com/science/article/pii/S0925231222008669.

[126] T. Soni Madhulatha. *An Overview on Clustering Methods*. 2012. arXiv: 1205.1117 [cs.DS].

[127] D. Magalhães et al. “Workload Modeling for Resource Usage Analysis and Simulation in Cloud Computing”. In: *Comput. Electr. Eng.* 47 (2015), pp. 69–81.

[128] Trisha Mahoney, Kush Varshney, and Michael Hind. *AI Fairness.*: O’Reilly Media, Incorporated, 2020.

[129] S. G. Makridakis and M. Hibon. *Evaluating Accuracy (or Error) Measures*. Fontainebleau: INSEAD. 1995.

[130] Alessia Mammone, Marco Turchi, and Nello Cristianini. “Support Vector Machines”. In: *Wiley Interdisciplinary Reviews: Computational Statistics* 1.3 (2009), pp. 283–289.

[131] O. Marcu et al. “Spark Versus Flink: Understanding Performance in Big Data Analytics Frameworks”. In: *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. 2016, pp. 433–442.

[132] Manuel Martinez and Rainer Stiefelhagen. “Taming The Cross Entropy Loss”. In: *Pattern Recognition: 40th German Conference, GCPR 2018, Stuttgart, Germany, October 9-12, 2018, Proceedings 40*. Springer. 2019, pp. 628–637.

[133] Ruben Mayer, Christian Mayer, and Larissa Laich. “The Tensorflow Partitioning and Scheduling Problem: It’s The Critical Path!” In: *Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning*. 2017, pp. 1–6.

[134] John McCall. “Genetic Algorithms for Modelling and Optimisation”. In: *Journal of computational and Applied Mathematics* 184.1 (2005), pp. 205–222.

[135] Brendan McMahan et al. “Communication-Efficient Learning of Deep Networks from Decentralized Data”. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. Fort Lauderdale Florida, USA: PMLR, 2017, pp. 1273–1282.

[136] Frank McSherry, Michael Isard, and Derek G. Murray. “Scalability! But at What COST?” In: *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*. Kartause Ittingen, Switzerland: USENIX Association, May 2015. URL: https://www.usenix.org/conference/hotos15/workshop-program/presentation/mcsherry.

[137] Larry R Medsker and LC Jain. “Recurrent Neural Networks”. In: *Design and Applications* 5.64-67 (2001), p. 2.

[138] Ninareh Mehrabi et al. “A Survey on Bias and Fairness in Machine Learning”. In: *ACM Computing Surveys* 54.6 (2022), 115:1–115:35.

[139] Xiangrui Meng et al. “Mllib: Machine learning in apache spark”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1235–1241.

[140] Aghiles Ait Messaoud et al. “Shielding Federated Learning Systems against Inference Attacks with ARM TrustZone”. In: *Proceedings of the 23rd ACM/IFIP International Middleware Conference*. Quebec, QC, Canada: ACM, 2022, pp. 335–348.

[141] Jeremy Miles. “R-squared, Adjusted R-squared”. In: *Encyclopedia of statistics in behavioral science* (2005).

[142] Azalia Mirhoseini et al. “Device Placement Optimization with Reinforcement Learning”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2430–2439.

[143] Azalia Mirhoseini et al. “Hierarchical Planning for Device Placement”. In: (2018).

[144] Sparsh Mittal, Poonam Rajput, and Sreenivas Subramoney. “A Survey of Deep Learning on CPUs: Opportunities and Co-Optimizations”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.10 (2022), pp. 5095–5115. DOI: 10.1109/TNNLS.2021.3071762.

[145] Sparsh Mittal and Jeffrey S. Vetter. “A Survey of Methods for Analyzing and Improving GPU Energy Efficiency”. In: *ACM Comput. Surv.* 47.2 (2014). ISSN: 0360-0300.

[146] *MLBench: Distributed Machine Learning Benchmark*. https://mlbench.github.io/. Accessed: 2023-12-06.

[147] *MLPerf*. https://mlperf.org/. Accessed: 2023-12-06.

[148] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. “Agnostic Federated Learning”. In: *International Conference on Machine Learning*. California, USA: PMLR, 2019, pp. 4615–4625.

[149] Jose G Moreno-Torres et al. “A Unifying View on Dataset Shift in Classification”. In: *Pattern recognition* 45.1 (2012), pp. 521–530.

[150] Ali Mostafaeipour et al. “Investigating The Performance of Hadoop and Spark Platforms on Machine Learning Algorithms”. In: *The Journal of Supercomputing* 77 (Feb. 2021). DOI: 10.1007/s11227-020-03328-5.

[151] David C Mowery. *International Computer Software Industry*. Oxford University Press, Inc., 1995.

[152] Fionn Murtagh. “Multilayer Perceptrons for Classification and Regression”. In: *Neurocomputing* 2.5-6 (1991), pp. 183–197.

[153] Rena Nainggolan et al. “Improved the Performance of the K-Means Cluster Using the Sum of Squared Error (SSE) Optimized by Using the Elbow Method”. In: *Journal of Physics: Conference Series*. Vol. 1361. 1. IOP Publishing. 2019, p. 012015.

[154] Deepak Narayanan et al. “PipeDream: Generalized Pipeline Parallelism for DNN Training”. In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 2019, pp. 1–15.

[155] Arundhati Navada et al. “Overview of Use of Decision Tree Algorithms in Machine Learning”. In: *2011 IEEE control and system graduate research colloquium*. IEEE. 2011, pp. 37–42.

[156] *News 20 Dataset*. http://qwone.com/~jason/20Newsgroups. Accessed: 2021-02-04.

[157] Dinh C. Nguyen et al. *Federated Learning for Smart Healthcare: A Survey*. 2021. arXiv: 2111.08834.

[158] Nhan Nguyen, Mohammad Maifi Hasan Khan, and Kewen Wang. “Towards Automatic Tuning of Apache Spark Configuration”. In: *11th IEEE International Conference on Cloud Computing, CLOUD 2018, San Francisco, CA, USA, July 2-7, 2018*, pp. 417–425.

[159] Nhan Nguyen, Mohammad Maifi Hasan Khan, and Kewen Wang. “Towards Automatic Tuning of Apache Spark Configuration”. In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 2018, pp. 417–425. DOI: 10.1109/CLOUD.2018.00059.

[160] Cyprien Noel and Simon Osindero. “Dogwild!-Distributed Hogwild for CPU & GPU”. In: *NIPS Workshop on Distributed Machine Learning and Matrix Computations*. 2014, pp. 693–701.

[161] Anant V. Nori et al. “REDUCT: Keep It Close, Keep It Cool! Efficient Scaling of DNN Inference on Multi-Core CPUs with near-Cache Compute”. In: *Proceedings of the 48th Annual International Symposium on Computer Architecture*. ISCA ’21. Virtual Event, Spain: IEEE Press, 2021, pp. 167–180. ISBN: 9781450390866. DOI: 10.1109/ISCA52012.2021.00022. URL: https://doi-org.ins2i.bib.cnrs.fr/10.1109/ISCA52012.2021.00022.

[162] Aaron van den Oord et al. *WaveNet: A Generative Model for Raw Audio*. 2016. arXiv: 1609.03499 [cs.SD].

[163] Yosuke Oyama et al. *The Case for Strong Scaling in Deep Learning: Training Large 3D CNNs with Hybrid Parallelism*. 2020. arXiv: 2007.12856 [cs.DC].

[164] Sharnil Pandya et al. “Federated Learning for Smart Cities: A Comprehensive Survey”. In: *Sustainable Energy Technologies and Assessments* 55 (2023), p. 102987.

[165] Gang-Min Park, Yong Seok Heo, and Hyuk-Yoon Kwon. “Trade-Off Analysis Between Parallelism and Accuracy of SLIC on Apache Spark”. In: *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*. 2021, pp. 5–12. DOI: 10.1109/BigComp51126.2021.00011.

[166] Jay H. Park et al. “HetPipe: Enabling Large DNN Training on (Whimpy) Heterogeneous GPU Clusters through Integration of Pipelined Model Parallelism and Data Parallelism”. In: *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, July 2020, pp. 307–321. ISBN: 978-1-939133-14-4. URL: https://www.usenix.org/conference/atc20/presentation/park.

[167] Adam Paszke et al. “Pytorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[168] Norman Paton et al. “Optimizing Utility in Cloud Computing through Autonomic Workload Execution.” In: *IEEE Data Eng. Bull.* 32 (Jan. 2009), pp. 51–58.

[169] PCM. *Processor Counter Monitor (PCM)*. https://software.intel.com/content/www/us/en/develop/articles/intel-performance-counter-monitor.html. 2021. (Visited on 01/27/2021).

[170] Leif E Peterson. “K-Nearest Neighbor”. In: *Scholarpedia* 4.2 (2009), p. 1883.

[171] Leonardo Piga et al. “Empirical Web server power modeling and characterization”. In: *2011 IEEE International Symposium on Workload Characterization (IISWC)*. 2011, pp. 75–75. DOI: 10.1109/IISWC.2011.6114200.

[172] Sasanka Potluri et al. “CNN Based High Performance Computing for Real Time Image Processing on GPU”. In: *Proceedings of the Joint INDS’11 & ISTET’11*. IEEE. 2011, pp. 1–7.

[173] David M. W. Powers. *Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation*. 2020. arXiv: 2010.16061 [cs.LG].

[174] Drazen Prelec. “The Probability Weighting Function”. In: *Econometrica* 66.3 (1998), pp. 497–527. ISSN: 00129682, 14680262. (Visited on 10/24/2022).

[175] Philipp Probst, Marvin N Wright, and Anne-Laure Boulesteix. “Hyperparameters and Tuning Strategies for Random Forest”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9.3 (2019), e1301.

[176] Aurick Qiao et al. “Fault Tolerance in Iterative-Convergent Machine Learning”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 5220–5230. URL: https://proceedings.mlr.press/v97/qiao19a.html.

[177] J.R. Quinlan. “Decision Trees and Decision-Making”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 20.2 (1990), pp. 339–346. DOI: 10.1109/21.52545.

[178] Benjamin Recht et al. “Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent”. In: *Advances in neural information processing systems* 24 (2011).

[179] Angie K Reyes, Juan C Caicedo, and Jorge E Camargo. “Fine-tuning Deep Convolutional Networks for Plant Recognition.” In: *CLEF (Working Notes)* 1391 (2015), pp. 467–475.

[180] C.J. Van Rijsbergen. “Information Retrieval.” In: *Journal of the American Society for Information Science* 30.6 (1979), pp. 374–375. DOI: https://doi.org/10.1002/asi.4630300621.

[181] Luis Rios and Nikolaos Sahinidis. “Derivative-Free Optimization: A Review of Algorithms and Comparison of Software Implementations”. In: *Journal of Global Optimization* 56.3 (2013), pp. 1247–1293.

[182] Isabelly Rocha et al. “PipeTune: Pipeline Parallelism of Hyper and System Parameters Tuning for Deep Learning Clusters”. In: *Proceedings of the 21st International Middleware Conference*. Middleware ’20. Delft, Netherlands: Association for Computing Machinery, 2020, 89–104. ISBN: 9781450381536. DOI: 10.1145/3423211.3425692. URL: https://doi.org/10.1145/3423211.3425692.

[183] Abhijit Guha Roy et al. *BrainTorrent: A Peer-to-Peer Environment for Decentralized Federated Learning*. 2019. arXiv: 1905.06731 [cs.LG].

[184] Sebastian Ruder. “An Overview of Gradient Descent Optimization Algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[185] Frank Seide and Amit Agarwal. “CNTK: Microsoft’s Open-Source Deep-Learning Toolkit”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 2135–2135.

[186] Alexander Sergeev and Mike Del Balso. *Horovod: Fast and Easy Distributed Deep Learning in TensorFlow*. 2018. arXiv: 1802.05799 [cs.LG].

[187] Amna Shahid and Malaika Mushtaq. “A Survey Comparing Specialized Hardware and Evolution in TPUs for Neural Networks”. In: *2020 IEEE 23rd International Multitopic Conference (INMIC)*. IEEE. 2020, pp. 1–6.

[188] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. “Activation Functions in Neural Networks”. In: *Towards Data Sci* 6.12 (2017), pp. 310–316.

[189] Roberto L. Shinmoto Torres et al. “Sensor Enabled Wearable RFID Technology for Mitigating the Risk of Falls Near Beds”. In: *2013 IEEE International Conference on RFID (RFID)*. Johor Bahru, Malaysia: IEEE, 2013, pp. 191–198.

[190] S. R. Shishira, A. Kandasamy, and K. Chandrasekaran. “Workload Characterization: Survey of Current Approaches and Research Challenges”. In: *Proceedings of the 7th International Conference on Computer and Communication Technology*. ICCCT-2017. Allahabad, India: Association for Computing Machinery, 2017, 151–156. ISBN: 9781450353243. DOI: 10.1145/3154979.3155003. URL: https://doi.org/10.1145/3154979.3155003.

[191] Meshal Shutaywi and Nezamoddin N Kachouie. “Silhouette Analysis for Performance Evaluation in Machine Learning with Applications to Clustering”. In: *Entropy* 23.6 (2021), p. 759.

[192] Konstantin Shvachko et al. “The Hadoop Distributed File System”. In: *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee. 2010, pp. 1–10.

[193] Michael W Sjoding et al. “Racial Bias in Pulse Oximetry Measurement”. In: *New England Journal of Medicine* 383.25 (2020), pp. 2477–2478.

[194] Linghao Song et al. *HyPar: Towards Hybrid Parallelism for Deep Learning Accelerator Array*. 2020. arXiv: 1901.02067 [cs.DC].

[195] *SparkMeasure, A Tool for Performance Troubleshooting of Apache Spark Workloads*. https://db-blog.web.cern.ch/blog/luca-canali/2018-08-sparkmeasure-tool-performance-troubleshooting-apache-spark-workloads. Accessed: 2021-02-04.

[196] Dave Steinkraus, Ian Buck, and PY Simard. "Using GPUs for Machine Learning Algorithms". In: *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. IEEE. 2005, pp. 1115–1120.

[197] Rainer Storn and Kevin P. Price. "Differential Evolution: A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces". In: *Journal of Global Optimization* 11 (1997), pp. 341–359.

[198] Dimitris Stripelis and José Luis Ambite. "Accelerating Federated Learning in Heterogeneous Data and Computational Environments". In: *arXiv preprint arXiv:2008.11281* (2020).

[199] Weigao Su et al. "Towards Device Independent Eavesdropping on Telephone Conversations with Built-in Accelerometer". In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5.4 (2022).

[200] Xiaogang Su, Xin Yan, and Chih-Ling Tsai. "Linear Regression". In: *Wiley Interdisciplinary Reviews: Computational Statistics* 4.3 (2012), pp. 275–294.

[201] Shan Suthaharan and Shan Suthaharan. "Decision Tree Learning". In: *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning* (2016), pp. 237–269.

[202] Pierre Tchounikine. *Computer Science and Educational Software Design: A Resource for Multidisciplinary Work in Technology Enhanced Learning*. Springer, 2011.

[203] Gernmanno Teles et al. "Machine Learning and Decision Support System on Credit Scoring". In: *Neural Computing and Applications* 32 (2020), pp. 9809–9826.

[204] The Apache Software Foundation. *Hadoop Commands Guide*. URL: https://hadoop.apache.org/docs/r1.2.1/cluster_setup.html#Configuration (visited on 2021).

[205] *The Artificial Intelligence Act*. EC. 2021. URL: https://artificialintelligenceact.eu/.

[206] *The National AI Initiative Act*. U.S. House. 2020. URL: https://www.ai.gov/.

[207] Alexander Tong, Guy Wolf, and Smita Krishnaswamy. "Fixing Bias in Reconstruction-based Anomaly Detection with Lipschitz Discriminators". In: *J. Signal Process. Syst.* 94.2 (2022), pp. 229–243.

[208] Leslie G Valiant. "Why BSP Computers? (Bulk-Synchronous Parallel Computers)". In: *[1993] Proceedings Seventh International Parallel Processing Symposium*. IEEE. 1993, pp. 2–5.

[209] SS Vallender. "Calculation of the Wasserstein Distance Between Probability Distributions on the Line". In: *Theory of Probability & Its Applications* 18.4 (1974), pp. 784–786.

[210] George Vavoulas et al. "The Mobiact Dataset: Recognition of Activities of Daily Living Using Smartphones". In: *International Conference on Information and Communication Technologies for Ageing Well and e-Health*. Vol. 2. Rome, Italy: SCITEPRESS, 2016, pp. 143–151.

[211] Praneeth Vepakomma et al. *No Peek: A Survey of Private Distributed Deep Learning*. 2018. arXiv: 1812.03288 [cs.LG].

[212] Joost Verbraeken et al. "A Survey on Distributed Machine Learning". In: *Acm computing surveys (csur)* 53.2 (2020), pp. 1–33.

[213] Alexander Vergara et al. "Chemical Gas Sensor Drift Compensation Using Classifier Ensembles". In: *Sensors and Actuators B: Chemical* 166-167 (May 2012), pp. 320–329. ISSN: 0925-4005. DOI: 10.1016/j.snb.2012.01.074.

[214] Mengdi Wang et al. "Characterizing Deep Learning Training Workloads on Alibaba-PAI". In: *IEEE International Symposium on Workload Characterization, IISWC 2019, Orlando, FL, USA, November 3-5, 2019*. IEEE, 2019, pp. 189–202. DOI: 10.1109/IISWC47752.2019.9042047. URL: https://doi.org/10.1109/IISWC47752.2019.9042047.

[215] Shaoqi Wang et al. "Aggressive Synchronization with Partial Processing for Iterative ML Jobs on Clusters". In: *Proceedings of the 19th International Middleware Conference*. 2018, pp. 253–265.

[216] Qizhen Weng et al. "MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters". In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, 2022, pp. 945–960. ISBN: 978-1-939133-27-4. URL: https://www.usenix.org/conference/nsdi22/presentation/weng.

[217] *What is Data Privacy - and Why Is It Important?* https://www.integrate.io/blog/what-is-data-privacy-why-is-it-important. Accessed: July 2023.

[218] Tom White. *Hadoop: The Definitive Guide*. 1st. O'Reilly Media, Inc., 2009. ISBN: 0596521979, 9780596521974.

[219] Timothy J Wroge et al. "Parkinson's Disease Diagnosis Using Machine Learning and Voice". In: *2018 IEEE signal processing in medicine and biology symposium (SPMB)*. IEEE. 2018, pp. 1–7.

[220] Yongkai Wu, Lu Zhang, and Xintao Wu. "On Convexity and Bounds of Fairness-Aware Classification". In: *The World Wide Web Conference*. WWW '19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 3356–3362. ISBN: 9781450366748.

[221] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).

[222] Bangzhou Xin et al. “Private FL-GAN: Differential Privacy Synthetic Data Generation Based on Federated Learning”. In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Virtual Barcelona: IEEE, 2020, pp. 2927–2931.

[223] Reynold S Xin et al. “Shark: SQL and Rich Analytics at Scale”. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*. 2013, pp. 13–24.

[224] Eric P Xing et al. “Petuum: A New Platform for Distributed Machine Learning on Big Data”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2015, pp. 1335–1344.

[225] Qiang Yang, Lixin Fan, and Han Yu. *Federated Learning: Privacy and Incentive*. Vol. 12500. Springer Nature, 2020.

[226] Zehua Yang et al. “Tear Up the Bubble Boom: Lessons Learned From a Deep Learning Research and Development Cluster”. In: *2022 IEEE 40th International Conference on Computer Design (ICCD)*. 2022, pp. 672–680. DOI: 10.1109/ICCD56317.2022.00103.

[227] Chunrong Yao et al. “EAIS: Energy-Aware Adaptive Scheduling for CNN Inference on High-Performance GPUs”. In: *Future Generation Computer Systems* 130 (2022), pp. 253–268.

[228] Amir Yazdanbakhsh et al. “An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks”. In: *arXiv e-prints* (2021), arXiv–2102.

[229] Madhu Yedla, Srinivasa Rao Pathakota, and TM Srinivasa. “Enhancing K-Means Clustering Algorithm With Improved Initial Center”. In: *International Journal of computer science and information technologies* 1.2 (2010), pp. 121–125.

[230] Bayya Yegnanarayana. *Artificial Neural Networks*. PHI Learning Pvt. Ltd., 2009.

[231] Sofia Yfantidou et al. *Beyond Accuracy: A Critical Review of Fairness in Machine Learning for Mobile and Wearable Computing*. 2023. arXiv: 2303.15585.

[232] Han Yu et al. “A Fairness-aware Incentive Scheme for Federated Learning”. In: *AIES ’20: AAAI/ACM Conference on AI, Ethics, and Society, February 7-8, 2020*. Ed. by Annette N. Markham et al. New York, NY, USA: ACM, 2020, pp. 393–399.

[233] Xubo Yue, Maher Nouiehed, and Raed Al Kontar. “Gifair-fl: A Framework for Group and Individual Fairness in Federated Learning”. In: *INFORMS Journal on Data Science* 2.1 (2023), pp. 10–23.

[234] Muhammad Bilal Zafar et al. “Fairness Constraints: A Flexible Approach for Fair Classification ”. In: *Journal of Machine Learning Research* 20.75 (2019), 75:1–75:42.

[235] Muhammad Bilal Zafar et al. “Fairness Constraints: Mechanisms for Fair Classification”. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. Fort Lauderdale Florida, USA: PMLR, 2017, pp. 962–970.

[236] Matei Zaharia et al. “Apache Spark: A Unified Engine for Big Data Processing”. In: *Communications of the ACM* 59.11 (2016), pp. 56–65.

[237] Matei Zaharia et al. “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for in-Memory Cluster Computing”. In: *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 2012, pp. 15–28.

[238] Matei Zaharia et al. “Spark: Cluster Computing With Working Sets”. In: *HotCloud* 10.10-10 (2010), p. 95.

[239] Yuchen Zeng, Hongxu Chen, and Kangwook Lee. *Improving Fairness via Federated Learning*. 2021. arXiv: 2110.15545.

[240] Daniel Yue Zhang, Kou Ziyi, and Wang Dong. “FairFL: A Fair Federated Learning Approach to Reducing Demographic Bias in Privacy-Sensitive Classification Models”. In: *Proceedings of the 2020 IEEE International Conference on Big Data, BigData2020*. United States: IEEE, 2020, pp. 1051–1060.

[241] Jiafeng Zhang. “Asynchronous Decentralized Consensus ADMM for Distributed Machine Learning”. In: *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)*. IEEE. 2019, pp. 22–28.

[242] Jilin Zhang et al. “A Parameter Communication Optimization Strategy for Distributed Machine Learning in Sensors”. In: *Sensors* 17.10 (2017), p. 2172.

[243] K. Zhang, S. Alqahtani, and M. Demirbas. “A Comparison of Distributed Machine Learning Platforms”. In: *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. 2017, pp. 1–9.

[244] Yue Zhao et al. “Federated Learning With Non-IID Data”. In: *arXiv preprint arXiv:1806.00582* (2018).

[245] Lichun Zhou. “Product Advertising Recommendation in E-Commerce Based on Deep Learning and Distributed Expression”. In: *Electronic Commerce Research* 20.2 (2020), pp. 321–342.

[246] Pengyuan Zhou et al. “Are You Left Out? An Efficient and Fair Federated Learning for Personalized Profiles on Wearable Devices of Inferior Networking Conditions”. In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6.2 (2022).

[247] Hongyu Zhu et al. “Benchmarking and Analyzing Deep Neural Network Training”. In: *IEEE International Symposium on Workload Characterization (IISWC’18)*. North Carolina, 2018.

[248] Xiaonan Zou et al. “Logistic Regression Model Optimization and Case Analysis”. In: *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*. IEEE. 2019, pp. 135–139.