Doctoral Dissertation
Doctoral Program in Modeling and Data Science (XXXVI cycle)

Pushing Federated Learning Boundaries

Three Innovative Distributed Intelligence Approaches

Gianluca Mittone

* * * * *

Supervisor
Prof. Marco Aldinucci

Doctoral examination committee
Prof. Pietro Liò - University of Cambridge
Prof. Lydia Y. Chen - University of Neuchâtel, Delft University of Technology

University of Turin
November 15th, 2024

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Gianluca Mittone
Turin, November 15th, 2024

Summary

The pervasivity of artificial intelligence is structurally changing how societies envision their development. The perception of what is valuable is rapidly shifting, and data is in the eye of the storm. This process affects every level of organisation in the modern world: from single individuals to multinational companies, from public administration to international organisations, everybody is developing the consciousness that data is a new, powerful social and economic device. Data is precious, and the conflict between who tries to harvest it and who instead tries to keep it private and protect its owners' rights rages. This friction constitutes the natural environment for developing new ideas and concepts to mitigate this contention. Federated learning arose in 2016 as one possible solution to the abovementioned struggle. The fundamental concept of this paradigm is to eliminate the need to exchange data between entities in favour of exchanging the knowledge extracted from it. This approach bypasses the privacy limit imposed by data owners, effectively enabling large-scale cooperative machine learning model training and eliminating the need for invasive data harvesting.

Starting from the real-world problem of creating a machine learning-based risk score for cardiological pathologies, the unsustainability of creating data lakes for such sensible tasks is described and discussed. Federated learning is identified as a solution to such an issue, and its main positive and negative aspects are investigated and analysed. Three assumptions of current-days federated learning software constitute the starting points for the main contributions of this dissertation: 1) the centralised structure currently implemented by many commercial frameworks, 2) their inner workings being strictly tied to deep learning models, and 3) their assumption of being deployed on private, specialised computing infrastructures. The proposed research expands the federated learning paradigm to handle scenarios in which these three conditions do not hold. Such research problems are addressed methodologically and practically during the dissertation, and three open-source, proof-of-concept software is made freely available as tangible research results: *FastFL*, *OpenFL-x*, and *xFFL*. All the software discussed is experimentally tested on many computational infrastructures (cluster, cloud, and high-performance computing facilities) and different micro-architectures (x86-64, ARM-v8, RISC-V), and the obtained learning and computational performance is collected, presented, and discussed.

- *FastFL* is a Python and C/C++ software based on the FastFlow parallel programming framework. It aims to prove that a different take on federated learning systems' structure design is possible. Different topologies and efficient communication protocols are exploited to push federations beyond the standard master-worker structure. These changes allow moving to more flexible and resilient structures, reducing the computation and communication overhead of commercial federated learning software.

- *OpenFL-x* targets the machine learning models compatible with current federated learning frameworks. It shows how implementing the AdaBoost.F algorithm inside the Intel® OpenFL framework allows federated learning to exploit non-deep machine learning models. This model-agnosticism property is crucial for scenarios where deep learning is not applicable, and thus, neither are current federated learning frameworks.

- *xFFL* aims at bringing federated learning to complex, shared, distributed computing infrastructures. A cross-facility federated learning workload is deployed on many high-performance computing through the StreamFlow workflow management system, allowing seamless interaction with job queue managers and containerisation technologies. The data moving and deployment logic is handled automatically, allowing this approach to be exploited efficiently despite its complexity.

Additionally, this dissertation makes a more theoretical contribution by highlighting the possibility of modelling FL processes from a higher-level perspective than the one used in current studies. Based on a RISC-pb^2l formal language extension, a domain-specific language for federated learning is introduced, allowing more abstract reasoning about different federated learning topologies' properties.

Acknowledgements

A PhD is not a mere career choice or a one-more title to add to one's CV. A PhD is a long, complex, and life-modeling journey that shapes one person's mind and world vision into something else: it turns a student into a researcher. Such a transformation requires time, dedication, and hard academic work, but not just that: it requires traveling, seeing different research realities, developing social skills, discussing with foreign people, and getting involved in new and unforeseen situations. Simply put, a PhD means growth, both academic and personal.

Such a growth process requires the dedication of precious time and resources by a PhD supervisor. In this context, I express my deepest gratitude to my supervisor, Prof. *Marco Aldinucci*. Despite his packed and busy academic life, he always did his best to answer my questions, clear my doubts, and guide me through the intricacies of the academic world. He never made me lack any resources to experience my PhD path fully, and he always gave me the right advice in every situation. Thank you sincerely for all of that.

However, a PhD is a team effort, not a one-person effort. The team that supported me during these intense years is numerous and varied, and I could not have found better people. Thank you, *Iacopo*, for all the evenings and nights spent in video call during the pandemic period of my PhD and for all the subsequent beer consumed in person after that; your technical and personal suggestions will not be forgotten. Thank you, *Robert*, for all the support given to me since you joined our research group: you were like a second supervisor to me. And thank you to all of you: *Barbara*, *Doriana*, *Alessia*, *Adriano*, *Bruno*, *Giulio*, *Samuele*, *Alberto*, *Lorenzo*, *Marco*, and *Emanuela*. Each day of my life spent in the laboratory would not have been the same without you.

Finally, I would like to express my gratitude to Prof. *Lydia Y. Chen* and Prof. *Pietro Liò* for accepting to review my PhD dissertation and being part of my doctoral examination committee; the latter also for hosting me at the Computer Laboratory at the University of Cambridge during my abroad period, which has been a terrific experience, fundamental for my personal growth.

A mia madre Teresa,
mio padre Marco,
mia sorella Giulia.
Grazie, sinceramente.

Grazie anche agli amici che mi sono stati vicini in questi lunghi anni: Alessio, Cristiano, Federico, Francesca, Francesco, Gabriele, Gaia, Giulia, Luca, Martina, Michele, Miriana, Nicoletta, Roberta. Un pensiero speciale è per te, Camilla, per tutte le emozioni che mi stai regalando.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter introduces all the main ideas and notions necessary to grasp the complex and lively research environment in which this dissertation is rooted. Artificial Intelligence (AI), data, society, ethics, and many other deeply intersected concepts are used to shape a precise world vision, giving meaning and context to the subsequent discussion. A description of this dissertation structure is then presented, stating the conceptual flow of the discussion. Each chapter is briefly introduced, allowing the reader to quickly identify the sections of more interest for his/her knowledge. Then, all the significant contributions achieved by the author are listed and described, assessing the practical impact and appreciation of the research work of which this dissertation is just the summa. Produced software, publications, and participation in national and international projects are listed, subdivided according to their topic, and briefly introduced and contextualised.

1.1 Motivations

Humans are becoming increasingly addicted to AI. Each aspect of the developed world is now flooded by ”smartness”: smartphones, which allow perpetual connection; smart automotive, which increases road security; smart homes, which ease household management; smart healthcare, which improves patient recovery; smart cities, which optimise citizen-city relations, and so on. Whether appreciated or not, the current reality is soaked with the idea that ”smart is better”. Some people do not like this trend, especially those whose jobs and competencies have been built over a lifetime and may be obliterated in a few years. Somebody else sees in this process the future itself, imagining a future where mechanical, repetitive, and tiresome jobs will not bother humans anymore. Whatever side the reader feels near to him/her has no relevance: the change (or the evolution) is happening, and current-day society is in the eye of it. The best bet is to try to manage this smartness addiction at best, making the most of it and consciously accepting its inevitable drawbacks. However, managing requires knowledge;

in the following paragraphs, a short background on how AI is mutating society will give the reader enough elements to grasp the complexity of the scenario.

First, it is essential to distinguish the meaning of two concepts that are often confused: AI and machine learning (ML). AI is a broader term than ML and generally refers to any automated system capable of making decisions in a given environment according to a specific policy. Such intelligent systems can be either human-programmed, like a rule-based system with hand-written rules, or automatically deducted from a data set thanks to specific learning algorithms, like a simple linear regression. ML identifies this last class of objects: intelligent systems that can extract relations existing in a given dataset through a specific training ("learning") algorithm. While classical, non-learning-based AI systems have existed for quite some time (the Dartmouth workshop in which the term "artificial intelligence" was first introduced dates back to 1956), ML systems have struggled more to become public knowledge. Nevertheless, now that most of the world knows ML, there is no turning back. One subset of ML systems, which are more correctly known as models, is currently shaking public opinion: Deep Learning (DL). This field encompasses all ML models based on deep neural networks (DNNs), which provide humans with unforeseen learning performance, knowledge extraction capabilities, and data generation features. This new property of generative DNNs is the latest cornerstone in the global AI addiction process. Humanity has discovered that AI is not limited to knowledge extraction from data but can produce new data based on its acquired knowledge. AI-generated images, sounds, videos, and text will change (and possibly dominate) the future consumption of multi-medial material. However, as the careful reader would easily sense, ML models are based on data, and so the more high-quality data fed to the AI, the better the performance obtained.

Thus, data is central to the global efforts to build better, more effective AI systems. However, this fact is not always known to people. As Shoshana Zuboff depicts in "Surveillance Capitalism," one of her best-known and appreciated works, large multinational corporations pervasively collected user data for many years in the past, exploiting the scarce people's consciousness of their personal data's value. As such, knowledge extracted from that data has been exploited for large-scale profit-making through personalised advertisements targeting people with unforeseen efficiency. Zuboff claims that the last step of this process is the current aim of large tech giants to no longer adapt their advertisements to the people but to use their massive knowledge bases about each individual to influence their personal choices opaquely, effectively directing individuals for the companies' economic benefit. A large portion of the digitalised population started to notice these malicious behaviours by observing how devices, social networks, apps, and websites dynamically adapt their advertisements according to their needs, often in a too-direct and suspicious way. Thus, the general concern about personal data harvesting and usage is growing. National and international institutions are producing different legal devices in a concrete effort to contrast this dangerous, data-based manipulation mechanism. Europe, for example, introduced the General Data Protection Regulation (GDPR) in 2016 to strengthen and homogenise European citizen's personal

data control and rights. While enhancing citizen's privacy rights, this new regulation also strongly limits data movements across companies and institutions, posing severe issues in designing data-dependant software, such as AI-based systems. This limitation is particularly evident when personal data is not used for profit and advertisement but to offer public and research services. One representative example is the medical field, in which AI-assisted diagnosis is becoming more widespread every day, helping physician in their daily job, reducing human errors and increasing diagnosis reliability.

This thesis finds its ground in this disputation: as time passes, society needs and desires more intelligent devices and powerful AIs and cannot help it. However, society also recognises the rights of individual citizens to refuse to nurture such systems with their data. The contrast between individual staticity and societal dynamicity leads to the research of new methodologies combining both needs. The solution explored in this dissertation was introduced by Google, one of the actors that first caused the data issue. To continuously feed data to its AIs (specifically, the next-word predictor models for Android smartphones), Google experimented with an ML paradigm never devised before: the data held by each device was no longer subject to harvesting, moving, or accumulation in centralised data lakes but left where it belonged; the subject of exchange and movement this time was the knowledge extracted from the data itself. Thus, the key idea behind federated learning (FL) is not to move the data but the knowledge derived from it. Such knowledge is not subject to any law or restriction since it is impossible (or, better, really difficult) to derive the data from which it was extracted anymore. This paradigm opened many doors for both research and industry to continue developing even better and smarter AI systems, which can cope with the new regulations protecting individual privacy. FL can be seen as a powerful cooperative tool, enabling individuals to take advantage of their cumulative data potential without disclosing sensible information to others and maintaining control over data. This dissertation deals with FL, exploring it under many different circumstances, and discusses how this innovative paradigm can be further expanded and improved.

1.2 Dissertation structure

This section summarises this dissertation's contents, briefly describing them section-by-section; please refer to Table 1.1 for a more schematic representation.

Chapter 1 constitutes the introduction to this PhD dissertation, describing the broader frame in which the proposed research work is located. Section 1.1 focuses mainly on the high-level social and technological scenario in which this thesis is rooted, giving a wider scientific context to the reader. Section 1.2 describes the thesis's structure, briefly introducing each topic explored during the PhD path and giving them a global logical structure. Section 1.3 lists all contributions produced by the author during his PhD, especially focusing on open-source software published on journals and conferences.

Chapter 2 introduces all the concepts needed to better understand and appreciate

Table 1.1: Schematic structure of this dissertation. Each chapter is briefly introduced, reporting its main contents and software contributions; each study leads to the topics examined in the following one.

Chapter	Title	Contents	Contributions
1	Introduction	Introducing all the main ideas and notions underlying this work; Describing this dissertation's structure and contributions.	
2	Background	Introducing all the main concepts required to understand this dissertation's main scientific contributions.	
3	Model-Agnostic Federated Learning	The PRAISE study pros and cons justify the investigation of model-agnostic FL approaches; MAFL is described and validated reproducing the PRAISE score study.	PRAISE *OpenFL-x*
4	High-Performance Federated Learning	The analysis of commercial FL frameworks reveals common poor design choices; an alternative high-performance is proposed together with a FL-specific DSL.	*FastFL*
5	Cross-Facility Federated Learning	The scaling/learning performance FL trade-off is investigated; FL is thus interpreted as a cross-facility enabling tool and geographically -distributed LLM training experiments are discussed.	*xFFL*
6	Conclusions	Summarising and wrapping up the dissertation; limitations and future work are discussed.	
Appendices	Federated Learning Applications	FL learning benchmarks on two applicative domains at varying hyperparameters are reported and discussed: - drug-target interaction; - solar wind prediction.	

the scientific contributions of this dissertation. Section 2.1 briefly introduces the fundamentals of ML, focusing mainly on the difference between "classical" (2.1.1) and "deep" (2.1.2) ML models. Section 2.2 introduces FL, the central concept explored in this doctoral thesis: FL is defined (2.2.1), its evolution and flavours are examined (2.2.2), and the major open-source software implementing this technique are analysed (2.2.3). Finally, Section 2.3 introduces the main concepts behind distributed computing systems, describing the most widespread ones available at the time of writing from the most centralised and performing to the most distributed and low power: High-Performance Computing (HPC) (2.3.1), Cloud (2.3.2), and Edge (2.3.3).

Chapter 3 presents the more ML-oriented contribution of this dissertation: Model-Agnostic Federated Learning (MAFL). Section 3.1 introduces the chapter describing the PRAISE study since it constitutes the starting point from which the decision to explore FL originated: the statistical (3.1.1) and ML (3.1.2) predictive tools currently used in the cardiological field are analysed, then the PRAISE model is proposed (3.1.3), and an alternative distributed implementation based on the concept of federated pooling is proposed (3.1.4). Section 3.2 describes OpenFL-extended, the first practical contribution of this doctoral thesis: it analyses the architecture of the open-source FL Intel® OpenFL framework (3.2.1), explain how to make FL model-agnostic (3.2.2), implementing such approach into Intel® OpenFL (3.2.3), and providing an experimental evaluation of the proposed software (3.2.4). Finally, Section 3.3 describes how the MAFL apprach can be

applied practically on a critical financial (3.3.1) use case in collaboration with Intesa SanPaolo: through the use of a public credit card transaction dataset (3.3.2) a proof-of-concept MAFL federation is deployed and the obtained performance analysed (3.3.3).

Chapter 4 explores the second main contribution of this dissertation, discussing the computational performance of current FL frameworks and proposing a more flexible and scalable approach. Section 4.1 introduces the main issues underlying the current FL framework design, focusing primarily on the communication topologies (4.1.1), communication backends (4.1.2), and programming languages (4.1.3). Section 4.2 showcases *FastFL*, an open-source C/C++ software based on the FastFlow parallel programming library (4.2.1) trying to overcome the abovementioned issues, presenting three practical use cases (4.2.2) and providing experimental results. Section 4.3 provides a more theoretical contribution to FL presenting RISC-pb^2l, a formal language for modelling parallel computations (4.2.2), discussing possible way of modelling FL processes through it (4.2.2), and creating Python wrapper for *FastFL* based on a FL domain specific language (DSL) based on it. Finally, Section 4.4 describes two experimental deployments analysing *FastFL*'s scaling performance compared to off-the-shelf FL frameworks (4.4.1) and implementing a real-time distributed multiview detection system (4.4.2).

Chapter 5 delves into the third and last main contribution of this dissertation: cross-Facility Federated Learning (*xFFL*). Section 5.1 explores the compute divide issue (5.1.1), how to deploy a FL workload onto multiple, geographically distributed HPC facilities to exploit the publicly available computing power (5.1.3), some *xFFL* preliminary experiments (5.1.2), and the effort in building the first European HPC federation through FL to train state-of-the-art large-scale ML models (5.1.3).

Concluding, Chapter 6 concludes the dissertation, summarising the provided contributions and the scientific impact of this dissertation. Section 6.1 analyses the limitations of the discussed approaches, pointing to interesting research directions to overcome them. Finally, Section 6.2 states final remarks and concludes the dissertation.

Additionally, Appendix A describes two FL learning performance benchmarking efforts carried out by the author during his PhD visiting period at the University of Cambridge, UK. Section A.1 investigates the task of predicting drug-target interaction (DTI) by first exploring this task (A.1.1) and then presenting experimental results in a vast amount of scenarios, including different client counts and data non-IIDness (A.1.2). Section A.2 carries out a very similar investigation in the domain of solar wind prediction: in this case, the applicative domain is explored (A.2.1) and then experimental results under various FL situations are presented (A.2.2). Finally, Section A.2.2 reports the funds and projects that support the presented research work.

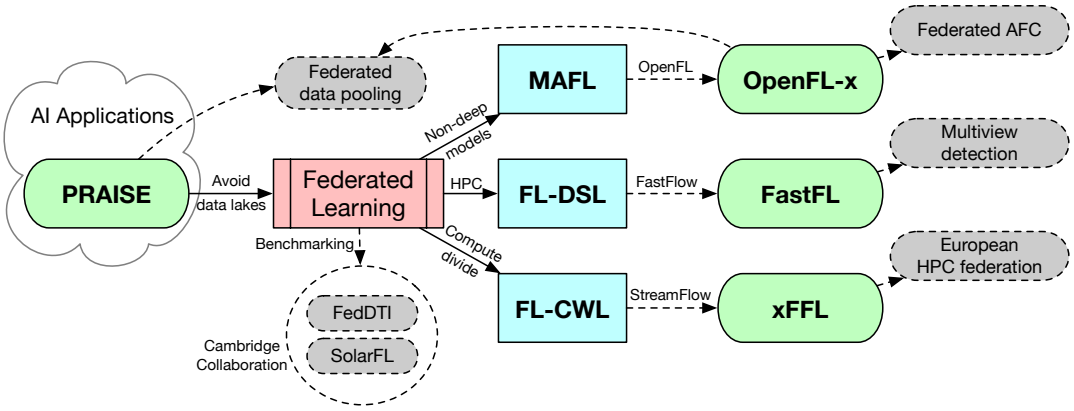


Figure 1.1: Logical flow of this dissertation's main contributions. The PRAISE score, the first AI application developed in the author's PhD path, lays the foundation for investigating FL techniques: Can data lakes be avoided? This led to the investigation of FL under three main research scenarios. The green background identifies software, cyan methodological, and grey applicative contributions. Dashed lines represent implementation efforts.

1.3 Contributions

The following lists and briefly exposes the major contributions made by the author during his PhD career; each one is then analysed more in-depth throughout the dissertation. Figure 1.1 represents the logical connections and flow between them, following closely the author's PhD path. The developed contributions consist of innovative methodologies that aim to investigate and establish new state-of-the-art practices in the ML/FL fields. This methodological approach is fundamental for producing structured innovation deeply rooted in the current scientific literature and innovative ideas capable of impacting the scientific community; however, without proof-of-concept implementation and experimental validation, such contributions may lose solidity. Thus, the proposed methodologies are investigated theoretically and practically, with software implementations aiming to validate their functional and nonfunctional aspects. The following paragraph summarises the software produced to validate such methodologies experimentally, constituting the concrete contribution of this PhD thesis. All the presented software should be considered research tools and are not intended for production environments. They implement experimental algorithms and concepts, and their source code is open-source and freely available on GitHub. These choices allow other researchers and practitioners to examine what goes on under the hood, possibly improving them and inspiring new research. All the presented software is supported by peer-reviewed publications and, thus, is validated by the scientific community. Table 1.2 summarises all of them more schematically.

Table 1.2: Acronym, research objective, and brief technical description of the main software contributions described in this dissertation; as can be seen, each contribution raises topics and questions leading to the subsequent contribution.

Contribution	Research objective	Technical description
PRAISE	Predicting the risk of adverse events (death, major bleeding, acute myocardial infarction) over a one-year period in patients who suffered from ACS.	Three AdaBoost models are trained on the largest ACS patient's data lake available at the time of writing.
OpenFL-x	Allowing ML practitioners to exploit traditional (non-deep) ML models in FL scenarios in a model-agnostic way.	The Intel® OpenFL FL framework is extended to support the AdaBoost.F model-agnostic algorithm, and computationally optimised.
FastFL	Overcoming the current limitations in FL framework scaling, allowing the exploitation of larger infrastructures (e.g., HPCs) more easily and efficiently.	The high-performance C/C++ FastFlow framework is used as the distributed computing framework under the hood of a high-level FL Python DSL (RISC-pb^2l).
xFFL	Bridging the compute divide between academia, SMEs and Big Tech, allowing them to use FL as a scaling tool, allowing new trade-offs between scaling and learning performance.	FL is modelled as a cross-facility computation expressed as a StreamFlow workflow exploiting the computing power of many Top500 European HPCs.

The PRAISE score[1] is a freely available web service designed for cardiological use. It aims to help physicians in designing the correct therapy for patients who suffer from Acute Coronary Syndrome (ACS) and are now under medication. The system accepts four types of medical information as input: clinical, therapeutical, angiographical, and procedural variable, for a total of 25 medical indicators. These variables are then fed to an AdaBoost model trained on the PRAISE dataset, a cohort of 19,826 adult patients who suffered from an ACS in the past, gathered from the BleeMACS and RENAMI registries, which included patients across several continents. The outputs of the system are three different risk score indicators, each one for a specific adverse event: acute myocardial infarction, major bleeding, and death. The predictions are calibrated over a one-year period, effectively assessing the risk of one of the abovementioned adverse events occurring in patients who have already suffered from an ACS up to a year after the follow-up. The PRAISE score is particularly useful for cardiologists since it helps to obtain a clear vision of the major risks the patient is currently experiencing, thus helping the physician formulate the right therapy for each individual. From an ML point of view, the PRAISE dataset was split into a training cohort (80%) and an internal validation cohort (20%), and many AdaBoost models have been trained to explore the hyperparameters space. The three final models, one for each adverse event investigated, have been externally validated on a cohort of 3,444 patients collected from the

[1] https://praise.hpc4ai.it

European SECURITY randomised controlled trial, two prospective registries from the University of Ferrara (the FRASER study, and the Prospective Registry of Acute Coronary Syndromes), and from the Clinical Governance in Patients with ACS project of the Fondazione IRCSS Policlinico S. Matteo of Pavia. The PRAISE score is an officially recommended tool by the European Society of Cardiology in their 2023 ACS guidelines [33].

OpenFL-x[2] is an open-source extension of Intel® OpenFL supporting FL involving any ML model. This software aims to expand the FL paradigm from the DNN niche to the broader scenario of non-deep ML models, encompassing Linear and Logistic Regressions, Naïve Bayes Models, K-nearest neighbours, Decision Trees, et similia. This capability of supporting any ML model is referred to as *model-agnosticism*. The underlying learning algorithm allowing this property to emerge in an FL context is AdaBoost.F, a federated version of the well-known AdaBoost ensemble boosting algorithm. The key aspect of AdaBoost.F is that the local models are not aggregated into a single global model anymore; instead, the best-performing local model of each federation round is added to the global ensemble boosting model, with the consequent continual re-tuning of the AdaBoost parameters. In this way, each new local model added to the global ensemble is focused on correcting the errors made by the previously accepted models. Such an approach is beneficial in contexts where the black-box behaviour of DNNs is not tolerable, or when, due to the data structure, DNNs are not the best choice, or even when it is already known that a specific ML model works well on the local data. The *OpenFL-x* development process also led to the discovery of many performance issues in the Intel® OpenFL framework, which are addressed in the proposed software, leading to a 5 times speedup over the base framework.

FastFL[3] is an open-source FL framework aiming at introducing new design concepts in the FL community. Inspired by the vast majority of current ML framework, *FastFL* exposes a simple Python interface, allowing practitioners to write working FL code quickly, but, under the hood, translates the specified logical operations into high-performance C/C++ code for efficient execution. The backend allowing for such execution is the FastFlow high-performance parallel programming library, the serialisation infrastructure is provided by *Cereal*, and *libtorch* provides support for DNN execution. The basic workflow of *FastFL* is simple: the user gives a description of an FL experiment, a DNN model, and a dataset, detailing all the required parameters; the framework then translates and compiles the specified computation into a C/C++ FastFlow source file, compiles the provided DNN model to a TorchScript format, and executes the computation. The advantage of using FastFlow as the backend is that the produced executable files can be executed in local or shared memory, effectively allowing both simulation and distributed deployments of FL tasks. Furthermore, FastFlow allows the specification

[2] https://github.com/alpha-unito/Model-Agnostic-FL

[3] https://github.com/alpha-unito/FastFederatedLearning

of many different communication topologies, thus allowing experimentation with different FL topologies and allowing researchers and practitioners to escape the standard master-worker structure. Lastly, since a one-to-one correspondence exists between the RISC-pb^2l constructs and the FastFlow building blocks, the latest versions of *FastFL* now support a Python DSL designed explicitly for FL. In this way, an innovative, declarative way of specifying FL computation can be used to deploy FL tasks, abstracting the user from the low-level details.

xFFL[4] is an innovative methodology aiming at fighting the compute divide by exploiting publicly-available computing power. Since developing large-scale AI systems is restricted to large private companies capable of acquiring sizeable private computing clusters, new tools and technologies must be developed to make such processes available to a broader public audience. *xFFL* demonstrate that this is possible through the exploitation of publicly available computing power distributed on many geographically distributed computing infrastructures. The proposed software stack leverages the StreamFlow WMS to deploy a specified computation on multiple computing infrastructures, retrieving and broadcasting the intermediate data and repeating the process until the desired final result is obtained. Such large-scale deployment implies interacting with complex systems such as SLURM and PBS, containerisation technologies such as Docker and Singularity, and even different microarchitectures. FL represents a perfect use case to showcase the potentiality of this approach; thus, an experimental LLaMA training distributed on three HPC centres is successfully designed and deployed through *xFFL* to prove the effectiveness of the proposed approach. The *xFFL* code is open-source and freely available, ready to be customised to be adapted to different use cases.

[4] https://github.com/alpha-unito/xffl

Chapter 2

Background

This chapter introduces all the main concepts required to understand and fully appreciate this dissertation's main scientific contributions. First, the basics of ML are discussed, focusing mainly on the difference between classical ML models, such as decision trees, naïve Bayes, logistic regression, et similia, and DNNs. Then, FL is introduced and formally defined, and its main typologies and frameworks are illustrated. The discussion then moves to the other aspect of this work, distributed computational systems. An overview of modern-day distributed systems paradigms is given, and HPC, cloud, and edge infrastructure characteristics are discussed.

2.1 Machine learning

ML is the branch of AI focused on developing algorithms capable of adapting and improving their predictive or generative performance by feeding on data. Adapting or improving the system's behaviour based on the provided data is called *learning* since the process remembers the human learning process in many aspects. The same ML algorithm, usually referred to as *model*, trained on different data will thus expose different capabilities and can therefore solve different tasks. In this context, when using the word *performance*, the underlying learning, predictive, or generative performance is considered, not the computational one, unless otherwise specified. This dissertation identifies two different subsets of ML models and explores them separately: classical ML models, i.e. non-neural networks, and DNNs.

2.1.1 Classical machine learning

Data is the starting point of the ML process. A set of *data points*, or *data samples*, is commonly referred to as a *dataset*. Each data sample comprises a set of *features*, which are information associated with the data sample, and optionally one or more *target variables*, the values a possible ML model will try to predict based on the values

of the features. The partition of the dataset used for the learning phase of the model (called *training*) is called *training set*; conversely, the partition eventually used for the evaluation phase (called *test*) is called *test set*. If the dataset presents target variable values, it is said to be *labelled*, *unlabelled* otherwise. From this starting point, many different ML algorithms can be applied to produce different models targeting different tasks.

As summarised by Flach [69], two types of ML models and three types of learning algorithms are identifiable. An ML model can be *predictive* if it aims to predict the target variables' values according to the provided features or *descriptive* if it aims to identify patterns and subgroups in the provided data. Recent research trends, however, firmly push towards another new type of ML model, called *generative* [77], whose aim is not to derive some information or structure from the provided training data but, instead, to generate new data congruently with the training set characteristics. Conversely, learning algorithms can be classified as *supervised* if the training data includes some target variable and *unsupervised* if not. However, this binary distinction became more complex with time. Current research strongly focuses on *semi-supervised learning* [38], a learning methodology capable of exploiting labelled and unlabelled data at the same time, and also on *reinforcement learning* [94], where an ML model is trained in a trial-and-error fashion, interacting with the surrounding environment and receiving feedback from his actions. Despite being known for a long time, these last two learning algorithms and the generative ML models have recently gained traction due to the vast amount of data, computational power, and learning power of DNNs accumulated in the last decade.

ML models can be trained to solve many tasks, but this dissertation will focus only on predictive ML models trained with supervised learning approaches. This ML setting is currently used in industry and research and is the most understood one. This particular setting can efficiently target two different tasks [64]: *classification* and *regression*. An ML model trained for classification tasks is called *classifier*, and it aims to assign the correct class label to each provided test data sample; on the other hand, an ML model trained for *regression* tasks is specialised in predicting continuous values assigned to each test sample. Such models can also be exploited to solve other ML tasks, such as *scoring* (*ranking*), in which the predicted value is a vector of scores among the different classes, and *probability estimation*, that is like scoring but in which the scores are in the range $[0,1]$. What these models learn under the hood is a probability estimation of the target variable given the feature values or, according to another point of view, an approximation function estimating the actual function associating a set of feature values to a specific target variable value [69]. This dissertation will almost exclusively deal with classification tasks, often targeting binary target variables.

There exist many *metrics* measuring the practical efficiency of a trained ML model. Specifically targeting classification [148], this research will mainly present three different types of learning performance metrics: *accuracy*, *area under the receiving operating characteristic curve (AUC)*, and F_β*score*. The accuracy of an ML is very straightforward

in its definition, being the ratio between the correctly classified samples and the total data samples number:

$$Accuracy = \frac{\#Correct\,predictions}{\#Datasamples}$$

Accuracy is a straightforward and effective way of measuring the performance of an ML model. However, it can be misleading when dealing with heavily unbalanced datasets in which most data samples are part of the same class. In this scenario, a classifier can learn to assign the majority class label to each data sample and still obtain a high accuracy score. The AUC can be used for binary classification tasks to alleviate this issue. The AUC is the area individuated under the *receiving operating characteristic (ROC)* curve, which is obtained by plotting the values of the *sensitivity*, also called *true positive rate (TPR)*, against 1-*specificity*, also called *false positive rate (FPR)*, for different values of the classification decision threshold. In formulas, being t the classification decision threshold, sensitivity is defined as the ratio between the correctly predicted positive data samples and all the positive data samples:

$$Sensitivity(t) = \frac{\#TruePositives(t)}{\#TruePositives(t) + \#FalseNegatives(t)}$$

while the FPR is the ratio between the number of false positive predictions and all negative data samples:

$$FalsePositiveRate(t) = \frac{\#FalsePositives(t)}{\#TrueNegatives(t) + \#FalsePositives(t)}$$

The AUC is a value bounded in the $[.5, 1]$ range, where .5 indicates random classification and 1 perfect classification. Lastly, when discussing systems in which the tradeoff between *false positive* predictions and *false negative* predictions is crucial, like the medical domain, the F_β score comes into play. Given that *precision* identifies the ratio between the true positive predictions and all the positive data samples:

$$Precision = \frac{\#TruePositives}{\#TruePositives + \#FalsePositives}$$

and *Recall* the ratio between the true positive predictions and all the positive predicted values:

$$Recall = \frac{\#TruePositives}{\#TruesPositives + \#FalseNegatives}$$

the F_β score is defined as:

$$F_\beta = (1 + \beta^2)\frac{Precision \times Recall}{(\beta^2 \times Precision) \times Recall}$$

This formula implies that the values of β establish the relevance of recall over precision, with higher values of β implying more recall weight over precision. These metrics can be applied to any ML model, provided that the model is a classifier.

Many classical ML models will be named through this dissertation, but mainly two will be used consistently: *decision trees* and *support vector machines (SVMs)*. Decision trees [100] are tree-like systems in which a data sample is fed to the root of the tree structure; a series of hierarchical conditions are then applied to the data sample features, routing it through the internal nodes towards a leaf, identifying the final data sample predicted value. Decision trees are, by their nature, hierarchical structures capable of partitioning the feature space into arbitrarily small subsets. From a statistical point of view, they are known as *recursive partitioning* methods [28] since they are based on the recursive partitioning of a population into smaller subgroups. This splitting process is usually guided by a metric measuring the effectiveness of the split; the most commonly employed splitting metric is the Gini index [37]. SVMs [82], on the other hand, produce linear classifiers on the provided data, maximising the gap obtained between the two identified classes. The SVMs build one or more hyperplanes in a multidimensional or infinite-dimensional space subdividing the space region containing samples from different classes, maximising the distance between the samples nearer to the bound and the bounds themselves. It is also possible to apply SVMs to non-linearly separable data by utilising the kernel trick, which involves remapping the input data samples into a different multidimensional space. Other types of ML models will be named through this dissertation, like linear regression [66], k-nearest neighbours (KNN) [65], and Gaussian naïve Bayes [68], but the discussion will not delve deeply into the inner workings of such models.

Sometimes, however, a single ML model cannot offer enough learning power to offer decent learning performance over a dataset. Remaining in classical ML, a possible solution to this issue is offered by the *ensemble learning* mehods [67]. These methods combine single ML models, called *weak learners* or *weak hypostesis* in this context, to construct a more powerful collective model, called *strong learners* or *strong hypostesis*. Two widespread methods exist to construct such ensemble models: *bagging* and *boosting*. Bagging [27] is based on collecting many weak learners trained independently on the same dataset. When predicting, all the weak learners calculate their prediction over the given data sample; the strong learner prediction is then calculated over the set of weak learners and can be obtained as a majority vote, the average, or any other chosen policy. Boosting [146], on the other side, iteratively trains different weak learners on the same dataset by re-weighting each time the misclassified data samples, thus forcing each weak learner to focus more learning power on the most misclassified instances to fill the prediction gaps of the previously trained weak learners. The final ensemble prediction is then obtained as the weighted sum of the weak learners' predictions, each weighted according to its learning performance. Lastly, another ensemble technique is employed, called *stacking* [172], in which an ML model receives all the weak learner predictions and produces the final prediction. However, it is not used in this dissertation.

A software framework is needed to implement and experiment with all the exposed

concepts. Nowadays, most of the publicly available ML code is developed with the open-source *SciKit-Learn* [102] library. SciKit-Learn is the de-facto standard library to build ML systems: it offers a complete and user-friendly Python interface, is built upon the very mature NumPy, SciPy, and matplotlib packages, and comprises all the necessary logic to build complete ML pipelines, from data handling to model training ad validation. Many experiments related to classical ML models reported in this dissertation have been implemented through this library.

2.1.2 Deep learning

DL started as any other branch of classical ML. The research work to open up this field first is by Rosenblatt and dates back to 1958 [141], proposing the *perceptron*, a probabilistic model inspired by the human neuron. In his work, Rosenblatt demonstrated that a system made up of randomly connected perceptron is capable of learning patterns intrinsic to randomly provided stimuli and that if such stimuli are differentiable into different classes by similarity, the probability that the final system learns a better-than-chance association between the stimuli and its class increases in the number of examples feed to the system itself. While a single perceptron is capable of distinguishing only linearly separable data [118], it is also true that multiple perceptrons stacked up in layers are capable of approximating functions of any type [20]. From this correlation between the perceptron structure and the underlying human neural network inspiration, systems of multiple connected perceptrons became called Artificial Neural Networks (ANNs).

From this historical basis, the field of DL slowly started to be studied until it exploded in the last decade with the ubiquitous availability of the necessary computational power to simulate and build such systems. As the number of layers in ANNs started to grow deeper and deeper to approximate more complex functions, ANNs started to be called DNNs, and, as a consequence, the ML branch dealing with such structure is named DL. DNNs are usually trained through the *back-propagation algorithm*, in which a data sample is fed to the network, the output prediction is compared to the ground truth label assigned to the data sample, an error metric is calculated on such error, and then all the DNN parameters are adjusted to accommodate the error. The DNN parameters are usually called *weights* since they are the weights of the weighted sum of the inputs of each neuron, and the error function calculated on the output is named *loss function*. Empirically, the training process can be described as exploring a complex multidimensional surface: the loss function and the training data describe such a surface, while the weights configuration indicates a position on it. During training, the model explores the loss surface, trying to find a good local minimum value of the loss function. Such a point assures a low loss function value and good learning performance. A commonly used algorithm for finding such a minimum point is the *stochastic gradient descent* (SGD) [26].

Due to their nature, DNNs are highly variable in their structure and can be modelled to deal with many different tasks. Notably, DNNs revolutionalised the field of

computer vision, thanks to the *convolutional neural networks* (CNNs), such as the residual DNNs [178] and the dense DNNs [87], and the field of natural language processing (NLP), through the use of long-short term memory (LSTM) DNNs [186]. Especially in this last research field, DNNs are currently shaking the public opinion on AI due to the public diffusion of generative DNN-based NLP services such as ChatGPT, the chatbot created by OpenAI® capable of quickly and easily generating complex and incredibly natural text outputs in response to natural language text inputs. ChatGPT is but one example of *generative* DNNs; other models capable of generating human-like text, usually called large language models (LLMs) due to their massive sizes and almost prohibitive training times and computational requirements, are Google® OpenFL Bard [154] and Meta® LLaMA [165]. However, generative DNNs targeting images, such as OpenAI® DALL-E [114], also exist.

Such large DNNs cannot be trained on a standard processor, at least not in a humanly-compatible time. Due to their inner workings, DNNs can be efficiently trained on graphical processing units (GPUs), which are indeed very common nowadays, but also new specialised processors are now designed to fit this kind of workload precisely, called *accellerators*, such as tensor processing units (TPUs) [93], field-programmable gate arrays (FPGAs) [166], and application-specific integrated circuits (ASICs) [98]. Often, a single accelerator is still insufficient to complete a DNN's training in a sufficiently brief time. In this scenario, multiple accelerators can be exploited in parallel to achieve shorter execution times through distributed training algorithms such as synchronous SGD [55], in which a copy of the model is allocated on each accelerator and then the intermediate gradients are exchanged in an all-to-all fashion, averaged, and redistributed back, or with a layer-wise partitioning of the model on multiple accelerators [4], thus parallelising the forward and backward pass of the learning algorithm, exploiting a pipeline-like principle.

2.2 Federated learning

FL is a relatively recent distributed ML methodology [116] aiming to bridge the gap between the need to train ever bigger ML models on ever larger datasets and the individual and companies' will to protect and not share their private data. From another point of view, FL is also a way to distribute the training of an ML model even more than before. However, it should be considered that the learning performance of FL is usually lower than that of traditional centralised learning [119].

2.2.1 Definition

A first attempt to define what FL is can be found in a comprehensive paper discussing the challenges and open issues of FL, authored by the same authors of the seminal paper giving birth to this research trend, Kairouz and McMahan [95]: *"Federated learning*

is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client's raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective." The practical implementation of an FL system is usually less general than this definition states. Most often, the models trained on the clients are DNNs, and the updates exchanged can be the DNNs' weights or gradients: since these values are tensors, they are easy to serialise, send over the network, and combine through mathematical operators. Usually, it is expected that all clients train the same DNN architecture and that each client's data schema is the same as all the others; these are strong assumptions to make, especially in the real world, but many works are trying to overcome them [106, 56, 139]. FL is thus an iterative process: the clients calculate the local updates and send them to the central server, which combines them and sends them back to the client, and the cycle restarts from the beginning. A single iteration of this cycle is called *federated round*, or simply *round*.

The basic strategy to combine the updates from different clients, usually called *aggregation strategy*, is *federated averaging* (fedAvg) [116], in which the clients' contributions are averaged between themselves. This approach, albeit simple, can provide decent learning results if the data distribution among the clients is particularly favourable, i.e., IID [108]. A straightforward improvement of fedAvg can be obtained by weighting the updates according to the amount of data they are calculated on; in this way, updates based on more data will weigh more than the others. In cases where the data non-IIDness does not come just from the quantity but also from the internal distribution of the data, other approaches are needed, such as SCAFFOLD [97], which takes advantage of variance reduction techniques to reduce the clients' performance drift due to the biased data distribution, or Favor [168], which routinely chooses which clients' updates to aggregate each round to balance the introduced biases. FedAvg is inherently synchronous, which can pose a limit to this approach's computational performance; thus, asynchronous approaches have been proposed, like ASO-Fed [43], which combines continual learning on the client side and asynchronous aggregation on the server side, or FedBuff [125], which exploits buffers to aggregate batches of updates asynchronously. Also, the communication cost has a crucial impact on FL computational performance. Approaches such as quantisation [152], compression [145], and distillation [174] have been proposed to reduce the update size, allowing for faster communications, thus reducing the FL overhead.

However, FL introduces another issue other than simple computational overhead and learning performance instability: handling *statefull* objects. The standard optimisers used in traditional DL maintain an internal state linked to the model's current weight configuration, and performance loss can happen when the local model's weights are substituted with the global model's ones. Federated optimisation techniques can be put into place to mitigate such performance issues [175], such as FedSplit [133], FedProx [107], and federated versions of adaptive algorithms such as ADAM [140]. At the

same time, another component of modern-day DNNs maintains an internal state: batch normalisation layers. When aggregating different models, it can be assessed that not considering these layers can lead to performance degradation [36]. Two straightforward solutions to this issue are averaging the batch normalisation parameters and the model's weights or substituting them with stateless normalisation layers, such as layer normalisation layers [59]. Another way is to implement a modified version of batch normalisation, capable of handling the weights shift, such as FixBN [191].

Apart from the technical issues of deploying an FL system, privacy aspects should be considered since FL is mainly adopted in scenarios where data privacy is of interest. Many different approaches can be implemented to enforce privacy guarantees in FL; the most widely used one is briefly exposed below. *Homomorphic encryption* (HE) [184] is an encryption technique allowing data to be converted in an encrypted format which supports a set of operations congruently with the original data format, i.e., working in the encrypted space will yield a result that, when decrypted, will correspond to the exact result that would have been obtained if the same operations were done on the non-encrypted data. This technique can be applied in the FL setting [187], usually encrypting the updates sent to the central server, which operates on the encrypted space, and then decrypting back the aggregated model. *Secure multi-party computation* (SMC) [52] is a cryptographic technique allowing many entities to collaboratively calculate a function dependent on their local data without disclosing them to the others; this approach takes into account the correctness of the result and the robustness to adversarial attacks. This approach can be combined with FL [109], obtaining a secure exchange of updates with the assurance that no client will be able to inspect other clients' updates, together with robustness to malicious clients. *Differential privacy* (DP) [60] is a privacy measure consisting of adding a controlled amount of noise to a system in the effort of not significantly altering its performance while assuring the impossibility of inferring the presence of a particular data sample in the underlying dataset. This approach is advantageous in FL [170], where the controlled noise can be added in many different steps of the workload (local batch update, update sent to the server, global model parameters, et similia), obtaining different performance and privacy effects, primarily hindering membership inference attacks. Finally, other techniques can be combined with the ones mentioned above to improve further the robustness and privacy-preserving properties of an FL system: *trusted execution environments* (TEEs) [143] can be exploited to run the local DL training or aggregation on non-trusted infrastructures [123], while *blockchain* [128] can be used to securely log in clients and keeping track of the update history, while also handling aggregation through smart contracts [136].

2.2.2 Categories

FL systems can be roughly classified according to two orthogonal dimensions: *deployment scale* e *data partitioning*. Starting from this first parameter, deployment scale, two possible scenarios are identifiable: small-scale deployment on powerful hardware,

called *cross-silo*, and large-scale deployments on low-power hardware, called *cross device*. Cross-silo FL [86] typically takes place on small federations (<100 clients) of server-grade machines or even entire data centres. This scenario encompasses federations of hospitals, assurance companies, universities, banks, and similar institutions. The underlying assumption is that the hardware is powerful enough to handle heavy DL workloads efficiently, that the clients are reliable machines always online, that the federation's network connection is fast and stable, and that this process is handled by expert personnel. This scenario is particularly suited for training large DL models on critical data. Thus, there is a significant focus on security issues, developing a common DL model suiting every client's data distribution without disclosing any information. On the other hand, cross-device FL [96] usually targets smartphones or, in general, edge devices. Thus, this scenario comprehends all the FL deployments on a large number of clients (>100 clients), which are usually unreliable, equipped with low-end hardware and, under power constraints, connected to the internet via a mobile connection. In this context, small DL models are usually trained locally on the device, and the aggregation process randomly samples only a subset of all the available local models due to the inherent bottleneck issue that the master-server approach presents when dealing with so many clients. This dissertation will deal with cross-silo FL only.

Three FL categories can be identified when moving to the data partitioning dimension according to how the feature and sample spaces are split among the clients: *horizontal*, *vertical*, and *hybrid* FL [183]. Horizontal FL is by far the most common FL scenario: in this context, the local datasets held by each client share the same *feature space*, i.e., they present the same feature schema, while their *sample space*, i.e., the set of data samples, does not overlap. Vertical FL [177], on the other hand, presents the inverse situation: the clients do not share the same feature space but overlapping subsets of it, while they share the same sample space. These two FL typologies take their names from how a tabular dataset would have been subdivided in each case: in horizontal FL, the data table would have been split between the clients through horizontal lines, while in vertical FL, it would have been split by vertical lines. Finally, hybrid FL [189] is a situation in which both the feature and sample spaces do not perfectly match between the clients but present some overlap. Each of these scenarios uses different DL models, learning algorithms, and aggregation strategies to accommodate the different feature/sample space characteristics; this PhD thesis will focus on the most common FL scenario: horizontal FL.

2.2.3 Open-source frameworks

Many frameworks have been designed to implement and deploy FL systems; the following will briefly list and describe the open-source ones.

Intel® OpenFL[1] [71], developed in collaboration with the University of Pennsylvania, is a cross-silo FL framework implemented in Python. It relies on the gRPC library for handling client/server communication, offers long and short-lived components to handle the FL workflow, and fully supports PyTorch but only partially other frameworks (i.e., not all the framework features such as all aggregation algorithms are already implemented in all frameworks). It is hosted by The Linux Foundation and offers support for the Intel® *SGX* TEE.

Flower[2] [23], initially developed at the University of Oxford, is a cross-silo/cross-device FL framework implemented in Python with gRPC communications. It is framework-agnostic, supporting software like PyTorch and TensorFlow, as well as Pandas and NumPy. It is developed with AI research in mind and designed to be extensible and customisable.

NVIDIA FLARE[3] [142] is a cross-silo FL framework implemented in Python with gRPC communications. It is framework-agnostic and explicitly supports horizontal and vertical FL, focusing on privacy-preserving techniques. The same FL code can be run in simulation, proof-of-concept, or distributed deployment.

FedML[4] [81] is an enterprise-level service aiming to provide full-stack tools for decentralised and federated learning settings, from the software framework to the computational infrastructure. Mainly developed in Python with gRPC-based communication, it aims to be a comprehensive environment for distributed and cross-silo federated learning workloads at scale. It supports all the main DL frameworks and communication libraries like gRPC, MPI, and MQTT.

TensorFlow Federated[5] (TFF) is the federated branch of TensorFlow. Mainly developed in Python with gRPC communications, it supports only Keras/TensorFlow and targets the cross-silo scenario. It also offers facilities for implementing DP and attacks by malicious clients [158].

PySyft[6] [194] is a Python-based framework offering a wrapping for other commonly used DL frameworks like PyTorch exploiting websockets communications. It aims to enable privacy-preserving techniques, such as DP and SMC, in the distributed DL scenario. It is based on a client-server approach to interact with remote datasets.

FATE[7] [44] is a Python-based, industrial-grade cross-silo FL framework based on gRPC communications. It offers HE and SMC computation protocols and supports many FL algorithms. It is hosted by The Linux Foundation.

[1] https://github.com/securefederatedai/openfl

[2] https://github.com/adap/flower

[3] https://github.com/NVIDIA/NVFlare

[4] https://github.com/FedML-AI/FedML

[5] https://github.com/tensorflow/federated

[6] https://github.com/OpenMined/PySyft

[7] https://github.com/FederatedAI/FATE/

FederatedScope[8] [179] is a Python-based event-driven FL platform based on gRPC communications, designed for both industry and research. It offers modules for implementing DP, privacy attacks, graph FL, recommendation systems, and other functionalities.

LEAF[9] [34] is a simulation-only FL framework developed in Python. It aims to establish benchmarks for various FL tasks and datasets.

2.3 Distributed systems

Distributed systems are computational systems made up of autonomous computational units called *computation nodes* interconnected together, appearing to the user as a single coherent system [156]. Each computation node is a complete computational unit composed of processors, memories, and, eventually, accelerators that could theoretically operate independently without being a part of a larger computational infrastructure. Cooperation between nodes is handled through different *communication protocols*, usually based on *remote procedure call* (RPC) or Message Passing Interface (MPI) implementations. These systems allow large-scale applications to run efficiently, exploiting the cumulative computational power expressed by the computation nodes. However, designing software capable of efficiently exploiting all the computational power provided by the underlying distributed infrastructure is not simple [45] since it has to correctly take advantage of the intra-node parallelism exposed by muti-core systems, and the iter-node parallelism offered by distributed systems. This software property, called *scalability* [110], is the primary metric for discussing parallel and distributed software performance.

2.3.1 High-performance computing

HPC systems, sometimes referred to as *computing clusters* and *supercomputers*, are distributed systems explicitly focusing on high computational performance [58]. These systems are massively parallel, collecting from hundreds to thousands of high-end computation nodes interconnected by low-latency, high-bandwidth networks, offering different file systems subdivided by capacity and performance. The computation nodes composing these systems are usually homogeneous, meaning that all of them are exact replicas of the others, thus making virtually indistinguishable the code execution on different subsets of computation nodes since each one of them offers precisely the same performance as any other and can reach tens of tera *floating point operations per second (FLOPS)*. Large HPC infrastructures can be subdivided into partitions, each using a different computation node type, allowing better resource usage according to the

[8] https://github.com/alibaba/FederatedScope

[9] https://github.com/TalwalkarLab/leaf

computation characteristics. The most common *communication topologies* for interconnecting the computation nodes are three: torus, fat-tree, and dragonfly [91]. Practical implementation of such low-latency, high-bandwidth networks can reach hundreds of Gb/s. The most well-known examples of fast interconnection networks for HPC systems are NVIDIA® *InfiniBand* [134] (initially developed by Mellanox®) and Intel® *Omni-Path* [24]. Finally, a fast data storage system can reach up to a few TB/s [29], completing the overview of the performance of an HPC infrastructure.

However, these numbers refer to the single performance peak of the components of an HPC cluster and do not reflect the system's true computing capabilities. Seamless interoperability between all these components is not guaranteed; faults are to be expected when dealing with such a large amount of hardware. Also, the software stack handling all the infrastructure has to be tailored and optimised to provide the best possible performance. To this end, many benchmark software has been proposed by the scientific community to test the actual capabilities of HPC systems, with the most widely employed one being HPL, a portable implementation of the LINPACK benchmark written in FORTRAN [57]. A list of the most powerful supercomputers in the world based on this benchmark is the TOP500[10] [117], which is updated two times a year. The current top-10 supercomputers in the world have the computing power of a few hundred up to a thousand PFLOPS, consuming thousands of KW of power. These computational powers are near the *exascale* frontier [151], the next objective in HPC infrastructure, aiming at building supercomputers capable of computing at the EFLOPS ratio.

HPC systems are not directly under the user's control due to system architecture design choices (see, as an example, the University of Turin OCCAM HPC cluster architecture [9]). Such systems usually allow users access through SSH connection to *frontend* servers, computing nodes designed to divide the HPC infrastructure from the external world. On these nodes, users can develop and, if it requires a small amount of time, compile and test their code since the frontend nodes have the same structure as the cluster's computing ones. The user cannot directly install new software; the one available has been specifically compiled to adapt to the computing platform, and technicians must install new software. Software tools such as Spack [75] can help in managing the software installation in this context. Once the code is ready for execution, the user cannot directly launch it on the computing nodes: HPC centres use a *batch execution model*, implying that no interaction with running code is allowed. The user should prepare in advance all the necessary scripts to ensure that his computation will run autonomously from the beginning to the end; such an auto-contained software execution is called *job*. A job can then be submitted to a *queue system*, such as SLURM [185] or HTCondor [63], which take care of starting the job on the requested resource as soon as they are free from other computation. In this way, code execution can be automatised while maximising computing nodes utilisation and the global infrastructure throughput.

[10] https://www.top500.org

2.3.2 Cloud computing

Cloud computing infrastructures offer a virtually unlimited amount of heterogeneous computational resources with an on-demand policy, trying to accommodate as many types of workload as possible (see, as an example, the University of Turin HPC4AI cloud system [8]). The idea behind this concept is simple: since different computations require different kinds of computational resources, let the user specify (and pay for) the exact amount and type of resource needed. This intuition implies two fundamental design principles: resources should be heterogeneous and on-demand. *Heterogeneity* refers to the vast offer of computational architectures in cloud environments. Computational nodes can be equipped with CPUs, GPUs, TPUs, or any other kind of processor and come in many different flavours, such as those with high memory, high computational power, and high-bandwidth capabilities. Resource heterogeneity thus allows cloud providers to satisfy all the possible spectrum of computational needs and budget constraints, even if introducing resource allocation issues that should be carefully handled to maintain the system's efficiency *wang2015*. Cloud resources are available *on-demand*: users can ask the cluster management software to be allocated a particular set of resources, which usage will be reserved and private. These two properties create the illusion of a personal, unlimited resource pool that can be bent to any computational need [155].

However, this illusion has a drawback: all the resources available to the user are *virtualised*, meaning there is no one-to-one correspondence between a virtual resource and a physical one [153]. Virtualisation makes distant and shared resources appear to the user as a single, coherent, and private resource cluster, fostering the illusion of a private and possibly unlimited computational cluster. Such an agglomerate of virtual resources appears to the user as a *virtual machine (VM)*, a full-stack virtual environment comprising virtualised hardware and software. Under the hood, many physical resources are shared among different VMs, implying heavy influences on the actual computational performance and scheduling issues. Different VMs can physically share sockets, memories, hard drives, and network interfaces, thus making execution times of a software variable depending on how busy the shared resources are by other users. Furthermore, many cloud providers tend to *overcommit* the available resources, i.e., allocate more virtual resources than the available physical ones [46]. This fact means that cloud clusters seem unlimited in terms of the resources available to the user, but not all the already created VMs can be instantiated simultaneously.

Due to the abovementioned design choices, cloud infrastructures are not desirable for high-performance computation or to take precise performance measurements [90]. They are much more fit for implementing service-oriented software, requiring high availability and replication [62]. Furthermore, since there is no queue waiting time, cloud infrastructure can be exploited to offload computation from the edge, alleviating low-power edge devices from heavy computations [162]. Another critical characteristic of cloud infrastructures is that they are usually geographically distributed: providers

such as Google Cloud [25], Amazon AWS [115], and Microsoft Azure [47] offer different physical deployment for their cloud services, allowing companies and developers to deploy their services across the world, reducing network communication times and providing a better user experience.

2.3.3 Edge computing

Another computing infrastructure declination is *edge computing*. This paradigm exploits the computing power at the edge of the network, directly contrasting the centralised approach used by HPC systems and taking the concept of heterogeneity and spatial distribution of cloud computing to the extreme, bringing into play new opportunities and challenges that need to be carefully handled [167]. Edge computing exploits the computational power of the uncountable number of computational devices scattered worldwide, independently from their capabilities: smartphones, wearable devices, domotic equipment, and similar devices can all be part of an edge computing system. An edge system composed of sensors and actuators capable of little computing power, interconnected with a network, is referred to as *internet of things* (IoT) [101]. When the edge computation is directly executed on the network infrastructure, such as routers and switches, this approach is also referred to as *fog computing* [42].

These kinds of infrastructure do not find their usefulness in the amount of computing power available or its availability: *pervasivity* and *responsiveness* are its most vital properties. An infrastructure composed of many edge devices communicating between themselves can quickly harvest large amounts of data directly at the source and eventually process them at the edge without needing to communicate it to a central infrastructure, coping with both privacy issues and network communication unreliability. Due to its characteristics, an edge system is not highly reliable: edge devices' network can be unstable, their available battery may not be sufficient to conclude a computation, they can be switched off, and so on.

Edge devices can be widely different, ranging from modern smartphones with a large amount of computational power and memory, maybe even with accelerators on board, always connected to fast mobile networks, to tiny, low-power sensors or wearable devices with limited capabilities. To efficiently exploit this extremely heterogeneous computing power, a *hierarchical organisation* is often adopted between edge devices, thus exploiting the tiniest devices to harvest and often preprocess data locally, offloading then more complex computations, such as AI training or inference, on more specialised and powerful hardware available in the edge system, or even to cloud or HPC infrastructures, as stated earlier. Such an approach has become increasingly popular in the last years, leading to the definition of the *computing continuum* [18], a vision in which computation flows seamlessly between different computing infrastructures, such as HPC, cloud, and edge, exploiting each one's properties at its best.

Chapter 3

Model-Agnostic Federated Learning

This chapter discussed this PhD thesis's first main scientific contribution: MAFL. The PRAISE score case study is introduced, first with an analysis of the primary statistical tools used in cardiological risk prediction and then with an analysis of the advantages of using ML models. The idea of bypassing the creation of a large data lake through a federated pooling approach is then proposed through the analysis of the PRAISE score experience. A simulative study demonstrates that using classical ML models on federated datasets can lead to results comparable to having the same algorithms applied to the whole dataset in a centralised fashion. FL is thus identified as a possible tool for designing the future's privacy-preserving medical tools (and not only). Since non-deep ML models are preferred in such a scenario due to their intrinsic characteristics, the idea of developing an FL model-agnostic methodology is introduced. The Intel® OpenFL framework is considered as a representative FL framework supporting just DNNs: its software design is analysed and extended to accommodate the AdaBoost.F algorithm, the core of the MAFL concept. *OpenFL-x* is thus proposed as the first MAFL framework, and experimental evidence of its correctness and performance are provided. A real-world study on financial crime detection conducted in collaboration with Intesa Sanpaolo is presented as a further experimental use case proving the validity of the MAFL approach. A synthetic, open-source credit card transaction dataset is used to test FL and MAFL's potential in this domain.

3.1 The PRAISE score

Management strategies for patients suffering from ACS lack an individualised approach. Such patients are at risk of many different adverse prognosis events, such as myocardial infarction and major bleeding [76]. Many tools have been developed to predict

these risks, some of which also aim at establishing the optimal duration of the dual antiplatelet therapy. Such tools are usually built on top of statistical methods like the Cox PH model, and modest results are obtained. ML can be an innovative tool that captures and models the complex, non-linear correlation between patients' data and adverse cardiological events risks. A state-of-the-art, ML-based risk-stratification model capable of predicting the risk of all-cause death, recurrent acute myocardial infarction, and major bleeding after ACS with unforeseen accuracy is built to prove this latter sentence: the PRAISE score. An in-depth literature analysis reveals that classical ML techniques are preferred over deep ML models for several different reasons. adaBoost is thus chosen as the ML algorithm for implementing this cardiological risk score. The PRAISE score model is then re-trained in a federated context, i.e. maintaining the training data separated according to the original hospitals, alleviating the issue of creating a data lake by collecting data from multiple parties. It resulted in a new experimental trade-off between patients' privacy, the medical effort to collect data and make agreements between different institutions, and the final ML model performance.

3.1.1 Statistical prediction in cardiology

From a statistical point of view, the tools allowing to predict the occurrence of an event over time are collected under the *survival analysis* branch. More specifically, a state-of-the-art review on cardiological risk scores [13] highlighted that basically all of those built on top of survival analysis techniques rely on the Cox PH model [51]. This model is based on the concept of *histantaneous hazard rate* ($\lambda(t)$), that is the rate at which events occur given the total population:

$$\lambda(t) = lim_{\Delta x \to 0} \frac{Ev(t, t+\Delta t)/N(t)}{\Delta t}$$

where $Ev(t, t+\Delta t)$ is the number of events occurring between t and $t+\Delta t$ and $N(t)$ is the number of individual at risk at time t. To compare the risks in two different populations (for example, the control population and the population undergoing a treatment), the *hazard ratio* (HR) between the two populations' instantaneous hazard rate:

$$HR = \frac{\lambda_1(t)}{\lambda_2(t)}$$

If $HR > 1$, then population 1 is at higher risk than population 2 and vice versa, with the magnitude of HR measuring this difference in risk. Given these definitions, the Cox PH model is defined as:

$$\lambda(t|X_i) = \lambda_0(t) \exp(X_i \beta)$$

where X_i is the ith individual's risk factors values vector (usually referred to as covariates in this context), β is the regression coefficients vector, and λ_0 is the baseline hazard supposing all risk factors are 0. The values of the β coefficients establish the impact of the risk factors on the population survival probability.

Many risk scores declined this simple yet effective model to grasp increasingly more complex patterns and relations in cardiological data, such as the Heart Failure Survival Score [1], the Seattle Heart Failure Model [105], ORBIT [129], PARIS [17], and PRECISE-DAPT [50]. However, all these risk scores are limited by the inherent assumptions of the Cox PH model, which are:

- the survival capability of an individual is independent of the other individuals in the population;
- the risk factors and the hazard are multiplicatively related (i.e., incrementing one of the risks multiplies the hazard);
- the HR over time is constant.

Different approaches try to cope with these assumptions differently, but overcoming them requires a radical model change. The Cox PH model, despite being widely used and effective, may as thus have reached its limit.

3.1.2 Machine learning prediction in cardiology

ML techniques can harvest the complex non-linear relationship in vast amounts of data with fewer underlying assumptions than the Cox PH model. Furthermore, ML is a vast world offering many models, each with its particular pros and cons; thus, ML results in a flexible and powerful tool capable of adapting to many different scenarios. This flexibility comes with different issues, though: choosing the best model and the corresponding hyperparameters for any given situation is not trivial since no model is inherently better than any other (no-free-launch theorem) [173], and different models bring with themselves different assumption and properties that may or may not be acceptable in different scenarios.

In the state-of-the-art review presented by the author [13], these concepts emerge clearly: out of the 44 papers selected by the study, only 12 described a cardiological risk score based on the Cox PH model. The remaining 32 papers focused on ML-based techniques, exploiting 16 different algorithms. These models range from simple linear and logistic regressions to complex DNN, from single models to model ensembles. Most notably, 9 algorithms over the 16 found in the selected literature are ensemble models, i.e. collections of individual ML models: random forest, adaBoost, gradient boosting, eXtreme gradient boosting, light gradient boosting, logitBoost, gradient boosting of decision trees, explainable boosting machines, and catBoost. The remaining 7 single models identified are logistic regression, SVMs, naïve Bayes, KNN, decision trees, Lasso logistic regression, and DNNs.

As can be seen from Figure 3.1, logistic regression is the most popular approach adopted in the literature, immediately followed by random forest. Despite being very different models, this trend can be interpreted as a research effort to model the available

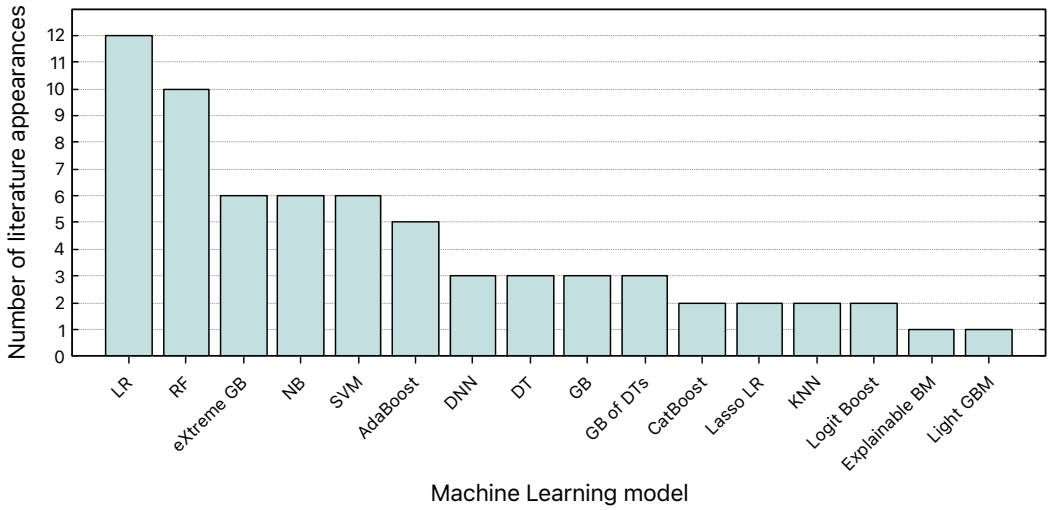


Figure 3.1: Number of ML models appearances in the cardiological risk score literature (up to 2022). ML models' names are abbreviated for convenience: logistic regression (LR), random forest (RF), gradient boosting (GB), naïve Bayes (NB), support vector machine (SVM), adaptive boosting (adaBoost), deep neural network (DNN), decision tree (DT), KNN (KNN), Boosting Machine (BM), Gradient BM (GBM).

data with the least complex model possible. This strategy allows obtaining a high *interpretability*-complexity ratio, ensuring that the model learns and generalises the principal relations present in the data without losing learning power on spurious correlations and noise. Interpretability is fundamental in medical applications since ML-based tools aim to support medical diagnosis, not to substitute physicians: their output should as thus be as interpretable as possible, and their decisional process should be inspectable and, eventually, *explainable*.

Interpretability and explainability, despite being often used interchangeably, retain two different and specific meanings in the ML landscape. Interpretability [35] refers to the possibility of understanding how an ML model makes a prediction. This property involves understanding the model's inner workings and inspecting the learned relationship between features and outputs, allowing experts and users to understand the decisional process clearly. Explainability [31], on the other hand, refers to the capability of an ML model to explain to the user why a specific prediction has been made. This property is especially desirable when working with high-complexity ML models, such as DNNs, since their predictions are very accurate but their decisional process is not always clear or transparent, resulting in the so-called *black-box* behaviour. Thus, interpretability and explainability are two different and complementary properties that an ML model trained on medical data should possess to be trustable and usable in practice.

Another insight that can be inferred from Figure 3.1 is the interest in *ensemble* ML

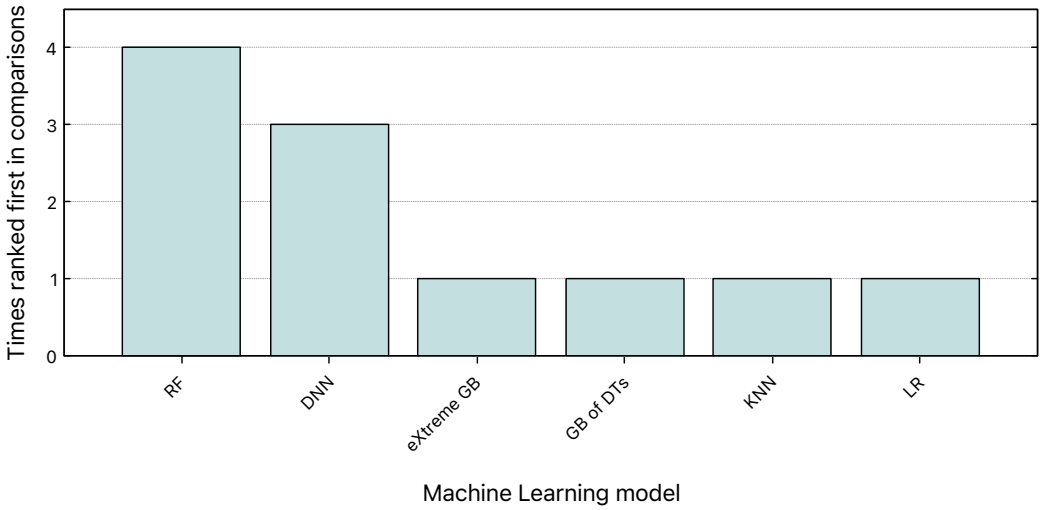


Figure 3.2: Number of times each ML model ranked first in a comparison against other ML models in the cardiological risk score literature (up to 2022). ML models' names are abbreviated for convenience: random forest (RF), deep neural network (DNN), gradient boosting (GB), decision tree (DT), k-nearest-neighbours (KNN), logistic regression (LR).

techniques: the attempts to exploit such methodology are more numerous than the ones trying to exploit DNNs. This phenomenon can be due to ensemble learning allowing the exploitation of multiple simple and interpretable ML models, such as decision trees, to build a higher-performance model, achieving interpretability and predictive power simultaneously. More specifically, it can be seen that *boosting* is investigated under many different declinations, indicating a shared trust in the capabilities and properties of this ML technique.

Figure 3.2 further highlights the capabilities of ensemble learning techniques in this field. Out of all the comparisons found in the literature, more than half the times, the best-performing model results in an ensemble model (4 times random forest and 2 times boosting ensembles, against 5 achieved by single models). Among the single models, DNNs achieved the highest number of winning comparisons, further proving the learning capabilities of these models. This analysis shows a strong trend towards ensemble ML models and DNNs for the cardiovascular risk scores of the future.

3.1.3 The PRAISE score approach and results

The PRAISE dataset collects 19,826 adult patients (≥18 years) who suffered an ACS and had a 1-year follow-up. 15,401 of those patients are collected from the BleeMACS registry. Their data were collected between 2003 and 2014 by 15 hospitals in North and South America, Europe, and Asia. The remaining 4,425 patients' data have been collected between 2012 and 2016 by 12 European hospitals. To further assess the predictive

capabilities of the trained models, an external validation dataset of 3,444 adult patients has been collected from four different sources. 442 patients came from the European SECURITY randomised controlled trial between 2009 and 2014. 402 patients are collected from the FRASER study and 1063 from the Prospective Registry of Acute Coronary Syndromes in Ferrara, both held in Ferrara, Italy, between 2014 and 2016. Lastly, 1537 patients are harvested from the Clinical Governance in Patients with ACS project of the Fondazione IRCSS, Policlinico S. Matteo, Pavia, Italy, held between 2015 and 2019. As can be seen, the available data is highly heterogeneous, including patients from all over the world and data collected over a long period.

The PRAISE dataset obtained after many data-preprocessing iterations is composed by *25 features*: 16 clinical variables (age, sex, diabetes, hypertension, hyperlipidaemia, peripheral artery disease, estimated glomerular filtration rate (EGFR), previous myocardial infarction, previous percutaneous coronary intervention, previous coronary artery bypass graft, previous stroke, previous bleeding, malignancy, ST-segment elevation myocardial infarction (STEMI) presentation, haemoglobin, and left ventricular ejection fraction (LVEF)), five therapeutic variables (treatment with β blockers, angiotensin-converting enzyme inhibitors or angiotensin-receptor blockers, statins, oral anticoagulation, and proton-pump inhibitors), two angiographic variables (multivessel disease and complete revascularisation), and two procedural variables (vascular access and percutaneous coronary intervention with drug-eluting stent). The PRAISE dataset is split into training (80%) and evaluation (20%) subsets. The nomenclature 'evaluation set' is used since it is more prevalent in the medical environment, but it is equivalent to the 'test set' expression commonly used in ML. Conversely, the external validation dataset is converted to the same feature schema as the one presented above but is not split into any subset.

Three different outcomes are selected as prediction targets: *all-cause death*, *recurrent acute myocardial infarction (ReAMI)*, and *bleeding academic research consortium major bleeding (BARC-MB)*. All outcomes refer to one year after the follow-up and are binary: the adverse event either happened (1) or not (0) during the first year after the follow-up. The PRAISE dataset's rich information and the variety of the identified prediction targets require adequate learning power to be investigated and exploited efficiently. To this end, four ML models have been selected according to the insights discussed in Section 3.1.2: *adaBoost*, *naïve Bayes*, *KNN (KNN)*, and *random forest*. adaBoost and random forest use decision trees as weak learners to improve the final model's interpretability. DNNs have been considered possible ML models worth investigating but have been excluded from the PRAISE study due to their relatively low interpretability level (black-box behaviour). Each ML model is trained, evaluated, and validated according to each outcome, leading to a set of three trained models for each model type. Each training is run tens of times, exploring each model's hyperparameter space, and only the best-performing model for each target outcome is presented. The ML models have been evaluated and compared through *F2 score*, *AUC*, and *calibration plot* values. The F2 score is preferred over the standard F1 due to the highly unbalanced dataset

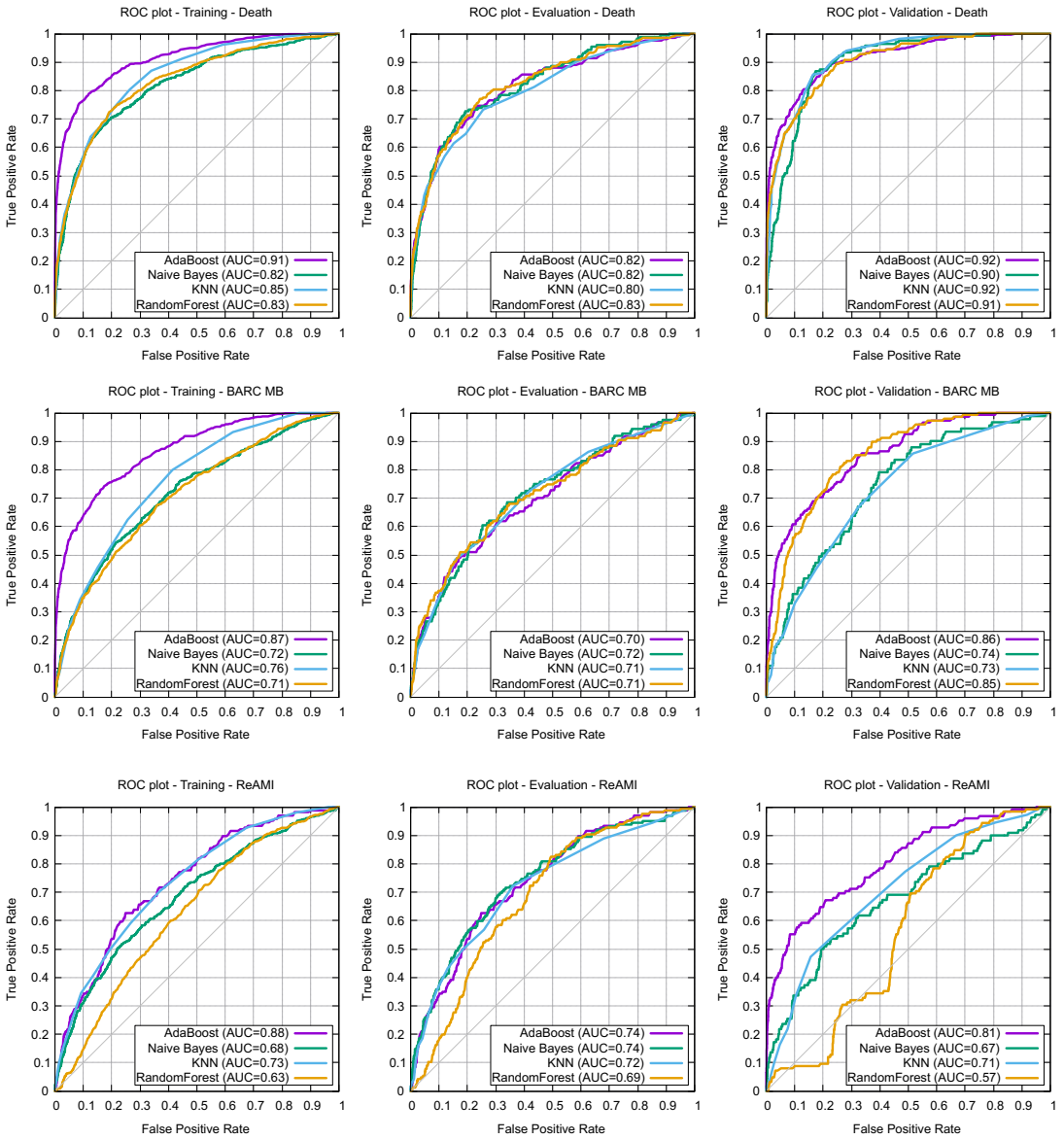


Figure 3.3: ROC curves obtained by the four tested ML models (adaBoost, naïve Bayes, KNN, random forest) on the PRAISE dataset splits (training, evaluation, validation) on the three selected outcomes (all-cause death, BARC-MB, ReAMI). AUC values are reported in the legend.

(3.3%, 3.1%, and 2.8% of all-cause death, ReAMI, and BARC-MB incidence, respectively) and because a false negative prediction is considered more harmful from the medical perspective than a false positive one.

Figure 3.3 reports the ROC curves obtained by the ML models tested on the PRAISE and validation datasets. Different prediction targets exhibit different patterns. All-cause death appears to be the most straightforward outcome to be predicted, producing smooth and relatively high AUC values for the medical field on all three subsets ($.80 < AUC < .92$). Conversely, BARC-MB and ReAMI show lower AUC values and higher differences in performance between the models ($.70 < AUC < .87$ BARC-MB, $.57 < AUC < .88$ ReAMI). These differences are exacerbated in the external validation dataset, which makes it possible to state that adaBoost seems to achieve the best and most stable performance of all the models. These properties can be observed mainly in the ReAMI validation ROC plot, where adaBoost stands out as the best-performing model while random forest exhibits particular difficulties. Thus, ReAMI results the most challenging outcome to predict among the selected ones. Looking at Figure 3.3 by columns, it can be seen that adaBoost has a clear tendency to overfit the training data more than the other models ($.88 < AUC < .91$ adaBoost, $.63 < AUC < .85$ others). Conversely, all models produce similar results on the evaluation set, with only slightly lower AUC values for the random forest ReAMI prediction ($AUC < .70$). The external validation performance results offer more room for discussion. While the peak performance for the all-cause death and BARC-MB outcomes are generally higher than the evaluation set ones, the ReAMI target exhibits a very differentiated pattern between the different models, highlighting adaBoost as the only model capable of obtaining stable and acceptable learning performance ($AUC > .80$). These behaviours highlight that the external validation dataset exhibits a different distribution than the PRAISE dataset, thus strongly testing the ML model's generalisation capabilities. Overall, the only model which successfully handled this distribution shift appears to be adaBoost since it maintains high predictive performance overall data subsets and outcomes, obtaining the highest AUC value in every presented ROC plot (apart from the all-cause death evaluation, in which random forest obtains a negligible .01 more AUC).

Table 3.1 reports many learning metrics collected during the PRAISE experiments. The F2 score is highlighted since it is taken particularly into consideration for this study, giving more weight to the recall than the precision, preferring false positives over false negatives. This behaviour is desired in the medical field since the cost of a false negative is much higher than that of a false positive; in other words, it is preferred to be extra cautious and examine more mid/low-risk patients than neglect a high-risk one. According to the F2 score, no ML model is inherently superior at predicting all-cause death: naïve Bayes obtained the highest scores on the evaluation set (.45), while KNN on the external validation set (.71), with adaBoost obtaining slightly lower values, and random forest much lower ones. The scenario completely changes when looking at the BARC-MB and ReAMI predictions. In these two cases, adaBoost obtains the highest values in both evaluation (.31 BARC-MB, .32 ReAMI) and validation (.40 BARC-MB, .39 ReAMI).

Finally, the calibration plots produced by the various ML models are examined. Figure 3.4 provides such plots as comparisons between the predicted and observed adverse

Table 3.1: Learning performance obtained by the four tested ML models (adaBoost, naïve Bayes, KNN, random forest) on the PRAISE dataset splits (training, evaluation, validation) with respect to the three selected outcomes (all-cause death, BARC-MB, ReAMI).

		adaBoost			**naïve Bayes**			**K-nearest neighbours**			**random forest**		
		Train	Evaluation	Validation	Train	Evaluation	Validation	Train	Evaluation	Validation	Train	Evaluation	Validation
Death	Threshold	.08	.08	.08	.15	.15	.15	.04	.04	.04	.08	.08	.08
	F2 score	.58	.43	.60	.43	.45	.68	.43	.42	.71	.39	.43	.57
	NPV	.98	.97	.99	.98	.98	.99	.98	.97	.98	.98	.98	.98
	PPV	.30	.23	.26	.20	.23	.33	.19	.20	.42	.17	.19	.23
	Accuracy	.92	.89	.78	.88	.88	.77	.86	.86	.84	.86	.86	.74
	Sensitivity	.76	.55	.88	.60	.60	.92	.64	.57	.86	.60	.63	.90
	Specificity	.92	.91	.77	.89	.89	.74	.87	.88	.84	.87	.87	.72
BARC-MB	Threshold	.05	.05	.05	.06	.06	.06	.03	.03	.03	.08	.08	.08
	F2 score	.45	.31	.40	.26	.25	.25	.26	.24	.24	.23	.28	.34
	NPV	.99	.98	.99	.98	.98	.98	.98	.98	.98	.97	.97	.99
	PPV	.19	.12	.14	.09	.09	.09	.08	.07	.07	.10	.12	.09
	Accuracy	.90	.87	.74	.80	.80	.80	.74	.72	.72	.88	.89	.58
	Sensitivity	.69	.49	.78	.51	.48	.48	.62	.57	.57	.34	.41	.90
	Specificity	.90	.88	.74	.81	.81	.81	.74	.73	.73	.80	.90	.56
ReAMI	Threshold	.04	.04	.04	.06	.06	.06	.04	.04	.04	.07	.07	.07
	F2 score	.43	.32	.39	.26	.32	.36	.27	.30	.38	.18	.26	.27
	NPV	.99	.98	.98	.97	.97	.95	.97	.97	.95	.97	.97	.96
	PPV	.15	.11	.13	.09	.12	.12	.10	.12	.14	.05	.07	.07
	Accuracy	.80	.78	.66	.77	.78	.57	.79	.80	.65	.59	.64	.38
	Sensitivity	.84	.58	.76	.50	.57	.69	.50	.50	.65	.54	.67	.79
	Specificity	.80	.79	.65	.78	.79	.56	.81	.81	.65	.60	.63	.35

event probability by deciles of predicted risk. Ideally, in such plots, the predicted risk should match the observed one as closely as possible, and the deciles of observed risk should increase monotonically from the first decile to the last. However, due to medical concerns, calibration plots highlighting a sharp subdivision between middle/low-risk patients (deciles 1-9) and high-risk ones (decile 10) are preferred; also, a slight overestimation of the predicted risk of high-risk patients is appreciable. From Figure 3.4, it can be deduced that random forest and KNN do not fit the abovementioned properties. adaBoost and naïve Bayes offer interesting performances, but adaBoost behaviour appears more similar to the one wanted. adaBoost calibration plots are well balanced between the predicted and observed risk, with a sharp subdivision of the 10-th decile, but lack the overestimation of high-risk patients; this property is provided by naïve Bayes, which instead tends to overestimate risk in general, which is not desirable.

Based on all the provided experimental pieces of evidence, adaBoost is thus chosen as the best-performing model on the PRAISE dataset, and the three best-performing adaBoost models trained respectively on all-cause death, BARC-MB, and ReAMI are referred to as *the PRAISE models*. To further help medical staff interpret, explain, and use the PRAISE score, Figure 3.5 analyses the most important predictors of the underlying adaBoost models for the different outcomes. These scores are normalised and consider the importance of each variable in each weak learner (i.e., their importance in the decision trees) and the respective weak learner's weight. LVEF, age, haemoglobin level, and statin therapy at discharge were the most important features to predict all-cause death.

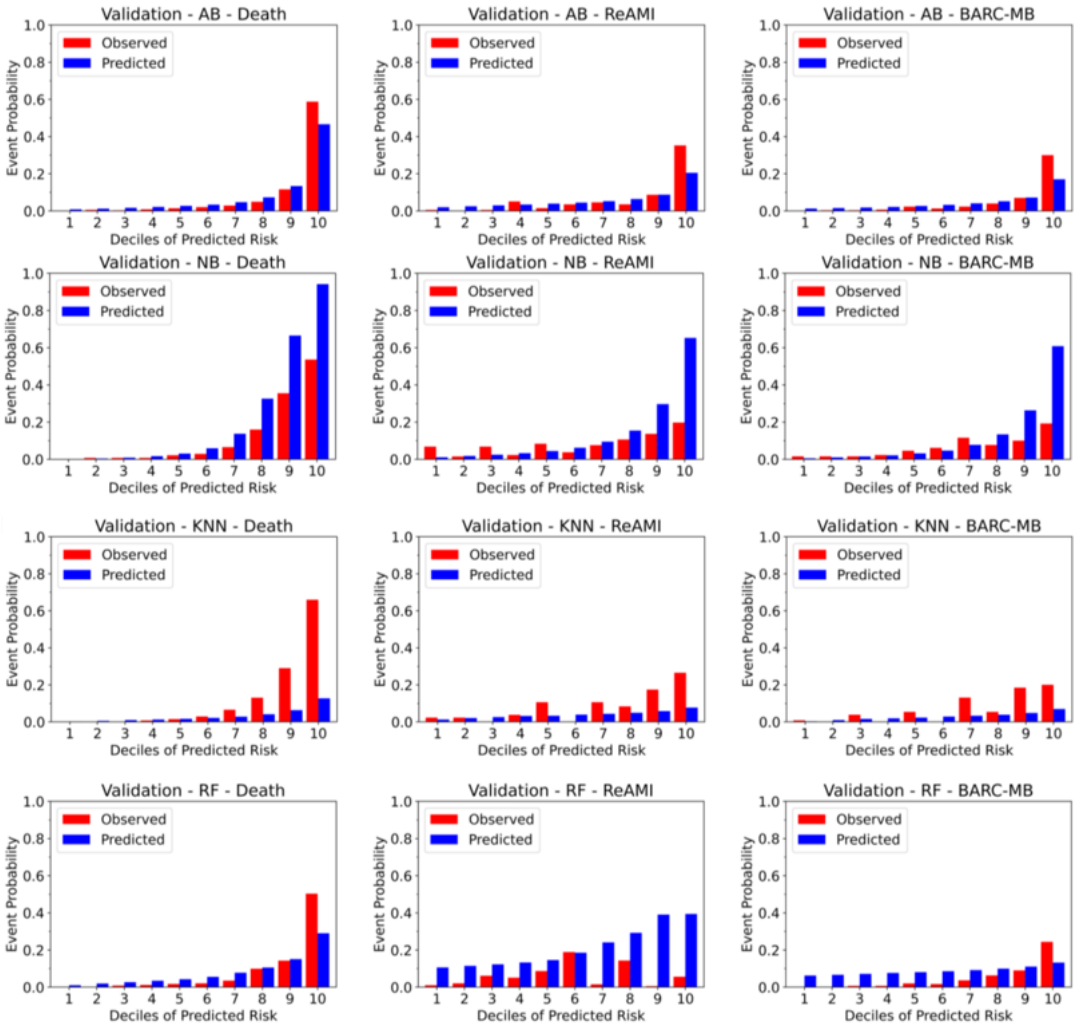
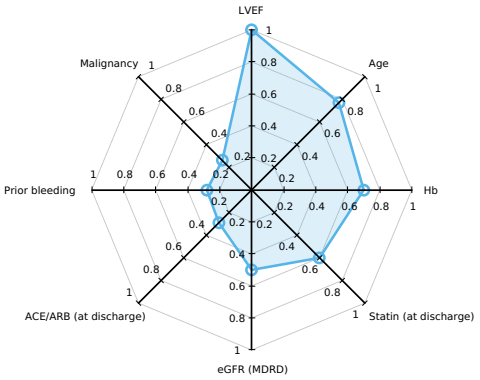
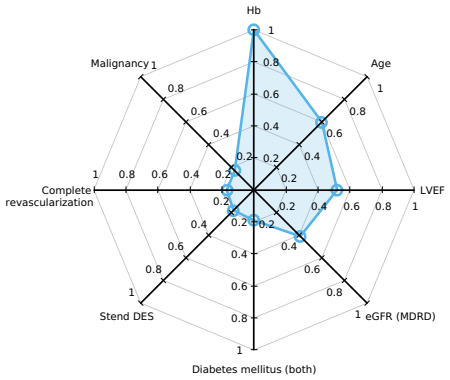


Figure 3.4: Calibration plots obtained by the four tested ML models (adaBoost, naïve Bayes, KNN, random forest) on the PRAISE dataset splits (training, evaluation, validation) on the three selected outcomes (all-cause death, BARC-MB, ReAMI).

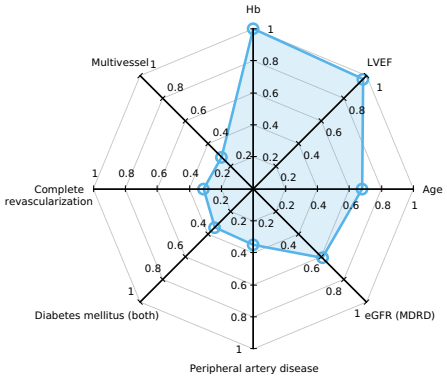
In contrast, although they have different relative importance, haemoglobin level, age, LVEF, and EGFR emerged as essential features for predicting ReAMI and BARC-MB. To further assess the PRAISE score's predictive power, additional tests are run, training and predicting all the outcomes of the evaluation and validation datasets using only the respective eight most important predictors. The obtained AUC values result only slightly lower than those obtained with all the available features (.93, .78, .87 all-cause death, .90, .62, .74 BARC-MB, .90, .68, .68 ReAMI for training, evaluation, and validation respectively). These numbers suggest that while an abundance of information can be



(a) All-cause death



(b) BARC-MB



(c) ReAMI

Figure 3.5: Radar plots reporting the PRAISE model (adaBoost) eight most important predictors for all-cause death (top left), BARC-MB (top right), and ReAMI (bottom).

helpful, having a well-selected subset of features strongly correlated with the desired outcome is a totally viable strategy for training an ML model efficiently. This strategy reduces the waste of learning power and allows for the use of simpler and lighter ML models.

The PRAISE models are available online through a web service called *the PRAISE web calculator*, allowing physicians to directly input into the web page their patients' clinical data and obtain predicted all-cause death, BARC-MB, and ReAMI risk scores in real-time. The PRAISE score web calculator is freely available online[1] and is hosted by HPC4AI, the High-Performance Computing for Artificial Intelligence cluster of the University of Turin, Italy. The PRAISE score is an officially recommended tool by the European Society of Cardiology in their 2023 ACS guidelines [33].

[1] https://praise.hpc4ai.it

3.1.4 Federated data pooling

The success and high predictive performance of the PRAISE score can be mainly attributed to the high quality of the underlying PRAISE dataset. Notably, such a dataset is one of the largest and most comprehensive ever created in the field of cardiological research. Its creation required collaboration between tens of national and international institutions, transnational agreements, and the effort to shape all different local datasets to fit into a shared and commonly designed feature schema. Such process is often referred to as *data pooling*. These efforts culminated in properly preprocessed and anonymised data, leaving the original research institutes to reach a common data lake, a *centralised* infrastructure in charge of analysing them. The data lake approach is currently the standard approach for running ML analyses, but its use may become increasingly hindered in the future. Sensible data, like medical data, is becoming increasingly more regulated and protected, and moving it from its original collecting infrastructure is also becoming harder. Newly introduced laws, such as the GDPR, impose strict conditions on sensible data sharing and processing, enforcing the data owner's rights. While protecting people's privacy and rights, these choices pose new and robust limits to the traditional centralised techniques adopted in ML research by strongly hindering data pooling processes.

Conversely, traditional centralised ML approaches are not keeping pace with the increasing ubiquity of data-harvesting devices. Data is not only collected by large, centralised institutions but pervasively. From tiny, almost invisible wearable devices to smartwatches, smartphones, tablets, and laptops to smart cars, smart homes, smart cities, and even satellites, people are immersed in an environment flooded with data-hungry devices. Such ubiquitous devices usually harvest personal and private data and, ideally, people would like such data to remain local to the device and be used just for purposes related to the device itself. It is not desirable to have such local data shared with or collected by third parties, not knowing which analyses or processes they will undergo. Thus, edge computing techniques can be deployed, easing data processing by exploiting the ubiquitous computing power available at the network's edge. This scenario opens up vast possibilities for *distributed* ML approaches, especially for those not requiring any data movement from their local infrastructure, like FL. Such an FL-based approach to distributed datasets is defined in the following *federated pooling*, in direct contraposition to traditional centralised data pooling.

While FL with DNNs is already well-known and studied, non-DNN-based FL is still in its infancy. A federated version of adaBoost is needed to reproduce the behaviour of the PRAISE model in a distributed environment. DistBoost [104] and PreWeak [32] are two distributed adaptations of adaBoost available in the literature that could be adapted to the FL environment. Polato et al. already made the effort to formalise their federated versions, respectively named DistBoost.F and PreWeak.F, and contestually proposed a

Algorithm 1: AdaBoost.F (server)

Input: C: number of clients
T: dimension of the ensamble
K: number of classes
Output: $\text{ens}(\mathbf{x}) \triangleq \text{vote}([h^{t\star}]_{t=1}^{T}, [\alpha^t]_{t=1}^{T}, \mathbf{x})$

for $t \in \{1 \dots T\}$ **do**
  $Z \leftarrow \| [\mathbf{receive}_Z(c)]_{c=1}^{C} \|_1$
  $\mathbf{h}^t \leftarrow [\mathbf{receive}_h(c)]_{c=1}^{C}$
  $\mathbf{broadcast}_{\mathbf{h}}(\mathbf{h}^t)$
  $\mathbf{E}^t \leftarrow \frac{1}{Z} [\mathbf{receive}_\epsilon(c)]_{c=1}^{C}$ ▷ $C \times C$ errors matrix
  $c^{t\star} \leftarrow \arg\min_c \sum_{c'=1}^{C} \mathbf{E}^t_{cc'}$
  $\epsilon^{t\star} \leftarrow \sum_{c=1}^{C} \mathbf{E}^t_{cc^{t\star}}$
  $\alpha^t \leftarrow \log\left(\frac{1-\epsilon^{t\star}}{\epsilon^{t\star}}\right) + \log(K-1)$
  $\mathbf{broadcast}_\alpha(\alpha^t)$
  $\mathbf{broadcast}_c(c^{t\star})$
$\mathbf{broadcast}_{\text{stop}}(\text{stop})$

new, innovative federated adaBoost version offering state-of-the-art learning performance: *AdaBoost.F* [135]. The training phase of AdaBoost.F is similar to the one of adaBoost, but it happens in a distributed manner; Algorithm 1 reports the server pseudocode, while Algorithm 2 the client's one. At each iteration t, a new weak hypothesis is learned from each client c and sent to the aggregator. The aggregator collects the weak hypotheses and broadcasts them all to all clients. The clients evaluate the received hypotheses on the local dataset and send the weighted errors vector ϵ to the aggregator, which can then aggregate these values into an errors matrix E^t. Values in E^t are then used to find the best hypothesis for the current round c^{t*} and to compute the current adaBoost coefficient α^t. By propagating these pieces of information to the clients, they can then update their local copy of the ensemble and the local examples' weights $\mathbf{d}$. Overall, the algorithm has strong resemblances with the original adaBoost algorithm; one interesting difference is that $\mathbf{d}$ is kept un-normalised in the client; this is important to make it possible to compute a global normalisation factor in the aggregator.

In the following, the PRAISE study is repeated through a federated pooling approach, investigating if the same learning performances of the PRAISE score could also be achieved through FL without requiring the patient's data to be moved from their local hospitals. To test the proposed federated pooling approach's real-world performance, traditional DNN-based FL and AdaBoost.F algorithms are run on the PRAISE dataset and compared with the PRAISE models' performances. The chosen DNN is a two-layer perceptron with 35 inputs, 35 hidden units, a single output and a binary

Algorithm 2: AdaBoost.F (client)

Input: $\mathscr{A}$: weak learner
$\mathbf{X} \in \mathbb{R}^{n \times m}$: training data
$\mathbf{y} \in \{1, \dots, K\}^n$: training labels

$\mathbf{d} \leftarrow \mathbf{1}$
while *not stop* **do**
  $\mathbf{send}_Z(\text{aggregator}, \|\mathbf{d}\|_1)$
  $h \leftarrow \mathscr{A}\left(\mathbf{X}, \mathbf{y}, \frac{\mathbf{d}}{\|\mathbf{d}\|_1}\right)$
  $\mathbf{send}_h(\text{aggregator}, h)$
  $\mathbf{h} \leftarrow \mathbf{receive}_{\mathbf{h}}(\text{aggregator})$
  $\epsilon \leftarrow \left[\mathbf{d}^\top [\![\mathbf{y} \neq h_c(\mathbf{X})]\!]\right]_{c=1}^{|\mathbf{h}|}$
  $\mathbf{send}_\epsilon(\text{aggregator}, \epsilon)$
  $\alpha^\star \leftarrow \mathbf{receive}_\alpha(\text{aggregator})$
  $c^\star \leftarrow \mathbf{receive}_c(\text{aggregator})$
  $\mathbf{d} \leftarrow \left[d_i \exp(-\alpha^\star [\![h_{c^\star}(x_i) \neq y_i]\!])\right]_{i=1}^n$

cross-entropy loss. We trained it for 100 rounds of one epoch each, using the Adam optimizer [99] ($lr = 10^{-3}$, $\beta_1 = .9$, $\beta_2 = .999$, $\epsilon = 10^{-7}$), and fedAvg as aggregation strategy. Conversely, the federated adaBoost ensemble is built by running 100 rounds of the AdaBoost.F algorithm using a decision tree with at most 10 leaves as weak learners. All FL runs have been orchestrated using Intel® OpenFL with a single aggregator and up to 16 collaborators, and both strong and weak scaling properties are tested. The strong scaling performance is tested by subdividing the entire PRAISE dataset into n i.i.d. subsets without replacement and assigning a subset to each of the n collaborators involved in the federation. This configuration keeps the same number of rows for each experimental configuration. Conversely, 16 subsets of the complete datasets are sampled and assigned to each collaborator to test the weak scaling performance. In this setting, the size of the problem increases linearly with the number of collaborators involved in the federation. Learning performances are measured on a virtualised environment on top of the OpenStack-based HPC4AI cloud infrastructure, with the aggregator running on a virtual machine with 4 cores and 8GB RAM and up to 16 collaborators hosted in virtual machines with 8 cores and 8GB RAM each. Conversely, computational times are measured on the C3S HPC facility, allocating an entire bare metal node with 2 Intel® Xeon E5-2697 sockets (18 cores, 2.30GHz) and 128GB RAM to each component of the Intel® OpenFL deployment.

Table 3.2 and Table 3.3 report the learning metrics achieved by standard FL and AdaBoost.F, respectively. AdaBoost.F dominates the accuracy metrics, achieving high values in all experiments ($.94 < accuracy < .95$). The DNN accuracies are much more erratic ($.39 < accuracy < .90$), showing a higher variance as the number of collaborators

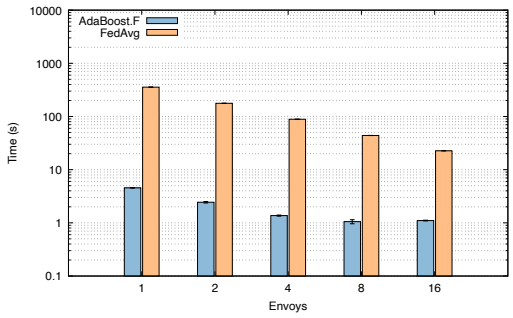
Table 3.2: Learning performance obtained by DNN-based FL on the PRAISE dataset up to 16 clients. The reported values are the average of 5 runs ± the standard deviation. The strong scaling setting with a single client is equivalent to the non-federated case.

	Clients	Accuracy	F1 Score	F2 Score	Precision	Recall
Strong scal.	1	.39 ± .47	.14 ± .08	.22 ± .04	.17 ± .09	.72 ± .39
	2	.56 ± .47	.19 ± .09	.26 ± .06	.15 ± .09	.61 ± .36
	4	.88 ± .01	.23 ± .01	.30 ± .01	.17 ± .01	.39 ± .02
	8	.72 ± .38	.20 ± .06	.27 ± .04	.16 ± .06	.48 ± .29
	16	.90 ± .01	.24 ± .01	.29 ± .01	.12 ± .01	.35 ± .02
Weak scal.	1	.56 ± .47	.16 ± .07	.22 ± .03	.12 ± .07	.56 ± .40
	2	.69 ± .37	.17 ± .05	.25 ± .04	.12 ± .06	.49 ± .30
	4	.72 ± .38	.20 ± .07	.27 ± .05	.15 ± .06	.49 ± .29
	8	.90 ± .04	.18 ± .10	.24 ± .13	.13 ± .08	.30 ± .17
	16	.55 ± .46	.17 ± .08	.26 ± .06	.11 ± .06	.63 ± .34

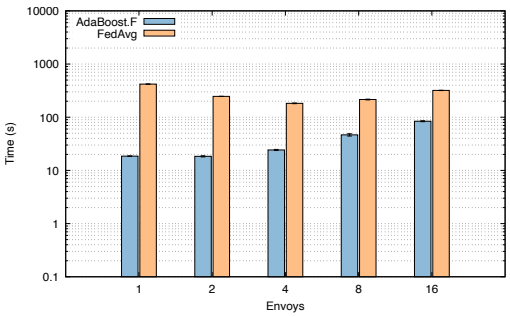
Table 3.3: Learning performance obtained by decision tree-based AdaBoost.F on the PRAISE dataset up to 16 clients. The reported values are the average of 5 runs ± the standard deviation. The strong scaling setting with a single client is equivalent to the non-federated case.

	Clients	Accuracy	F1 Score	F2 Score	Precision	Recall
Strong scal.	1	.95 ± .00	.19 ± .07	.15 ± .06	.35 ± .10	.13 ± .05
	2	.95 ± .00	.23 ± .03	.19 ± .03	.36 ± .04	.17 ± .03
	4	.94 ± .00	.19 ± .02	.16 ± .02	.26 ± .04	.15 ± .02
	8	.94 ± .00	.20 ± .04	.17 ± .03	.28 ± .06	.16 ± .03
	16	.94 ± .00	.19 ± .03	.17 ± .03	.25 ± .04	.16 ± .03
Weak scal.	1	.95 ± .00	.09 ± .02	.06 ± .01	.33 ± .05	.05 ± .01
	2	.95 ± .00	.10 ± .02	.07 ± .01	.45 ± .05	.05 ± .01
	4	.95 ± .00	.15 ± .04	.12 ± .04	.32 ± .06	.10 ± .10
	8	.95 ± .00	.17 ± .02	.14 ± .01	.28 ± .04	.13 ± .01
	16	.94 ± .00	.20 ± .03	.18 ± .02	.27 ± .04	.16 ± .02

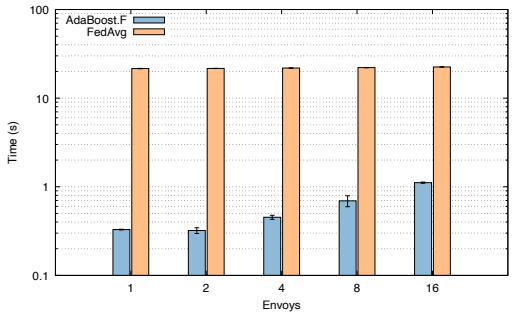
grows and a higher variance in the 5 experiment repetitions. However, accuracy should be used carefully when dealing with highly unbalanced datasets such as this, and these high values suggest that the ensemble model categorises most of the examples as the majority class (negatives). The remaining metrics confirm this intuition: AdaBoost.F recall values ($.50 < recall < .17$) are much lower than the DNN ones ($.30 < recall < .72$). However, DNN F1 and F2 scores ($.14 < F1 < .24, .22 < F2 < .30$) result much better



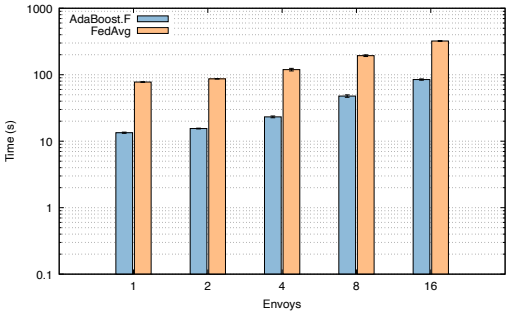
(a) Strong scaling - single client computation time



(b) Strong scaling - total execution time



(c) Weak scaling - single client computation time



(d) Weak scaling - total execution time

Figure 3.6: Strong (top) and weak (bottom) scaling wallclock time performance obtained by AdaBoost.F (decision trees) and FL (DNNs) training for 100 rounds on the PRAISE dataset. Experiments executed on the C3S HPC infrastructure.

than the AdaBoost.F ones ($.09 < F1 < .23, .07 < F2 < .19$). Indeed, results reported in Table 3.2 are even better than those shown in the original PRAISE score paper. However, it is not possible to state that these models are better than the PRAISE model since they have not been thoroughly evaluated, examined, and validated. Another interesting facet of the reported DNN results is that the F metrics do not follow a monotonically growing pattern in the number of collaborators: they show an inverted v shape. At least in the weak scaling setting, the performances were expected to continue growing due to the increase in data provided by the additional collaborators. A possible explanation might be that FL hardly leverages all the available data when the number of involved parties grows, almost to the point of making adding new parties to the federation no longer worthwhile if their local data is not a significant amount.

Figure 3.6 shows the execution times of 100 training rounds for federated DNN and AdaBoost.F in the strong and weak scaling settings, respectively. The strong scaling

setting with a single envoy, identical to a non-federated process, is considered a baseline. Training a DNN with fedAvg is between 5 and 30 times longer than training an ensemble of decision trees with AdaBoost.F in both settings. However, this gap is much more evident when considering only the actual computation time, as the overhead introduced by serialisation and communication is much more evident with AdaBoost.F. Plus, the communication time appears to be the actual bottleneck in the overall execution, as the total training time increases with the number of federation members. When analyzing the two algorithms' strong and weak scaling behaviour, it is worth noting that the DNN follows a common trend in both settings. Indeed, the total time to solution decreases with more collaborators in the strong scaling setting, while it remains almost constant in the weak scaling one. Conversely, the time to solution decreases only up to 8 collaborators in the strong scaling setting with AdaBoost.F, and it linearly grows up in the weak scaling setting. This behaviour is justified because the second phase of the algorithm requires each collaborator to evaluate n decision trees on the local data to determine the best one, where n is the number of collaborators in the federation. This finding suggests that the benefit of using AdaBoost.F will be much more evident with a more efficient, high-performance FL framework; also, standard FL would witness better scaling and computational performance if current FL frameworks could better target the main computational issues of this approach.

3.2 OpenFL-extended

Having proven that FL is a suitable path for the future development of distributed, efficient, and privacy-preserving ML techniques, it is now time to analyse the technical computer science challenges underlying it. More specifically, an off-the-shelf, open-source, industry-grade FL framework, Intel® OpenFL, is considered. Intel® OpenFL is deconstructed and analysed from a parallel and distributed computing point of view, and some internal issues are improved and corrected. This software engineering effort formed the basis for re-structure part of Intel® OpenFL to accommodate the AdaBoost.F algorithm, making Intel® OpenFL officially the first *model-agnostic* FL framework on the market. The produced software, *OpenFL-extended* (*OpenFL-x*), is then experimentally tested to prove both the AdaBoost.F algorithm implementation correctness and the framework computational performance. All this work is done in collaboration with the Intel® OpenFL official developer team, confirming the industrial interest in this project and its impact on the FL community. *OpenFL-x* is open-source and freely available on GitHub[2].

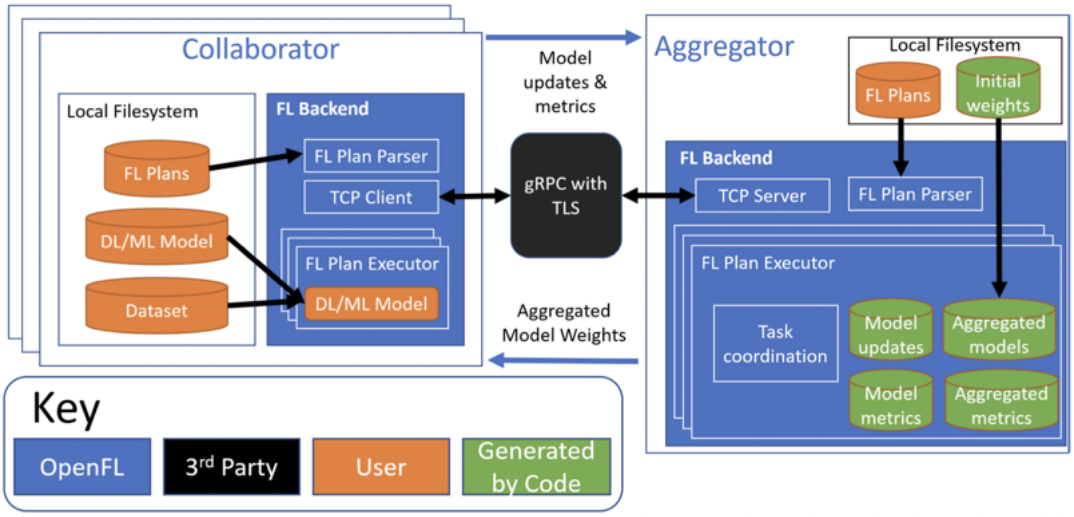


Figure 3.7: Intel® OpenFL core software architecture. Higher-level software components that allow more structured and durable federations are omitted. The internal framework's components (depicted in blue) are the target to be modified for the AdaBoost.F implementation.

3.2.1 Intel® OpenFL software architecture

Intel® OpenFL is an open-source FL framework developed by Intel® Labs and Intel® Internet of Things Group. It is a project hosted by the Linux Foundation that aims to be community-driven software constantly improved by users' contributions. Intel® OpenFL is entirely developed in Python 3 and relies on gRPC and protobuf for its communications. It is DL-framework-agnostic, supporting any DL library available through a plugin mechanism. Intel® OpenFL aims to support long-lasting federations by creating an overlay network between the participant to the federation; such a distributed service is then in charge of scheduling and handling the execution of the different FL experiments pushed to the system's queue. Underlying these long-lived components, the core classes of Intel® OpenFL run the actual FL experiment.

As depicted in Figure 3.7, the `Collaborator` and `Aggregator` classes are the main actors instantiated by OpenFL, which then exploit all the other software components. These two classes create a client-server structure, where a single `Aggregator` orchestrates the FL workload on the `Collaborators` connected to it. Each `Collaborator` at start-up time tries to connect to its designated `Aggregator`; if it is not found, the `Collaborator` shuts down. These connections are TCP-based and are supposed to remain active for the whole duration of the FL process. In this sense, Intel® OpenFL is

[2] https://github.com/alpha-unito/Model-Agnostic-FL

a connection-oriented software, exposing minimal functionalities for handling disconnected and straggler `Collaborators`. Once the `Aggregator` and all the `Collaborators` are correctly set up and connected, the FL experiment execution can start.

The FL experiment is described entirely by the user and is the only part of the framework that the user is supposed to write/modify him/herself. Figure 3.7 represents such software components in orange and specifies their location on the local file system, suggesting user ownership. The `DL/ML Model` and `Dataset` specification are usually given in one Python file, named in the following *experiment file*, together with the definition of the train and test functions, data preprocessing, metrics collection, et similia. As such, the experiment file defines all the traditional ML algorithms and hyperparameters of the FL process. All these specifications will then be injected into the FL process in the right places through the `FL Plan`, a YAML file describing the actual FL process. This configuration file specifies a wide range of information guiding the FL process concretely, such as which classes to instantiate as `Aggregator` and `Collaborator`, which path to use to save the trained models, which hyperparameters to use, networking parameters, which tasks to accomplish, and many others. The standard Intel® OpenFL plans support only three different tasks:

- `aggregated_model_validation`: test set validation of aggregated model;
- `train`: local training of the model;
- `locally_tuned_model_validation`: test set validation of local model.

Each task is a complex field, allowing the user to specify which function to use to realise it; this is how Intel® OpenFL allows the user to inject custom Python code into the framework to customise the FL computation. The specified tasks are then executed cyclically for the number of specified federated rounds.

The inner working classes of OpenFL, represented in blue in Figure 3.7, are then in charge of computing the FL workload. Such software components are numerous and complex; here, only a high-level perspective of the resulting workflow is given. The first step in the Intel® OpenFL computation is the `FL Plan` parsing, allowing the user's specifications and custom code to percolate into the framework. Then, the `Aggregator` instantiate the `DL/ML Model` *initial weights*; such values can be randomly generated at runtime or loaded from the local file system by adequately configuring the `FL Plan`. Such weights are then serialised and sent to each `Collaborator` that can instantiate a local `DL/ML Model`. Each `Collaborator` execute the tasks specified in the `FL Plan` iteratively for the number of specified federated rounds. At the end of each task, its final result is communicated back to the server. Such communication can contain performance metrics if the task runs evaluation workloads or a new set of `DL/ML Model` weights if it runs a training workload. Notice that each time weights are exchanges, the `DL/ML Model` are locally re-instantiated. The `Aggregator` takes care of synchronising all `Collaborators` at the end of each task, collecting the global performance metrics, and concretely running the aggregation of the `DL/ML Models`. Both `Aggregator`

and `Collaborators` keep track of the collected metrics and the trained models locally through a Pandas database. The stored information is represented in green in Figure 3.7.

All the communications in the framework are handled through the RPC model, specifically with the gRPC library, represented in black in Figure 3.7. This communication model is implemented by having a passive `Aggregator`, which methods are remotely invocated by the `Collaborators`. The sequence in which the `Aggregator` methods are invocated is then established by the `Collaborators`, which themselves are directed by the tasks sequence specified in the `FL Plan`. Such gRPC calls require the exchange of the functions' parameters and the methods' results; these objects are serialised before being sent through the *protobuf* library. Since this exchanged data can be subject to interception or sniffing, communications can be TLS-encrypted.

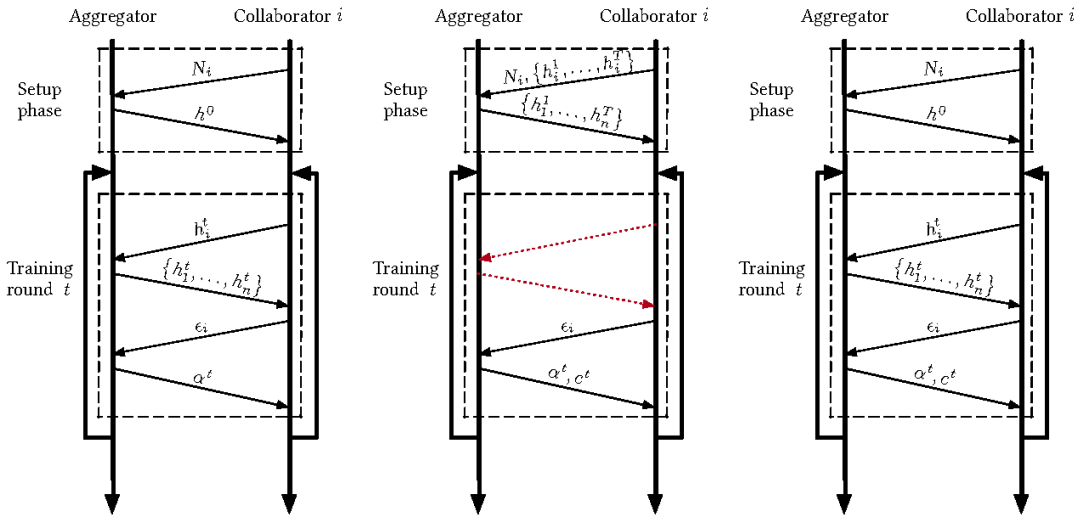
The high-level software components are now briefly introduced. To build long-lasting federations, `Aggregator` and `Collaborator` are abstracted into distributed services, named `Director` and `Envoy`. Such software components are supposed to be started on the respective institutions and run as services. The user is then able to submit to the `Director` one or more FL experiments through a Python API; the `Director` takes then care of instantiating the necessary `Aggregator` and `Collaborators`, dispatching the experiments, and collecting the results.

3.2.2 Model-agnostic federated learning

Intel® OpenFL does not support non-DNN models by default, as almost all of the FL framework discussed in Section 2.2.3. The few that do, support only a few special cases, such as gradient-boosting decision trees or other gradient-based techniques. No FL framework available on the market can be defined as *model-agnostic*, i.e., capable of supporting an FL workload independently from the ML trained locally. The reason for this is twofold. On the one hand, modern FL frameworks still try to achieve sufficient technical maturity rather than add new experimental functionalities. On the other hand, model-agnostic federated algorithms are still new and little investigated, and their efficacy has yet to be comprehensively proven. This effort of bringing AdaBoost.F to a stable, industry-grade, open-source FL framework such as Intel® OpenFL thus results also in the creation of the first freely available *Model-Agnostic* FL framework available for research purposes, since AdaBoost.F works independently from the underlying ML model chosen as a weak learner.

Analysing more in-depth the three federated adaBoost algorithms proposed in [135] and briefly discussed in Section 3.1.4 (namely DistBoost.F, PreWeak.F, AdaBoost.F), a common underlying communication protocol can be extracted. The result of this analysis can be observed in Figure 3.8, and can be summarized as:

1. The aggregator receives the dataset size N from each collaborator and sends them an initial version of the weak hypothesis.



(a) DistBoost.F (b) PreWeak.F (c) AdaBoost.F

Figure 3.8: Graphical representation of the DistBoost.F, PreWeak.F, and AdaBoost.F protocols. N is the dataset size, T is the number of training rounds, h the weak hypothesis, ϵ the classification error, α the adaBoost coefficient. The subscript $i \in [1, n]$ indices the collaborators and the superscript t the training rounds (with 0 standing for an untrained weak hypothesis). $c \in [1, n]$ is the index of the best weak hypothesis in the hypothesis space. The red dotted line in PreWeak.F highlights the absence of communication.

2. The aggregator receives the weak hypothesis h_i from each collaborator and broadcasts the entire hypothesis space to every collaborator.

3. The errors ϵ committed by the global weak hypothesis on the local data are calculated by each client and sent to the aggregator.

4. The aggregator exploits the error information to select the best weak hypothesis c, adds it to the global strong hypothesis and sends the calculated adaBoost coefficient α to the collaborators.

Note that N is needed to adequately weigh the errors committed by the global weak hypothesis on the local data, allowing to compute α correctly.

These generic model-agnostic federated protocols are more complex than the standard FL one. One more communication for each round and the exchange of complex objects across the network (the weak hypotheses) are required, strongly impacting the computational performance. Note that each arrow going from collaborator i to the aggregator in Figure 3.8 implies a synchronisation barrier among all the collaborators in the federation. Increasing the number of global synchronisation points reduces concurrency and increases the sensitivity to stragglers. Once an FL framework handles the common protocol structure, implementing any of the three algorithms requires the

same effort. Thus, implementing AdaBoost.F into Intel® OpenFL not only creates the first MAFL framework on the market but also opens the door to experimenting with other MAFL algorithms.

Different MAFL algorithms provide different solutions to the same FL problem. Despite being similar once abstracted from their low-level details, the three MAFL algorithms reported in Figure 3.8 all explore a different hypothesis space. While step 1 is inherently a setup step, DistBoost.F and AdaBoost.F repeat steps 2-4 cyclically; PreWeak.F instead fuses steps 1 and 2 at setup time, receiving from each collaborator T instances of already trained weak hypotheses (one for each training round) and broadcasting $n \times T$ models to the federation. Then, each federated round t loops only on steps 3 and 4 due to the different *hypothesis space* the algorithms explore. While DistBoost.F and AdaBoost.F create a weak hypothesis during each federated round, PreWeak.F creates the whole hypothesis space during step 2 and then searches it for the best solution. All three algorithms produce the same strong hypothesis and adaBoost model, but they differ in the selection of the best weak hypothesis at each round:

- DistBoost.F uses a committee of weak hypotheses;
- PreWeak.F uses the weak hypotheses from a fully trained adaBoost model;
- AdaBoost.F uses the best weak hypothesis trained in the current round.

This is just one example of how opening the FL software environment to adopt more flexible communication protocols and more powerful serialisation techniques would allow ML practitioners to find innovative solutions to already-known ML problems.

3.2.3 OpenFL-extended implementation

Redesigning Intel® OpenFL comprises two main goals: allowing more flexible protocol management and making the whole infrastructure model-agnostic. These operations require modification of the core classes of OpenFL. All the changes to the original software explored in the following are made in the least invasive way possible, trying to respect the original Intel® OpenFL design principles. This choice implies that the software components most subject to modifications are those depicted in blue in Figure 3.7, i.e., the core Intel® OpenFL classes. Minor fixes are also applied to the orange elements, i.e., the user interface, to allow the user to personalise the framework workflow completely. Such changes make it possible to use Intel® OpenFL with its standard behaviour, i.e., standard FL, or to use the newly introduced AdaBoost.F algorithm with any ML model as a weak learner. The obtained software, *OpenFL-x*, is thus both FL and MAFL compliant.

The `FL Plan` directs the Intel® OpenFL run time. The original Intel® OpenFL Plan is rather primitive in its functions. It is not entirely customisable by the user, and many of its fields are overwritten at run time with default values. Due to its unused power, the parsing of the plan file has been extended and empowered, making it capable of

handling new types of tasks and a higher range of arguments (and also making it evaluate *every* parameter in the file). The new model-agnostic workflow can be triggered by specifying the `nn: False` argument under the `Aggregator` and `Collaborator` fields. The specific steps of the protocol can then be listed in the `tasks` section. In *OpenFL-x*, the tasks vocabulary comprises three additional tasks:

- `weak_learners_validate`: test set validation of the weak hypothesis;
- `adaboost_update`: update of the global parameters of AdaBoost.F on the `Collaborators` and the ensemble model on the `Aggregator`;
- `adaboost_validate`: local test set validation of the aggregated AdaBoost.F model.

The `weak_learners_validate` task is similar to `aggregated_model_validation`; however, it returns additional information for AdaBoost.F, such as which samples are correctly predicted/mispredicted and the norm of the samples' weights. This extended set of tasks allows the users to use new MAFL algorithms, such as AdaBoost.F. Additionally, if the `adaboost_update` task is omitted, it is possible to obtain a simple *Federated Bagging* behaviour. Switching behaviour requires small actions other than changing the Plan; however, both functionalities are documented with tutorials in the code repository.

The communications between the `Aggregator` and the `Collaborators` are revised to allow these new tasks to be executed correctly. New messages are implemented into the original *communication protocol*, allowing data exchange other than ML/DL models and performance metrics since AdaBoost.F relies on exchanging locally calculated parameters. Furthermore, Intel® OpenFL only implements two synchronisation points in its original workflow: one at the end of the federation round and one when the `Collaborator` asks the `Aggregator` for the aggregated model. These synchronisation points are hard-coded into the software and cannot be generalised for other uses. For the AdaBoost.F workflow, a more general synchronisation point is needed: each task specified in the `FL Plan` requires a synchronisation point at its end since each task depends on the previous one. AdaBoost.F thus requires more synchronisation points than standard FL. A new `synch` message is thus added to the original *gRPC* protocol. The working mechanism of this synchronisation point is straightforward: each `Collaborator` asks for a `synch` at the end of each task, and if all `Collaborators` have not finished the current task, then it waits a fixed amount of time before trying the synchronisation again; otherwise, it is allowed to continue to the next task. This solution, even if not highly efficient, respects the Intel® OpenFL internal synchronisation mechanisms and does not require any disruptive redesigning of the original software.

Switching to the central core classes of Intel® OpenFL, both `Aggregator` and `Collaborator` are revised to allow their behaviour to adapt to both DL and ML models and to handle correctly the new tasks specifiable in the `FL Plan`. The `Collaborator` class now offers different behaviours according to the ML model used in the computation. Suppose that the `FL Plan` specifies that the training will not involve DNNs: the

`Collaborator` will actively keep track of the parameters necessary to the AdaBoost.F algorithm, like the mispredicted examples, the weight associated with each data sample, and the weighted error committed by the models. In this case, the internal database changes policies, changing tags and names associated with the entries to make possible finer requests. Counterwise, if the `FL Plan` specifies a standard FL behaviour, the `Collaborator` falls back to its original behaviour, maintaining backward compatibility. On the other hand, the `Aggregator` is extended to handle any `DL/ML Model` (instead of only DNNs weights), aggregation functions instantiated dynamically from the `FL Plan` file, and the synchronisation needed at the end of each task. New methods allow the `Aggregator` to query the internal database more finely, thus allowing it to read and write the `DL/ML Model` with the same tags and name as the `Collaborator`, maintaining coherency between the two entities and allowing correct communication when exchanging stored values.

`TensorDB`, the internal class used for storage, is modified to accommodate the new behaviours described above. This class implements a simple *Pandas* data frame responsible for model storage and retrieving done by the `Aggregator` and `Collaborator`. Furthermore, its `clean_up` method has been revised, making it possible to maintain a fixed amount of data in memory. This fix has an essential effect on the computational performance since the `TensorDB` query performance is directly proportional to the amount of data it contains, and, in the standard Intel® OpenFL, the dataset size grows linearly in the number of federated rounds.

Finally, the more high-level and interactive classes, namely `Director` and `Envoy`, and the serialisation library are updated to work correctly with the new underlying code base. This effort results in a model-agnostic FL framework that supports the standard DNNs-based FL workflow and the new adaBoost.F algorithm. Using the software in one mode or another does not require any additional programming effort from the user; a few simple configuration instructions are enough. Additionally, the installation procedure has been updated to incorporate all new module dependencies of the software. The provided *OpenFL-x* repository provides a complete set of tutorials, making it easy for any developer to get started with this experimental software and reproduce the proposed experimental results.

Using traditional ML models instead of DNNs drastically reduces the computational load that Intel® OpenFL has to sustain. Moreover, AdaBoost.F requires one additional communication phase per round compared to standard FL. This fact exacerbates the impact of time spent in communication and synchronisation on the overall system performance. Many optimisations are implemented during the extension of Intel® OpenFL to reduce such communication impact, and their impact on the system overhead is now analysed. As depicted in Figure 3.9, optimised *OpenFL-x* achieves a 5.5x speedup on a simple FL use-case compared with its not optimised version. A 10-leave decision tree is trained on the `adult` dataset over 100 federated rounds using 9 nodes (1 aggregator plus 8 collaborators) as a baseline workload. Physical machines are used to obtain stable and

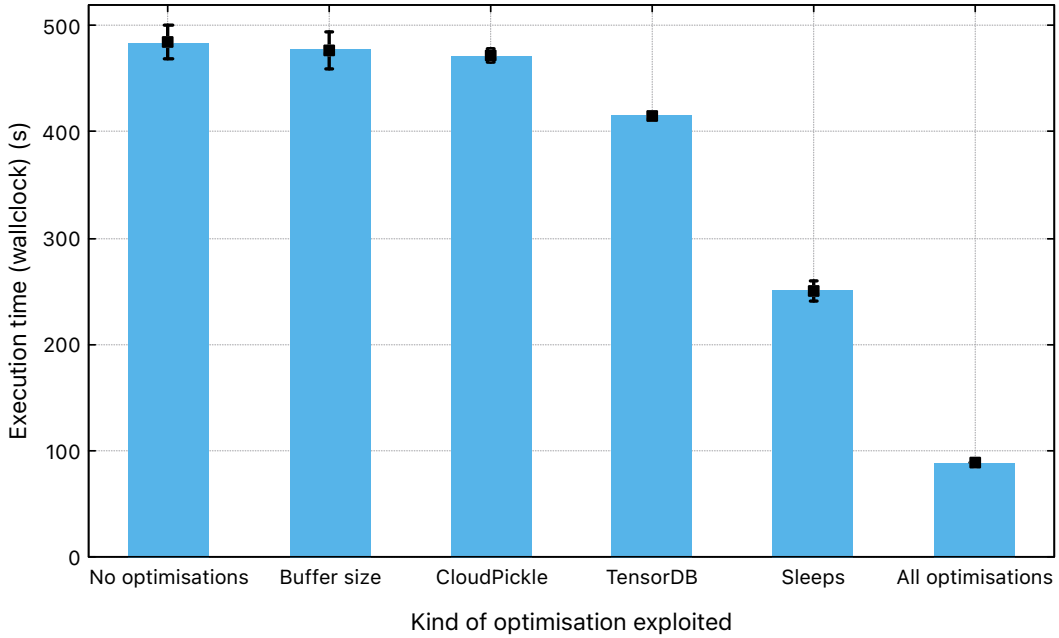


Figure 3.9: Ablation study of the *OpenFL-x* optimisations implemented. The test federation comprehends 8 collaborators trained on the IID split `adult` dataset for 100 rounds. The 95% CI has been obtained over 5 executions.

reliable computing times, as execution times on bare-metal nodes are more deterministic than on cloud infrastructures. Each HPC node is equipped with two 18-core Intel® Xeon E5-2697 v4 @2.30 GHz and 126 GB of RAM, and a 100Gb/s Intel® Omni-Path network interface (in IPoFabric mode) is used as an interconnection network. Reported times are an average of five runs ± the 95% CI. Such optimisations are now examined individually and, despite being devised for *OpenFL-x*, they are also applicable to the standard Intel® OpenFL, since none of them is strictly related to the ML model being trained.

The baseline wall-clock execution time is 484.13±15.80 seconds; this is the starting point of this ablation study. The first optimisation introduced into *OpenFL-x* is to adapt the buffer sizes used by gRPC to accommodate larger models and avoid resizing operations. Increasing the gRPC buffer from 2MB to 32MB reduced the use-case execution time to 477.0±17.5 seconds, an improvement of ~ 1.5%. Notice that the impact of this tuning increases in the size of the exchanged ML model. The second optimisation concerns the serialisation library, which is necessary to prepare complex objects for exchange through the network. By using *Cloudpickle*, the use-case execution time is reduced to 471.4±6.1 seconds, an improvement of ~2.6%. Next, `TensorDB` performance is examined, which access operations slow down linearly in the number of federated rounds due to the increase in the database size. Through a proper configuration,

Table 3.4: Comparison between the F1 scores obtained by AdaBoost.F and *OpenFL-x* over all the datasets tested in the original AdaBoost.F paper. The reported *OpenFL-x* values are the average of 5 runs ± the standard deviation. The number of classes of each dataset is included for reference.

Dataset	**Classes**	**AdaBoost.F**	***OpenFL-x***
Adult	2	85.58 ± 0.06	85.60 ± 0.05
ForestCover	2	83.67 ± 0.21	83.94 ± 0.14
Kr-vs-kp	2	99.38 ± 0.29	99.50 ± 0.21
Splice	3	95.61 ± 0.62	96.97 ± 0.65
Vehicle	4	72.94 ± 3.40	80.04 ± 3.30
Segmentation	7	86.07 ± 2.86	85.58 ± 0.06
Sat	8	83.52 ± 0.58	84.89 ± 0.57
Pendigits	10	93.21 ± 0.80	92.06 ± 0.44
Vowel	11	79.80 ± 1.47	79.34 ± 3.31
Letter	26	68.32 ± 1.63	71.13 ± 2.02

`TensorDB` is set to store only the essential information of the last two federation rounds, sufficient to guarantee the correct execution of the AdaBoost.F algorithm. This configuration thus results in a stable memory occupation and access time, independently from the number of federated rounds: the use-case execution time drops to 414.8±0.9 seconds, an improvement of ~14.4% over the baseline. Lastly, the two `sleep` in the *OpenFL-x* code are examined. One is needed for the end-round synchronisation, and the other for the general synchronisation point (`synch`) needed at the end of each task; their original values are 10 seconds and 1 second, respectively. Their values were both empirically lowered to 0.01 seconds to reduce waiting times: this is the lowest waiting time still improving the global execution time. This choice is possible due to the computational infrastructures exploited in this work; it may not be suitable for wide-scale implementations in which servers and clients are geographically distant, compute and energy-constrained, or connected to an unreliable network. With this sleep calibration, the global use-case execution time results in 250.8±9.6 seconds, a ~48.2% less than the baseline. Overall, with all the optimisations applied together, *OpenFL-x* achieves a final mean execution time of 88.6± seconds, i.e. a 5.46x speedup over the baseline.

3.2.4 Experimental validation

The original AdaBoost.F paper's experiments [135] are here replicated to investigate the correctness of the *OpenFL-x* implementation. The results are compared to the original ones, proving the soundness of *OpenFL-x*. These experiments involve ten different datasets: `adult`, `forestcover`, `kr-vs-kp`, `splice`, `vehicle`, `segmentation`, `sat`,

pendigits, vowel, and letter. These are standard ML datasets targeting classification tasks, both binary (adult, forestcover, kr-vs-kp) and multi-class (all the others), with a varying number of features (from the 14 of adult up to the 61 of splice), and a different number of samples (from the 846 of vehicle up to the 495.141 of forestcover). Each training set is split in an IID way across all the Collaborators, while the testing has been done on the entire test sets. The adaBoost model class is implemented manually, while a simple decision tree from SciKit-Learn with 10 leaves is used as a weak learner. The computation is run on VMs allocated on a cloud infrastructure to simulate a real-world cross-silo FL scenario. Respecting the original AdaBoost.F experimental setting, ten computational nodes are allocated, one for each Collaborator, plus one more node for the Aggregator. Each VM allocates an 8-core Intel® Xeon Processor (Skylake, IBRS) working at 2.1GHz and 8GB of RAM; the interconnection network is a 10Gb Ethernet.

Table 3.4 reports each dataset's original and reproduced F1 score (mean value ± the standard deviation over 5 runs). The experimental values are fully compatible with the results reported in the original study, thus assessing the correctness of the implementation. It can be observed that the standard deviation intervals are particularly high for the vehicle, segmentation, and vowel. This fact can be due to the small size of the training set of these datasets, respectively 677, 209, and 792 samples, which, when split up across ten Collaborators, results in an even smaller quantity of data per client, determining the creation of low-performance weak learners. Also letter reports a high standard deviation: this could be due to the difference between the classification capabilities of the employed weak learner (a 10-leaves decision tree) compared to the high number of labels present in this dataset (26 classes), making it hard to obtain high-performance weak learners. Nonetheless, all the reported F1 scores are comprehended inside the original paper's margin of error, making *OpenFL-x* a reliable tool for further experimentation.

The mean F1 score curve for each dataset can be observed in Figure 3.10. After an initial dip in performance, almost each learning curve continues to grow monotonically to higher values. This fact is expected since AdaBoost.F is supposed to improve its classification performance the more weak learners are included in the model, i.e., in the number of federated rounds. A new weak learner is added to the aggregated model after round: AdaBoost.F thus grows linearly in size with the number of federated rounds. This characteristic of the algorithm has many consequences, like the increasingly longer time needed for inference and for moving the aggregated model over the network. From Figure 3.10, it is noticeable that a few tens of federated rounds are more than enough to obtain a decent level of F1 scores in the vast majority of cases. This property is attractive since obtaining a small and efficient AdaBoost.F model with little training effort is possible. Instead, it is possible to obtain better-performing models with more extended training for more complex datasets like letter and vowel. This property implies that AdaBoost.F can produce bigger and heavier models at need, according to the desired performance and inference complexity.

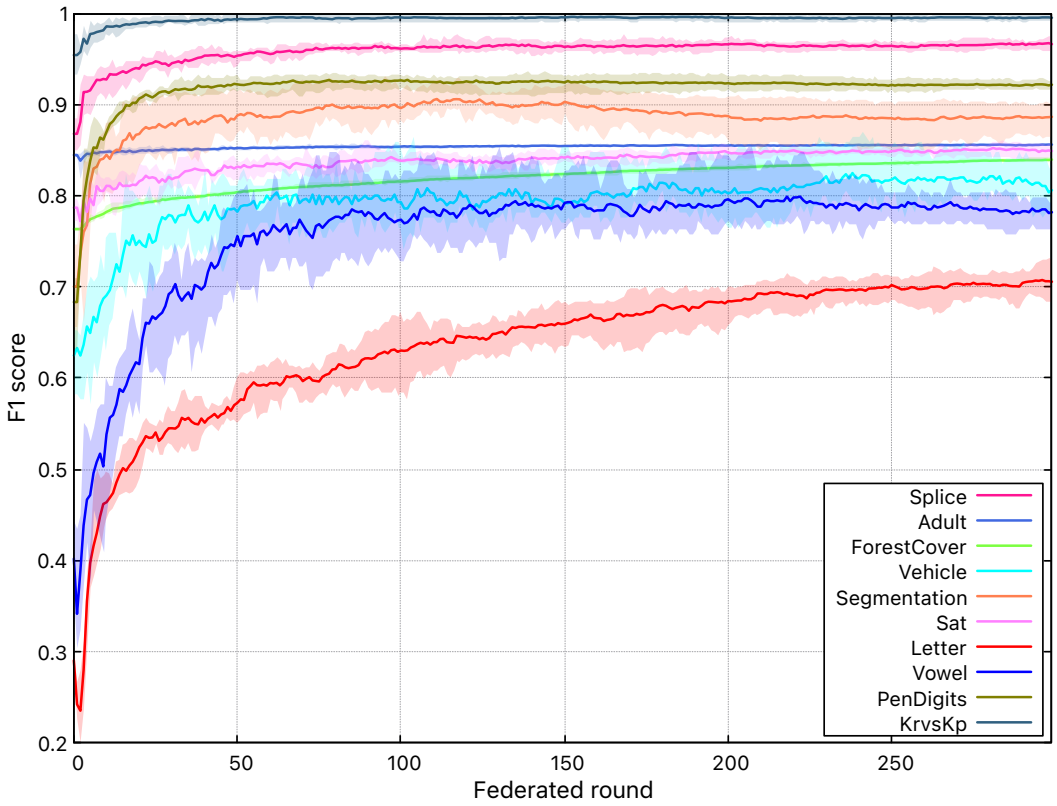


Figure 3.10: F1 score curves (± the standard deviation) obtained by *OpenFL-x* with a 10-leaves decision tree as weak learner on each dataset tested in the AdaBoost.F original paper. Each experiment is run for 300 federated rounds and is replicated 5 times.

Figure 3.11 shows the F1 curves obtained using *OpenFL-x*'s AdaBoost.F implementation with different weak learners on the `vowel` dataset. One representative ML model has been chosen from each multi-label classifier family available on SciKit-Learn: extremely randomised trees (trees), ridge linear regression (linear models), multi-layer perceptrons (MLP) (ANNs), KNN (neighbours), Gaussian naïve Bayes (naïve Bayes), and simple 10-leaves decision trees as baselines. Each model has been used out-of-the-box, without hyper-parameter tuning, using the default parameters set by SciKit-Learn (v1.1.2). All ML models work straightforwardly in *OpenFL-x* without needing to code anything manually: changing the weak learner class name in the experiment file results is sufficient. Such easiness of use proves that data scientists can leverage *OpenFL-x* to experiment with the MAFL approach without requiring any heavy effort.

The computational performance of *OpenFL-x* is now analysed. The scalability study uses the HPC infrastructure reported earlier in this Section and Monte Cimone [19], the first available HPC-class cluster based on RISC-V processors. Monte Cimone comprises 8 computing nodes equipped with a U740 SoC from SiFive integrating 4 U74

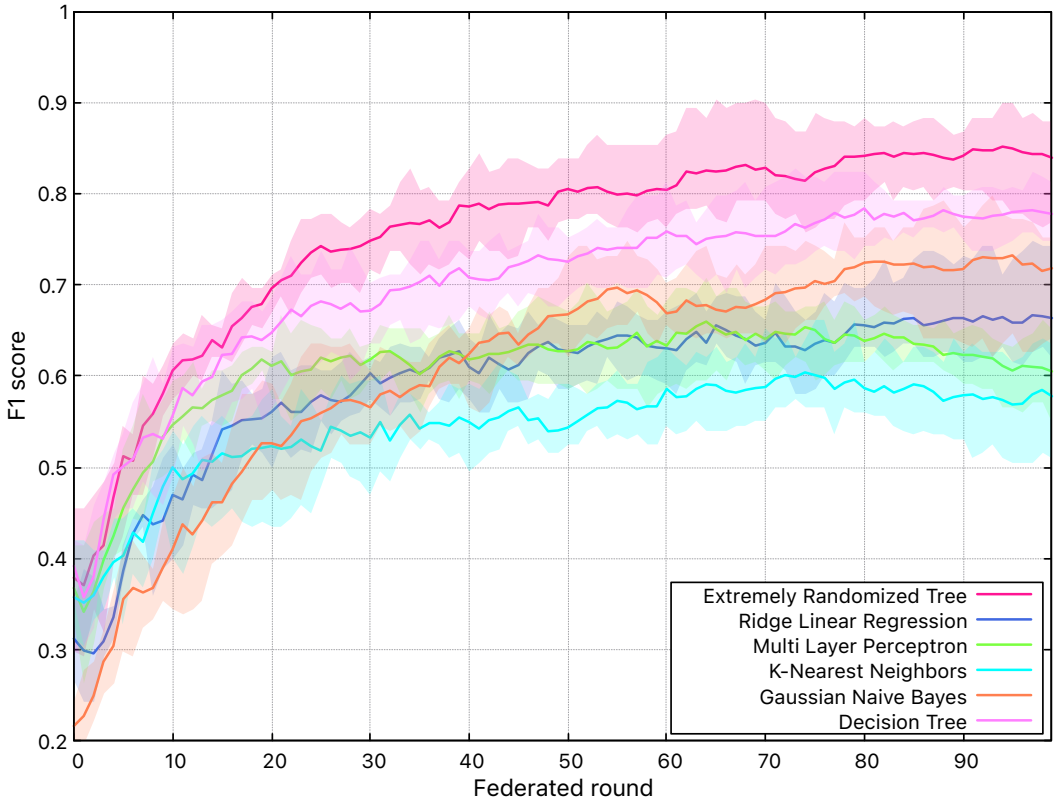
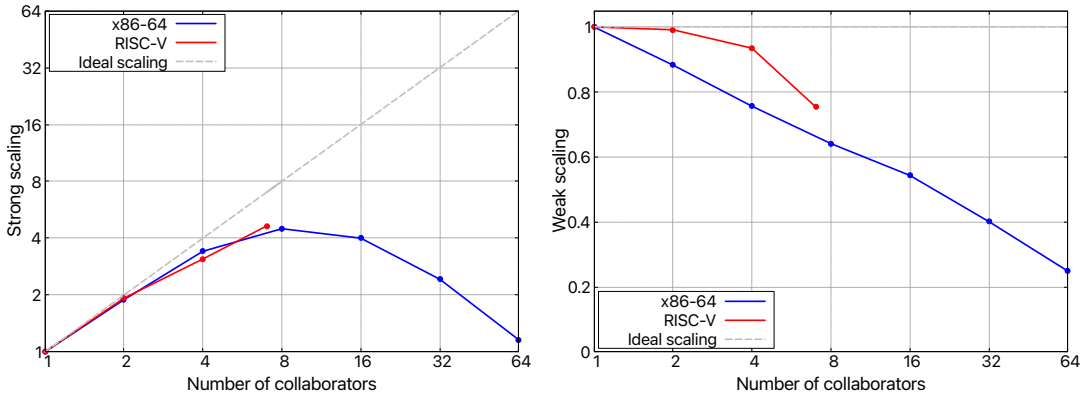


Figure 3.11: F1 score curves (± the standard deviation) obtained by *OpenFL-x* using a different ML model as weak learners each time on the `vowel` dataset. Each experiment is run for 100 federated rounds and is replicated 5 times.

RV64GCB cores @ 1.2 GHz, 16GB RAM, and a 1 Gb/s interconnection network. The `forestcover` dataset is selected for running these experiments, being the largest used in this study, and split into 485K training and 10K testing sample sets. The chosen weak learner is a 10-leaves SciKit-Learn decision tree, the same used earlier in this Section. These experiments use just 100 federated rounds, which are enough to provide acceptable and stable results (10 federated rounds on the RISC-V system due to the longer computational times required). Different federations have been tested, varying numbers of Collaborators from 2 to 64 by powers of 2. No larger federations are tested since Intel® OpenFL is designed to suit a cross-silo FL scenario, meaning a few dozen clients. Two different scenarios are investigated: *strong scaling*, where a fixed problem size is split uniformly across the increasing number of `Collaborators`, and *weak scaling*, where a fixed size problem split is assigned to each `Collaborator` regardless of the number of `Collaborators` involved. In both cases, the baseline reference is the time taken by a federation comprising an aggregator and a single collaborator.

Figure 3.12 shows the strong and weak scaling properties of *OpenFL-x*; each data



(a) Strong scaling

(b) Weak scaling

Figure 3.12: Strong and weak scaling performance obtained by *OpenFL-x* with a 10-leaves decision tree weak learner on the `forestcover` dataset on two HPC infrastructures (C3S, Monte Cimone) with different microarchitectures (x86-64, RISC-V). Note that Monte Cimone offers up to 8 computing nodes (1 aggregator + 7 collaborators), limiting the scale of the experiments.

point is the mean over 5 runs. The RISC-V plot stops at 7 `Collaborators` since Monte Cimone offers just 8 computational nodes, and sharing a node between the Aggregator and Collaborator would affect the computational time measurements. *OpenFL-x* does offer notable strong scaling performance, being unable to scale efficiently beyond 8 nodes as the execution becomes increasingly communication-bound. The same effect can also be noted for the weak scaling performance; nevertheless, the degradation appears sublinear. This fact is essential since the main benefit of an FL system is the additional training data each contributor node provides. Thus, a real-world federation resembles a weak scaling scenario from the computational performance perspective. Noticeably, the RISC-V cluster exhibits a better strong scaling than the other HPC infrastructure. This performance is justified by the slower computational speed of the RISC-V cores, which leads to higher training times and makes the execution more compute-bound, especially with fewer nodes. Also, the weak scalability on the RISC-V cluster suffers from the lower network speed available. Given the discussed experimental results and that real-world cross-silo federations rarely count more than a dozen participants, it is possible to assess that *OpenFL-x* is suitable for real-world MAFL experimentation and prototyping, offering both proved AdaBoost.F support and sufficient scaling performance.

The implementation experience of *OpenFL-x* and the subsequent experimentation made it evident that current FL frameworks are not designed to be as flexible as the current research environment needs them to be. The standard OpenFL workflow is

not customisable without modifying the code, and the serialisation structure is DNN-specific, suggesting that a new, workflow-based FL framework is needed. Such a framework should not implement a fixed workflow but allow the user to express any number of workflow steps, entities, the relations between them, and the objects that must be exchanged. This property implies the generalisation of the serialisation infrastructure, which cannot be limited to tensors only. Such an approach would lead to a much more straightforward implementation of newer and experimental approaches to FL, both from the architectural and ML perspectives.

Furthermore, the use of asynchronous communication can help better manage the concurrent architecture of the federation. These systems are usually slowed down by stragglers that, since the whole system is supposed to wait for them, will slow down the entire computation. In *OpenFL-x*, the waiting time between the different collaborators participating in the training determines a significant part of the scalability issues. While such an approach would improve the scalability performance of any FL framework, it also underlies the need to investigate how to handle newer and older updates simultaneously. This capability would improve the computational performance of gradient and non-gradient-based systems: the relative aggregation algorithms must be revised to accommodate this new logic. This matter is not trivial and deserves research interest. Furthermore, an exciting feature that can be added to *OpenFL-x* is using different weak learners for different collaborators or federated rounds. This property would allow a diversification that can benefit particular applications, such as IoT or data on which the efficacy of specific ML models has already been proven. Lastly, due to the possibility of exploiting less computationally requiring models, *OpenFL-x* can easily be used to implement FL on low-power devices, such as RISC-V-based systems.

3.3 Federated anti-financial crimes

A real-world MAFL case study is now presented and discussed, proving the potentialities of such an approach. In collaboration with *Intesa SanPaolo*'s *Anti-Financial Crime Digital Hub* (AFC Digital Hub), an MAFL-based proof-of-concept anti-financial crime system is deployed. *Intesa SanPaolo* is the largest Italian bank at the time of writing, and its primary legal and administrative headquarters are in Turin, Italy. The AFC Digital Hub is a consortium constituted by the *Intesa SanPaolo Innovation Center*, the *University of Turin*, and the *Polytechnic University of Turin*, among others, promoting the use of new technologies and AI-based systems to detect and prevent financial digital crimes. A public, freely available dataset resembling *Intesa SanPaolo*'s financial data is used as a proxy to study this kind of data and its particularities. After carefully preprocessing the available data, two FL-based approaches are tested, investigating the possibility of collaboration between different financial institutions on such an issue. According to the recent literature, two ML models are selected for this study: *TabNet* and *one-class SVM*. TabNet is a DNN specialised for tabular data, and thus, its use constitutes a standard FL scenario. On the other hand, one-class SVM is a traditional ML model, and

creating a federation out of it requires significant effort to develop an FL algorithm capable of handling it; this is an MAFL use case. The obtained results are then reported and discussed, highlighting the difficulties of working with such complex and heavily unbalanced data and strict regulations and company policies restricting their use.

3.3.1 Digital financial crimes detection

The continuous trend toward society's digitalisation is impacting every aspect of people's lives, including money management. With an increasing portion of everyday money exchange moving to online platforms, financial crimes are also moving towards the digital world. Frauds, money theft and laundering, and other illegal financial activities are moving from the physical to the digital world, and financial institutions are rapidly adapting to this change. Fighting such a phenomenon requires advancements and investments in digital competencies and technologies to keep up with ever-evolving criminal techniques. Current systems adopted by financial institutions are usually semi-automatic, with complex rule-based systems selecting suspect transactions to be manually analysed by expert personnel. Such systems are hard-coded, with hundreds of hand-written and experience-based rules, while manual transaction revisions require many financial experts to devote their time to the job. The current systems thus require a large economic effort, especially considering that criminal transactions represent a minimal amount of the total, according to *Intesa SanPaolo*'s information, reaching even one malicious transaction in a million. Furthermore, each financial institution handles such a process independently, leading to different detection policies for each institution commensurate with the bank's size, capabilities, and management.

FL is a viable option for financial institutions to collectively develop a common ML-based system capable of integrating every single bank's knowledge without disclosing their precious private data. An AI-empowered system capable of continually learning would significantly improve banks' ability to spot and prevent criminal transactions. Moreover, if more companies develop such a system together, the deriving ML model would also be capable of learning the characteristics of the different criminal transactions handled by different banks, significantly improving the security systems of both large and small banks. Large companies would benefit from the knowledge the ML model acquires from other large institutions. In contrast, smaller ones could then exploit ML model performance they could not obtain by themselves. Such an ML-based system would thus be used in conjunction with the existing rule-based systems, improving their performance. Humans would still be needed in such a framework, making the final decision on suspicious transactions. The explainability of such ML models is also extremely important. It allows the human component of the loop to understand why a particular transaction has been highlighted and eases the final decision process. Thus, FL seems an interesting ML technique for pushing different financial institutions to collaborate on developing new-generation anti-financial crime systems.

However, literature addressing financial crimes and FL is still very poor. On the one

hand, FL is still not so widespread in practice. It is a relatively novel ML approach, and its adoption requires mature software frameworks allowing it and expert ML practitioners to deploy it. On the other hand, it takes a long time for a regulated industry like the financial world to open up to new technologies. Such companies must comply with many laws and security precautions, adopting a technology only when it is sufficiently mature. Also, FL is a collaborative approach, and collaboration with such large institutions is complex and slow, especially considering the value of each bank's private data and its sensibility.

Interpreting transactions as complex graphs, [159] propose a graph-FL-based solution in which local and global feature sets are calculated across each bank's transaction graph. This interesting approach merges FL and graph-learning approaches, but unfortunately, it does not describe the actual procedures involved in great detail. Furthermore, the dataset used for experimentation is not freely available, making the provided results non-reproducible. [32] explores the potentiality of FL in financial applications, especially considering privacy constraints. This paper proposes an approach based on SMC and DP to design an FL deployment that is fully compliant with regulation standards and avoids possible data leakages. [112] also exploits DP, but in conjunction with HE, to carry out a vertical FL application across multiple financial institutions. In this case, FL represents but one step of the proposed pipeline, which aims to connect the behaviour of each individual on all available financial databases. However, the experimental assumptions made in this study are unrealistic. They suppose that half of the financial institution's clients are fraudulent, undermining the proposed approach's real-world applicability. Another application of DP and FL in financial applications is found in [147], where they are used to train an autoencoder exploiting data from multiple entities. This study models the financial crime detection problem as an unsupervised anomaly detection problem, where fraudulent transactions are identified by examining the autoencoder reconstruction error. Finally, [190] proposes a federated self-supervised learning system exploiting contrastive learning to handle heavily unbalanced data effectively. This approach proves to be experimentally effective, providing state-of-the-art results.

3.3.2 The synthetic financial datasets for fraud detection

Considering the literature mentioned above in digital financial crime detection, it is clear how restricted and regulated access to real bank transaction data is. This is due to various reasons, such as the banks' customers' right to privacy, the current data processing laws, and the data's value. However, accessing real data is the first step in researching and developing innovative techniques impacting the real world. Unfortunately, access to actual financial data is not granted for this research work; a synthetic alternative is thus considered. Such synthetic data is generated from a real, publicly available transaction dataset: PaySim [111]. This dataset is based on a real-world case study in which a real company developed an electronic wallet application that allows

users to exchange money between themselves through their smartphones. The existing transaction base has been subsequentially expanded with the system's transnational financial logs, obtaining a database similar to a real financial company dataset.

From this starting point, the *synthetic financial datasets for fraud etection* is generated through a repeated simulation process iterated over the PaySim dataset. This synthetic dataset is scaled down to a quarter of the original size of PaySim and is freely available on Kaggle[3]. The dataset consists of 6,362,620 transactions, of which only 8,213 are labelled as fraudulent, respecting the roughly 1 in a million proportion of actual financial data. 11 features describe each data sample, here summarised:

- `step`: numerical time step of the transaction. The PaySim simulation is run for 30 days, for 744 1-hour time steps.
- `type`: categorical value describing the transaction type (CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER).
- `amount`: numerical value describing the amount of money moved by the transaction expressed in the local currency.
- `nameOrig`: the customer's name starting the transaction.
- `oldbalanceOrg`: numerical value reporting the customer's balance before the transaction.
- `newbalanceOrig`: numerical value reporting the customer's balance after the transaction.
- `nameDest`: name of the customer recipient of the transaction.
- `oldbalanceDest`: numerical value reporting the recipient's balance before the transaction (omitted if merchant).
- `newbalanceDest`: numerical value reporting the recipient's balance after the transaction (omitted if merchant).
- `isFraud`: flag signalling that the transaction is requested by one of the fraudulent customers of the simulation.
- `isFlaggedFraud`: flag signalling that the transaction violates the business model rules (i.e., transferring an `amount` $\geq$ 200,000 in a single transaction).

In this synthetic dataset framework, fraudulent agents aim to profit by emptying regular customers' accounts and then cashing out of the system.

[3] https://www.kaggle.com/datasets/ealaxi/paysim1

A preliminary analysis of this dataset reveals that no duplicated samples and no missing values are present. A correlation analysis between the features exposes some trivial correlations, such as between the new and old balances and the balances and the transaction type. Interestingly, no strong correlation arises between the fraud flag and the other features; just some minor correlations are found, especially with the amount of money transferred, the transaction type, and the violation of the business model flag. A principal component analysis (PCA) analysis reveals that the fraudulent transactions are scattered among two of the three principal components, meaning they share one. This fact impacts the similarity between the fraudulent samples; in fact, a similarity analysis reveals that the positive (fraudulent) samples have a mean cosine similarity of .36, while it is generally .03.

3.3.3 Proof-of-concept FL and MAFL experimental results

Defining a congruous experimental setting is the first step in designing a realistic proof-of-concept federation of financial institutions. Realistically, such a federation would not span many institutions due to the amount of work required to establish agreements between them and to reach a common technical ground (software, algorithms, goals, data schema, et similia). Another characteristic of such a federation is that the participants can access large computing infrastructures and stable interconnection networks and hire expert personnel to handle the computation. Furthermore, it is assumed that each participant has access to a vast amount of data, and no particular restrictions on availability are considered. Based on these characteristics, the proposed FL system is a perfect example of a cross-silo FL setting, respecting all the standard assumptions of this scenario. This fact implies that the main aim of this proof-of-concept study is the learning metrics obtained by the federated model over the local dataset, software design choices, and the consequential computation and communication performances obtained by the system are taken into consideration since the parties involved in the federation can allocate as many computational resources as needed.

Considering the cross-silo scenario and the considerations expressed by *Intesa Sanpaolo Innovation Center*, a proof-of-concept federation with 4 clients is proposed. Half of these clients are supposed to be large institutions, possessing most of the data available. In contrast, the other half are supposed to be small institutions, holding much less data than the others. This choice is made to investigate the impact of highly imbalanced data distribution across the client on the final federated ML model. Ideally, the federated model could exploit the large amount of data provided by the larger institutions to learn as many fraudulent patterns as possible while exploiting the smaller ones to learn more particular fraudulent patterns not found in the larger ones. This problem is particularly interesting in FL research since it is highly problematic to exploit each data source at its best without letting the knowledge contained in larger datasets overwhelm the smaller ones and vice versa. The exact data split used in the experiment is reported in Table 3.5. As can be seen, split 1 and 2 possess the same amount of data

Table 3.5: Number of data samples, positive samples, and their relative percentage for each of the four dataset splits adopted in the anti-financial crimes experiments. The table's bottom section reports the dataset's global information.

Dataset	# samples	# positive samples	% positive samples
Training set (split 1)	2,217,065	3,016	.14
Training set (split 2)	2,217,066	2,974	.13
Training set (split 3)	204,556	208	.10
Training set (split 4)	228,717	106	.05
Training set	4,867,404	6,304	.13
Validation set	540,823	688	.13
Test set	954,393	1,221	.13

(2,217,000 samples), which in turn is an order of magnitude larger than split 3 and 4 (200,000 samples); however, all of them share a similar percentage of positive sample (.13), apart from split 4, which has a notably lower one (.05).

According to the reviewed literature, two different models are selected for investigation: *TabNet* and *one-class SVM*. TabNet [15] is an open-source[4] PyTorch implementation of a deep tabular data learning architecture. It exploits sequential attention to choose which feature to focus on at each decision step, concentrating the learning power of the model on the most impacting features. TabNet outperforms the other deep tabular data learning models available at the time of writing and guarantees interpretability at both feature and global levels. On the other hand, one-class SVM is an anomaly-detection-oriented model based on SVM. Anomaly detection is a branch of ML that identifies data samples that do not conform to the data's expected behaviour. In this setting, anomaly detection can be helpful since fraudulent data can be considered anomalous among all the fair ones. The PCA has previously confirmed this consideration: fraudulent transactions present different characteristics from the others. These two models offer the possibility of investigating standard FL and MAFL approaches to this research problem.

Obtaining a federation out of TabNet is straightforward; once the TabNet PyTorch model is extracted from the SciKit-Learn wrapper, it can be treated as any other DNN. Any standard FL framework can handle such a PyTorch DNN; it should only be noticed that the SciKit-Learn wrapper provided some hyper-parameter tuning and preprocessing functionalities that must be re-implemented in the FL framework. On the other hand, federating one-class SVM requires an algorithmic effort. Anaissi et al. [12] propose a federated one-class SVM training algorithm based on fedAvg, in which the one-class SVM local models are trained through an SGD-based procedure sharing the

[4] https://github.com/dreamquark-ai/tabnet

Table 3.6: Learning performance obtained by one-class SVM and TabNet on the PaySim dataset. The centralised model performance are reported as a baseline, together with the performance that each client obtains on the local data split and the performance of the federated model.

		run 1	**run 2**	**run 3**	**run 4**	**run 5**	$\mu \pm \sigma$
one-class SVM	Local split 1	.789	.928	.663	.785	.912	$.815_{\pm .108}$
	Local split 2	.72	.843	.906	.783	.843	$.819_{\pm .070}$
	Local split 3	.802	.782	.529	.739	.862	$.743_{\pm .127}$
	Local split 4	.763	.805	.731	.528	.675	$.700_{\pm .107}$
	Centralised	.923	.94	.907	.971	.955	$.939_{\pm .025}$
	Federated	.677	.727	.601	.604	.608	$.643_{\pm .056}$
TabNet	Local 1	.971	.960	.962	.982	.992	$.973_{\pm .014}$
	Local split 2	.971	.964	.963	.981	.965	$.969_{\pm .007}$
	Local split 3	.838	.818	.676	.875	.792	$.800_{\pm .076}$
	Local split 4	.351	.698	.549	.600	.695	$.579_{\pm .142}$
	Centralised	.963	.988	.920	.984	.981	$.967_{\pm .028}$
	Federated	.905	.919	.914	.910	.870	$.904_{\pm .019}$

coefficients of the features in the kernel space. This algorithm allows the server to aggregate the local models by simply averaging their parameters. Only the above-median loss models are considered for aggregation, strengthening it against possible low-performing parameters. The proposed procedure is thus very similar to a standard FL scenario and does not require heavy software refactoring of an off-the-shelf FL framework.

Given the proposed proof-of-concept federation's enterprise level, an enterprise-level framework is needed to provide the necessary FL software infrastructure. NVIDIA FLARE is selected among all the reviewed FL frameworks due to its technical maturity, security features, and enterprise-level support. NVIDIA FLARE is an open-source, DL framework-agnostic FL framework. It offers out-of-the-box support for advanced privacy features such as HE and DP, which can be extremely important for financial institutions. Furthermore, it eases the development process of a federation by allowing both simulated, proof-of-concept, and real-world deployment. Furthermore, it offers basic support for some FL algorithms based on traditional ML models, such as XGBoost. All the characteristics mentioned above make it possible to implement both the standard FL (TabNet) and the MAFL (one-class SVM) workload in reasonable development time. All the presented experiments are run in simulation mode.

Table 3.6 summarises the obtained experimental results. As can be seen, TabNet AUC values obtained by the smaller clients (.800, .579) are noticeably lower than both

the centralised (.967) and federated ones (.904). This difference in performance determines an advantage for small institutions participating in such a federation since they will gain a better model than the one they could obtain. However, this knowledge transfer does not happen the other way around: the larger institution obtains higher AUC values (.973, .979) than the centralised (.967) and the federated (.904) scenarios. These results mean that the standard fedAvg procedure used in the experiments cannot produce a federated model containing the knowledge of both the small and large parties. Instead, it produces a model that is the average of them. This consideration implies that some of the knowledge contained in the large institutions' models is lost in the aggregation process. These results would thus push large institutions to exclude small clients from their federation since they would not gain anything from it and would obtain a federated model worse than the one they would obtain by themselves. This phenomenon is further highlighted by the fact that the federated model AUC mean value (.904) is much lower than the centralised one (.967). A different, more complex aggregation strategy needs to be implemented to justify the real-world deployment of such a federation. Ideally, the knowledge of large and small institutions should be contained in the final aggregated model, reaching performance comparable to those of the centralised model. Furthermore, TabNet is very sensitive to its randomly selected initial weights: it can happen that some runs do not converge at all. Such runs are excluded from the presented results.

On the other hand, one-class SVM results are, unfortunately, a complete failure. While the centralised model obtains an AUC value (.939) that is comparable to the centralised TabNet one (.967), the federated model's AUC (.643) is not comparable to TabNet's one (.904). Even worse, no institution actually gains anything from the federation: each local individual model AUC (.815, .819, .743, .700) is higher than the federated one (.643), indicating that no knowledge accumulation is occurring in the federated model. The only positive result obtained by one-class SVM is that it obtains better performance when dealing with highly unbalanced datasets: the mean AUC of client 4 (.700) is much higher than the one obtained by TabNet on the same client (.579). However, one-class SVM is also heavily influenced by the random initialisation of its parameters, and it performs very poorly if not finely tuned.

As an additional experiment, AdaBoost.F has been tested on this proof-of-concept financial federation. Standard adaBoost is run on the whole (centralised) dataset as a comparison metric. AdaBoost.F obtained the same accuracy value of centralised adaBoost but with slightly better precision (.971) and recall (.857).

Chapter 4

High-Performance Federated Learning

This chapter discusses this PhD thesis's second main scientific contribution: *FastFL*. An analysis of modern commercial FL frameworks introduces all the design flaws that could negatively impact their computational performance. The communication topologies, the chosen communication backend, and the programming languages used in such software implementations are analysed, highlighting how much these elements can impact computational performance. Since current FL frameworks are based on usability, extensibility, and stability, their computational performance is not particularly developed. As discussed in the previous chapter, many new types of FL algorithms may arise, each stressing the software's computation or communication capabilities. Modern software is not ready to adapt to newer, non-DL-based FL algorithms or the growing scale of FL applications. An innovative top-down approach to FL is proposed to address the abovementioned computational issues. The FastFlow parallel computing framework is selected to handle both the communication topology and backend, and the *FastFL* FL framework is built on top of it. It is high-performance oriented due to its C/C++ implementation, supports TCP/MPI communications, and allows shared and distributed memory executions of the same code. Having the bottom software layer, the abstract, top-level interface to such framework is provided by the RISC-pb^2l formal language, able to describe abstractly any computation implementable with FastFlow. These two elements are then included into *FastFL* by offering automatic translation from the high-level RISC-pb^2l description of a federation into a low-level FastFlow implementation. A Python DSL wrapping the RISC-pb^2l language is then described, completing the software stack offered by *FastFL*. The computational performance of *FastFL* is then analysed and discussed, proving that *FastFL* is a successful proof-of-concept of a high-performance oriented FL framework.

Table 4.1: Brief overview of the most widespread and mature FL frameworks available in the literature. For each of them are reported the targeted applications, the deployment scenarios, the communication protocols used, the programming language used for implementation, and the possibility of building federations based on custom communication graphs.

Framework	**Target**	**Scenario**	**Comm. protocol**	**Impl.**	**Custom comm. graph**
TFF [16]	Cross-silo	Simulation Real world	gRPC/protobuf	Python	✗
PySyft [194]	Cross-silo Cross-device	Simulation Real world	Websockets	Python	✗
SecureBoost [44]	Cross-silo	Simulation Real world	gRPC/protobuf	Python	✗
FederatedScope [179]	Cross-silo Cross-device	Simulation Real world	gRPC/protobuf	Python	✗
LEAF [34]	Cross-silo	Simulation	—	Python	✗
FedML [81]	Cross-silo	Simulation Real world	gRPC/protobuf MPI, MQTT	Python	✗
OpenFL [71]	Cross-silo	Simulation Real world	gRPC/protobuf	Python	✗
Flower [23]	Cross-silo Cross-device	Simulation Real world	gRPC/protobuf	Python	✗
FastFL [121]	Cross-silo	Simulation Real world	**TCP/Cereal** **MPI/Cereal**	**C/C++**	✓

4.1 Current federated learning frameworks' limits

Many off-the-shelf FL frameworks are currently available, each with specific characteristics. Figure 4.1 summarises the most well-known and mature of them based on [21]. As can be seen, most of them share the same design choices: all support the cross-silo scenario, are implemented in Python, and offer simulation functionalities. Almost all share the same communication protocol based on gRPC and protobuf. These similitudes imply that the core design ideas behind many current FL frameworks are similar. It derives that the derived software offers very similar functions engineered in different ways. This fact does not hinder the different FL frameworks' utility and efficacy, especially considering the highly regulated environment in which FL has to foster cooperation. However, it highlights that the FL scenario is currently getting fixed on certain expectations and axioms without trying to follow other possible development paths. This section addresses this issue by considering three different aspects of current FL frameworks that can particularly hinder computational performance.

First, standard FL is mainly based on a master-worker approach, overlooking other possible structures to handle communications between different parties. Despite many pros, master-worker is not a remarkably scalable communication structure and is liable to the single-point-of-failure issue. Second, commonly used FL frameworks offload their communication logic to a gRPC and protobuf backend, which is not particularly fast or flexible. Standard HPC approaches, like MPI, could offer better communication costs, even if they require a more careful implementation. Lastly, most current FL frameworks are implemented in Python. While this approach is convenient from a development point of view, the amount of resources required by modern Python software and the resulting computational performances are not compatible with a vision of FL aiming at supporting the ever-evolving computing continuum environment. Additionally, no FL framework is designed to natively support emerging computational architectures, such as RISC-V. The emergence of alternative *instruction set architectures (ISAs)* to x86-64 and ARM-v8 exacerbates the challenges in porting (optimised) libraries and guaranteeing interoperability between heterogeneous systems. This research strand proposes that the modern FL framework should be flexible in defining the system architecture and communication patterns to open up research on alternative FL systems and exploit optimised distributed runtimes for optimal performance on different ISAs.

4.1.1 Communication topologies

Communication topologies are a well-known topic in parallel and distributed computing. When multiple processes cooperate to solve a common task, allowing fast and reliable information sharing between them is critical to enhance computational performance. Processes can be organised according to several different topologies, each with pros and cons. Thus, if it exists, the best communication topology for a given task has to be chosen according to the specific deployment scenario. When making these choices, communication link bandwidths, local computational power, available storage, and business requirements should all be considered. It is worth noting that different communication topologies can be exploited to carry out the same computation but with different properties, like the number of communications required or the total amount of computation of the whole system. Three communication topologies used later in this chapter to model DML workloads are characterised: master-worker, peer-to-peer, and tree. They are graphically represented in Figure 4.1.

The master-worker topology is one of the most common and widely used. In this schema, one process distinguishes itself from the "master", while all the others are known as "workers". The master coordinates the workers, devoting its computing capabilities to split the computation into smaller shards, assigning them to the worker processes and retrieving the partial results. The master is thus the pivot of the computation and the only process directly communicating with all the other ones. This latter aspect can be identified as this topology's strongest and weakest property. On

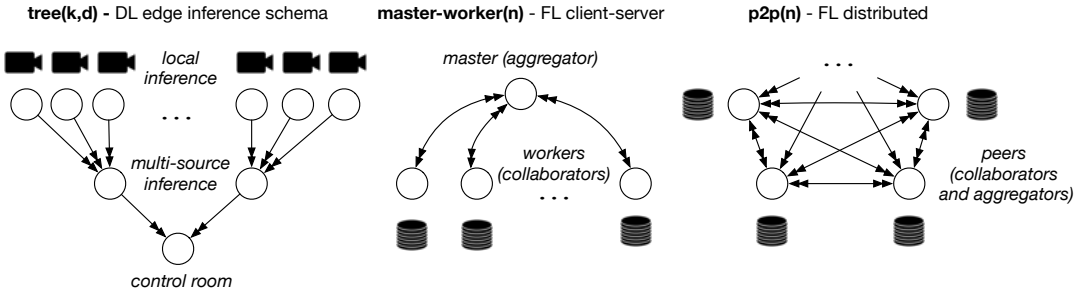


Figure 4.1: Three examples of communication topologies (tree, master-worker, and peer-to-peer) that are not all supported by current FL frameworks. Each has its properties: the tree can be exploited for EI tasks (for example, a multiview detection system), the master-worker for centralised FL workloads, and peer-to-peer for FL deployment in which a central point of failure is not desirable.

the one hand, having a single synchronisation, communication, and management process makes software development and deployment simple, simultaneously achieving a decently low number of communications in the system. On the other hand, having a multitude of processes continuously communicating towards a single process can easily lead to difficulties in obtaining timely responses from the master, possibly saturating its bandwidth and resources, thus slowing down the entire system. This fact can limit the scalability properties of this approach. Also, having a single process handling a large, complex system constitutes a single point of failure, meaning that the whole system fails if this single process fails; it also opens up privacy issues since the master process would be the preferred target for attacks. FL frameworks commonly implement this communication topology and usually decline it into the client (worker)—server (master) perspective. The workers are deployed on the client side, where data resides, receive the ML centralised ML model from the master, actively train it on the local hardware, and send it back to the master. The master is instead deployed server-side, usually on a third-party service, like a could VM: it instantiates the client processes and provides them with the initial ML model; when it receives back the established number of locally trained models, aggregates them and sends the result back to the workers. The process continues iteratively until the ending condition is met; at this point, the master ends the workers and terminates the FL systems.

The peer-to-peer topology offers an entirely different take on processes' organisation and communication. In this case, no process distinguishes itself from the others: they are all peers, each executing the same functions. The result is a completely distributed structure without a central communication point and, consequently, without a single point of failure. It follows that the processes should handle computation and partial computation results accumulation simultaneously, effectively acting as both clients and servers, complicating code development and deployment. Performance-wise, this

communication topology offers extended flexibility and attack resiliency. However, it requires a higher number of communications to carry out the same computation, which strongly limits scalability when dealing with a large number of peers. In the FL context, this communication topology requires each process to train the local model on the local data and then send it to all the other processes while simultaneously receiving all the other processes' locally trained models. As can be seen, the amount of data exchange in this context grows exponentially in the number of peers, while in the master-worker scenario, it grows linearly in the number of workers.

Finally, another communication topology that will be exploited later in this chapter is the tree. A tree structure is a hierarchical system in which each process masters its subgroup of processes. This structure starts iteratively from a single process, called root, and proceeds recursively up to the last processes in the tree, called leaves, constituting a direct, acyclic graph. This topology retains both master-worker and peer-to-peer properties. It maintains a single point of failure, the root node, but all the intermediate nodes of the tree can fail without making the whole structure fail. It also limits the amount of communications directed to a single process, thus limiting the communication channels bottleneck effect, and the total number of communications grows slower than the peer-to-peer topology. When looking at FL, this structure can turn particularly useful for distributed EI tasks, where the leaves of the tree act as data-harvesting processes, the intermediate nodes as partial aggregation or feature extraction steps, and the root as the decisional endpoint.

4.1.2 Communication backends

Having a lower-level look at how communications are implemented in practice, it can be noticed that almost all of the FL frameworks listed in Table 4.1 rely on the gRPC/protobuf combination to provide this functionality. This design choice can be justified from a software engineering point of view, easing the communication modelling effort. gRPC (Google RPC) is an open-source, cross-platform framework implementing RPC functionalities. It uses HTTP/2 for transport and protocol buffers (protobuf) as interface description language. The RPC paradigm revolves around the procedure concept, meaning that procedures and their inputs and outputs are the fundamental objects of remote communication. In gRPC, this paradigm is notably declined towards web services: a client-server architecture is assumed, in which the server exposes the procedures the clients remotely call. To realise this, clients possess a stub, a gRPC object describing the procedures available for remote calling on the server side. Thus, client-server communications acquire point-to-point properties, with procedure execution as granularity level. While effective in web-based, service-oriented software, it may not be the best solution in the FL context due to its underlying assumptions. Having a communication protocol capable of handling collective communications and working at a lower granularity could greatly improve the malleability of FL software's distributed structure while obtaining even better communication performance as a byproduct.

A possible programming model offering these properties is MPI. MPI is a de facto standard in parallel and distributed application development and a must for HPC-oriented software. The MPI standard establishes point-to-point and collective communication primitives, allowing them to behave blocking/non-blocking and synchronously/asynchronously according to the programmer's needs. MPI thus defines a portable, flexible standard that is open to implementation. OpenMPI and MPICH are the most common open-source MPI implementations used in HPC practice. They have been widely proven to be extremely efficient and scalable, and they are the communication backend chosen by the vast majority of current HPC scientific software. Current HPC distributed-memory systems are thus programmed mainly through the MPI standard to achieve peak performance. The Top500, the list ranking the top 500 most powerful supercomputers in the world, exploits the high-performance LINPACK benchmark to measure peak Flop/s, which implements inter-node communications through the MPI standard. Most MPI implementations target the C, C++, and Fortran languages, but software packages extending this support to C#, Java, and Python are also available. MPI is a lower-level, more general approach towards process communication than the RPC protocol. Thanks to its broader set of primitives, it models almost any possible inter-process communication topology. It also allows arbitrarily small granularity in the communication contents. Thus, if correctly handled, this approach's richness, flexibility, and complexity can also offer better computational performance in the FL scenario.

4.1.3 Programming languages

All FL frameworks depicted in Figure 4.1 are mainly implemented in Python. Python is the go-to programming language for ML practitioners and represents the most used programming language at the time of writing, followed by C, C++, and Java. Its multi-paradigm approach, dynamic variable typing, and high-level syntax allow for fast code development. All these characteristics make Python an excellent scripting language, particularly well-suited to creating program prototypes quickly and easily. However, these commodities came with a cost: performance. Python is inherently a single-threaded programming language, offering limited support for multi-threaded applications. Thus, it cannot efficiently exploit the computational power of modern multi-core CPUs. Furthermore, the dynamically typed variables imply a huge overhead at runtime intrinsic to the language itself, and the pre-compilation phase to bytecode and the following interpretation add levels of complexity that spoil the runtime performance. Python syntax does not push developers to think about low-level details and efficiency. Conversely, it makes it extremely easy to use high-level abstractions, leading to unbearable low performance when translated to low-level operations.

This effect can be seen especially in scientific computing and, consequently, in AI/ML code. All major current AI/ML library ad frameworks, such as Sci-Kit Learn, PyTorch, TensorFlow, et similia, are structured as a Python interface wrapping high-performance

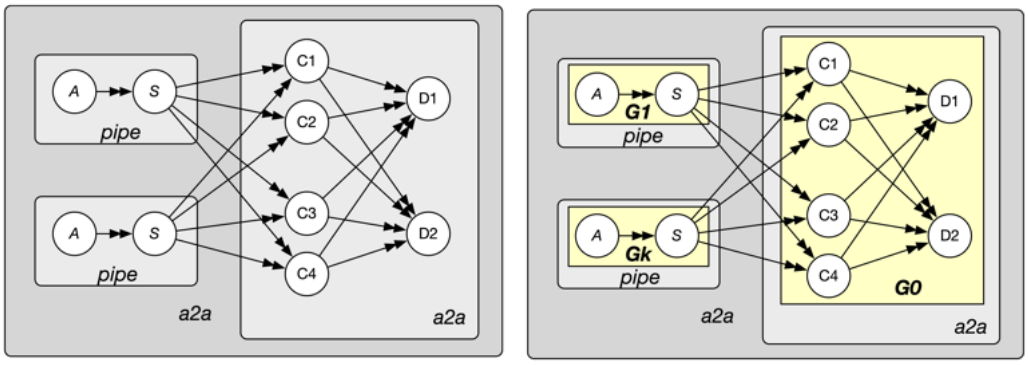
C/C++ code. This design choice makes the difference, allowing ML practitioners to exploit the computational performance of well-designed, low-level routines without worrying about the technical details. This compromise came at the cost of the Python wrapping overhead, which is usually bearable but not negligible, especially on low-power devices. For example, it is possible to verify experimentally the overhead introduced by using PyTorch's Python interface in place of the C++ LibTorch interface on the RISC-V platform. The RISC-V choice allows stressing the overhead on a platform with low computational power and few optimised libraries. Using the MNIST benchmark from the official PyTorch examples, a simple CNN is trained to compare the runtime performance of the two APIs. Note that the two scripts train the same CNN architecture and differ only in language and API use. The results averaged across 5 runs show that the C++ API (314.5s) is about 30% faster than the Python API (442.8s). With this simple example, it is already possible to understand how a hybrid Python/C/C++ FL framework's performance would improve over a full Python one. Such an FL framework can be designed as a currently appreciated ML framework: a Python wrapper handling a high-performance C/C++ core.

4.2 FastFederatedLearning

In this section, *FastFL* is introduced. Starting from discussing a lower-level oriented FL framework capable of delivering higher computational performance than the state-of-the-art, the FastFlow parallel programming language is described. FastFlow defines basic computational units, i.e., building blocks, that can be composed to express a wide range of parallel computations. A FastFlow program can then be deployed on shared and distributed memory systems without needing any modification, effectively giving *FastFL* the capability of handling both simulated and real-world FL deployments. The proposed approach is straightforward to implement and exploits C/C++ and libtorch to provide state-of-the-art ML results in a computationally efficient way. A first version of *FastFL* is then described with its features, and experimental results on various microarchitectures are presented, proving the efficacy of the proposed approach.

4.2.1 The FastFlow parallel programming language

The C++ header-only FastFlow library [7] provides application designers with essential features for parallel programming via suitable abstractions (e.g., Pipeline, ordered Task-Farm, Divide&Conquer, Parallel-For-Reduce, Macro Data-Flow) and a carefully designed runtime system (RTS). At the lower software layer of the library, there are the so-called *Building Blocks*, recurrent data-flow compositions of concurrent activities working in a streaming fashion, which are used as the primary abstraction layer



(a) Shared-memory streaming graph

(b) Distributed-memory streaming graph with K+1 groups (G0,..., Gk)

Figure 4.2: Graphical representation of a FastFlow application and how it can be run interchangeably in shared and distributed memory. The communication topology is represented as a composition of building blocks in a data-flow graph, partitioned into distributed groups.

for building FastFlow parallel patterns and, more generally FastFlow streaming topologies [163, 6]. Parallel building blocks are: pipeline (`ff_pipeline`), task-farm (`ff_farm`), and all-to-all (`ff_a2a`). Sequential building blocks are: standard node (`ff_node`), multi-input/output node (`ff_minode/monode`), and the node combiner (`ff_comb`). They can be combined and nested in different ways, forming either acyclic or cyclic concurrency graphs, where nodes are FastFlow concurrent entities and edges are communication channels carrying data pointers. Building blocks mainly target system programmers who want to build new frameworks or RTSs. All high-level parallel patterns offered by the FastFlow library have been implemented using building blocks. Following the principles of the structured parallel programming methodology, a parallel application (or one of its components) is conceived by selecting and adequately assembling a small set of well-defined building blocks modelling data and control flows. These can be combined and nested differently, forming acyclic or cyclic concurrency graphs.

The original FastFlow release implemented nodes as concurrent entities and edges as communication channels carrying data pointers. The FastFlow runtime system has recently been extended to deploy FastFlow programs in distributed-memory environments [161]. The distributed RTS has been implemented by leveraging building blocks and extending them to preserve the original data-flow streaming programming model. By introducing a small number of edits, the programmer may port shared-memory parallel FastFlow applications to a hybrid implementation (shared-memory plus distributed-memory) in which parts of the concurrency graph will be executed in parallel on different machines according to the well-known SPMD model. Such refactoring involves

introducing distributed groups (*dgroups*) that identify logical partitions of the building blocks that compose the application streaming graph according to a small set of graph-splitting rules. An example of a FastFlow application partitioned in distributed groups is given in Figure 4.2. Inter-dgroup (i.e., inter-process) communications leverage raw TCP/IP or MPI, whereas intra-dgroup communications use highly efficient lock-free shared-memory communication channels [5]. A distributed FastFlow application is accompanied by a JSON configuration file that defines the dgroup-to-machine mapping and a set of non-functional properties helpful to tune the application execution, e.g., the communication protocol, the thread-to-core affinity setting for each dgroup, the amount of transparent message batching for outbound communications. Concerning program launching, the `dff_run` software module is responsible for deploying and launching the FastFlow distributed application. It launches the application processes, each with the appropriate parameters (e.g., dgroup name), following the mapping dgroup-host described in the JSON configuration file. For applications using the MPI library as a communication protocol, the `dff_run` is just a wrapper of the well-known `mpirun` launcher.

In the distributed FastFlow RTS, data serialisation can be carried out in two ways. The programmer may select the best approach between the two for each data type flowing into the inter-group channels (i.e., the data types produced/received by the edge nodes of a dgroup). The first approach employs the *Cereal* serialisation library [79]. It can automatically serialise base C++ types and compositions of C++ standard-library types; it just requires implementing simple mapping functions for custom or user-defined types. The second approach lets the user specify its serialisation and deserialisation function pair. When feasible, this latter might help avoid the need for extra copies by the serialisation process itself. This work uses *Cereal* based serialisation, which guarantees portable representations over different architectures.

FastFlow targeted `x86_64`, `ARM` and `Power` processors. Porting the FastFlow's low-level thread-based RTS to support RISC-V systems has been painless. We just needed to extend the machine-dependent implementation of the *Write Memory Barrier* and the ticks cycle counter used for implementing the Single-Producer Single-Consumer lock-free queue and the non-blocking RTS behaviour. The blocking RTS is instead implemented atop standard POSIX condition variables.

4.2.2 Three *FastFL* use case examples

Three DML communication topologies are considered to showcase the flexibility of the proposed approach (see Figure 4.1): master-worker, a mesh made of peers, and a tree-based structure. These three topologies have been described and discussed in Section 4.1.1. The first two are use cases suitable for FL tasks, while the latter is for EI. Three software implementations are created, one for each use case, all relying on FastFlow to describe the architecture and communication topology and the Cereal library to perform serialisation. ML operations such as loading, training, and testing a DNN,

are based on the PyTorch library via the C++ API, namely libtorch. Models are specified as the generic PyTorch `torch::nn::Module` class, allowing *FastFL* to work with any PyTorch model.

The master-worker and the peer-to-peer topologies are based on the all-to-all building block (`ff_a2a`), which efficiently models together the reduce and broadcast operators required by both workflows and rounds are modelled as cycles (i.e., feedback channels) created by activating the `wrap_around` feature. The `ff_a2a` building block defines two sets of nodes connected according to the shuffle communication pattern; each node in the first set is thus connected to all nodes in the second set. A master-worker topology is obtained if the first set contains the aggregator node and the right set all worker nodes. The `ff_a2a` building block is also exploited to create the mesh topology. Peers are split into a modified aggregator, which additionally trains and includes a model trained on local data, and a distributor, which sends the local model to the other peers' aggregators. All aggregators are assigned to the left set and all distributors to the right set of the `ff_a2a` building block. Aggregator and distributor of the same peer are tied together by assigning them to the same distributed group. Leveraging the message routing options, each aggregator sends its aggregated model only to its distributor; in contrast, each distributor forwards the aggregated model to all other aggregators on the feedback channel. All nodes are designed for modularity and implemented as derived `ff_node` C++ classes. Multi-input multi-output nodes, i.e., aggregator node in the master-worker topology and peer nodes in the mesh topology, are implemented by exploiting the combiner building block (`ff_comb`) by combining a multi-input adapter forwarding messages and a multi-output node containing the logic. The aggregation methodology is specified as a separate policy class for future extensibility, e.g., `FedAvg` for federated averaging. Similarly, workers allow specifying the training strategy, i.e., optimiser, to use as (automatic type inferred) template argument. The serialisation of models with Cereal requires specifying overloaded `save` and `load` functions for the specific class defining the PyTorch model. These are implemented as simple wrappers around the `load` and `save` functions offered by the PyTorch frontend for model (de-) serialisation. The two abovementioned FL use cases exploit a simple Multilayer Perceptron (MLP) comprised of three fully connected layers as DNN. The selected task is training to recognise digits from the MNIST dataset[1]. As hyperparameters, cross-entropy is used as a loss function and SGD as the optimiser, with a learning rate of 0.01 and momentum of 0.5. Concerning the MNIST dataset, the training set is split into equally sized random subsets assigned to each worker; this has been done to simulate a genuine federation in which each client possesses only a subset of the whole data distribution.

The EI is showcased by using a tree topology for modelling a control-room use case, which aims to solve the underlying problem of raising alerts for man-on-the-ground

[1] http://yann.lecun.com/exdb/mnist/

events. Leaf nodes containing multiple cameras feed into local pre-trained standard YOLOv5 networks pre-processed video images by resizing them to the correct resolution (e.g., 640x640), adding a border, and ensuring the image has the correct tensor shape. After applying the YOLOv5 model, the aggregation nodes post-process the result to extract the bounding boxes with a classification score larger than a given threshold (detecting man-on-the-ground events) and output aggregating results along the tree until the control room located at the root. The inference runs on a 2-level tree topology with 148 frames at each leaf node with levels connected using the all-to-all building block (`ff_a2a`). The subsequent ff implementation resulted in nesting two levels of all-to-all BB, similar to an extended master-worker topology. The network provided by the YOLOv5 maintainers is exported into a TorchScript archive to work with PyTorch-based networks in a C++-only environment. TorchScript archives have the advantage of serialising the Python code describing the network and the necessary weights. The Python code is represented in the archive using a subset of Python itself (TorchScript is the name of this subset); weights are serialised in pickle format. When a TorchScript archive is deserialised, the Python is just-in-time compiled, and the binary is dynamically linked into the running program.

4.2.3 Experimental results

Each *FastFL* use case is tested on three hardware architectures hosted on the two research systems described below. The Monte Cimone [19] system is the first physical prototype and test-bed of a complete RISC-V (RV64) compute cluster, integrating not only all the essential hardware elements besides processors, namely main memory, non-volatile storage, and interconnect, but also a complete software environment for HPC, as well as a full-featured system monitoring infrastructure. Monte Cimone comprises eight computing nodes running Linux Ubuntu 21.04 and is enclosed in four computing blades. Each computing node is based on the U740 SoC from SiFive and integrates four U74 RV64GCB application cores, running up to 1.2 GHz and 16GB of DDR4, 1 TB node-local NVME storage, and PCIe expansion cards. Each Monte Cimone computing node integrates separated shunt resistors in series with each of the SiFive U740 power rails as well as for the on-board memory banks, which can be leveraged to attain fine-grained power monitoring of power rails, including the core complex, IOs, PLLs, DDR subsystem and PCIe one. The power rails current and voltage are monitored by a PXIe-4309 board from National Instruments. The PXIe-4309 module features 8 ADC devices and supports a maximum data acquisition rate of 2 MSamples per second. Collected current and voltage traces are then post-processed to obtain an average power consumption for each application run. The EPI-TO system is a modular cluster designed to experiment with the technologies under development in the European Processor Initiative (EPI). It includes an ARM-v8 module (4 nodes), an x86-64 module (4 nodes), and a RISC-V module (2 nodes). The x86-64 module comprises 4 Supermicro servers, each including 2 Intel Xeon Gold 6230 CPU (20-cores@2.10GHz) sockets, and 1536GB RAM. The

Table 4.2: Experimental *FastFL* setting. The chosen model and dataset general information are reported for the three tested topologies (master-worker, peer-to-peer, tree). The forward and backward pass FLOPs are estimated using the PyTorch profiler.

Topology	Model	# param.	Aggregator data	Worker/Peer data	Forward FLOPs	Backward FLOPs
Master-worker	Custom MLP	52.6K	10K test images	7.5K train images	105.1K (1 image)	109.8K (1 image)
Peer-to-peer	Custom MLP	52.6K	—	7.5K train + 10K test images	105.1K (1 image)	109.8K (1 image)
Tree	YOLO v5n	1.87M	—	148 frames	2.68G (1 frame)	—

ARM-v8 module comprises 4 ARM-dev kits, each including one socket Ampere-Altra Q80-30 (80-core@3GHz), 512GB RAM, 2 x NVIDIA BF-2 DPU, and 2 x NVIDIA A100 GPU. The Intel and Ampere servers are connected via an Infiniband HDR and a 1Gb/s Eth networks. The GPUs are not used in the present experiments. All nodes run Linux Ubuntu 20.04 and share a high-performance BeeGFS file system. All the servers are set to use the "performance governor" mode. The RISC-V module is composed of 2 servers identical to Monte Cimone servers.

Many steps have been taken to ensure a fair comparison between the different architectures since they have different degrees of computational power and maturity: x86-64 and ARM-v8 platforms are server-grade machines, while RISC-V is still a young and experimental embedded-like platform (even if some more HPC-oriented prototypes, such as the Ventana processors, are being actively developed). First of all, the number of cores available to each machine has been taken into account; since the least powerful platform from this point of view is SiFive RISC-V (4 cores per node), all the processes in the Intel and Ampere experiments are capped consequently so that precisely 4 cores would be assigned to each of them: this is done through the `taskset` command. Monte Cimone offers a maximum of 8 computational nodes, so the proposed experiments are calibrated on a federation of a maximum of 8 workers. Since the number of Intel and Ampere servers is limited to 4, two nodes per server are placed when running 8 node configurations. In such cases, the node threads are placed in different areas of the processors and near the memory banks to limit the interference between them as much as possible. Due to the invasive PyTorch threading policy, only using the `taskset` command is insufficient in this scenario. Even if the process is restricted to 4 cores, PyTorch still creates a thread pool of as many threads as available cores. This behaviour can lead to many threads swapping between each other, which can spoil PyTorch's performance and subsequently ruin the fairness of the comparison. This problem has been resolved by setting `OMP_NUM_THREADS=4`, thus limiting the OpenMP threads created by PyTorch to 4, making it behave on Ampere exactly like on the SiFive platform. The `MKL_NUM_THREADS=4` has also been set to prevent the MKL library from creating more threads than the assigned cores to obtain the same behaviour on Intel. Another precaution to exploit the few cores available at maximum is to use FastFlow's TCP backend instead of the MPI one. The motivation behind this is the computational behaviour of the OpenMPI blocking `receive`. When issued, it actuates a busy waiting policy, directly occupying a core at 100% of its capability, which would skew the energy results.

Moreover, in the context of SiFive RISC-V, this means wasting 1/4 of the available computational power. We thus resorted to the TCP backend. Lastly, a shared workload for the different experiments has to be set. Table 4.2 summarises the data details used and estimated model complexity. Due to the massive difference in the computational performance of the various machines, the choice has been made with particular attention to the SiFive RISC-V. We have chosen to train the MLP on MNIST for 100 epochs, subdivided into 20 federated rounds composed of 5 epochs each. This choice allows the learning curve to stabilise and, at the same time, allows the assessment of relevant measurements (e.g., communication and computation costs). We also experimented with a larger ML workload: training a ResNet18 network on the CIFAR10 dataset. Unfortunately, such a configuration required 24 hours on the RISC-V processor to complete a single training epoch, which is incompatible with the experimental setting.

A brief video of 148 frames containing people moving has been chosen for the YOLO-v5 experiments. Note that these choices, especially using only a subset of the available cores, do not hinder x86-64 and ARM platforms in favour of RISC-V. In fact, due to the chosen workload, the total computation time increases with the number of involved cores. This effect is due to the reduced benefits gained in parallelising the training of a small model and the additional costs required by thread handling and synchronisation (particularly under the Python threading model).

The results of the experiments are reported extensively in Table 4.3. As can be seen, SiFive is an order of magnitude slower than the other systems, being almost always between 25-35 times slower than Intel and Ampere. This is to be expected due to the platform's young stage of development, the absence of optimised libraries for deep learning-specific computations, and the lack of vectorial accelerators. The lack of vectorial units is particularly detrimental to the SiFive processor. The code running on the other two platforms is compiled with libraries explicitly optimised for exploiting vectorial units (the oneMKL library for Intel and the Arm Performance Library for Ampere). On the other hand, the gap in performance between Intel and Ampere is negligible: Ampere is almost as fast as Intel in the computation while consuming an order of magnitude less power. The difference in power consumption is discussed in more detail in the following. However, it can be easily assessed that the Ampere system, at least in this restricted context, is energetically far more optimised than the Intel one; SiFive, on the other hand, presents an energy consumption of 2-5 times Ampere's, while Intel is up to of an order of magnitude.

From a scaling perspective, all experiments behaved as expected: the recorded execution times are congruous with the weak scalability law since they remain constant or slightly increase with the number of processes; this confirms the goodness of the software and the capabilities and flexibility of the FastFlow runtime.

The heterogeneous experiments are another strong point in favour of the FastFlow capabilities: thanks to the Cereal serialisation backend, it is possible to create a federation in which different workers are hosted on different computational systems. This

Table 4.3: Computational performance obtained by *FastFL* with the three tested topologies (mater-worker, peer-to-peer, tree) on the three tested microarchitectures (x86-64, ARM-v8, RISC-V). The number of OpenMP (and MKL on Intel) threads has been set to 4 and bound to 4 physical cores, and each result is averaged over 5 runs. The hybrid Intel-Ampere experiments are executed by allocating processes equally on each microarchitecture's cluster.

(a) MNIST master-worker training results. Performance metrics are collected on 20 federation rounds of 5 training epochs each (100 epochs total); each client is assigned 1/8 of the entire dataset.

	master + 2 workers		**master + 4 workers**		**master + 7 workers**	
	time (s)	energy/worker (J): Δ (tot)	time (s)	energy/worker (J): Δ (tot)	Time (s)	energy/worker (J): Δ (tot)
x86-64 (Intel)	23.84	973 (1992)	**23.56**	1011 (2069)	**24.38**	1049 (2146)
ARM-v8 (Ampere)	**23.33**	**133** (**483**)	25.66	**146** (**531**)	25.86	**148** (**535**)
RISC-V (SiFive)	674.47	269 (2562)	673.70	269 (2560)	687.03	274 (2610)
Intel-Ampere	29.50	—	29.55	—	33.34	—

(b) MNIST peer-to-peer training results. Performance metrics are collected on 20 federation rounds of 5 training epochs each (total 100 epochs); each client is assigned 1/8 of the entire dataset.

	2 peers		**4 peers**		**8 peers**	
	time (s)	energy/peer (J): Δ (tot)	time (s)	energy/peer (J): Δ (tot)	time (s)	energy/peer (J): Δ (tot)
x86-64 (Intel)	**23.15**	2082 (4261)	**24.05**	2162 (4422)	**24.95**	2210 (4522)
ARM-v8 (Ampere)	24.39	**169** (**535**)	24.90	**173** (**546**)	26.65	**185** (**585**)
RISC-V (SiFive)	819.35	409 (3195)	815.55	407 (3180)	933.62	466 (3641)
Intel-Ampere	45.20	—	39.13	—	50.88	—

(c) YOLO tree-based inference results. Performance metrics are collected by assigning each leaf a 148-frame video.

	root + 2 leaves		**root + 4 leaves**		**root + 7 leaves**	
	time (s)	energy/leaf (J): Δ (tot)	time (s)	energy/leaf (J): Δ (tot)	time (s)	energy/leaf (J): Δ (tot)
x86-64 (Intel)	**19.76**	1520 (2389)	**19.38**	1491 (2343)	**19.01**	1462 (2298)
ARM-v8 (Ampere)	37.16	**291** (**848**)	39.88	**312** (**910**)	43.15	**338** (**985**)
RISC-V (SiFive)	1201.51	841 (4926)	1205.77	844 (4943)	1212.77	848 (4972)
Intel-Ampere	35.65	—	35.65	—	36.10	—

feature is not trivial since moving data from one system to another usually implies conversion issues. No compatibility issue emerged in these experiments, and the heterogeneous cooperation worked smoothly from a computational performance perspective. We successfully ran a distributed master-worker training across all the computational

Table 4.4: Comparison between the tested systems' (x86-64, ARM-v8, RISC-V) CPU-only power consumption per FLOP and thermal design characteristics. The Joule/FLOP values are obtained as the mean of 3 different master-worker configurations.

	Δ Energy/FLOP	**Energy/FLOP**	**Avg power (idle)**	**TDP/socket**
x86-64 (Intel)	6.3 nJ	12.8 nJ	44 W	125 W
ARM-v8 (Ampere)	0.9 nJ	3.2 nJ	15 W	250 W
RISC-V (SiFive)	1.7 nJ	15.9 nJ	3.4 W	5 W

platforms exploited in this study (Intel, Ampere, and SiFive), highlighting the proposed software stack's flexibility and compatibility features. This fact is crucial since the federation of different entities cannot come with the requirement that all entities employ the same underlying computational infrastructure.

While not explicitly relevant to this discussion, it is worth reporting that the measured classification accuracy of the tested systems acts as a sanity check. All the proposed FL architectures achieve the expected learning performances. Specifically, the MLP model achieved more than 95% of accuracy in all configurations, reaching up to 97% in most runs (YOLOv5 performances are not relevant here since pre-trained models are used).

Finally, to give a perspective to the presented performance, one of the proposed scenarios (the master-worker structure with 4 workers) is reproduced with OpenFL, one of the mature FL frameworks from related work. To further highlight the effort towards the RISC-V developers community, OpenFL is ported to this computational platform. This porting required recompiling in ad-hoc ways the software dependencies (ninja, openblas, grpc/grpcio, and crytography). As reported in table 4.3a, the proposed SiFive implementation can complete the training of 100 epochs over the MNIST dataset in 673.70 seconds on average (11 minutes and 7 seconds). Conversely, OpenFL, with the same model, hyper-parameters, data pre-processing and distribution, achieved an average running time of 2,486 seconds (41 minutes and 26.42 seconds). Additionally, the same experiment is repeated on the x86-64 platform: even in this case, the efficiency of the proposed implementation is confirmed (23.56 seconds) with respect to OpenFL (59.15 seconds). The difference between the two execution times is stunning, and its motivations have already been discussed throughout the entire paper; this is just another example of how a more high-performance-orientated FL framework would benefit the overall FL research environment.

Looking at Table 4.4, it can be assessed that the three tested systems belong to different computing classes. Indeed, the table reports idle CPU and system power consumption as well as the Thermal Design Power (TDP) per node and system type. The Ampere and Intel CPUs are server processors with TDPs of over 100 Watts, while the SiFive CPU is an embedded-class processor with TDPs below 5 Watts. Table 4.3 reports the dynamic energy (Δ) as well as total energy per worker/peer/leaf when training the

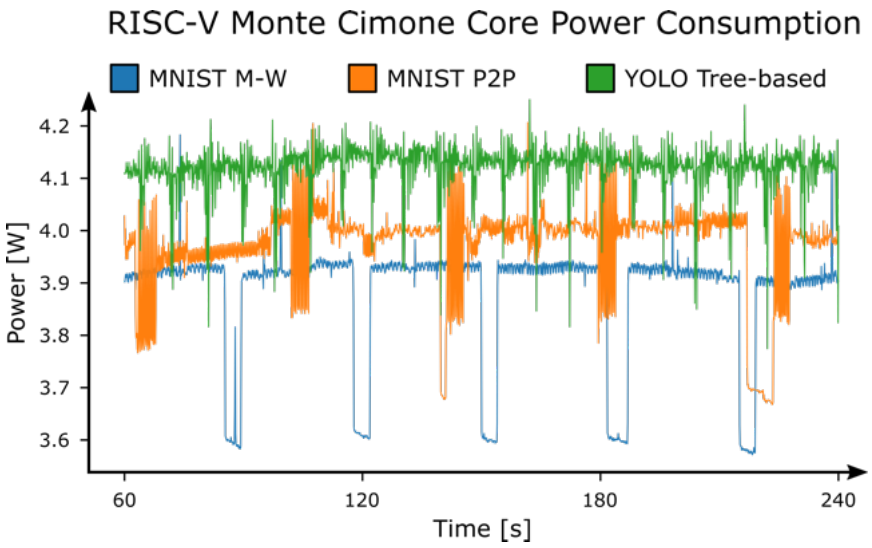


Figure 4.3: Three minutes of power consumption traces of *FastFL* training/inference on the Monte Cimone RISC-V cluster. It can be seen that different communication topologies tested (master-worker, peer-to-peer, tree) imply different power consumption patterns.

MNIST model (a,b) and inference of the YOLOv5 model (c). The Intel and Ampere CPUs have comparable performance, but the Ampere CPU has a lower power consumption of almost an order of magnitude. This fact makes the Ampere CPU the most performing one and the most energy-efficient one in this study. When compared to the Intel CPU, the Ampere one attains an average reduction for delta and total energy of 8.3x (4.3x) for the MNIST master-worker case, 5.7x (3.7x) for the MNIST peer-to-peer case and 4.7x (2.6x) for the YOLOv5 tree-based case. The SiFive CPU, compared to the Ampere one, attains an average delta (total) energy consumption that is only 1.9x (5.0x) higher for the MNIST master-worker case, 2.4x (6.0x) for the MNIST peer-to-peer case, and 2.7x (5.4x) for the YOLOv5 tree-based case.

While this is an unexpected result, given the absence of SIMD/Vector extension for the SiFive compute nodes, the significantly longer compute times drive up the consumed energy. We expect improved results once new silicon implements the RISC-V ISA vector extensions. Figure 4.3 reports an extract of three minutes of the CPU power consumption (in the y-axis) for a single SiFive compute node for the three experimental settings. We can notice that the MNIST master-worker (MNIST M-W in the figure) achieves significantly lower power consumption than the MNIST peer-to-peer (MNIST P2P in the figure) while having a lower time to solution. The additional computation can explain this fact due to each peer computing its global model and the increased traffic to handle peer-to-peer communication. Data transfers are visible in both traces as lower power consumption segments, each corresponding to a federated round. The

YOLOv5 tree-based experiment has a higher computational intensity due to the higher model complexity, which translates into a higher peak power consumption.

When comparing the different configurations, an overall increase in the time-to-solution can be noticed while increasing the number of workers/peers/leaves, which directly translates into an increase in the energy consumption of the models. The scalability is different for the peer-to-peer and master-worker communication topologies, where the master-worker time-to-solution overhead saturates with the 4 workers case; in contrast, the peer-to-peer time-to-solution overhead continues to scale with the number of peers. This behaviour is expected due to the latter's exponential cost of communication scalability.

In Table 4.4, the different processors' energy efficiency (Joules per floating-point operation) is reported by extracting the floating-point operations for each training epoch and the energy consumed by a single worker in the master-worker scenarios. The reported values are calculated following the equation:

$$E_{\text{FLOP}} = \frac{P \cdot t_{\text{epoch}}}{[N_{\text{images}} \cdot (N_{\text{Forward}} + N_{\text{Backward}})]}$$

where P is the mean consumed power, t_{epoch} the time to train one epoch, N_{images} the number of images, and N_{Forward}, N_{Backward} the profiled FLOPS for forward and backward pass. Taking into account only the energy effectively consumed by the computation (Δ energy), both Ampere and SiFive result to be more energy efficient than the Intel CPU, respectively 7x and 3.7x; this is good, especially for the SiFive platform, given the system's novelty. On the other hand, if the whole system consumption is taken into account, then the results are dramatically different: in this case, Ampere and Intel are more energy efficient than SiFive, by 5x and 1.2x, respectively; this is due to the higher execution time required by the SiFive platform to complete the same amount of computation done by the other two processors.

4.3 A federated learning domain-specific language

The following Section proposes a top-down approach to implementing an FL (or EI) system. The typical FL abstraction stack is modified by building upon the *FastFL* approach, proving that a more flexible and powerful FL framework architecture is possible. This approach deals with high-level issues, such as modelling distributed computation, and low-level issues, such as software portability to different microarchitectures. The process starts with defining the system through a formal language describing distributed processes, an adapted version of the RISC-pb^2l [6] language. This definition is then implemented by mapping it to the building blocks of the FastFlow parallel library [7]. On top of the RISC-pb^2l language, a DSL is then formalised. This FL-oriented DSL is inspired by state-of-the-art ML frameworks, in which an easy-to-use Python wrapper wraps the complexity of the underlying high-performance C/C++ implementation. The

proposed DSL acts then as a Python wrapper of the underlying *FastFL* implementation, allowing it to exploit its potentialities and performances, abstracting from the technicality of its use.

4.3.1 The RISC-pb^2l formal language

The first step in creating an abstraction layer wrapping *FastFL* is to design a high-level modelling language. Such a language should be sufficiently abstract to allow modelling a wide range of distributed computations while still being sufficiently specific to be easily applicable in different contexts. With this aim in mind, the RISC-pb^2l formal language [6] for parallel processes is chosen and adapted to describe distributed ML workloads. RISC-pb^2l is based on basic logical units called building blocks, recurrent data-flow compositions of concurrent activities working in a streaming fashion, which are the primary abstraction layer for building parallel patterns and, more generally, streaming topologies [163, 6]. The RISC-pb^2l building blocks match the FastFlow building blocks in a one-to-one fashion, thus allowing for a straightforward implementation of a system given its formal specification. The FastFlow software was born in 2010 and was developed along the RISC-pb^2l language. They are two sides of the same coin: that is the reason why it is possible to map one formalism into the other so straightforwardly. This fact makes FastFlow an already available high-performance software implementation of RISC-pb^2l, which is also already compatible with the RISC-V ISA. These facts were crucial for choosing RISC-pb^2l as language abstraction for this research idea. The translation from RISC-pb^2l into FastFlow is as follows: the sequential and sequential wrappers can be mapped into a `ff_node`, the spread and 1-to-N operators to a `ff_loadbalancer`, the reduce and N-to-1 to a `ff_gatherer`, the feedback into a `wrap_around`, the parallel into a `ff_farm` and, finally, the pipe into a `ff_pipeline`. As can be seen, the correspondence between the two formalisms is almost 1-to-1. Right now, the translation from the formalism to the code has to be done manually, but a small compiler can do such a job. This concept will be discussed later in Section 4.3.2.

However, since RISC-pb^2l is born for modelling parallel computation, it must be adapted to handle the distributed workflows involved in DML. The RISC-pb^2l language is thus made location-aware by introducing a *distribute* building block, which expresses the distributed computation of a function on a set of nodes specified via a superscript. Table 4.5 reports the syntax and description of all RISC-pb^2l building blocks used in this dissertation. Parallelism is no longer specified in terms of the number of threads but in sets of nodes, maintaining a coherent semantic. Formally, the numerical subscripts used by the *parallel* operator is replaced with a superscript referring to sets of nodes. The switch to superscripts underlines the semantic change from the original RISC-pb^2l formal language. The advantage of using such a formalism is the possibility of designing and discussing the distributed system clearly and structured before its implementation. Furthermore, this formalism allows discussion of a designed computation's computational properties and applies optimisations before the implementation

Table 4.5: Brief description of the RISC-pb^2l building blocks. The sequential and parallel wrappers constitute the basic elements of the language, effectively abstracting the domain-specific computations that are then combined through the other operators to model complex parallel (and distributed) computations.

Syntax	Semantics
$((\ f\ ))$	**Sequential wrapper** Wraps any sequential code into a RISC-pb^2l "function".
$(\mid f \mid)$	**Parallel wrapper** Wraps any parallel code into a RISC-pb^2l "function".
$[\mid \Delta \mid]^N$	**Distribute** Computes $\mid N \mid \Delta$ distributively on the node set N producing $\mid N \mid$ outputs.
$\Delta_1 \bullet \ldots \bullet \Delta_n$	**Pipe** Uses n different programs as stages to process the input data items and to obtain output data items.
$\left(_g \triangleright\right)$	**Reduce** Computes a output item using an l level ($l \geq 1$) k-ary tree. Each node in the tree computes a k-ary function g.
$(_f \triangleleft)$	**Spread** Computes n output items using an l level ($l \geq 1$) k-ary tree. Each node in the tree computes a k-ary function f.
$\triangleleft_{D-Pol}$	**1-to-N** Sends data received on the input channel to one or more output channels. $D - Pol \in [Unicast(p), Broadcast, Scatter]$.
$\triangleright_{G-Pol}$	**N-to-1** Sends data from the n input channels on the single output channel. $G - Pol \in [Gather, Gatherall, Reduce]$.
$\overleftarrow{(\Delta)}_{cond}$	**Feedback** Routes output data y back to the input channel according to $Cond(x)$.

phase. The interested reader is encouraged to refer to [6] for the complete set and detailed composition of the original RISC-pb^2l formal language.

By using the exposed RISC-pb^2l formalism, the master-worker FL process can be described as

$$((\text{init})) \bullet \overleftarrow{\left([|(|\text{test}|) \bullet (|\text{train}|)|]^W \bullet \left(_{\text{FedAvg}} \triangleright\right) \bullet \triangleleft_{\text{Bcast}}\right)_r}$$

where ((init)) is the function initialising the communication graph and creating the starting models, W is the set of workers, and r is the condition checking if the prefixed number of rounds has been reached. On the other hand, the peer-to-peer mesh FL

process can be formalised through RISC-pb^2l as

$$[|((\text{init}))|]^P \bullet \overleftarrow{\left(\left[\left|(|\text{test}|) \bullet (|\text{train}|) \bullet \lhd_{\text{Bcast}} \bullet \left(_{\text{FedAvg}} \rhd\right)\right|\right]^P\right)_r}$$

Finally, the three-level tree topology can be modelled through RISC-pb^2l as

$$((\text{init})) \bullet \overleftarrow{\left([|\text{infer}|]^L \bullet (_{\mathscr{F}} \rhd) \bullet [|\text{combine}|]^C \bullet (_{\mathscr{F}} \rhd) \bullet ((\text{alert}))^R\right)_\infty}$$

where L, C, R are the sets of leaf, control and root nodes, and $\mathscr{F}$ is the function routing to the father node.

As can be seen, the two formalisations are similar, which is not casual; it is possible to prove logically that the two formulas yield the same outputs if given the same inputs, modulo a different number of computations and communications. This fact can be easily seen by comparing the last two terms of both feedback blocks; with a bit of rewriting, it is possible to state that

$$(_{\text{FedAvg}}\rhd)\bullet \lhd_{\text{Bcast}} \equiv [|\lhd_{\text{Ucast}_A}|]^W \bullet (_{\text{FedAvg}}\rhd)$$

and

$$[|\lhd_{\text{Bcast}} \bullet (_{\text{FedAvg}}\rhd)|]^P \equiv [|\lhd_{\text{Bcast}}|]^P \bullet [|\rhd_{\text{FedAvg}}|]^P$$

With this new formulation, it is easy to see that, even if they are equivalent output-wise, these two computations exploit different amounts of communications ($[|\lhd_{\text{Ucast}_A}|]^W$ vs. $[|\lhd_{\text{Bcast}}|]^P$) and computations ($(_{\text{FedAvg}}\rhd)$ vs. $[|\rhd_{\text{FedAvg}}|]^P$). This result implies that the two computations are mathematically equivalent output-wise: the two communication topologies produce the same final ML model, with the same learning performances, assuming the same hyper-parametrisation and modulo the differences in communication involved in the process. This simple analysis exemplifies the potential of using a formal tool such as RISC-pb^2l for modelling and discussing distributed systems. Given the two above descriptions, it is straightforward to translate them into ff programs.

4.3.2 High-level federated learning modelling

Having successfully modelled the three provided use cases with the RISC-pb^2l formal language exposed some critical aspects of this tool. On the one hand, RISC-pb^2l is excellent at modelling distributed workloads but is not intended for a large audience. RISC-pb^2l is a theoretical tool, subject to future investigation, changes, and updates. Furthermore, it is not straightforward to use, requiring the user's theoretical knowledge and abstraction efforts, which pose a significant barrier to its use. On the other hand, modelling a DML scenario with RISC-pb^2l does not automatically provide a working implementation. The user is still in charge of manually translating the abstract RISC-pb^2l building block sequence to a practical FastFlow implementation, dealing with all the

low-level coding aspects involved. It should also be considered that the *FastFL* approach entirely relies on a C/C++ software stack. While this design choice preserves computational efficiency, it also implies programming skills that may be outside the common knowledge of ML practitioners. All these factors introduce a huge usability overhead compared to other off-the-shelf FL frameworks, making *FastFL* not the best choice for typical deployments.

Inspired by state-of-the-art ML frameworks, a Python wrapper can be exploited to overcome these burdens. As done by PyTorch, TensorFlow, Sci-Kit Learn, and many others, the high-performance C/C++ implementation is wrapped by a simple, easy-to-use Python wrapper. This approach has the advantage of abstracting the ML practitioner from the low-level details of the implementation while allowing fast and straightforward code writing. Furthermore, Python is the de facto standard programming language for ML workloads, thus not requiring ML practitioners to learn another language to use *FastFL*. Guided by these principles, a Python wrapper for *FastFL* is proposed. It is designed as a top-level abstraction on top of RISC-pb^2l, which abstracts the FastFlow backend. How this abstraction stack is concretely turned into a working FL framework is described in Section 4.3.3. In the following, the proposed Python interface is formalised and discussed.

The creation of the *FastFL* Python wrapper follows two different principles. On the one hand, it should be coherent with the RISC-pb^2l syntax, meaning expressing a significant subset of all the possible computations formalisable through RISC-pb^2l, if not all. As a reference, the three communication topologies discussed in Section 4.3.1 are considered the minimum support target. The first step is to create a sufficiently simple yet expressive enough RISC-pb^2l Python interface. The sequential and parallel wrappers are modelled by a single class, namely `Wrapper`, containing the actual business code to be run on the systems. The two wrappers have been merged since their differences are insignificant at this level of abstraction. Each other operator involved in modelling the selected use cases is then modelled as a different Python class maintaining its original name, apart from the distributed wrapper, which is renamed `Parallel`, indicating that the same function is executed in parallel on different devices. All classes modelling the behaviour of a RISC-pb^2l building block are derived from the base abstract class `BuildingBlock`. Each class contains template C/C++ FastFlow code parametrised at runtime based on the provided arguments.

On the other hand, the proposed Python interface should have simple syntax to allow a seamless construction of FL topologies. To this aim, a list-based syntax is chosen, similar to the one adopted by common ML frameworks. The FL system is then described as a list of BuildingBlock objects, each appropriately parametrised, eventually containing sublists of other building blocks. This structure defines a `FLGraph` object, an abstract description of an FL learning process ready to be materialised. Until the materialisation, this object can be modified freely, allowing the automatic parametric building of FL topologies.

Given the above definitions, a master-worker FL process can be formalised as

```
FLGraph([
 Wrapper("Initialisation"),
 Feedback([
 Parallel([
 Wrapper("Train"),
 Wrapper("Test")
 ]),
 Reduce("FedAvg"),
 Broadcast(),
 ])
])
```

It can be seen that this code snippet closely resembles the formula RISC-pb^2l master-worker formula given in Section 4.3.1. Similarly, the peer-to-peer mash can be defined as

```
FLGraph([
 Parallel([
 Wrapper("Initialisation")
 ]),
 Feedback([
 Parallel([
 Wrapper("Train"),
 Wrapper("Test"),
 Broadcast(),
 Reduce("FedAvg"),
 ]),
 ])
])
```

while the tree-based EI use case as

```
FLGraph([
 Wrapper("Initialisation_inference"),
 Parallel([
 Wrapper("Inference"),
 Reduce("Father"),
 Wrapper("Combine")
 ]),
 Reduce("Father"),
 Wrapper("Control_room")
])
```

Note that these code snippets are almost all needed to run a complete federation. The federation is ready to run with a few more lines of code defining the DNN model and the dataset.

The proposed *FastFL* Python wrapper can be considered a DSL for FL having seen these experiments. It is a programming language devised explicitly to efficiently model a restricted class of problems unrelated to low-level implementational details. This

fact also implies that this FL DSL is not restricted to model RISC-pb^2l and FastFlow computations but could be used with any other parallel programming framework backend. Exactly as FastFlow abstracts the programmer from the communication backend, allowing a seamless deployment on shared memory or distributed memory through TCP or MPI communication, this FL DSL abstracts the programmer from the backend, concretely implementing the computation. For example, it is possible to envision a workflow-based backend for the FL DSL, which would allow the execution of an FL task as a workflow and exploit all the benefits of this technique.

4.3.3 Practical implementation into *FastFL*

Now that the whole abstraction stack is ready, the automatic top-to-bottom translation must be combined. Once the `FLGraph` object is correctly configured, it is sufficient to call its `compile` method to start the process. Compiling the `FLGraph` means recursively inspecting each `BuildingBlock` constituting it and correctly instantiating the template code associated. Some particular `BuildingBlock` sequences are recognised and merged during this process, thus obtaining specific and optimised code for the most common situations. This process is particularly delicate since the template code is written in C/C++ with FastFlow. Any effort in augmenting the flexibility of the `BuildingBlock` classes thus corresponds to generating correct and efficient C/C++ code from a high-level Python specification. This step is thus the pivot point of the entire process, hiding the efficient C/C++ implementation under the Python DSL's hood.

The compilation and linking steps occur once the C/C++ source code is ready. Such commands are handcrafted carefully, and the library paths can be customised by appropriately configuring *FastFL*. Since this process is done on the local machine, it derives that at the time of writing, *FastFL* can only run on clusters of homogeneous machines. Supporting heterogeneous use cases, as discussed in Section 4.4, requires moving the source code on each machine for the compilation and linking, also considering the differences in local library paths, compiler versions, and software stack in general. Building such a flexible deployment process is a work in progress and one of the main aims of the near future developments of *FastFL*. An example of FL deployment on a heterogeneous cluster of machines in a partially automated way is exposed in Section 5.1 exploiting Singularity containers and the StreamFlow WMS.

Once the C/C++ FastFlow executable is ready, the deployment configuration takes place. FastFlow relies on a JSON configuration file to handle distributed deployments. Such a file declares all the groups participating in the computation (usually one per node), identifying them through an IP address and explicitly indicating which port to use for communication. It also specifies the name associated with each group, which must match the group names inserted in the C/C++ FastFlow code automatically generated. It can optionally specify custom commands to be appended to the distributed execution commands. This JSON configuration is generated automatically based on the *FastFL* runtime configuration, allowing it to quickly move from a shared memory

deployment to a large distributed one without any effort requested by the programmer.

Finally, the provided DNN model is analysed. *FastFL* can handle both DNN definitions and TorchScript-compiled models. In the first case, however, the model is instantiated and compiled into a TorchScript file. This process is done for two reasons. On the one hand, the TorchScript compilation can be configured to optimise the saved model, thus increasing the obtained performance even more. On the other hand, this makes the model definition portable and easily transferrable between different machines. Now that everything is ready, the FL deployment begins, and the FL tasks start.

As seen, *FastFL* takes care of all low-level aspects of the FastFlow backend. This approach allows the ML practitioner to focus on the FL process's high-level details, abstracting from the technicalities of a real-world high-performance deployment. This fact is particularly evident when comparing the coding effort required from the high-level point of view to the actual low-level code produced. For example, the following code snippet produces a master-worker deployment on 5 computational nodes of a cluster:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from python_interface.DSL.flgraph import *
from python_interface.DSL.flgraph.flgraph import *
from python_interface.configuration import Configuration
from python_interface.dataset import Dataset
from python_interface.experiment import Experiment
from python_interface.model import Model

DATA_PATH = "_PATH_TO_DATASET_"
nodes = ["small-0" + str(rank) + ":800" + str(rank) for rank in range(
                                    1, 5)]

class Net(nn.Module):
 ...

ff_executable = FLGraph([
 Wrapper("Initialisation"),
 Feedback([
 Parallel([
 Wrapper("Train"),
 Wrapper("Test")
 ]),
 Reduce("FedAvg"),
 Broadcast(),
 ])
]).compile()

config = Configuration(endpoints=nodes, executable_path=ff_executable)
```

```
model = Model(model=Net(), example=torch.rand(128, 1, 28, 28),
                                    optimize=False)
dataset = Dataset(DATA_PATH)
experiment = Experiment(config, model=model, dataset=dataset)

experiment.run_experiment()
```

Many customisable aspects are hidden in this example to show the simplicity with which it is possible to create a working FL deployment with *FastFL*.

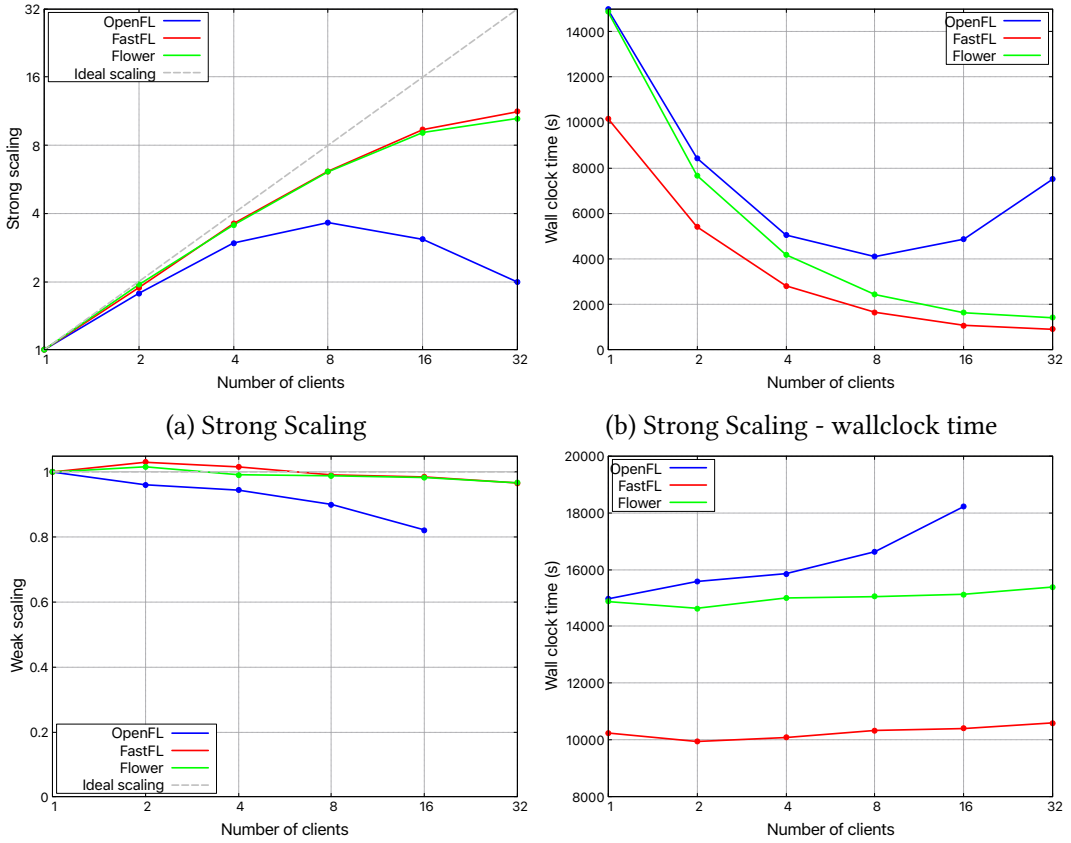
4.4 First *FastFL* experimental results

This section briefly summarises two experimental uses of *FastFL*. First, an experimental investigation of its scaling performance is proposed. *FastFL* weak and strong scaling performance are collected on an HPC infrastructure and discussed. The other two mature FL frameworks used in this dissertation, OpenFL and Flower, are used as comparisons. Following, a practical, real-world use of *FastFL* is proposed. A multi-view pedestrian detection system is implemented, inspired by MVDet [84]. Since this use case fits an EI workload, *FastFL* is exploited for its implementation. The proposed system is then compared to an alternative centralised implementation, always based on *FastFL*.

4.4.1 Scaling performance

The success of many disruptive ML models is not solely due to innovative technical breakthroughs but also to the vast quantity of data on which they have been trained. Fl can be seen as a further step in developing the DML scaling capabilities, offering an innovative tradeoff between the federation size and the learning performance of the final ML model: the more nodes in the federation, the more data is processed simultaneously, but the more unpredictable is the effect on the final ML model. FL thus exhibits two significant costs from the computational perspective: ML model training and parameters communication. These aspects stand out from the ML field. They should be analysed from the distributed computing perspective, where the decomposition of a workload into computing and communication costs is a well-established practice. The following proposes a preliminary study of the scalability property of the most mature and widely used FL framework available on the market, investigating how different software designs can handle federations with varying numbers of clients and varying data quotas. This scenario does not consider learning performance since the focus is on computational performance and FL frameworks' software design.

The literature lacks studies on scalability and flexibility, which are fundamental when leveraging FL in a production-ready deployment where computational efficiency is critical. On the one hand, scalability identifies the FL framework's capability of dealing with a growing number of clients. This property is a crucial feature of such software,



(a) Strong Scaling

(b) Strong Scaling - wallclock time

(c) Weak Scaling

(d) Weak Scaling - wallclock time

Figure 4.4: Strong (top) and weak (bottom) scaling performance comparison of OpenFL, Flower, and *FastFL* training a ResNet18 on the MNIST dataset for 100 epochs on the C3S HPC infrastructure. Both scaling (left) and wallclock times (right) are reported; missing data indicates an execution time greater than 6 hours.

and many studies consider it a future direction for research in FL [92, 21]. In particular, [171] highlights the lack of comparison between FL frameworks as hindering the scientific advancement toward high-performance frameworks. On the other hand, flexibility is crucial when deploying an FL system since it identifies the framework's capabilities to adapt to different scenarios. Almost every framework provides specific use-case examples, but these are rarely tested in realistic scenarios involving different computing power available, network speeds, microarchitectures, and data splitting. For example, according to the systematic literature review of [171], 29 out of 34 reviewed papers that conducted experiments on classification choose MNIST or CIFAR-10 as datasets, and only 11 out of 34 choose a non-iid setting. Some FL frameworks provide examples even with different communication protocols; others have incomplete documentation,

Table 4.6: Strong and weak scaling wallclock execution times (s) obtained by the tested FL frameworks (OpenFL, Flower, *FastFL*) on various federation configurations, ranging from 1 up to 32 clients. Missing data means that the setting required more time to complete than the maximum allowed by the C3S HPC cluster, i.e., 6 hours.

		1 client	2 clients	4 clients	8 clients	16 clients	32 clients
Strong	OpenFL	14967	8433	5051	4104	4870	7517
	Flower	14872	7672	4184	2435	1633	1415
	FastFL	10175	5414	2821	1656	1085	905
Weak	OpenFL	14967	15578	15853	16624	18216	—
	Flower	14872	14636	14999	15046	15128	15385
	FastFL	10249	9951	10090	10340	10407	10607

making deployment very difficult when changing datasets, models, or data processing pipelines.

A widespread FL scenario is identified to produce a reliable and practically applicable analysis [183]. As previously discussed, the most widely used FL scenario is horizontal, cross-silo FL with a master-worker topology. These keywords imply a situation with a small number of institutions (<100), each with high storage capacity, high computational capabilities, fast and reliable network connection, and data that share the same feature space. A central server coordinates the participants, aggregates the clients' models, and broadcasts back the aggregated model. The C3S HPC centre of the University of Turin is used to provide sufficiently reliable computational performance for the experiments. The federation's nodes are allocated to different computational nodes (OmniPath network, 2xintel Xeon CPU E5-2697 v4 per node). A standard ResNet18 is trained as a benchmark workload for 50 federated rounds on MNIST. This model/dataset combination is sufficiently standard in the ML community to provide widely appreciable results. All experiments are deployed on bare metal without any containerisation technology involved.

Figure 4.4 and Table 4.6 report the computational performance of two mature FL frameworks, i.e., OpenFL and Flower, and *FastFL*. OpenFL and Flower are based on similar technologies. They are built entirely in Python and exploit gRPC/protobuf as a communication backend. Refer to Table 4.1 for further details. It is immediately apparent that the two selected commercial FL frameworks exhibit radically different scaling behaviours despite adopting a very similar software architecture. Flower efficiently scales up to the maximum number of nodes tested (32 nodes), while OpenFl's performance radically spoils with federations larger than 8 clients. The apex of this behaviour is the 32-node scenario, in which OpenFL cannot complete the training in the maximum node allocation time allowed by C3S, i.e., 6 hours. However, with restricted federation

(<4 nodes), it can be observed that OpenFL and Flower do not exhibit so radically different computational performance. This fact implies that the baseline performance of both software is determined by the same technological choices made by both frameworks. Consequently, it can be deduced that the different software design choices adopted by the two software play a crucial role in determining the scaling performance: it is evident that OpenFL's communication protocol is unable to efficiently handle a large number of clients despite using the same gRPC/protobuf approach as Flower. Regarding *FastFL*, it can be seen as a Flower competitor: they expose very similar scaling behaviours, modulo a wallclock time offset in favour of *FastFL*. It can be hypothesised that *FastFL* is thus capable of handling large numbers of clients as nicely as Flower while constantly obtaining lower wallclock times thanks to its underlying computational choices (FastFlow backend and C/C++ implementation).

4.4.2 Multiview detection

EI is a research field undergoing strong experimental efforts due to the convergence of many phenomena. The abundance of pervasive computational devices [124] offers lots of spare computational power exactly where data are harvested. Many approaches exploit this latent computing power to process data timely, especially for medical purposes [138, 80]. However, as ML models became increasingly complex and computationally power-consuming, edge devices adapted like in a co-evolution schema [181]. Thus, AI-specialised edge devices have flourished, like TPUs [149], NPUs [160], FPGAs [78], and many others based on innovative computing paradigms, like in-memory computing [85]. All this hardware-related research effort is devoted to increment the efficiency of AI-related computations, trying to mitigate as much as possible the natural computing and battery life limitation of such devices, but many real-world still incorporate only CPUs as compute due to cost, thermal and power constraints [11].

ML-based detection systems can be modelled in many ways, but reliance on DNNs is one constant. Single-view detection [192, 132, 188, 83] addresses the case in which just a single image of a scenario is available for inference at a time. Such methods can be anchor-based or not and can handle occlusion issues by exploiting techniques such as part-recognition, non-maximal suppression, and repulsion loss. Additional information exploited by the inference can be domain-specific, like the detection of heads and feet when searching for people, or can be obtained through the use of particular image acquisition tools, such as RGB-D cameras and LIDAR detectors, to obtain single images correlated with more spacial information. Multiview detection exploits multiple sources of image acquisition to produce a single inference [84, 70, 157] to overcome occlusion problems. The principal research focus in this scenario is aggregating the information retrieved from multiple data sources. Popular approaches target combining multiple single-view detection, aggregating the features extracted from each image, and applying geometrical transformation to the camera outputs.

MVDet (MultiView Detector) is a state-of-the-art multiview detector that identifies

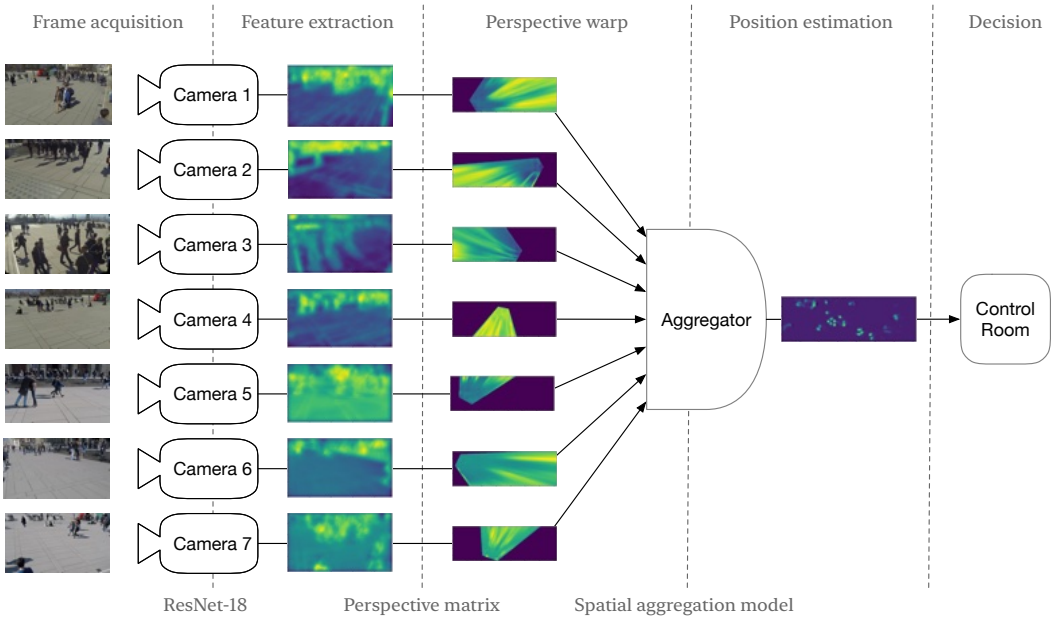


Figure 4.5: Distributed implementation of a multiview detection system with *FastFL*. The workflow starts with each camera acquiring the current time step frame; the frame features are then extracted by a ResNet18 and warped according to the camera perspective matrix directly on the edge; then all the warped feature maps are collected by the aggregator, which aggregates them via the spatial aggregation model, producing the position estimation map; this last result is then sent to the control room for operational decision. This process is repeated iteratively.

persons standing and moving across an open public area. The main idea is to use a trained base model, e.g. ResNet18 or VGG11, to extract the features from each camera frame and transform it via a homography, i.e. perspective warp, projecting it to the bird eye view of the area. The results are then fed into an aggregation model, which detects the position of all the persons in the area. Bringing the MVDet model to a real-world edge environment requires methodological and implementational choices. Starting from the original MVDet code, two different computational stages are identified: a parallelisable one that comprehends the frame acquisition, feature extraction, and perspective warping, and, conversely, a sequential one that is the multiview aggregation. Since the system is synchronous at the frame level, the multiview aggregation is data-dependent on the previous processing steps: it can not start until all the camera frames from the current time step have been acquired and processed. Following the original MVDet paper, the number of camera inputs expected by the aggregator is seven.

Given this computational scenario, two different edge implementations of the MVDet

system are proposed: a more distributed one and a more centralised one. The distributed implementation is depicted in Figure 4.5. It takes full advantage of the model partitioning technique, allocating part of the ML models on the edge devices while trying to parallelise the execution as much as possible by assigning all the parallelisable code to a different camera. Conversely, the centralised implementation fully exploits model offloading, allocating all the computational burden on the server while requiring the camera to acquire the frames. This implementation takes full advantage of the computational power available on the edge, maintaining the computation as near the data as possible while alleviating the computational burden on the aggregation server. The main drawback of this structure is that the feature exchange between the cameras and the aggregator is heavier than the simple frame exchange.

Due to the targeted scenario, i.e. edge devices, energy efficiency is critical to the computation. Commercial DML software is Python-based, requiring computational and memory capabilities that not every edge device can afford. Furthermore, popular DML software is being developed primarily based on user-friendliness and additional security features, such as Homomorphic Encryption and Secure Multiparty Computation, rather than computational performance. While this strategy comes in handy when dealing with powerful computing devices, an edge-oriented DML framework should be as efficient as possible by reducing its consumption of resources (computation, memory, energy). Also, most commercial DML software is very solidly designed to handle only one communication topology, the master-worker one. Any other communication structure would require heavy software modifications, resulting in a not-as-intended use of the frameworks. This use case is particularly well-fit for a tree-based structure, which is not currently implementable with commercial DML software. Due to these two motivations, efficiency and communication topology malleability, *FastFL* is chosen as the DML framework for implementing the proposed system.

The application architecture is based on several building blocks, i.e. specialisations of the `ff_node` class, the primary logical unit exposed by FastFlow, connected in a tree-like structure. At the leaves of the tree, multiple `ff_monode_t`, named *camera* are placed, each representing a different camera in the system. These building blocks are executed in parallel independently. For the distributed implementation, a new frame is read at each time step, its features extracted by a pre-trained base model, i.e. ResNet18, and warped according to a perspective transformation to consider the camera view angle. Conversely, for the centralised implementation, the camera just acquires the current time step frame. Each *camera* sends the result to the next tree level. This level consists of a `ff_minode_t` called *aggregator*, which takes multiple inputs and returns a single output. In the distributed implementation, results collected from all children *Camera* are aggregated by the spatial aggregation model into the final position estimation map. Conversely, for the centralised implementation, the *aggregator* node also takes over the feature extraction and perspective warping for each received camera frame. Finally, the position estimation map is sent to the tree's root, i.e. `ControlRoom` node, where real-time decisions can be taken in a real-world deployment. Figure 4.5

summarises the distributed implementation architecture. In case of multiple open areas, the application supports multiple subtrees, one for each area, each consisting of an *aggregator* with multiple *camera* as children.

The original MVDet experiments are reproduced to prove the proposed approach's effectiveness experimentally, measuring the computation performance of the two implementations. The Wildtrack multiview dataset [39] comprises 7 static cameras capturing views of a public open area with dense groups of pedestrians standing and walking. The dataset provides each camera with the accurate position and view angle and 400 time-synchronised full-HD frames. All experiments are run on the HPC4AI cloud computing facility [8] exploiting 10 virtual machines, each one equipped with 8 64-bit vCPUs mapping to dedicated cores of an Intel Xeon Gold-6230 @2.10GHz processor (Skylake, IBRS), 16GB RAM, 1 Gb/s interconnection network, and running Ubuntu 22.04 as operating system. CPU-only nodes are used to better model the capabilities of edge devices, which often forgo GPUs for cost, energy, and thermal constraints. In contrast, the single-core performance of modern SoC can be in line with that of a vCPU. Each system component (7 Camera, 1 Sync, 1 Aggregator and 1 ControlRoom node) is deployed on a different VM. The time needed to process one complete set of camera frames and produce the estimated positions is assumed as the performance metric. Experiments are replicated 5 times; the mean computing time plus the 95% confidence interval is reported.

The proposed multiview detection systems are tested under different combinations of computational power assigned to the critical software components, i.e. Camera and Aggregator, thus simulating how different edge devices with varying performance impact the system. The computing power is modulated by varying the number of cores available to the different components using taskset and providing hints to the number of threads to spawn to libtorch via the `MKL_NUM_THREADS` and `OMP_NUM_THREADS` environment variables. Specifically, the systems are tested, assigning to each Camera 1, 2, 4, or 8 cores and to the Aggregator 4 or 8 cores for 16 different computational power configurations. Different network conditions are also emulated by limiting the bandwidth available to the different nodes to study how it affects the performance of the proposed system. Indeed, edge devices typically need to rely on slower networks, e.g. cellular connections. Kollaps is a decentralised container-based network emulator, exploiting Docker Swarm (or Kubernetes) to deploy and run distributed computations with specific network topology made of bridges and links with specific upload/download bandwidth, latency, and jitter characteristics enforced using the `traffic control` capability of the Linux kernel. The systems are tested with 10/10 Mbps per Camera and 100/100 Mbps for the aggregator, representing a low bandwidth scenario. The Camera bandwidth is then increased to 25/284 Mb/s, which represents the average upload/download speed achieved by the best 5G network in Italy in 2022 [131] and, consequently, the network bandwidth of the aggregator is increased to 1/1 Gb/s to cope with the aggregated bandwidth from the Cameras. The latter configuration is tested, assigning 4 or 8 cores to the aggregator.

Table 4.7: Computational performance obtained by the proposed MVDet *FastFL* implementation at varying computational power and network bandwidth. The first two scenarios adopt a bandwidth comparable with current Italian cities' 5G performance, while the third simulates a resource-constrained edge deployment.

Aggregator cores	Aggregator bandwidth up/down	Camera bandwidth up/down	Centralised (s/set)	Distributed (s/set)
4 cores	1/1 Gb/s	25/284 Mb/s	15.38	49.68
8 cores	1/1 Gb/s	25/284 Mb/s	11.98	46.89
8 cores	100/100 Mb/s	10/10 Mb/s	14.22	108.79

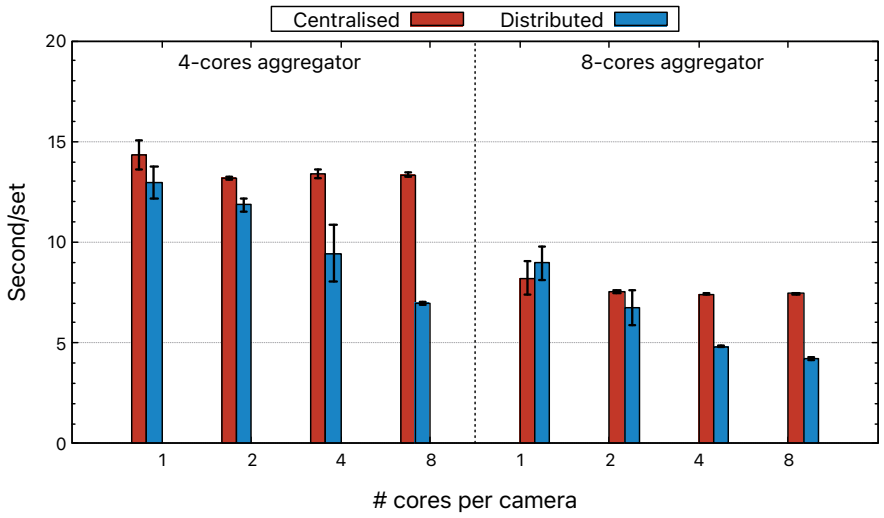


Figure 4.6: Computational performance comparison between the two MVDet *FastFL* implementation (centralised, distributed). The reported values are the seconds needed to process each set of 7 frames +/- the 95% confidence interval over 5 runs, in all combinations of computational power assigned to the server (4, 8 cores) and cameras (1, 2, 4, 8 cores).

Figure 4.6 presents the results across the 16 computational power configurations. It can be noticed how the centralised implementation performance is basically unaltered by the amount of computing power given to the camera, while doubling the number of cores given to the aggregator almost halves the amount of time required to process a single frame, i.e. from 13.57 s to 7.66 s on average. Conversely, the distributed approach is more sensible to the computing power given to both Camera and Aggregator, steadily increasing its computational performance when the computing resources given to the system (Camera and Aggregator nodes) increase from 12.97 s/set using a total of $7x1 + 4 = 11$ vCPUs to 4.23 s/set using a total of $7x8 + 8 = 64$ vCPUs.

By comparing the two systems, it is clear that the distributed implementation obtains globally lower computational times, achieving better computational performance with respect to the centralised approach thanks to exploiting the computational power spread over the computing continuum. The higher the computing power of the camera, the more significant the performance gap, i.e. up to 1.92x faster with 4 cores per aggregator and 8 cores per camera. However, also the network plays a role. Indeed, in the 1 (8) cores per camera (Aggregator), the centralised approach beats the decentralised approach by a small margin due to the increased communication time to transmit the 3.5x larger feature maps instead of the plain captured frames.

Looking at Table 4.7, observing how both implemented systems behave under different network bandwidth conditions is possible. To better simulate a low-power edge system, each camera node is allocated 2 cores. As can be seen, the distributed implementation particularly suffers from bandwidth limits. As explained before, the feature maps are way heavier than the plain frames; hence, bandwidth limits have a particular impact on the system performance, regardless of the amount of computing power allocated to each component. This behaviour can be noted especially in the last row of Table 4.7, in which the centralised system is 7.65x times faster than the distributed one. In a 5G network with higher bandwidths, this speedup decreases to 3.23x-3.91x, a reduction of 42.22%-51.11% compared to the previous scenario. Instead, when the network is not the bottleneck, the distributed approach is the best in almost all scenarios, as shown in Figure 4.6. Please note that the MVDet model is reported as is without any optimisation for communication so as not to alter the model performance. While possibly having a detrimental effect on the model performance, the communication cost could be lowered by compressing the feature maps or retraining the model with feature maps having a lower number of channels.

This comparison clearly shows how the environment can influence the real-world deployment of an edge inference system. Computational power and network bandwidth allocated to each computing continuum element are crucial, but there is no one-fits-all strategy to implement such systems. Workload distribution across the continuum can often become disadvantageous if it leverages critical resources that the centralised counterpart, instead, is not so reliant on. Nevertheless, the same distributed deployment can efficiently exploit all the available resources in an ideal scenario, thus outperforming the more centralised approach. Since environmental conditions change over time, future edge inference systems should consider this variability and try to accommodate and adapt to it, especially if they claim to be flexible, reliable, and efficient.

Chapter 5

Cross-Facility Federated Learning

5.1 Federating across multiple HPCs

This last chapter introduces the most recent advancements of the author's research work. Another vision of FL is proposed, aiming at exploiting geographically distributed computational power. In this sense, FL become more than a methodological tool for running privacy-preserving DML tasks; it become an additional tool capable of increasing the scalability of DML workloads outside the single data centre. FL thus exposes a new tradeoff between a DML system's scalability and learning performance while natively allowing privacy-preserving data handling. This property allows the exploitation of sparse computational power, making it possible to run large DML workloads that a single HPC or cloud infrastructure cannot handle. A federated training of the LLaMA LLM is deployed on three geographically distributed HPC centres, proving that the publicly available computational power can be exploited to run a state-of-the-art DML workload. Such large-scale deployment requires specific tools to be handled correctly; this dissertation identifies the StreamFlow WMS as suitable for the task. This proof-of-concept federation aims to democratise AI development, contrasting the big-tech companies' approach of building large, private data centres to train cutting-edge ML models. All the obtained results and updates on this topic are periodically released publicly on the official cross-Facility Federated Learning website[1].

5.1.1 The compute divide and the *xFFL* approach

The compelling growth in resource requirements of current ML models is reaching never-seen peaks [150]. With the emergence of LLMs, the number of parameters of

[1] https://hpc4ai.unito.it/hpc-federation/

state-of-the-art DNN suddenly increased from hundreds of millions to tenths or even hundreds of trillions, effectively blowing up an order of magnitude [113]. The most well-known examples of such trends are the GPT models. Such growth in models' size is upheld by the synchronously increasing size of available datasets, which can reach hundreds of terabytes. It is immediately apparent that enormous computational resources and time are needed to train such models on such data. Developing these AI models is outside the possibility of common research institutions and companies [22]. Thus, large private companies are the only organisations capable of accumulating the necessary computational power to produce these models. They usually build private clusters specifically designed for AI workloads. The frontiers of AI development are thus becoming prerogatives of privates, excluding publicly funded research. This friction between privately and publicly available computing power is just one example of the so-called *compute divide*.

Publicly available computing power is a reality, though. HPC centres are widespread worldwide, and many offer the possibility to obtain many computational hours for research purposes. For example, through the EuroHPC PRACE initiative, the European Union allows access to a publicly available portion of the computing hours of the HPC centres on its territory. This mechanism allows the researchers to obtain thousands of node hours in world-class European HPC centres. However, even if they are assigned sufficient computing power, public HPC centres' terms of use and regulation make them completely different from private clusters. Typically, a maximum number of computational nodes per cluster is fixed, and it is impossible to scale further on a single infrastructure. Furthermore, submitted jobs cannot run for an indefinitely long time, but the expected running time has to be declared ahead of time, respecting the policies of the HPC centre. Jobs exceeding their declared running time are killed by the cluster management system. Also, queue waiting time has to be taken into consideration. Private HPC centres have many users queuing for resource users, and job scheduling thus imposes waiting times before the requested resources are available. These control mechanisms are handled by complex scheduling systems such as SLURM or PBS, which are the gateway to access computational resources and whose use is not immediate.

Exploiting the joint computational power from multiple publicly available HPC clusters can be a solution to mitigate the computing divide between private and public institutions. Ideally, a large complex computation can be split into multiple sub-computations and run simultaneously on multiple HPC centres. The partial results obtained on each machine can then be aggregated on one of the machines, giving the final result of the global computation. If the process is iterative, such global results can be moved back to the clusters, and the computation can restart from where it stopped. This approach allows a large computation to be run on a set of HPC centres as a single, geographically distributed computational cluster while respecting the individual HPC centres' terms of use. FL appears to be particularly fit for this scenario [193, 180, 40] The presented methodology is named *cross-Facility Federated Learning* (*xFFL*) in the following.

Not all computations are suitable for this approach. The queue waiting time must be considered each time a computation has to be run on an HPC centre. Such time is generally unpredictable and can vary from minutes to hours based on the cluster utilisation, requested resources, external events like maintenance, etc. Furthermore, moving the partial results from one location to another relies on the public internet network. Data transfer between HPC centres is thus subject to varying bandwidths and network speeds, making the process slow and error-prone. The proposed approach is thus particularly suitable for large computations requiring a large amount of computational time and resources while not requiring fast communication between the sub-computations.

Manually handling the proposed process is not sustainable. Interaction with multiple HPC centres simultaneously is not straightforward. It requires a control system capable of interacting with the various scheduling systems adopted by the different centres while simultaneously handling the data transfer processes and checking on the running jobs. Thus, a WMS capable of handling hybrid workflows can be considered an appropriate tool for handling such a complex and distributed workload. StreamFlow [49] is chosen as WMS to orchestrate the cross-HPC federation. StreamFlow implements the CWL standard and is natively container-oriented, offering excellent support for multi-container-oriented tasks. Furthermore, it supports hybrid workflows, allowing the execution of different workflow steps on different computational infrastructures. It thus fits perfectly the nature of the proposed methodology.

5.1.2 Proof-of-concept experiments

If experimentally investigating an HPC-oriented methodology implies motivating HPC use in the first place, investigating a cross-HPC methodology intrinsically requires it even more. Large, computationally bound, timely use cases are needed to test the proposed approach's real-world efficiency and efficacy, and world-leading HPC infrastructures are required to provide solid experimental results. The complexity of a cross-HPC deployment is high indeed, encompassing many architectural and performance aspects. Preparing and optimising the experimental software for the different CPU/GPU architectures, considering the different intra-, inter-, and cross-communication costs, and ensuring software stack compatibility across the different computing centres are just a few aspects that should be considered when designing such a workload. Computations that are not adequately capable of scaling across many computing nodes or mainly rely on fast communications may not adequately exploit the proposed approach. A single-HPC deployment is still the best option in such cases.

FL is particularly suitable for cross-HPC workloads for many reasons. Data can be partitioned effortlessly on the different HPC centres, modulating the local workload according to each machine's computational power and available computing hours. The training of different data partitions is independent of one another; they can thus be executed in parallel on different infrastructures, independently of the local job queue waiting time. The intermediate result aggregation is not frequent, so even long data

transfer times become manageable.

A first cross-HPC FL small-scale experiment is set up to validate the feasibility of the proposed methodology. A VGG16 is trained over two datasets: a standard MNIST dataset, residing on the CINECA MARCONI100 HPC facility located in Bologna (single-node technical specifications: 2x16-core IBM POWER9 AC922, 256 GB RAM and 4 NVIDIA V100 GPUs), and a grayscaled version of SVHN, residing within HPC4AI facility located in Torino (single-node technical specifications: 80-core Arm Neoverse-N1, 512 GB RAM, 2 NVidia A100 GPU) [48]. This national-scale *xFFL* deployment allowed experimentation with the StreamFlow backend on a standard FL task, allowing the identification of the major bottlenecks and issues of the *xFFL* methodology. The federation is run in two configurations: 100 1-epoch federated rounds and 50 2-epoch federated rounds. The federations ran smoothly, and their final obtained models showed the expected accuracy performance on both datasets (>0.99 on MNIST and >0.90 on SVHN). The same setup is run on a pure cloud-based environment offered by HPC4AI and compared to an off-the-shelf solution: Intel® OpenFL. The obtained time-to-solution results confirmed that the *xFFL* approach does not inject overwhelming overheads at this scale: the 100 rounds configuration ran in 2h40m and the 50 rounds one in 2h20m when orchestrated by StreamFlow, while they executed respectively in 3h06m and 2h09m when handled by Intel® OpenFL. These results suggest that the *xFFL* approach offers interesting computational performance at this scale and that larger deployments should be tested to find its limits. The code of this first *xFFL* experiment is publicly available on GitHub[2].

An LLM FL use case is proposed as a proof-of-concept large scale workload for the proposed cross-HPC methodology. Meta®'s LLaMA-2 is chosen as an exemplar state-of-the-art LLM, while the cleaned mC4 Italian corpus (clean_mc4_it) is chosen as an exemplar linguistic dataset. LLaMA-2 is an open-source, freely available LLM produced by Meta®. Many versions of LLaMA-2 are available for download, ranging from 7 to 70 billion parameters. These transformer-based models express state-of-the-art conversational performance, with the larger one outperforming even GPT-4 [164]. Such models are trained on Meta®'s private cluster, the Meta Research Super Cluster, comprising 760 NVIDIA DGX A100 systems as computing nodes for a total of 6,080 GPUs. DGX systems are interconnected through the NVIDIA Quantum 1600 Gb/s InfiniBand two-level Clos fabric with no oversubscription. RSC's storage tier has 175 petabytes of Pure Storage FlashArray, 46 petabytes of cache storage in Penguin Computing Altus systems, and 10 petabytes of Pure Storage FlashBlade [89]. The Meta® training dataset encompasses 2 trillion data tokens; the training of all models took jointly 3,311,616 GPU computing hours, for an equivalent CO_2 emission of 539 tons [164]. LLaMA-2 with 7 billion parameters is thus selected for the experimental's simplicity and feasibility.

A cleaned version of the mC4 Italian split is always used as one of the experimental

[2] https://github.com/alpha-unito/streamflow-fl

Table 5.1: LLaMA-2 (7B version) estimated one epoch training wallclock time on the clean_mc4_it dataset (4,085,342 fixed 2048 token-length data samples) at different numbers of nodes, ranging from 2 to 128. The deployment is done bare-metal with a PyTorch version manually compiled on the Leonardo supercomputer (1 node = 4 NVIDIA A100 GPUs).

#Nodes	#GPUs	Loading time (s)	Queuing time (s)	Estimated execution time (hours)	Aggregate data processing speed (it/s)	Speedup	Efficiency
2	8	34	2	774	0.35	2.00	1.00
4	16	34	2	385	0.99	4.02	1.00
8	32	34	2	193	2.83	8.02	1.00
16	64	34	2	98	8.16	15.80	0.99
32	128	38	2	49	23.19	31.59	0.99
64	256	90	222	25	66.32	61.92	0.97
128	512	120	438	14	179.02	110.57	0.86

datasets in the following experiments [144, 182]. This dataset is a preprocessed version of the mC4 corpus Italian split. It omits documents containing despicable words, too short (<3 words) or too long (>1000 characters) sentences, sentences not ending with end-of-sentence punctuation, documents which are either too short (<5 sentences or 500 characters) or too long (>50,000 characters), and so on. The result is a cleaned Italian corpus of 103 million documents, comprising 41 billion words. The 'tiny' split of the corpus is chosen since the full corpus size is prohibitive and incompatible with the experiment time requirements. Thus, the experiments are run on 10 million documents, comprising 4 billion words, for a total of 4,085,342 training samples and 13252 testing samples, each comprising 2048 tokens.

The experimental setting is organised as follows. Data is preprocessed, manually split, and moved to each HPC centre according to the available computational power and time. StreamFlow is deployed on an external cloud machine, which will also act as the federation's aggregator. LLaMA is distributed among the computing nodes through the Fully Sharded Data Parallel (FSDP) methodology on the single HPC centre, using different sharding strategies for each infrastructure according to the local hardware characteristics. From a high-level perspective, the LLaMA model is sharded at the intra-node level and replicated on each node. This distribution strategy means that a model parallel approach is exploited intra-node, while a data parallel approach is exploited inter-node.

Previous scaling experiments on Leonardo (the Italian pre-exascale HPC system; its technical details are given in section 5.1.3) showed almost ideal scalability of the FSDP methodology up to 16 nodes (64 GPUs), as reported in Table 5.1, confirming the LLaMA scaling capabilities when trained distributedly with the FSDP strategy. Note that the model's weights are re-initialised at the beginning of the deployment and that all of them are trained during the process: this is a full-scale LLM training, not a fine-tuning. Due to limited resource availability, the total execution time required to train LLaMAv2

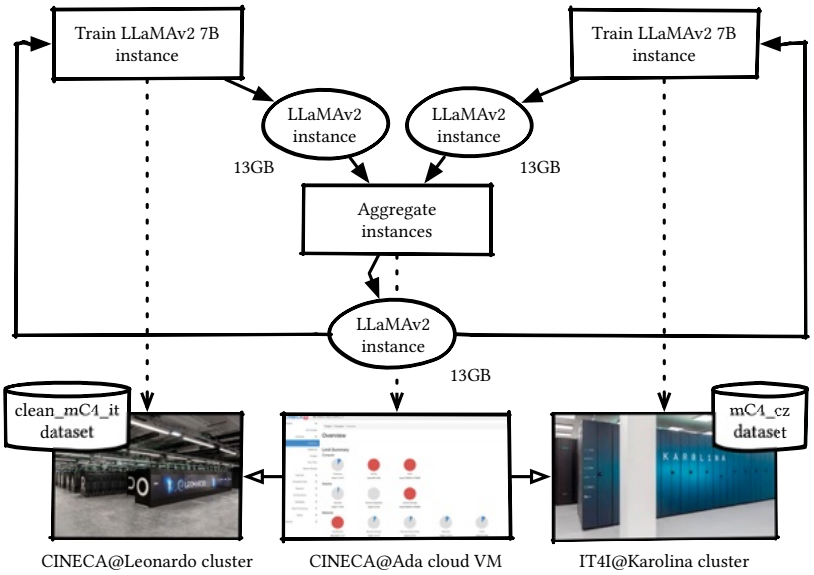


Figure 5.1: Schema of the proof-of-concept *xFFL* LLaMA-2 (7B version) hybrid workflow deployment. StreamFlow automatically deploys the model on the two HPC facilities (interacting with SLURM), retrieves the trained parameters, aggregates them on a third machine, and repeats the process until convergence.

7B on the whole dataset is estimated. This estimation is possible since the training time is linear in the training dataset size, modulo having fixed hyperparameters. Once the aggregate data processing speed (usually expressed in terms of iterations/second) is stable, estimating the total execution time becomes straightforward, knowing the number of iterations needed for the dataset (1 iteration = 1 data batch = 4 data samples = 8192 tokens in our testbed). During preprocessing, the documents are converted into fixed-length sequences of 2048 tokens through the standard LLaMAv2 tokeniser. As a result, the Italian dataset used consisted of 10 million documents comprising 4 billion words, converted into a total of 4,085,342 training samples and 13,252 testing samples of 2048 tokens each.

The first effective deployment of this LLaMA-2 7B experimental setup is done on two EuroHPC infrastructures: Leonardo and Karolina (as depicted in Figure 5.1). Leonardo is the Italian pre-exascale HPC system hosted by CINECA in Bologna; its technical details are given in section 5.1.3. Karolina is a petascale HPC ranking 113th in the Top500 list (Apollo 6500, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Infiniband HDR200, HPE) hosted by the IT4Innovations National Supercomputing Center at the VSB-Technical University of Ostrava, Czechia. The deployment is done bare-metal on both infrastructures, exploiting manually compiled PyTorch versions. Leonardo uses a subset of the cleaned version of the mC4 Italian split, while Karolina adopts a subset of the standard mC4 corpus Czech split [137]. This choice suggests the applicability of the

Table 5.2: LLaMA-2 (7B version) transfer times between the computing facilities. The cloud VM is located in Bologna, physically near Leonardo. LLaMA-2 7B weighs approximately 13 GB on disk (saved in half precision).

	Leonardo@CINECA		**Karolina@IT4I**		**Ada Cloud VM@CINECA**
	Queuing time (s)	Transfer time (s)	Queuing time (s)	Transfer time (s)	Model aggregation time (s)
Min	0	138	0	178	118
Avg	34	217	175	242	133
Max	164	360	699	344	143

xFFL approach in a real FL context, in which different computing infrastructures hosted in different countries have access to radically different private data. This use case thus aims to produce a single bilingual LLM through FL training on two different linguistic corpora. Learning performance, however, is not investigated here since the focus of this discussion is on the computational performance of cross-facility computations and not on the FL training performance of LLMs. We retained the same data quantity from the mC4 Czech dataset to ensure balanced execution times (4,085,342 training samples and 13,252 testing samples of 2048 tokens each). Each dataset occupies about 102GB of disk space, much more than the 13GB required by the half-precision representation of the LLaMAv2 7B weights.

Scalability in a *xFFL* execution is hard to achieve. In addition to the overheads of distributed training on a single HPC, FL exhibits several other potential overhead sources. The major sources of additional overhead in a *xFFL* deployment are:

- Model exchange time across a geographical-scale network;
- Aggregation time (either centralized or distributed);
- Load imbalance between different sites due to different dataset sizes and/or different computing power;
- Job queue waiting time (typically the main performance issue due to the inherent stochasticity of this time span).

The values of these additional overheads measured during the Leonardo-Karolina *xFFL* deployment experiments are reported in Table 5.2. The load imbalance overhead is not reported since the same amount of data is used on both infrastructures in the presented experiment. As can be observed, the model aggregation and transfer times are predictable and stable. In contrast, the queuing time is much more variable and unpredictable, making the system's overall execution time unpredictable. Note that each training round between the aggregation point is modelled as a single job that, thus, is

submitted by StreamFlow to the queuing system and then is subject to a queue time before execution. With frequent aggregations, the queuing time on public HPC infrastructures can quickly become the major bottleneck of the *xFFL* deployment. Nevertheless, *xFFL* proved to be suitable and stable enough to handle a real-world deployment on two HPC infrastructures. All the developed code for this deployment is open-source and freely available on GitHub[3].

5.1.3 Creating the first European HPC federation

Experimental setup

The last iteration of the xFFL experiment focuses more on broadening the distributed computation scale, targeting a more realistic use case. This time, LLaMA-3 8 billion is trained using a prompt-tuning approach for an open-ended generation task. The dataset used is again the cleaned mC4 Italian partition. Data is fed to the LLM with a generic prompt ("Scrivi un documento"-"Write a document"), and the perplexity between the generated text and the document passed on the template is calculated. Aggregation happens on a VM hosted by CINECA Ada cloud in Bologna as in the Leonardo-Karolina setting.

Three world-leading HPC centres are selected for this experiment: *Leonardo* (CINECA, Bologna, Italy), *LUMI* (CSC, Kajaani, Finland), and *MeluXina* (LuxProvide, Bissen, Luxembourg). Technical details describing these HPC infrastructures are given at the end of this paragraph. Their heterogeneous CPU/GPU architectural characteristics make it possible to showcase how to design and manage complex software capable of harvesting computational power, notwithstanding the underlying hardware. Leonardo offers the de facto standard Intel/NVIDIA architecture, LUMI the alternative full-AMD architecture, and MeluXina a hybrid AMD/NVIDIA solution. These three HPC centres represent the vast majority of HPC architectures populating the Top500 at the time of writing and offer an excellent scenario for obtaining reliable and generalisable experimental results.

The HPC centres' geographical distribution is also considered; a graphical representation is given in Figure 5.2. Cross-HPC workloads aim to exploit geographically distributed computational power to a level never seen before. Thus, it is fundamental to investigate how large distances between these infrastructures can interfere with computational performance. Considering that Leonardo is located in Bologna, Italy, LUMI in Kajaani, Finland, and MeluXina in Bissen, Luxembourg, the total geographical distance separating these three HPC centres is $\sim 5{,}183Km$ as the crow flies (much more than the $\sim 782Km$ between Bologna and Ostrava), covering a total land area of $\sim 678.061Km^2$ ($\sim$ 16% of the entire EU surface area). These massive distances, however,

[3] https://github.com/alpha-unito/xffl

Figure 5.2: Representation of the third *xFFL* experiment's geographical extent. The distance between these three HPC centres is $\sim$ 5,183Km as the crow flies (much more than the $\sim$ 782Km between Bologna and Ostrava), covering a total land area of $\sim$ 678.061Km^2 ($\sim$ 16% of the entire EU surface area).

influence cross-HPC distributed computations, especially spoiling the data transfer performance between geographically distant infrastructures: minimising data transfer is mandatory. Given this geographical setting, these experiments are thus considerable as the first concrete step toward a European cross-HPC federation, effectively bringing together the computational power scattered across Europe.

Another fundamental aspect characterising the three chosen infrastructures is their relative computing power. All the given data in the following are to be intended at the time of writing (Top 500 - November 2023 update). Leonardo is a pre-exascale system achieving a performance peak of 238.70 PFlop/s (LINPACK R_{max}), ranking 6th in the Top 500 list. LUMI is also a pre-exascale system and the largest supercomputer in Europe. It achieves a performance peak of 379.70 PFlop/s (LINPACK R_{max}), ranking 5th in the Top 500 list. These two supercomputers are the two largest currently

available in Europe. MeluXina, on the other hand, is a more modest infrastructure, achieving a performance peak of 10.52 PFlop/s (LINPACK R_{max}), ranking 71th in the Top 500 list. These computational performance differences highlight another key aspect of cross-HPC workloads: load balancing. Harvesting sparse computational power requires carefully managing the available resources, especially considering the capabilities of each device. Different-size machines can be exploited together successfully if the workload deployed on each is tailored to the available resources. Similar execution times on each machine directly reduce data transfer and synchronisation waits between them, speeding up the global cross-HPC execution time and improving resource usage simultaneously. This fact is especially true for iterative applications that require one or more synchronisation steps at each cycle, such as modern simulation software or distributed ML training algorithms.

Singularity containers are exploited to avoid platform-specific code issues while allowing the portability and reproducibility of the experiments [103]. The NVIDIA official optimised PyTorch images are used on the Intel-NVIDIA and AMD-NVIDIA architectures, Leonardo and MeluXina. To optimise the computational performance on LUMI, the LUMI's official Singularity image for PyTorch (lumi-pytorch-rocm-5.6.1-python-3.10-pytorch-v2.2.0-dockerhash-f72ddd8ef883.sif) is used after being updated with the necessary Python packages. This approach proved more computationally efficient than running bare-metal code, especially on NVIDIA architectures.

Technical details of Leonardo, LUMI, and MeluXina are given below to complete the experimental setting overview. Leonardo comprehends 4992 liquid-cooled compute nodes interconnected through an NVIDIA Mellanox network, with Dragon Fl+, capable of a maximum bandwidth of 200Gbit/s between each pair of nodes. The type of computing node exploited in these experiments is a custom BullSequana X2135 "Da Vinci" blade equipped with an Intel Xeon 8358 Ice Lake 32 cores 2.6GHz CPU, 512 (8 x 64) GB RAM DDR4 3200MHz, 4 NVIDIA A100 SXM6 custom Ampere 64GB HBM2 GPU, and 2 NVIDIA HDR 2x100Gb/s cards. LUMI comprehends 5026 compute nodes (2,978 GPU nodes + 2,048 CPU nodes) interconnected through an HPE Cray Slingshot-11 200Gbps network interconnect (NIC). The LUMI-C nodes (CPU) have a single endpoint, while the LUMI-G nodes (GPU nodes) have 4 endpoints - one for each GPU. Each endpoint provides up to 50 GB/s of bidirectional bandwidth. The LUMI-G nodes exploited in these experiments have an AMD EPYC "Trento" 64-core CPU, 512 (8x64) GB RAM DDR4, and 4 AMD MI250x 128GB HBM2e GPUs. MeluXina comprehends 813 compute nodes (573 CPU nodes + 200 GPU nodes + 20 FPGA nodes + 20 large-memory nodes) interconnected with an InfiniBand (IB) HDR 200Gb/s high-speed fabric. The GPU nodes exploited in these experiments have 2 AMD Rome 32 cores 2.35 GHz CPUs, 512 GB RAM, and 4 NVIDIA Ampere A100 40GB HBM GPUs.

Table 5.3: LLaMA-3 (8B version) execution time subdivided in its main components. The training is done on 20,000 training samples of 2,048 tokens each on the Leonardo HPC.

#Nodes	Model loading (s)	Distributed setup (s)	Training (s)	Testing (s)
1	120.6	87.8	3,321.4	54.4
2	123.6	95.3	1,700.0	27.0
4	120.6	158.4	788.4	13.0
8	121.8	94.5	432.6	6.0
16	131.8	95.7	223.4	3.0
32	131.4	110.0	115.0	1.0
64	122.0	109.5	63.2	~ 0
128	118.6	119.2	40.8	~ 0
256	117.2	124.8.8	22.8	~ 0

Experimental results

It is crucial to analyse the computational performance of the selected HPC infrastructure on AI-related tasks to give a fair perspective on the obtained experimental results. AI workloads exhibit specific computational performance and resource usage different from the standard computing benchmarks, i.e., LINPACK. The performance results presented in the following are obtained by training LLaMA-3 8 billion parameters on 20,000 training tokens from the 'tiny' cleaned mC4 Italian corpus on the Leonardo HPC.

The code's relatively poor scalability performance is the first thing to be noticed. As can be seen in Table 5.3 and graphically in Figure 5.3, the distributed setup time ("other sequential code" in the figure) starts increasing from 16 nodes (64 GPUs), suggesting even more significant overheads on larger-scale deployments. This setup time, together with the model loading time, determines the computation's non-scalable component, limiting the overall code scaling performance. The experiments target the LLaMA 8B model; larger LLMs will yield even larger overheads. The training and testing code, on the other hand, seems to scale almost perfectly up to 64 nodes (256 GPUs). Such a limit is intrinsic to the FSDP technique itself: since the global batch size of the training increases with larger deployments, the distributed computing libraries implementing FSDP are not optimised to span more than that. These performance issues do not seem to be correlated with the problem's size (i.e., the size of the training dataset) or to the FSDP distributed training technique itself. These statements are confirmed by the fact that increasing the training dataset size does not change the code's scaling behaviour and that the performance of the FSDP code section yields nice scalability performance up to 64 nodes, as can be observed in Figure 5.4.

This in-depth scalability analysis on Leonardo provides a fair baseline for comparing

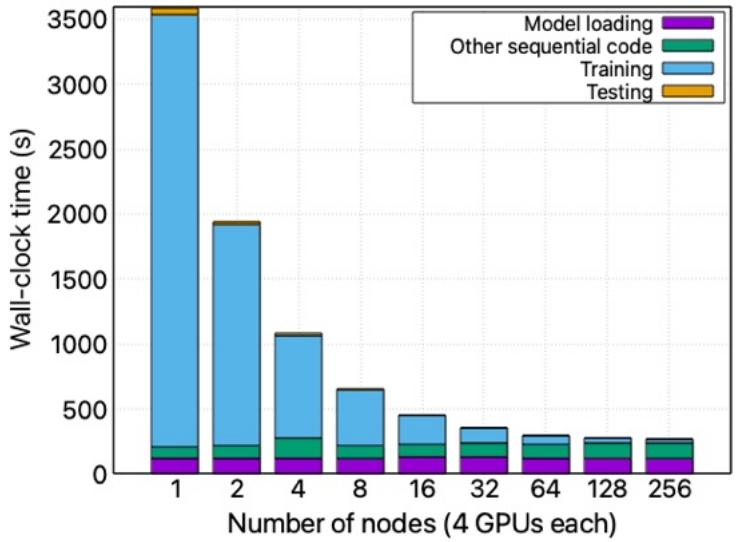


Figure 5.3: LLaMA-3 (8B version) execution time subdivided in its main components. The training is done on 20,000 training samples of 2,048 tokens each on the Leonardo HPC.

the performance offered by the other two HPC infrastructures selected for this experiment: LUMI and MeluXina. It should be noted that the different infrastructures differ in their node structure, memory and computational power; thus, the LLaMA-3 8B deployment has to be tailored specifically for each platform. For example, a single instance of LLaMA-3 8B fits into a single Leonardo or LUMI node but requires two nodes on MeluXina. This difference is due to the GPU RAM available on the computing nodes. Each Leonardo node is equipped with 4 custom NVIDIA A100 for a total of 256 GB of GPU RAM, each LUMI node is equipped with 4 AMD MI250x for a total of 512 GB of GPU RAM, while each MeluXina node is equipped with 4 NVIDIA A100 for a total of 160 GB of GPU RAM. Note that each LUMI GPU comprises two graphics compute dies, meaning that each LUMI node is equipped with 8 64GB RAM GPUs for a practical perspective. This difference in model allocation also implies greater communication overhead on MeluXina since training a single model requires inter-node communication, while this is not the case on Leonardo and LUMI. Finally, a different dataset size is selected for each HPC system to obtain equal workloads from a time-to-solution perspective. This process is based on selecting a workload lasting 1 hour on the smallest available deployment on each system. It resulted in 16,384 training samples on Leonardo, 4,096 on LUMI, and 8,192 on MeluXina.

The obtained computational performance on the three HPC systems is reported in

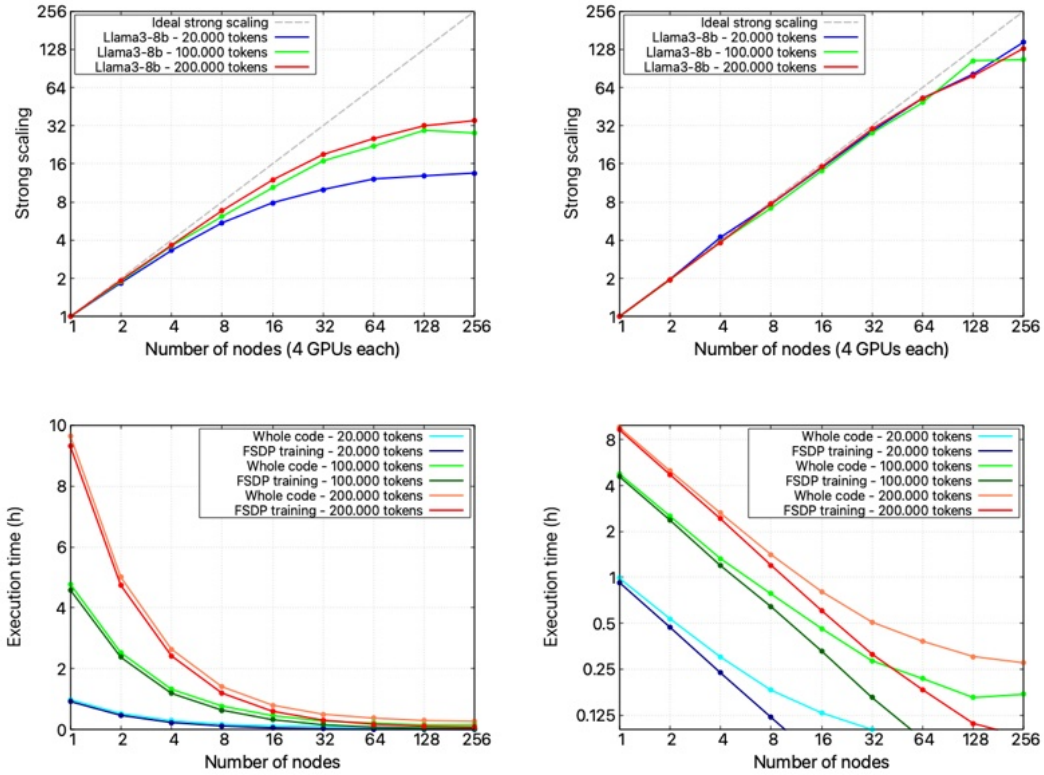


Figure 5.4: LLaMA-3 (8B version) scaling performance on Leonardo. Both the whole deployment code and the FSDP code are analysed. The training is done on samples of 2,048 tokens each on the Leonardo HPC.

Figure 5.5. It appears clear that the scalability problems discussed in the previous paragraphs are unrelated to the underlying hardware: also on LUMI and MeluXina the whole code's scalability performance starts to spoil after 16 nodes, while the FSDP code scales reasonably well on all infrastructures and also on a high number of nodes. From a scaling perspective, LUMI performs better than the other two infrastructures. While this is true from a strict perspective, it has to be noted that the three HPC systems expose radically different absolute computational times, meaning that the scalability performance on its own its not sufficient to determine the most efficient single-HPC deployment. Figure 5.6 depicts the difference in absolute compute time required to process the same amount of data on each HPC infrastructure (16,384 training samples). As can be seen, LUMI is slower in processing data from an absolute time perspective. This lower computing performance can mitigate the communication overheads implied by FSDP, thus exposing an 'artificially' better scaling performance compared to Leonardo and MeluXina.

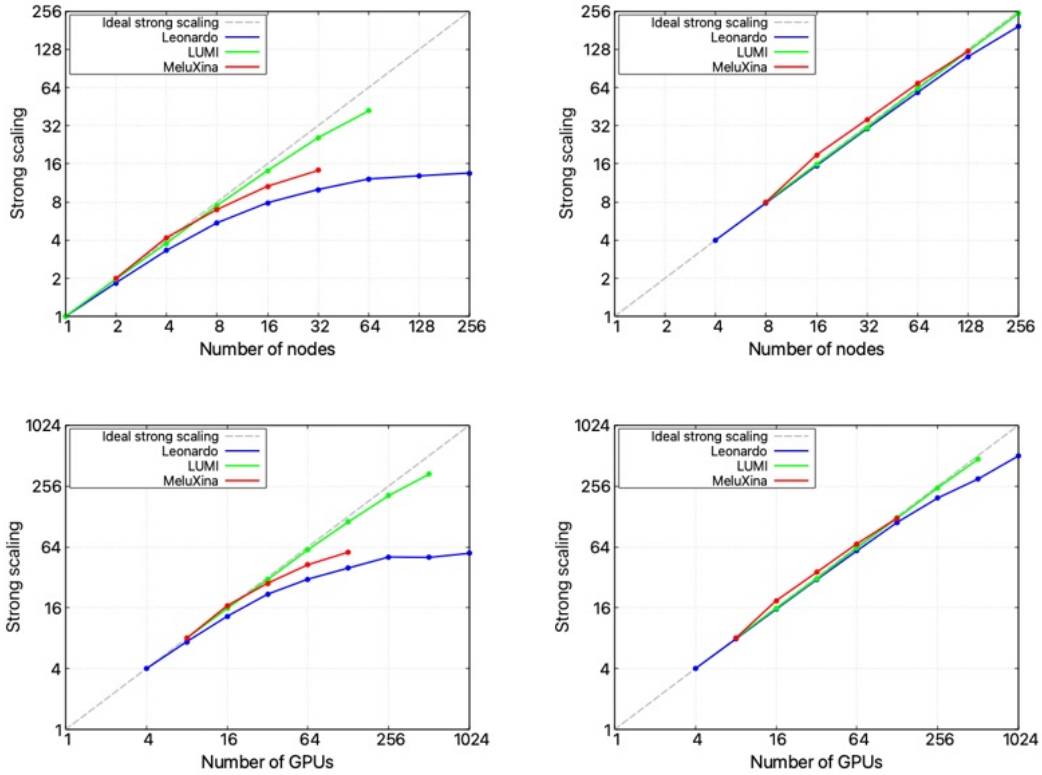


Figure 5.5: Comparison between LLaMA-3 (8B version) scaling performance on Leonardo, LUMI and MeluXina. Both the whole deployment code and the FSDP code are analysed. The training is done on a different number of tokens on each HPC infrastructure to accommodate the different computing power.

Table 5.4: LLaMA-3 (8B version) transfer time between Leonardo, LUMI, MeluXina and the Cloud VM. LLaMA-3 8B weights $\sim 15GB$ on disk saved in half precision. Experiments are repeated 33 times.

From	To	Max time (min:sec)	Min time (min:sec)	Average time (min:sec)
Cloud VM	Leonardo	06:33	03:57	04:23
Cloud VM	LUMI	13:55	08:20	09:32
Cloud VM	MeluXina	03:50	03:18	03:36
Leonardo	Cloud VM	06:34	05:45	06:00
LUMI	Cloud VM	76:38	16:04	27:51
MeluXina	Cloud VM	40:43	06:34	27:42

Considering all these single-HPC considerations, the *xFFL* performance results obtained on the three systems together are exposed. Many *xFFL* experiments are launched

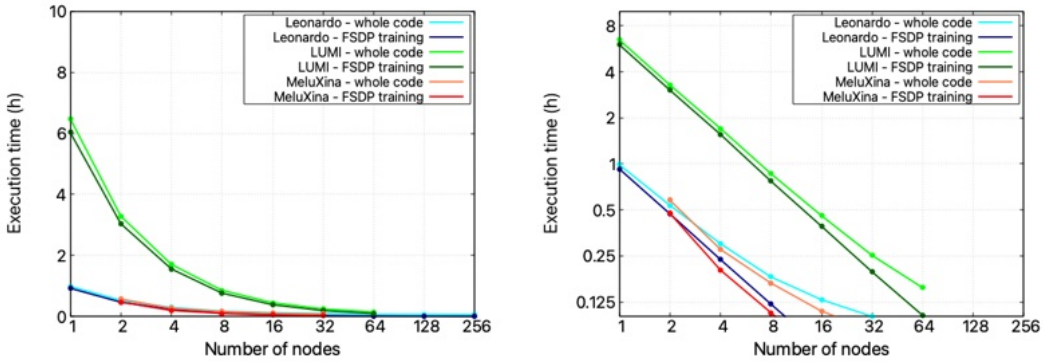


Figure 5.6: Comparison between LLaMA-3 (8B version) scaling performance on Leonardo, LUMI and MeluXina. Both the whole deployment code and the FSDP code are analysed. The training is done on 16,384 training samples of 2048 tokens each on each HPC infrastructure.

Table 5.5: LLaMA-3 (8B version) queuing and execution times on Leonardo, LUMI and MeluXina. The training is done on a different number of tokens on each HPC infrastructure to accommodate the different computing power.

Leonardo			**LUMI**			**MeluXina**		
# Nodes	Queue time (min:sec)	Execution time (min:sec)	# Nodes	Queue time (min:sec)	Execution time (min:sec)	# Nodes	Queue time (min:sec)	Execution time (min:sec)
1	51:08	49:49	1	13:35	50:50	2	00:01	34:33
2	03:03	27:05	2	05:05	26:36	4	11:28	16:46
4	04:35	15:18	4	07:39	14:37	8	50:58	10:36
8	04:29	09:37	8	09:28	08:27	16	41:38	07:26
16	38:11	06:09	16	03:42	05:25	32	00:18	06:09

to train LLaMA-3 8B (15 GB disk size in half-precision format). The base FL configuration consists of 1 Leonardo node, 1 LUMI node, and 2 MeluXina nodes. Data is split according to each facility's computing power to obtain balanced execution times, as exposed in the previous paragraphs. As shown in Table 5.4, there appear to be differences in the transfer time between the Cloud VM and the supercomputer and vice versa. It is possible that such unbalance is due to an unforeseen StreamFlow behaviour to be corrected; however, this issue is currently under active investigation. On the other hand, queue times present an extremely variable behaviour. Figure 5.5 summarizes the queuing and execution times of the *xFFL* experiments in this setting. As can be seen, high-usage periods can easily increase the queueing time needed for the *xFFL* to be executed. While these factors strongly hinder the scalability property of *xFFL*, it should be noted that the proposed system automatically and correctly manages such delays, meaning that such overheads will not be an issue on a private or free cluster. However,

data has been collected on a small set of runs; more extensive experimentation is required for proper evaluation. Queue-aware scheduling policies that adjust workloads based on predicted queue times should be investigated to mitigate the impact of outliers jobs.

Chapter 6

Conclusions

This last chapter wraps up the dissertation. The major limitations of the presented work are exposed, identifying possible interesting research scenarios derived from this dissertation. Some future works already foreseen are discussed, highlighting that the presented dissertation is an organic component of a live, evolving research path and not just a sterile academic milestone. Finally, the conclusions summarise the exposed concepts and proposed research ideas, integrating them into the broader social and technological framework described in the introduction, giving the dissertation cohesion and integrity.

6.1 Limitations and future work

The presented research path gives an overview of the major structural problems of current DML practice focusing on FL. Such issues' practical aspects are considered, always highlighting their implementational aspects. Such an approach makes this work particularly application-oriented, focusing on real-world use cases, data analysis, and large-scale distributed DML deployments. In this sense, this dissertation gives practical insights into the DML research scenario and proposes experimental tools to solve such issues. Such an approach favours the development of usable software, thus promoting a tangible impact in the research community. However, it partially neglects the investigated issues' more theoretical and methodological aspects. Each chapter of this dissertation tries to present at least some methodological considerations about the proposed works, but theoretical considerations are not always present. Thus, a limitation of the presented research work is the prevalence of a practical, implementation-oriented approach to the DML scenario, which may have overlooked more high-level methodological and theoretical aspects.

Federated Learning as a Service (FLaaS) is a proposed future work to solve this lack. FLaaS aims to overturn the traditional FL methodology completely. Instead of designing

a federation as a structure built from a set of institutions cooperating on the same learning problem, FLaaS envisions a federation as a service offered by a set of institutions to allow a third party to solve its learning problem. According to this vision, the data-holding institutions are no longer the active part of the federation but a coordinated net of servers willing to participate in a federation. This innovative vision allows institutions to allow external researchers to access potentially unused local datasets while maintaining traditional FL's privacy properties. In this sense, FLaaS can be envisioned as a new knowledge-shared paradigm in which a researcher with an exciting research idea can easily identify which nodes of the FLaaS platforms hold data that can be of interest for his/her research and run an FL workload on them. The main benefits of this approach are that researchers outside a data-holding institution can exploit the FL properties to conduct their research and that data-holding institutions that do not know how to exploit their local data can share them, helping scientific advancements (eventually under fee payment). FLaaS thus envisions a new FL paradigm, possibly impacting standard FL practices, software, and methodologies. It is currently being developed, and many companies have expressed interest in the project.

Another issue of this dissertation is the widespread spectrum of the proposed ideas. Many software and hardware concepts are discussed in this thesis. Cardiological and financial research, traditional, deep, and distributed ML, FL, EI, HPC, cloud, and edge computing, different microarchitectures including RISC-V, software design and implementation, parallel and distributed computing, workflows, containers, and so on. These topics are too many and different to be adequately discussed and exposed in one research work and too many to be deeply and profoundly understood in a three-year Ph.D. path. It derives that this dissertation lacks a well-focused and specialised contribution, preferring instead a more cross-topic and transversal contribution, often more towards HPC, ML, or even applicative domains.

A future research work merging all the proposed DML ideas is proposed to mitigate such an issue. Ideally, the future development of *FastFL* aims at incorporating all research ideas discussed before its exposition, such as distributed boosting, and after, such as cross-HPC deployments. In this sense, *FastFL* is the main experimental software actively developed by the author, and it will include all the experience and ideas developed during the PhD period. Notably, the FLaaS idea exposed just above is designed to support *FastFL* and containerisation technologies such as Docker and Singularity. Furthermore, integration with the StreamFlow WMS is envisioned to handle complex, distributed FL workloads as workflows. Such an approach guarantees many advantages, such as the native support of containerisation technologies, the already developed SLURM and PBS interface for HPC offload, and built-in fault tolerance in case of node failure. This way, all the proposed research ideas can converge into one single software encapsulating all the ideas developed across three research years, offering a unified, compact, and coherent research product.

Finally, the *xFFL* experimentation showed that LLM training workflows scale differently on different HPC facilities mainly due to overhead handling (model loading,

PyTorch distributed setup). FSDP training scales well up to 256 GPUs on all the tested HPC facilities, but with very different absolute compute times that should be considered. Carefully balancing data to obtain homogeneous execution times for each round is fundamental, but some overheads are unpredictable, i.e., queuing times. The *xFFL* deployment can be further optimised by reducing the model loading time using high-end storage and I/O optimisation techniques (e.g., GPUDirect storage), investigating strategies to avoid PyTorch cold restarts on all nodes (caching, faster setup algorithms), or reducing the amount of communication involved in the process. Furthermore, it should be noted that investigating *xFFL* performance at larger scales will probably result in discovering new computing and communication bottlenecks in the underlying communication libraries (e.g., NVIDIA NCCL). Larger deployments will also need different training hyperparametrisation to preserve learning performance. Finally, HPC instability has to be considered: node failures, network instabilities, and frequent maintenance periods make it difficult to use three facilities at the same time and would probably prevent using even more simoultaneously. Also, the extreme heterogeneity in their interfaces, structure, computing hours allocation policies, computing time accounting (node hours vs GPU hours vs core hours), container support (there is no standard way to build Singularity containers on the systems, and is even not allowed on some facilities), and so on pose a significant obstacle to the construction of large-scale international HPC federations.

6.2 Conclusion

This dissertation explores many aspects of the FL paradigm. All the proposed research ideas are designed to push the boundaries of current FL practice, ranging from low-level perspectives, such as the support and power consumption of different microarchitectures, to high-level perspectives, such as modelling distributed computation through theoretical tools. This PhD thesis highlights that FL is much more than what was originally defined by Kairouz and McMahan:

- FL is an enabling methodology to allow the pooling of distributed data, easing the process of data harvesting and agreement between the different parties. Section 3.1.4 proves this by replicating the performance of a state-of-the-art cardiological risk score, the PRAISE score, by training the same ML model on a federated version of the same dataset through the FL methodology, thus bypassing all the effort and time needed to create the original PRAISE data lake.
- FL can be generalised to classical ML models through a careful algorithmic design. Section 3.2 demonstrates the versatility of FL by implementing the AdaBoost.F model-agnostic FL algorithm into the Intel® OpenFL framework. This innovation enables the application of FL in fields where DNNs are not always viable, such as medicine (Section 3.1) and finance (Section 3.3).

- FL is not bound to the master-worker topology nor high-level Python implementations. Chapter 4 shows that through a high-level formal language (RISC-pb^2l) and careful low-level implementation (FastFlow), it is possible to envision a new full-stack vision of FL deployment, with both a more flexible communication topology and better computational performance.
- FL is a powerful, loosely synchronous methodology that enables new trade-offs between ML training scalability and learning performance. This power is demonstrated in Chapter 5.1, where FL is exploited to deploy large-scale, distributed LLM training on geographically distributed HPC infrastructures, bridging the digital divide.

The investigated topics already flow into practical and tangible contributions to the research community, i.e., open-source software implementations. This effort will allow the research community to experiment and explore the proposed FL research vision with the plurality of concepts and topics involved. The proposed work aims to empower a distributed vision of ML, pushing towards software, tools, and methodologies to allow AI to flourish pervasively together with the help of everybody, overcoming current computational and technological constraints restricting AI pioneering in the hands of few and large private companies. AI is a tool in the hands of the world, but not everyone can deploy, improve, and use it. The capability of developing effective AI systems is becoming a pivotal point of the current technological and social scenario, and the proposed research efforts and products aim at democratising it as much as possible.

Appendix A

Federated Learning Applications

This appendix collects two practical FL investigations conducted by the author during his abroad period. All the exposed studies and results are obtained in collaboration with the Computer Laboratory of the University of Cambridge, Cambridge, UK. The methodological idea behind these studies is the same: applying FL to a domain in which data is critical and investigating the approach's feasibility. The selected use cases are DTI and solar wind prediction. All the proposed experiments are based on actual data, with hundreds of experiments run on large HPC and cloud infrastructure. This appendix thus provides learning, computational, and domain-specific performance results, engaging the interest of three research communities.

A.1 Drug-Target Interaction

This section delivers the first-ever FL benchmark for the DTI task. At the time of writing, the only study in the literature that applies FL to DTI is FL-QSAR, which presents the first federated model trained for a related drug discovery task. Unfortunately, it stops short of analysing FL performance beyond demonstrating its feasibility for up to 4 clients [41]. A novel technique named diffusion is developed to provide a more comprehensive study framework for FL applications. Through diffusion, it is possible to identify and explain a significant and material difference in the sensitivity of FL to non-IID data in the DTI task. A state-of-the-art DTI scenario is thus proposed and analysed through the diffusion technique. The importance of data ownership structure in FL for DTI is discussed as a significant performance determinant and a key aspect when engaging real-world actors in cooperative model training. The reported experiments aim to represent the complexities a federation of pharmaceutical labs would entail as realistically as possible. The whole spectra of IID-ness and data ownership distribution are explored, and core FL algorithms are used to provide general and applicable

experimental results.

A.1.1 Background

This research work proposes to fit the GraphDTA graph neural network on the KIBA drug-target dataset split among multiple clients through FL, thus simulating a pharmaceutical laboratories federation. The Graph Drug-Target Affinity (GraphDTA) [126] mode is a graph DNN currently used as the backbone of many current state-of-the-art models [88, 127, 61]. GraphDTA regresses the drug-target pair onto a continuous measurement of binding affinity for that pair, the KIBA score, that is later exposed. This model requires the target to be a 1D sequence and the drug as a molecular graph, making it possible for the model to capture the bonds among atoms directly. No fancy aggregation strategies are implemented, and neither is the model's structure refined to fit the federated task. Only the stateful components of GraphDTA are modified and replaced with stateless ones to accommodate better the FL process: layer normalisation instead of batch normalisation and SGD instead of ADAM. These choices allow for more stable learning curves and cleaner convergence of the federated model without harming its performance. Both FL and non-FL experiments are run with the same adjusted architecture. On the other hand, the Kinase Inhibitor BioActivity (KIBA) dataset reports 246,088 scores for 52,498 chemical compounds and 467 kinase targets, originating from three separate large-scale biochemical studies. The target proteins are encoded in the SMILE format and converted to a suitable graph representation before the FL process.

The experimental setup builds on the experiments usually associated with FL benchmarks while substantially expanding them. First, the model is compared against a suitable alternative. Given the lack of prior work, there was no ready candidate for this comparison. A centralised model is unsuitable since its use is unrealistic due to regulatory and commercial considerations. Cryptographic approaches to data anonymisation would be usable in real life; however, they are not a direct competitor to FL. However, they can augment each other and provide joint solutions similar to what FL with differential privacy does [170]. Ultimately, a simple Bergman's ensemble [27] of models is chosen, with each model being trained separately on a different data split of the entire dataset. Data splits are maintained constants when comparing FL and Ensemble Learning. The choice of baseline algorithms for both FL and the ensemble is deliberate, as any extension applicable to one can be straightforwardly re-engineered for use with the other [135]. Therefore, working with simple implementations provides us with a fair, uncoloured comparison of the two approaches rather than of their two extensions. The metric used to evaluate each experiment is the Mean Squared Error (MSE); in the case of FL, the MSE of the global model is computed, while in the case of bagging, the MSE of the ensemble is considered. The test set is the same for all experiments, allowing for a fair comparison of different runs.

The reported experiments exploit the open-source FLOWER [23] framework to provide FL functionalities and the FedAvg aggregation strategy. The code is available on

Table A.1: Learning performance obtained by DTI-FL and an ensemble alternative on the KIBA dataset. The % difference columns refer to the federated values compared to the ensemble ones: positive difference in the MSE highlights worse relative FL performance and vice versa.

	IID distribution			**non-IID distribution**		
	Ensemble MSE	Federated MSE	% difference	Ensemble MSE	Federated MSE	% difference
2 clients	0.509	0.530	+4.08%	0.550	0.556	+1.19%
4 clients	0.563	0.577	+2.58%	0.556	0.556	-0.05%
8 clients	0.567	0.574	+1.30%	0.568	0.574	+1.20%
16 clients	0.576	0.578	+0.42%	0.573	0.578	+0.690%
32 clients	0.709	0.599	-15.53%	0.579	0.578	-0.024%

GitHub[1]. The provided code works with PyTorch but will eventually be compatible with TensorFlow. It is being shared for the benefit of the Biologists working on DTI and those interested in proving and capitalising on FL's usefulness as a secure, privacy-preserving, and performance-conserving platform for sharing pharmaceutical data under regulatory and commercial constraints.

A.1.2 Experimental results

Table A.1 shows the obtained experimental performance on the proposed system in different scenarios. Learning performance differences are reported between the federation of deep model architectures and an ensemble of the same architectures. An experiment is successful if the federated model performance matches the non-private distributed alternative. The reported experimental results show that FL can retain up to 15% better performance relative to the distributed alternative while ensuring no data or any other high-level summary is revealed [23]. The general trend in the IID results points to a relative advantage for the ensembles at very low client counts that quickly dissipates, turns into parity, and, from 16 clients up, fully reverses as the client count increases. Second, the non-IID data display effective parity practically at all client counts, indicating that FL can deal with unequal data distributions much better than the distributed alternative. This matched performance makes it a clear favourite for future distributed learning research in the DTI domain, alongside FL's solid privacy and security guarantees, which are entirely lacking in the distributed alternative, Furthermore, seeing that the IID and non-IID performances are effectively matched, the FL performance developed under varying non-IID conditions is investigated.

Data non-IID-ness in DTI is two-dimensional as there are two model inputs. The proteins and the chemicals are jointly considered to predict their interaction. Consequently, it is possible to investigate the distribution one dimension at a time, either

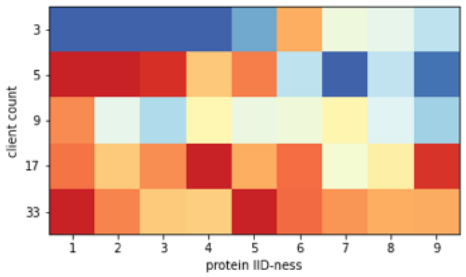
[1] https://github.com/Giemp95/FedDTI

non-IID to the protein or chemical inputs, or explore it in both dimensions simultaneously. Neither of these three approaches can be ruled out as a priory as the input classes are statistically independent. Consequently, the domain does not lend itself easily to the established notions of non-IID-ness in FL, and non-IID-ness should be tested under all three conditions. The proposed experiments investigate the entire continuum of IID-ness rather than just its two extremes. The IID data distribution is a random draw; each client has an equal chance of owning each data point. A non-IID distribution assigns proteins or drugs to specific clients, who then own all experiments that contain the protein or drug assigned to them. In the real world, these would be the laboratories looking for drugs targeting a specific protein or investigating the effects of a specific drug.

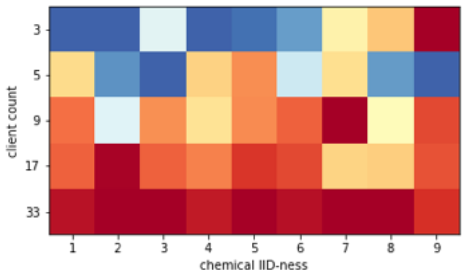
Each row of each map in Figure A.1 is obtained by first assigning to each client all experiments corresponding to an exclusive collection of either proteins or drugs. Then, at each step along the continuum, the clients exchange some of their data with their neighbours. This exchange follows a Gaussian curve, introducing an uneven representation of each data class outside its assigned client. This method is called diffusion since different IIDness levels are obtained through data diffusion from each client to its neighbours. This choice makes the distribution more realistic since it is unlikely that all clients but one would hold the same amount of data in any given class. The desired mix of protein- and drug-centric clients for the protein and drug experiments is obtained by splitting the data into two sub-datasets and then treating each as a separate one-class non-IID experiment. This scenario is closest to what is expected in the real world. Each square in the figure reports the average over ten training iterations of the given model's loss performance relative to the centralised case. The client counts presented in these figures reflect the cross-silo setup of this domain.

Figure A.1 shows the heat maps exploring the IID-ness space along the protein, drug, and both dimensions. As expected, having a higher client count hurts the performance at all non-IID-ness levels. The more fragmented the dataset, the more challenging the aggregation, as the larger client count implies fewer data per client in this setup, spoiling the individual client models. However, the different levels of IID-ness do not appear to be linked to the model's performance. In other words, while a general trend towards worse performance can be detected in each column, such a trend is less evident in the rows. This property is exciting, as it implies that it does not matter whether all client labs test the same combination of proteins or if each client has their own or substantially similar portfolio. It also means that what is a significant drain on FL's robustness in other domains is not a factor in the DTI domain.

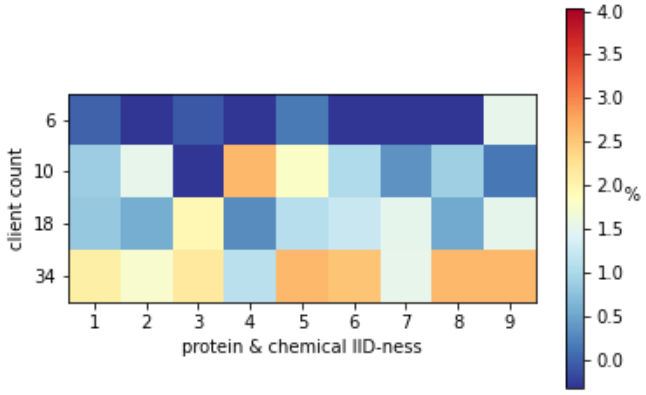
In summary, unlike in other domains in which FL has been investigated, in the DTI, due to its unique data structure, the input IID-ness does not play a significant role, making the domain singularly unique among FL domains. This observation is crucial as resilience to non-IID data distribution is usually the chief robustness metric for comparing different aggregation strategies in FL. With data distribution eliminated as a major limitation to the DTI-FL implementation's robustness, data quantity distribution, i.e.



(a) Protein-based IID-ness variation.



(b) Chemical-based IID-ness variation.

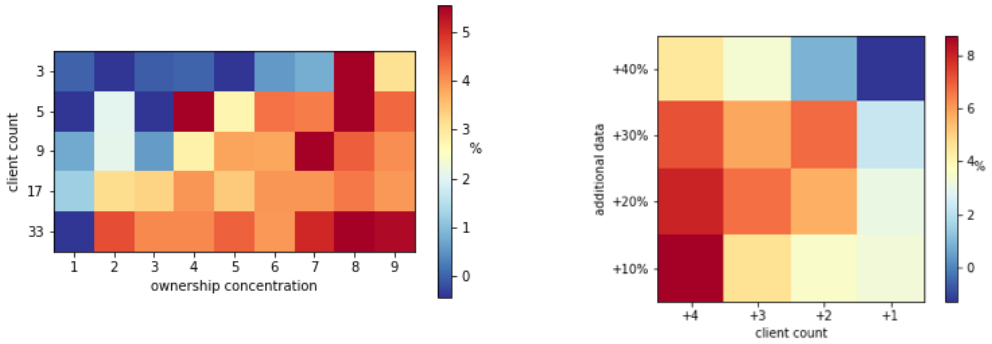


(c) Combined IID-ness variation.

Figure A.1: Learning performance (MSE) % change relative to the smallest client count and highest concentration obtained by FL-DTI at protein-based (top-left), chemical-based (top-right), protein- and chemical-based (bottom) IID-ness variation and different client counts (up to 33 clients).

uneven data ownership, is considered the next candidate for a significant performance driver.

Data distribution imbalance plays a significant role in FL's performance at DTI. Data distribution imbalance and unevenness in the data quantity among clients are of particular concern in the DTI domain, as the participant landscape is composed of a hodgepodge of big and small entities. While plausible in some domains, the often-made assumption that clients have access to about the same amount of data is contrary to the structure of the pharmaceutical industry. Moreover, when this assumption is relaxed, it is arguable that exploiting more clients will speed up the training process while, in contrast, data quantity distribution among the clients will ultimately not impact the model



(a) Data quantity IID-ness variation.

(b) Data quantity IID-ness variation over different numbers of clients.

Figure A.2: A.2a (left)): Learning performance (MSE) % change relative to the smallest client count and the highest concentration obtained by FL-DTI for a selection of client counts and a range of data quantity distributions sampled equidistantly. A.2b (right)): Learning performance (MSE) % change relative to training FL-DTI based on the dominant client's (60% of the) data is reported for adding up to 40% of extra data in increments of 10% and divided among 1 to 4 additional clients.

performance [176]. Figure A.2 challenges this assumption and examines data quantity distribution's impact on the model performance under varying client counts.

Figure A.2a investigates the interplay between client count and data quantity distribution profile. The dataset is distributed among multiple clients. The same single client is designated as the dominant client and receives a variable percentage of the data. The rest of the data is distributed unevenly among the rest of the clients following the Gaussian curve, achieving a reasonably uneven distribution in line with the approach exposed in the previous subsection.

Increasing the client count makes the problem harder, increasing the error. This time, however, the rate of performance deterioration depends on the unevenness of data allocation among the clients. At each client count, irrespective of the ownership inequality level, it holds that moving to a more concentrated data ownership favours the model's performance. This effect is significant throughout the tested conditions but grows stronger the closer the tested setup is to the highly centralised data ownership.

Crucially, the co-dependent effect is not only present in the overwhelmingly dominant client case (far left), where it could be discounted as a case of mode collapse into a pseudo-centralised setup, but it holds throughout the tested conditions. This persistence makes such observation particularly salient. There is a cost to having a diluted client data ownership structure. The next step is to investigate the interplay of this cost with the benefit of adding new data.

Figure A.2b investigates the trade-off between the benefit of adding more data to

an existing federation and the cost resulting from increasing the client count and thus diluting the client data ownership structure. The progression starts with a single client allocated a 60% share of the data. Without loss of generality, this can represent a preexisting federation of clients. The remaining 40% of the dataset is available for addition. The heat map reports the error implications from adding this data in increments of 10% distributed among 1 to 4 clients.

Predictably, increasing the amount of additional data and spreading this data among fewer clients improve model performance in Figure A.2b. What is less predictable is that the rate of improvement is about the same in both of these dimensions, which is indeed remarkable. In the tested situation, increasing the concentration of data ownership can, in some cases, have as strong a positive effect on the model's performance as adding 10% of the data. The benefit of additional data can substantially offset the cost due to the changed data ownership distribution. The symptom of this is that the top left to bottom right diagonal, where the forces work against each other, varies much less than the bottom left to top right diagonal, where they reinforce each other. The strength of this effect, and in particular its potential to overturn the benefits of substantial dataset increases, suggests questions beyond this paper's scope. Nevertheless, they are significant as they call for rethinking the data imbalance vision as a mere convergence speed issue. The leveraging of this observation and its use in the design of superior aggregation strategies is left as future work.

A.2 Solar wind speed prediction

DML is the future of onboard computation in space as it offers scalability, resilience, and flexibility that a centralised setup can not match. In the communication space, it trades in the cost of a full-dataset aggregation for an intermittent exchange of training messages. This research work explores the communication cost landscape of centralised and federated DNN training, considering a spatial use case. The issue of predicting solar wind speed days in advance is considered, and actual NASA data is exploited to carry out the investigation. Many experiments are run on Google Cloud computing resources to prove the feasibility of the proposed approach. They give materially significant recommendations relevant to the design of future space missions as they identify a substantial trade-off between the benefits of adding new data and the cost of adding more clients. This work is conducted under FDL supervision and founded by Trillium Technologies Inc.

A.2.1 Background

Onboard computation is relevant to current missions and necessary for future ones. It allows researchers to significantly reduce the communication costs associated with sending large amounts of data a long way to the Earth. Extreme solar winds can impact communication, disrupting satellites and spacecraft operativity. Accurately forecasting

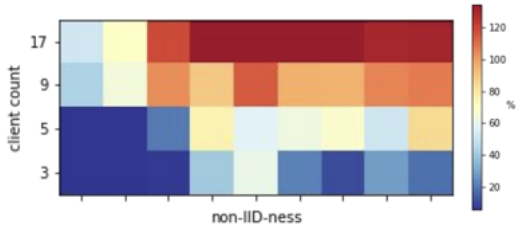
the solar wind speed onboard is thus an essential proving ground for future AI deployment in space. FL is thus proposed to train a state-of-the-art transformer model for solar wind speed prediction [30] on the Extreme UV images taken by the NASA Solar Dynamics Observatory [74, 130]. FL performance is investigated and compared to the centralised model under IID and non-IID conditions, searching for significant communication savings. Results for forecasting at a four-day lag from a single 211 Å image are presented.

A.2.2 Experimental results

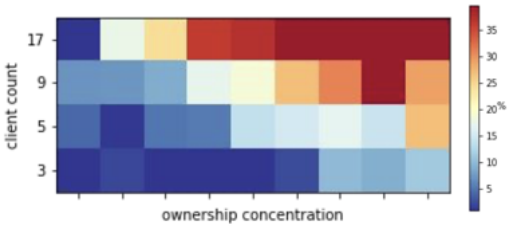
Figure A.3 present the obtained experimental results. Each subfigure investigates the interplay between the federation client count and a major driver of performance and generalisation. Heath maps are used to illustrate performance loss relative to the centralised setup. Non-IID-ness increases towards the right in Figure A.3a, as does equality of data split in Figure A.3b.

The centralised and federated training successfully replicates the expected performance observed in the original solar wind prediction paper. The proposed experiments follow the same general training curve patterns as the baseline centralised setup, achieving a statistically indistinguishable MSE of 0.098. However, they differ markedly in their communication costs. The basic distributed (non-federated) setup requires the communication of all gradients at each training step. In the proposed setup, this is 330MB of gradients 1095 times per epoch or about 361.35GB in total. This number is unfavourable to the 35GB size of the entire dataset. Meanwhile, FL can work with communicating a comparable 330MB tensor of data just once per epoch in the most conservative and basic setup.

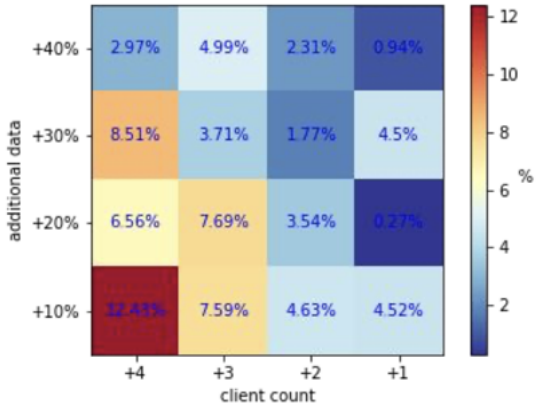
The benchmark performance experiments confirm that distributed setups replicate the centralised case for up to 8 clients. Beyond this point, there is a 20% performance fall. Ownership concentration experiments dive into the issue of client count tolerance dependence on the data ownership structure. They show that thinly spread data distribution patterns hamper convergence as client models fail to converge. Non-IID-ness tolerance experiments investigate FL's resilience to increasing client counts as data distribution changes. As each client's data became less diverse, i.e., focused only on a specific solar cycle period (more non-IID), the system's tolerance for a higher client count decreased. Indeed, the proposed experiments demonstrate this data-ownership tradeoff. A 40% of the data is held out, resulting in an apparent performance loss. Adding it back in tranches of 10% and spread among 1-4 clients clearly shows that the highest rate of improvement is achieved when the data is added in through the smallest number of clients. Indeed, adding the first 10% in a single chunk gives 3x relative improvement than when spread out. Notably, the similarity between the 1-way and 2-way splits suggests a minimal sufficient data requirement. Put together, it is necessary to pay close attention to the client data collection, ownership structure, and concentration for the benefits of distributed training to be realised, as these were shown to be persistent and



(a) Solar wind-based IID-ness variation.



(b) Data quantity IID-ness variation.



(c) Data quantity IID-ness variation over different numbers of clients.

Figure A.3: Learning performance (MSE) % change relative to the smallest client count and highest concentration obtained by FL at different levels of solar wind-based (A.3a) and data quantity-based (A.3b) non-IIDness. A.3c): Learning performance (MSE) % change relative to training solar-FL solely based on the dominant client's (60% of the) data is reported for adding up to 40% of extra data in increments of 10% and divided among 1 to 4 additional clients.

material determinants of performance.

It is finally possible to asses that FL significantly lowers the communication cost of message passing relative to its distributed peers. Furthermore, the proposed extensive battery of experiments shows that the observed results are robust to a wide array of changes in the client count and the degree of data distribution heterogeneity.

Acknowledgements

This work receives EuroHPC-JU funding under grant No. 101034126, with support from the Horizon2020 programme (the European PILOT). This work is also supported by the Spoke ”FutureHPC & BigData” of the ICSC - Centro Nazionale di Ricerca in ”High-Performance Computing, Big Data and Quantum Computing”, funded by the European Union - NextGenerationEU.

Nomenclature

Acronyms / Abbreviations

ACS Acute Coronary Syndrome

AFC Anti Financial Crime

AI Artificial Intelligence

ANN Artificial Neural Network

ASIC Application-Specific Integrated Circuit

AUC Area Under the ROC Curve

BARC-MB Bleeding Academic Research Consortium major bleeding

CNN Convolutional Neural Network

CWL Common Workflow Language

DL Deep Learning

DNN Deep Neural Network

DP Differential Privacy

DSL Domain Specific Language

EI Edge Inference

FL Federated Learning

FLaaS Federated Learning as a Service

FLOPS FLoating point Operations per Second

FPGA Field-Programmable Gate Array

FPR False Positive Rate

GDPR General Data Protection Regulation

GPU Graphical Processing Unit

HE Homomorphic Encryption

HPC High-Performance Computing

IID Independent and Identically Distributed

IoT Internet of Things

ISA Instruction Set Architecture

KNN K-Nearest Neighbours

LLM Large Language Model

LSTM Long-Short Term Memory

LVEF Left Ventricular Ejection Fraction

MACE Major Adverse Cardiac Events

MAFL Model Agnostic Federated Learning

ML Machine Learning

MLP Multi-Layer Perceptron

MPI Message Passing Interface

NLP Natural Language Processing

PCA Principal Component Analysis

PH Proportional Hazard

ReAMI Recurrent Acute Myocardial Infarction

ROC Receiving Operating Characteristic

RPC Remote Procedure Call

SGD Stochastic Gradient Descent

SMC Secure Multi-party Comuptation

SMP Secure Multi-party Comuptation

SVM Support Vector Machine

TEE Trusted Execution Environment

TPR True Positive Rate

TPU Tensor Processing Unit

VM Virtual Machine

WMS Workflow Management System

Bibliography

[1] Keith D. Aaronson et al. "Development and Prospective Validation of a Clinical Index to Predict Survival in Ambulatory Patients Referred for Cardiac Transplant Evaluation." In: *Circulation* 95.12 (1997), pp. 2660–2667. DOI: 10.1161/01.cir.95.12.2660. eprint: https://www.ahajournals.org/doi/pdf/10.1161/01.CIR.95.12.2660. URL: https://www.ahajournals.org/doi/abs/10.1161/01.CIR.95.12.2660.

[2] Giovanni Agosta et al. "TEXTAROSSA: Towards EXtreme scale Technologies and Accelerators for euROhpc hw/Sw Supercomputing Applications for exascale." In: *24th Euromicro Conference on Digital System Design, DSD 2021, Virtual Event / Palermo, Sicily, Italy, September 1-3, 2021*. Ed. by Francesco Leporati, Salvatore Vitabile, and Amund Skavhaug. Palermo, Italy: Ieee, Aug. 2021, pp. 286–294. ISBN: 978-1-6654-2703-6. DOI: 10.1109/dsd53832.2021.00051. URL: https://doi.org/10.1109/DSD53832.2021.00051.

[3] Giovanni Agosta et al. "Towards EXtreme scale technologies and accelerators for euROhpc hw/Sw supercomputing applications for exascale: The TEXTAROSSA approach." In: *Microprocess. Microsystems* 95.C (Nov. 2022), p. 104679. ISSN: 0141-9331. DOI: 10.1016/j.micpro.2022.104679. URL: https://doi.org/10.1016/j.micpro.2022.104679.

[4] Samson B. Akintoye et al. "Layer-wise partitioning and merging for efficient and scalable deep learning." In: *Future Gener. Comput. Syst.* 149 (2023), pp. 432–444. DOI: 10.1016/j.future.2023.07.043. URL: https://doi.org/10.1016/j.future.2023.07.043.

[5] Marco Aldinucci et al. "An Efficient Unbounded Lock-Free Queue for Multi-core Systems." In: *Euro-Par 2012 Parallel Processing - 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings*. Ed. by Christos Kaklamanis, Theodore S. Papatheodorou, and Paul G. Spirakis. Vol. 7484. Lecture Notes in Computer Science. Springer, 2012, pp. 662–673. DOI: 10.1007/978-3-642-32820-6_65. URL: https://doi.org/10.1007/978-3-642-32820-6%5C_65.

[6] Marco Aldinucci et al. "Design patterns percolating to parallel programming framework implementation." In: *Int. J. Parallel Program.* 42.6 (2014), pp. 1012–1031. DOI: 10.1007/s10766-013-0273-6. URL: https://doi.org/10.1007/s10766-013-0273-6.

[7] Marco Aldinucci et al. "Fastflow: High-Level and Efficient Streaming on Multi-core." In: *Programming multi-core and many-core computing systems*. John Wiley & Sons, Ltd, 2017. Chap. 13, pp. 261–280. ISBN: 978-1-119-33201-5. DOI: https://doi.org/10.1002/9781119332015.ch13. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119332015.ch13. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119332015.ch13.

[8] Marco Aldinucci et al. "HPC4AI: an AI-on-demand federated platform endeavour." In: *Proceedings of the 15th ACM International Conference on Computing Frontiers, CF 2018, Ischia, Italy, May 08-10, 2018*. Ed. by David R. Kaeli and Miquel Pericàs. Acm, 2018, pp. 279–286. DOI: 10.1145/3203217.3205340. URL: https://doi.org/10.1145/3203217.3205340.

[9] Marco Aldinucci et al. "OCCAM: a flexible, multi-purpose and extendable HPC cluster." In: *CoRR* abs/1709.03715 (2017). arXiv: 1709.03715. URL: http://arxiv.org/abs/1709.03715.

[10] Marco Aldinucci et al. "Practical parallelization of scientific applications with OpenMP, OpenACC and MPI." In: *J. Parallel Distributed Comput.* 157 (Nov. 2021), pp. 13–29. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2021.05.017. URL: https://doi.org/10.1016/j.jpdc.2021.05.017.

[11] Faisal Alsakran et al. "Intrusion Detection Systems for Smart Home IoT Devices: Experimental Comparison Study." In: *Security in Computing and Communications - 7th International Symposium, SSCC 2019, Trivandrum, India, December 18-21, 2019, Revised Selected Papers*. Ed. by Sabu M. Thampi et al. Vol. 1208. Communications in Computer and Information Science. Springer, 2019, pp. 87–98. DOI: 10.1007/978-981-15-4825-3_7. URL: https://doi.org/10.1007/978-981-15-4825-3%5C_7.

[12] Ali Anaissi, Basem Suleiman, and Widad Alyassine. "A Personalized Federated Learning Algorithm for One-Class Support Vector Machine: An Application in Anomaly Detection." In: *Computational Science - ICCS 2022 - 22nd International Conference, London, UK, June 21-23, 2022, Proceedings, Part IV*. Ed. by Derek Groen et al. Vol. 13353. Lecture Notes in Computer Science. Springer, 2022, pp. 373–379. DOI: 10.1007/978-3-031-08760-8_31. URL: https://doi.org/10.1007/978-3-031-08760-8%5C_31.

[13] Yasir Arfat et al. "Machine learning for cardiology." In: *Minerva cardiology and angiology* 70.1 (Feb. 2022), pp. 75–91. ISSN: 2724-5683. DOI: 10.23736/s2724-5683.21.05709-4. URL: https://doi.org/10.23736/s2724-5683.21.05709-4.

[14] Yasir Arfat et al. "Pooling critical datasets with Federated Learning." In: *31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2023, Naples, Italy, March 1-3, 2023*. Ed. by Raffaele Montella, Javier García Blas, and Daniele D'Agostino. Naples, Italy: Ieee, Mar. 2023, pp. 329–337. ISBN: 979-8-3503-3763-1. DOI: 10.1109/pdp59025.2023.00057. URL: https://doi.org/10.1109/PDP59025.2023.00057.

[15] Sercan Ö. Arik and Tomas Pfister. "TabNet: Attentive Interpretable Tabular Learning." In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 6679–6687. DOI: 10.1609/aaai.v35i8.16826. URL: https://doi.org/10.1609/aaai.v35i8.16826.

[16] The TensorFlow Federated Authors. *TensorFlow Federated*. Version 0.48.0. Google, Feb. 2019. URL: https://github.com/tensorflow/federated.

[17] Usman Baber et al. "Coronary Thrombosis and Major Bleeding After PCI With Drug-Eluting Stents." In: *Journal of the American College of Cardiology* 67.19 (2016), pp. 2224–2234. DOI: 10.1016/j.jacc.2016.02.064. eprint: https://www.jacc.org/doi/pdf/10.1016/j.jacc.2016.02.064. URL: https://www.jacc.org/doi/abs/10.1016/j.jacc.2016.02.064.

[18] Daniel Balouek-Thomert et al. "Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows." In: *Int. J. High Perform. Comput. Appl.* 33.6 (2019). DOI: 10.1177/1094342019877383. URL: https://doi.org/10.1177/1094342019877383.

[19] Andrea Bartolini et al. "Monte Cimone: Paving the Road for the First Generation of RISC-V High-Performance Computers." In: *35th IEEE International System-on-Chip Conference, SOCC 2022, Belfast, United Kingdom, September 5-8, 2022*. Ed. by Sakir Sezer et al. Ieee, 2022, pp. 1–6. DOI: 10.1109/socc56010.2022.9908096. URL: https://doi.org/10.1109/SOCC56010.2022.9908096.

[20] Eric B. Baum. "On the capabilities of multilayer perceptrons." In: *J. Complex.* 4.3 (1988), pp. 193–215. DOI: 10.1016/0885-064x(88)90020-9. URL: https://doi.org/10.1016/0885-064X(88)90020-9.

[21] Enrique Tomás Martínez Beltrán et al. "Decentralized Federated Learning: Fundamentals, State of the Art, Frameworks, Trends, and Challenges." In: *IEEE Commun. Surv. Tutorials* 25.4 (2023), pp. 2983–3013. DOI: 10.1109/comst.2023.3315746. URL: https://doi.org/10.1109/COMST.2023.3315746.

[22] Tamay Besiroglu et al. "The Compute Divide in Machine Learning: A Threat to Academic Contribution and Scrutiny?" In: *CoRR* abs/2401.02452 (2024). DOI: 10.48550/ARXIV.2401.02452. arXiv: 2401.02452. URL: https://doi.org/10.48550/arXiv.2401.02452.

[23] Daniel J. Beutel et al. "Flower: A Friendly Federated Learning Research Framework." In: *CoRR* abs/2007.14390 (2020). arXiv: 2007.14390. URL: https://arxiv.org/abs/2007.14390.

[24] Mark S. Birrittella et al. "Intel® Omni-path Architecture: Enabling Scalable, High Performance Fabrics." In: *23rd IEEE Annual Symposium on High-Performance Interconnects, HOTI 2015, Santa Clara, CA, USA, August 26-28, 2015*. IEEE Computer Society, 2015, pp. 1–9. DOI: 10.1109/hoti.2015.22. URL: https://doi.org/10.1109/HOTI.2015.22.

[25] Ekaba Bisong. "An Overview of Google Cloud Platform Services." In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 7–10. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_2. URL: https://doi.org/10.1007/978-1-4842-4470-8%5C_2.

[26] Léon Bottou et al. "Stochastic gradient learning in neural networks." In: *Proceedings of Neuro-Nımes* 91.8 (1991), p. 12.

[27] Leo Breiman. "Bagging Predictors." In: *Mach. Learn.* 24.2 (1996), pp. 123–140. DOI: 10.1007/bf00058655. URL: https://doi.org/10.1007/BF00058655.

[28] Leo Breiman et al. *Classification and Regression Trees*. Wadsworth, 1984. ISBN: 0-534-98053-8.

[29] André Brinkmann et al. "Ad Hoc File Systems for High-Performance Computing." In: *J. Comput. Sci. Technol.* 35.1 (2020), pp. 4–26. DOI: 10.1007/s11390-020-9801-1. URL: https://doi.org/10.1007/s11390-020-9801-1.

[30] Edward J. E. Brown et al. "Attention-Based Machine Vision Models and Techniques for Solar Wind Speed Forecasting Using Solar EUV Images." In: *Space Weather* 20.3 (2022). e2021SW002976 2021SW002976, e2021SW002976. DOI: https://doi.org/10.1029/2021SW002976. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2021SW002976. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2021SW002976.

[31] Nadia Burkart and Marco F. Huber. "A Survey on the Explainability of Supervised Machine Learning." In: *J. Artif. Intell. Res.* 70 (2021), pp. 245–317. DOI: 10.1613/jair.1.12228. URL: https://doi.org/10.1613/jair.1.12228.

[32] David Byrd and Antigoni Polychroniadou. "Differentially private secure multi-party computation for federated learning in financial applications." In: *ICAIF '20: The First ACM International Conference on AI in Finance, New York, NY, USA, October 15-16, 2020*. Ed. by Tucker Balch. Acm, 2020, 16:1–16:9. DOI: 10.1145/3383455.3422562. URL: https://doi.org/10.1145/3383455.3422562.

[33] Robert A Byrne et al. "2023 ESC Guidelines for the management of acute coronary syndromes: Developed by the task force on the management of acute coronary syndromes of the European Society of Cardiology (ESC)." In: *European Heart Journal* 44.38 (Aug. 2023), pp. 3720–3826. ISSN: 0195-668x. DOI: 10.1093/eurheartj/ehad191. eprint: https://academic.oup.com/eurheartj/article-pdf/44/38/3720/52018153/ehad191.pdf. URL: https://doi.org/10.1093/eurheartj/ehad191.

[34] Sebastian Caldas et al. "LEAF: A Benchmark for Federated Settings." In: *CoRR* abs/1812.01097 (2018). arXiv: 1812.01097. URL: http://arxiv.org/abs/1812.01097.

[35] Diogo V. Carvalho, Eduardo M. Pereira, and Jaime S. Cardoso. "Machine Learning Interpretability: A Survey on Methods and Metrics." In: *Electronics* 8.8 (2019). ISSN: 2079-9292. DOI: 10.3390/electronics8080832. URL: https://www.mdpi.com/2079-9292/8/8/832.

[36] Bruno Casella et al. "Experimenting With Normalization Layers in Federated Learning on Non-IID Scenarios." In: *IEEE Access* 12 (2024), pp. 47961–47971. DOI: 10.1109/access.2024.3383783. URL: https://doi.org/10.1109/ACCESS.2024.3383783.

[37] Lidia Ceriani and Paolo Verme. "The origins of the Gini index: extracts from Variabilitàe Mutabilità(1912) by Corrado Gini." In: *The Journal of Economic Inequality* 10.3 (2012), pp. 421–443. DOI: 10.1007/s10888-011-9188-x. URL: https://doi.org/10.1007/s10888-011-9188-x.

[38] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, eds. *Semi-Supervised Learning*. The MIT Press, 2006. ISBN: 978-0-262-03358-9. DOI: 10.7551/mitpress/9780262033589.001.0001. URL: https://doi.org/10.7551/mitpress/9780262033589.001.0001.

[39] Tatjana Chavdarova et al. "WILDTRACK: A Multi-Camera HD Dataset for Dense Unscripted Pedestrian Detection." In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 5030–5039. DOI: 10.1109/CVPR.2018.00528. URL: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Chavdarova%5C_WILDTRACK%5C_A%5C_Multi-Camera%5C_CVPR%5C_2018%5C_paper.html.

[40] Haokun Chen et al. "FedDAT: An Approach for Foundation Model Finetuning in Multi-Modal Heterogeneous Federated Learning." In: *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*. Ed. by Michael J. Wooldridge, Jennifer G. Dy, and Sriraam

Natarajan. AAAI Press, 2024, pp. 11285–11293. DOI: 10.1609/AAAI.V38I10.29007. URL: https://doi.org/10.1609/aaai.v38i10.29007.

[41] Shaoqi Chen et al. "FL-QSAR: a federated learning-based QSAR prototype for collaborative drug discovery." In: *Bioinform.* 36.22-23 (2021), pp. 5492–5498. DOI: 10.1093/bioinformatics/btaa1006. URL: https://doi.org/10.1093/bioinformatics/btaa1006.

[42] Songqing Chen, Tao Zhang, and Weisong Shi. "Fog Computing." In: *IEEE Internet Comput.* 21.2 (2017), pp. 4–6. DOI: 10.1109/mic.2017.39. URL: https://doi.org/10.1109/MIC.2017.39.

[43] Yujing Chen et al. "Asynchronous Online Federated Learning for Edge Devices with Non-IID Data." In: *2020 IEEE International Conference on Big Data (IEEE BigData 2020), Atlanta, GA, USA, December 10-13, 2020*. Ed. by Xintao Wu et al. Ieee, 2020, pp. 15–24. DOI: 10.1109/bigdata50022.2020.9378161. URL: https://doi.org/10.1109/BigData50022.2020.9378161.

[44] Kewei Cheng et al. "SecureBoost: A Lossless Federated Learning Framework." In: *IEEE Intell. Syst.* 36.6 (2021), pp. 87–98. DOI: 10.1109/mis.2021.3082561. URL: https://doi.org/10.1109/MIS.2021.3082561.

[45] Jaeyoung Choi, David W. Walker, and Jack J. Dongarra. "The Design of Scalable Software Libraries for Distributed Memory Concurrent Computers." In: *Proceedings of the 8th International Symposium on Parallel Processing, Cancún, Mexico, April 1994*. Ed. by Howard Jay Siegel. IEEE Computer Society, 1994, pp. 792–799. DOI: 10.1109/ipps.1994.288214. URL: https://doi.org/10.1109/IPPS.1994.288214.

[46] Maxime C. Cohen et al. "Overcommitment in Cloud Services: Bin Packing with Chance Constraints." In: *Manag. Sci.* 65.7 (2019), pp. 3255–3271. DOI: 10.1287/mnsc.2018.3091. URL: https://doi.org/10.1287/mnsc.2018.3091.

[47] Michael Collier and Robin Shahan. *Microsoft Azure Essentials-Fundamentals of Azure*. Redmond, Washington: Microsoft Press, 2015. ISBN: 978-0-7356-9722-5.

[48] Iacopo Colonnelli et al. "Federated Learning meets HPC and cloud." In: *Machine Learning for Astrophysics (ML4Astro 2022)*. Vol. 60. Astrophysics and Space Science Proceedings. Catania, Italy: Springer, Cham, June 2023, pp. 193–199. ISBN: 978-3-031-34167-0. DOI: 10.1007/978-3-031-34167-0_39. URL: https://doi.org/10.1007/978-3-031-34167-0%5C_39.

[49] Iacopo Colonnelli et al. "StreamFlow: Cross-Breeding Cloud With HPC." In: *IEEE Trans. Emerg. Top. Comput.* 9.4 (2021), pp. 1723–1737. DOI: 10.1109/tetc.2020.3019202. URL: https://doi.org/10.1109/TETC.2020.3019202.

[50] Francesco Costa et al. "Derivation and validation of the predicting bleeding complications in patients undergoing stent implantation and subsequent dual antiplatelet therapy (PRECISE-DAPT) score: a pooled analysis of individual-patient datasets from clinical trials." In: *The Lancet* 389.10073 (2017), pp. 1025–1034. DOI: 10.1016/s0140-6736(17)30397-5. URL: https://doi.org/10.1016/S0140-6736(17)30397-5.

[51] D. R. Cox. "Regression Models and Life-Tables." In: *Journal of the Royal Statistical Society: Series B (Methodological)* 34.2 (1972), pp. 187–202. DOI: https://doi.org/10.1111/j.2517-6161.1972.tb00899.x. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1972.tb00899.x. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1972.tb00899.x.

[52] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. ISBN: 978-1-107-04305-3. URL: http://www.cambridge.org/de/academic/subjects/computer-science/cryptography-cryptology-and-coding/secure-multiparty-computation-and-secret-sharing?format=HB%5C%7B%5C&%7Disbn=9781107043053.

[53] Fabrizio D'Ascenzo et al. "Forecasting the Risk of Heart Failure Hospitalization After Acute Coronary Syndromes: the CORALYS HF Score." In: *The American Journal of Cardiology* 206.1 (Nov. 2023), pp. 320–329. ISSN: 0002-9149. DOI: 10.1016/j.amjcard.2023.08.010. URL: https://doi.org/10.1016/j.amjcard.2023.08.010.

[54] Fabrizio D'Ascenzo et al. "Machine learning-based prediction of adverse events following an acute coronary syndrome (PRAISE): a modelling study of pooled datasets." In: *The Lancet* 397.10270 (Jan. 2021), pp. 199–207. ISSN: 0140-6736. DOI: 10.1016/s0140-6736(20)32519-8. URL: https://doi.org/10.1016/S0140-6736(20)32519-8.

[55] Jeffrey Dean et al. "Large Scale Distributed Deep Networks." In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. Ed. by Peter L. Bartlett et al. 2012, pp. 1232–1240. URL: https://proceedings.neurips.cc/paper/2012/hash/6aca97005c68f1206823815f66102863-Abstract.html.

[56] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. "Adaptive Personalized Federated Learning." In: *CoRR* abs/2003.13461 (2020). arXiv: 2003.13461. URL: https://arxiv.org/abs/2003.13461.

[57] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. "The LINPACK Benchmark: past, present and future." In: *Concurr. Comput. Pract. Exp.* 15.9 (2003), pp. 803–820. DOI: 10.1002/cpe.728. URL: https://doi.org/10.1002/cpe.728.

[58] Jack J. Dongarra and Aad J. van der Steen. "High-performance computing systems: Status and outlook." In: *Acta Numer.* 21 (2012), pp. 379–474. DOI: 10.1017/s0962492912000050. URL: https://doi.org/10.1017/S0962492912000050.

[59] Zhixu Du et al. "Rethinking Normalization Methods in Federated Learning." In: *CoRR* abs/2210.03277 (2022). DOI: 10.48550/arxiv.2210.03277. arXiv: 2210.03277. URL: https://doi.org/10.48550/arXiv.2210.03277.

[60] Cynthia Dwork. "Differential Privacy." In: *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*. Ed. by Michele Bugliesi et al. Vol. 4052. Lecture Notes in Computer Science. Springer, 2006, pp. 1–12. DOI: 10.1007/11787006_1. URL: https://doi.org/10.1007/11787006%5C_1.

[61] Pantelis Elinas, Edwin V. Bonilla, and Louis C. Tiao. "Variational Inference for Graph Convolutional Networks in the Absence of Graph Data and Adversarial Settings." In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/d882050bb9eeba930974f596931be527-Abstract.html.

[62] Patricia Takako Endo et al. "High availability in clouds: systematic review and research challenges." In: *J. Cloud Comput.* 5 (2016), p. 16. DOI: 10.1186/s13677-016-0066-8. URL: https://doi.org/10.1186/s13677-016-0066-8.

[63] E M Fajardo et al. "How much higher can HTCondor fly?" In: *Journal of Physics: Conference Series* 664.6 (Dec. 2015), p. 062014. DOI: 10.1088/1742-6596/664/6/062014. URL: https://dx.doi.org/10.1088/1742-6596/664/6/062014.

[64] Peter Flach. "Binary classification and related tasks." In: *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012, pp. 49–80. ISBN: 978-1-107-42222-3. DOI: 10.1017/cbo9780511973000.004. URL: https://doi.org/10.1017/CBO9780511973000.004.

[65] Peter Flach. "Distance-based models." In: *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012, pp. 231–261. ISBN: 978-1-107-42222-3. DOI: 10.1017/cbo9780511973000.010. URL: https://doi.org/10.1017/CBO9780511973000.010.

[66] Peter Flach. "Linear models." In: *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012, pp. 194–230. ISBN: 978-1-107-42222-3. DOI: 10.1017/cbo9780511973000.009. URL: https://doi.org/10.1017/CBO9780511973000.009.

[67] Peter Flach. "Model ensembles." In: *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012, pp. 330–342. ISBN: 978-1-107-42222-3. DOI: 10.1017/cbo9780511973000.013. URL: https://doi.org/10.1017/CBO9780511973000.013.

[68] Peter Flach. "Probabilistic models." In: *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012, pp. 262–297. ISBN: 978-1-107-42222-3. DOI: 10.1017/cbo9780511973000.011. URL: https://doi.org/10.1017/CBO9780511973000.011.

[69] Peter Flach. "The ingredients of machine learning." In: *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012, pp. 13–48. ISBN: 978-1-107-42222-3. DOI: 10.1017/cbo9780511973000.003. URL: https://doi.org/10.1017/CBO9780511973000.003.

[70] François Fleuret et al. "Multicamera People Tracking with a Probabilistic Occupancy Map." In: *IEEE Trans. Pattern Anal. Mach. Intell.* 30.2 (2008), pp. 267–282. DOI: 10.1109/TPAMI.2007.1174. URL: https://doi.org/10.1109/TPAMI.2007.1174.

[71] Patrick Foley et al. "OpenFL: the open federated learning library." In: *Physics in Medicine & Biology* 67.21 (Oct. 2022), p. 214001. DOI: 10.1088/1361-6560/ac97d9. URL: https://dx.doi.org/10.1088/1361-6560/ac97d9.

[72] William Fornaciari et al. "RISC-V-Based Platforms for HPC: Analyzing Non-functional Properties for Future HPC and Big-Data Clusters." In: *Embedded Computer Systems: Architectures, Modeling, and Simulation - 23rd International Conference, SAMOS 2023, Samos, Greece, July 2-6, 2023, Proceedings*. Ed. by Cristina Silvano, Christian Pilato, and Marc Reichenbach. Vol. 14385. Lecture Notes in Computer Science. Samos, Greece: Springer, July 2023, pp. 395–410. ISBN: 978-3-031-46076-0. DOI: 10.1007/978-3-031-46077-7_26. URL: https://doi.org/10.1007/978-3-031-46077-7%5C_26.

[73] Guglielmo Gallone et al. "Impact of Left Ventricular Ejection Fraction on Procedural and Long-Term Outcomes of Bifurcation Percutaneous Coronary Intervention." In: *The American Journal of Cardiology* 172.1 (June 2022), pp. 18–25. ISSN: 0002-9149. DOI: 10.1016/j.amjcard.2022.02.015. URL: https://doi.org/10.1016/j.amjcard.2022.02.015.

[74] Richard Galvez et al. "A Machine-learning Data Set Prepared from the NASA Solar Dynamics Observatory Mission." In: *The Astrophysical Journal Supplement Series* 242.1 (May 2019), p. 7. DOI: 10.3847/1538-4365/ab1005. URL: https://dx.doi.org/10.3847/1538-4365/ab1005.

[75] Todd Gamblin et al. "The Spack package manager: bringing order to HPC software chaos." In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*. Ed. by Jackie Kern and Jeffrey S. Vetter. Acm, 2015, 40:1–40:12. DOI: 10.1145/2807591.2807623. URL: https://doi.org/10.1145/2807591.2807623.

[76] Philippe Généreux et al. "Incidence, Predictors, and Impact of Post-Discharge Bleeding After Percutaneous Coronary Intervention." In: *Journal of the American College of Cardiology* 66.9 (2015), pp. 1036–1045. DOI: 10.1016/j.jacc.2015.06.1323. eprint: https://www.jacc.org/doi/pdf/10.1016/j.jacc.2015.06.1323. URL: https://www.jacc.org/doi/abs/10.1016/j.jacc.2015.06.1323.

[77] Harshvardhan GM et al. "A comprehensive survey and analysis of generative models in machine learning." In: *Comput. Sci. Rev.* 38 (2020), p. 100285. DOI: 10.1016/j.cosrev.2020.100285. URL: https://doi.org/10.1016/j.cosrev.2020.100285.

[78] Tiago Gomes et al. "Towards an FPGA-based network layer filter for the Internet of Things edge devices." In: *21st IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2016, Berlin, Germany, September 6-9, 2016*. IEEE, 2016, pp. 1–4. DOI: 10.1109/ETFA.2016.7733684. URL: https://doi.org/10.1109/ETFA.2016.7733684.

[79] Shane W. Grant and Randolph Voorhies. *Cereal a C++11 library for serialization*. Version 1.3.2. USCiLab, July 2013. URL: https://github.com/USCiLab/cereal.

[80] Luca Greco et al. "Trends in IoT based solutions for health care: Moving AI to the edge." In: *Pattern Recognit. Lett.* 135 (2020), pp. 346–353. DOI: 10.1016/J.PATREC.2020.05.016. URL: https://doi.org/10.1016/j.patrec.2020.05.016.

[81] Chaoyang He et al. "FedML: A Research Library and Benchmark for Federated Machine Learning." In: *CoRR* abs/2007.13518 (2020). arXiv: 2007.13518. URL: https://arxiv.org/abs/2007.13518.

[82] Marti A. Hearst. "Trends & Controversies: Support Vector Machines." In: *IEEE Intell. Syst.* 13.4 (1998), pp. 18–28. DOI: 10.1109/5254.708428. URL: https://doi.org/10.1109/5254.708428.

[83] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. "Learning Non-maximum Suppression." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 6469–6477. DOI: 10.1109/CVPR.2017.685. URL: https://doi.org/10.1109/CVPR.2017.685.

[84] Yunzhong Hou, Liang Zheng, and Stephen Gould. "Multiview Detection with Feature Perspective Transformation." In: *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part VII*. Ed. by Andrea Vedaldi et al. Vol. 12352. Lecture Notes in Computer Science. Springer, 2020, pp. 1–18. DOI: 10.1007/978-3-030-58571-6_1. URL: https://doi.org/10.1007/978-3-030-58571-6%5C_1.

[85] Tzu-Hsiang Hsu et al. "AI Edge Devices Using Computing-In-Memory and Processing-In-Sensor: From System to Device." In: *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2019, pp. 22.5.1–22.5.4. DOI: 10.1109/IEDM19573.2019.8993452. URL: https://doi.org/10.1109/IEDM19573.2019.8993452.

[86] Chao Huang et al. "Promoting Collaboration in Cross-Silo Federated Learning: Challenges and Opportunities." In: *IEEE Commun. Mag.* 62.4 (2024), pp. 82–88. DOI: 10.1109/mcom.005.2300467. URL: https://doi.org/10.1109/MCOM.005.2300467.

[87] Gao Huang et al. "Densely Connected Convolutional Networks." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 2261–2269. DOI: 10.1109/cvpr.2017.243. URL: https://doi.org/10.1109/CVPR.2017.243.

[88] Kexin Huang et al. "DeepPurpose: a deep learning library for drug-target interaction prediction." In: *Bioinform.* 36.22-23 (2021), pp. 5545–5547. DOI: 10.1093/bioinformatics/btaa1005. URL: https://doi.org/10.1093/bioinformatics/btaa1005.

[89] *Introducing the AI Research SuperCluster – Meta's cutting-edge AI supercomputer for AI research*. 2022. URL: https://ai.meta.com/blog/ai-rsc/ (visited on 04/18/2024).

[90] Keith R. Jackson et al. "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud." In: *Cloud Computing, Second International Conference, CloudCom 2010, November 30 - December 3, 2010, Indianapolis, Indiana, USA, Proceedings*. IEEE Computer Society, 2010, pp. 159–168. DOI: 10.1109/cloudcom.2010.69. URL: https://doi.org/10.1109/CloudCom.2010.69.

[91] Nikhil Jain et al. "Evaluating HPC networks via simulation of parallel workloads." In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, UT, USA, November 13-18, 2016*. Ed. by John West and Cherri M. Pancake. IEEE Computer Society, 2016, pp. 154–165. DOI: 10.1109/sc.2016.13. URL: https://doi.org/10.1109/SC.2016.13.

[92] Madhura Joshi, Ankit Pal, and Malaikannan Sankarasubbu. "Federated Learning for Healthcare Domain - Pipeline, Applications and Challenges." In: *ACM Trans. Comput. Heal.* 3.4 (2022), 40:1–40:36. DOI: 10.1145/3533708. URL: https://doi.org/10.1145/3533708.

[93] Norman P. Jouppi et al. "Motivation for and Evaluation of the First Tensor Processing Unit." In: *IEEE Micro* 38.3 (2018), pp. 10–19. DOI: 10.1109/mm.2018.032271057. URL: https://doi.org/10.1109/MM.2018.032271057.

[94] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. "Reinforcement Learning: A Survey." In: *J. Artif. Intell. Res.* 4 (1996), pp. 237–285. DOI: 10.1613/jair.301. URL: https://doi.org/10.1613/jair.301.

[95] Peter Kairouz et al. "Advances and Open Problems in Federated Learning." In: *Found. Trends Mach. Learn.* 14.1-2 (2021), pp. 1–210. DOI: 10.1561/2200000083. URL: https://doi.org/10.1561/2200000083.

[96] Sai Praneeth Karimireddy et al. "Breaking the centralized barrier for cross-device federated learning." In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato et al. 2021, pp. 28663–28676. URL: https://proceedings.neurips.cc/paper/2021/hash/f0e6be4ce76ccfa73c5a540d992d0756-Abstract.html.

[97] Sai Praneeth Karimireddy et al. "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning." In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. Pmlr, 2020, pp. 5132–5143. URL: http://proceedings.mlr.press/v119/karimireddy20a.html.

[98] J Kin, M Gupta, and WHM Smith. *Application specific integrated circuits*. Ieee, 1997.

[99] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[100] Carl Kingsford and Steven L Salzberg. "What are decision trees?" In: *Nature Biotechnology* 26.9 (2008), pp. 1011–1013. DOI: 10.1038/nbt0908-1011. URL: https://doi.org/10.1038/nbt0908-1011.

[101] Hermann Kopetz and Wilfried Steiner. "Internet of Things." In: *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Cham: Springer International Publishing, 2022, pp. 325–341. ISBN: 978-3-031-11992-7. DOI: 10.1007/978-3-031-11992-7_13. URL: https://doi.org/10.1007/978-3-031-11992-7%5C_13.

[102] Oliver Kramer. "Scikit-Learn." In: *Machine Learning for Evolution Strategies*. Cham: Springer International Publishing, 2016, pp. 45–53. ISBN: 978-3-319-33383-0. DOI: 10.1007/978-3-319-33383-0_5. URL: https://doi.org/10.1007/978-3-319-33383-0%5C_5.

[103] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. "Singularity: Scientific containers for mobility of compute." In: *Plos One* 12.5 (May 2017), pp. 1–20. DOI: 10.1371/journal.pone.0177459. URL: https://doi.org/10.1371/journal.pone.0177459.

[104] Aleksandar Lazarevic and Zoran Obradovic. "Boosting Algorithms for Parallel and Distributed Learning." In: *Distributed Parallel Databases* 11.2 (2002), pp. 203–229. DOI: 10.1023/a:1013992203485. URL: https://doi.org/10.1023/A:1013992203485.

[105] Wayne C. Levy et al. "The Seattle Heart Failure Model." In: *Circulation* 113.11 (2006), pp. 1424–1433. DOI: 10.1161/circulationaha.105.584102. eprint: https://www.ahajournals.org/doi/pdf/10.1161/CIRCULATIONAHA.105.584102. URL: https://www.ahajournals.org/doi/abs/10.1161/CIRCULATIONAHA.105.584102.

[106] Daliang Li and Junpu Wang. "FedMD: Heterogenous Federated Learning via Model Distillation." In: *CoRR* abs/1910.03581 (2019). arXiv: 1910.03581. URL: http://arxiv.org/abs/1910.03581.

[107] Tian Li et al. "Federated Optimization in Heterogeneous Networks." In: *Proceedings 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. URL: https://proceedings.mlsys.org/book/316.pdf.

[108] Xiang Li et al. "On the Convergence of FedAvg on Non-IID Data." In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=HJxNAnVtDS.

[109] Yong Li et al. "Privacy-Preserving Federated Learning Framework Based on Chained Secure Multiparty Computing." In: *IEEE Internet Things J.* 8.8 (2021), pp. 6178–6186. DOI: 10.1109/jiot.2020.3022911. URL: https://doi.org/10.1109/JIOT.2020.3022911.

[110] Henry H. Liu. *Software Performance and Scalability - A Quantitative Approach*. Wiley series on quantitative software engineering. Wiley, 2009. ISBN: 978-0-470-46253-9. URL: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470462531.html.

[111] Edgar Lopez-Rojas, Ahmad Elmir, and Stefan Axelsson. "Paysim : A financial mobile money simulator for fraud detection." In: *28th European Modeling and Simulation Symposium, EMSS 2016 :* Dime University of Genoa, 2016, pp. 249–255. ISBN: 978-88-97999-68-3.

[112] Boliang Lv et al. "Research on Modeling of E-banking Fraud Account Identification Based on Federated Learning." In: *IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress, DASC/PiCom/CBDCom/CyberSciTech 2021, Canada, October 25-28, 2021*. Ieee, 2021, pp. 611–618. DOI: 10.1109/dasc-picom-cbdcom-cybersci tech52372.2021.00105. URL: https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech52372.2021.00105.

[113] Zixuan Ma et al. "BaGuaLu: targeting brain scale pretrained models with over 37 million cores." In: *PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022*. Ed. by Jaejin Lee, Kunal Agrawal, and Michael F. Spear. ACM, 2022, pp. 192–204. DOI: 10.1145/3503221.3508417. URL: https://doi.org/10.1145/3503221.3508417.

[114] Gary Marcus, Ernest Davis, and Scott Aaronson. "A very preliminary analysis of DALL-E 2." In: *CoRR* abs/2204.13807 (2022). DOI: 10.48550/arxiv.2204.13807. arXiv: 2204.13807. URL: https://doi.org/10.48550/arXiv.2204.13807.

[115] Sajee Mathew and J Varia. "Overview of amazon web services." In: *Amazon Whitepapers* 105 (2014), pp. 1–22.

[116] 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, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*. Ed. by Aarti Singh and Xiaojin (Jerry) Zhu. Vol. 54. Proceedings of Machine Learning Research. Pmlr, 2017, pp. 1273–1282. URL: http://proceedings.mlr.press/v54/mcmahan17a.html.

[117] Hans Werner Meuer. "The TOP500 Project: Looking Back Over 15 Years of Supercomputing Experience." In: *Inform. Spektrum* 31.3 (2008), pp. 203–222. DOI: 10.1007/s00287-008-0240-6. URL: https://doi.org/10.1007/s00287-008-0240-6.

[118] Marvin L. Minsky and Seymour A. Papert. *Perceptrons: expanded edition*. MIT press, 1988. DOI: 10.5555/50066. URL: https://dl.acm.org/doi/abs/10.5555/50066.

[119] Gianluca Mittone et al. "A Federated Learning Benchmark for Drug-Target Interaction." In: *Companion Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*. Ed. by Ying Ding et al. Austin, TX, USA: Acm, Apr. 2023, pp. 1177–1181. ISBN: 978-1-4503-9419-2. DOI: 10.1145/3543873.3587687. URL: https://doi.org/10.1145/3543873.3587687.

[120] Gianluca Mittone et al. "Distributed Edge Inference: an Experimental Study on Multiview Detection." In: *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*. Ed. by Massimo Villari and Omer Rana. Ucc '23. Taormina, Italy: Acm, Dec. 2024, pp. 285–290. ISBN: 979-8-4007-0234-1. DOI: 10.1145/3603166.3632561. URL: https://doi.org/10.1145/3603166.3632561.

[121] Gianluca Mittone et al. "Experimenting with Emerging RISC-V Systems for Decentralised Machine Learning." In: *Proceedings of the 20th ACM International Conference on Computing Frontiers, CF 2023, Bologna, Italy, May 9-11, 2023*. Ed. by Andrea Bartolini et al. Bologna, Italy: Acm, May 2023, pp. 73–83. ISBN: 979-8-4007-0140-5/23/05. DOI: 10.1145/3587135.3592211. URL: https://doi.org/10.1145/3587135.3592211.

[122] Gianluca Mittone et al. "Model-Agnostic Federated Learning." In: *Euro-Par 2023: Parallel Processing - 29th International Conference on Parallel and Distributed Computing, Limassol, Cyprus, August 28 - September 1, 2023, Proceedings*. Ed. by José Cano et al. Vol. 14100. Lecture Notes in Computer Science. Limassol, Cyprus: Springer, Sept. 2023, pp. 383–396. DOI: 10.1007/978-3-031-39698-4_26. URL: https://doi.org/10.1007/978-3-031-39698-4%5C_26.

[123] Fan Mo et al. "PPFL: privacy-preserving federated learning with trusted execution environments." In: *MobiSys '21: The 19th Annual International Conference on Mobile Systems, Applications, and Services, Virtual Event, Wisconsin, USA, 24 June - 2 July, 2021*. Ed. by Suman Banerjee, Luca Mottola, and Xia Zhou. Acm, 2021, pp. 94–108. DOI: 10.1145/3458864.3466628. URL: https://doi.org/10.1145/3458864.3466628.

[124] Arun Narayanan et al. "Key Advances in Pervasive Edge Computing for Industrial Internet of Things in 5G and Beyond." In: *IEEE Access* 8 (2020), pp. 206734–206754. DOI: 10.1109/ACCESS.2020.3037717. URL: https://doi.org/10.1109/ACCESS.2020.3037717.

[125] John Nguyen et al. "Federated Learning with Buffered Asynchronous Aggregation." In: *International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event*. Ed. by Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera. Vol. 151. Proceedings of Machine Learning Research. Pmlr, 2022, pp. 3581–3607. URL: https://proceedings.mlr.press/v151/nguyen22b.html.

[126] Thin Nguyen et al. "GraphDTA: predicting drug-target binding affinity with graph neural networks." In: *Bioinform.* 37.8 (2021), pp. 1140–1147. DOI: 10.1093/bioinformatics/btaa921. URL: https://doi.org/10.1093/bioinformatics/btaa921.

[127] Tuan Nguyen et al. "Graph Convolutional Networks for Drug Response Prediction." In: *IEEE ACM Trans. Comput. Biol. Bioinform.* 19.1 (2022), pp. 146–154. DOI: 10.1109/tcbb.2021.3060430. URL: https://doi.org/10.1109/TCBB.2021.3060430.

[128] Michael Nofer et al. "Blockchain." In: *Bus. Inf. Syst. Eng.* 59.3 (2017), pp. 183–187. DOI: 10.1007/s12599-017-0467-3. URL: https://doi.org/10.1007/s12599-017-0467-3.

[129] Emily C. O'Brien et al. "The ORBIT bleeding score: a simple bedside score to assess bleeding risk in atrial fibrillation." In: *European Heart Journal* 36.46 (Sept. 2015), pp. 3258–3264. ISSN: 0195-668X. DOI: 10.1093/eurheartj/ehv476. eprint: https://academic.oup.com/eurheartj/article-pdf/36/46/3258/17355968/ehv476.pdf. URL: https://doi.org/10.1093/eurheartj/ehv476.

[130] *OMNI 1-min Data set.* 2020. DOI: 10.48322/45bb-8792. URL: https://doi.org/10.48322/45bb-8792 (visited on 04/11/2024).

[131] *OpenSignal Italy - Mobile Network Experience Report - November 22.* https://www.opensignal.com/reports/2022/11/italy/mobile-network-experience. Accessed: 2023-09-21.

[132] Wanli Ouyang, Xingyu Zeng, and Xiaogang Wang. "Partial Occlusion Handling in Pedestrian Detection With a Deep Model." In: *IEEE Trans. Circuits Syst. Video Technol.* 26.11 (2016), pp. 2123–2137. DOI: 10.1109/TCSVT.2015.2501940. URL: https://doi.org/10.1109/TCSVT.2015.2501940.

[133] Reese Pathak and Martin J. Wainwright. "FedSplit: an algorithmic framework for fast federated optimization." In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.* Ed. by Hugo Larochelle et al. 2020.

[134] Odysseas I. Pentakalos. "An Introduction to the InfiniBand Architecture." In: *28th International Computer Measurement Group Conference, December 8-13, 2002, Reno, Nevada, USA, Proceedings.* Computer Measurement Group, 2002, pp. 425–432. URL: http://www.cmg.org/?s2member%5C_file%5C_download=/proceedings/2002/2176.pdf.

[135] Mirko Polato, Roberto Esposito, and Marco Aldinucci. "Boosting the Federation: Cross-Silo Federated Learning without Gradient Descent." In: *International Joint Conference on Neural Networks, IJCNN 2022, Padua, Italy, July 18-23, 2022.* Ieee, 2022, pp. 1–10. DOI: 10.1109/ijcnn55064.2022.9892284. URL: https://doi.org/10.1109/IJCNN55064.2022.9892284.

[136] Youyang Qu et al. "Blockchain-enabled Federated Learning: A Survey." In: *ACM Comput. Surv.* 55.4 (2023), 70:1–70:35. DOI: 10.1145/3524104. URL: https://doi.org/10.1145/3524104.

[137] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." In: *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. URL: http://jmlr.org/papers/v21/20-074.html.

[138] Amir M. Rahmani et al. "Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach." In: *Future Gener. Comput. Syst.* 78 (2018), pp. 641–658. DOI: 10.1016/J.FUTURE.2017.02.014. URL: https://doi.org/10.1016/j.future.2017.02.014.

[139] Alain Rakotomamonjy et al. "Personalised Federated Learning On Heterogeneous Feature Spaces." In: *CoRR* abs/2301.11447 (2023). DOI: 10.48550/arxiv.2301.11447. arXiv: 2301.11447. URL: https://doi.org/10.48550/arXiv.2301.11447.

[140] Sashank J. Reddi et al. "Adaptive Federated Optimization." In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=LkFG3lB13U5.

[141] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519. URL: https://doi.org/10.1037/h0042519.

[142] Holger R. Roth et al. "NVIDIA FLARE: Federated Learning from Simulation to Real-World." In: *IEEE Data Eng. Bull.* 46.1 (2023), pp. 170–184. URL: http://sites.computer.org/debull/A23mar/p170.pdf.

[143] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. "Trusted Execution Environment: What It is, and What It is Not." In: *2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*. Ieee, 2015, pp. 57–64. DOI: 10.1109/trustcom.2015.357. URL: https://doi.org/10.1109/Trustcom.2015.357.

[144] Gabriele Sarti and Malvina Nissim. "IT5: Large-scale Text-to-text Pretraining for Italian Language Understanding and Generation." In: *CoRR* abs/2203.03759 (2022). DOI: 10.48550/arxiv.2203.03759. arXiv: 2203.03759. URL: https://doi.org/10.48550/arXiv.2203.03759.

[145] Felix Sattler et al. "Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data." In: *IEEE Trans. Neural Networks Learn. Syst.* 31.9 (2020), pp. 3400–3413. DOI: 10.1109/tnnls.2019.2944481. URL: https://doi.org/10.1109/TNNLS.2019.2944481.

[146] Robert E. Schapire. "The Strength of Weak Learnability." In: *Mach. Learn.* 5 (1990), pp. 197–227. DOI: 10.1007/bf00116037. URL: https://doi.org/10.1007/BF00116037.

[147] Marco Schreyer, Timur Sattarov, and Damian Borth. "Federated and Privacy-Preserving Learning of Accounting Data in Financial Statement Audits." In: *3rd ACM International Conference on AI in Finance, ICAIF 2022, New York, NY, USA, November 2-4, 2022*. Ed. by Daniele Magazzeni et al. Acm, 2022, pp. 105–113. DOI: 10.1145/3533271.3561674. URL: https://doi.org/10.1145/3533271.3561674.

[148] Naeem Seliya, Taghi M. Khoshgoftaar, and Jason Van Hulse. "A Study on the Relationships of Classifier Performance Metrics." In: *ICTAI 2009, 21st IEEE International Conference on Tools with Artificial Intelligence, Newark, New Jersey, USA, 2-4 November 2009*. IEEE Computer Society, 2009, pp. 59–66. DOI: 10.1109/ictai.2009.25. URL: https://doi.org/10.1109/ICTAI.2009.25.

[149] Jonah Sengupta et al. "High-Speed, Real-Time, Spike-Based Object Tracking and Path Prediction on Google Edge TPU." In: *2nd IEEE International Conference on Artificial Intelligence Circuits and Systems, AICAS 2020, Genova, Italy, August 31 - September 2, 2020*. IEEE, 2020, pp. 134–135. DOI: 10.1109/AICAS48895.2020.9073867. URL: https://doi.org/10.1109/AICAS48895.2020.9073867.

[150] Jaime Sevilla et al. "Compute Trends Across Three Eras of Machine Learning." In: *International Joint Conference on Neural Networks, IJCNN 2022, Padua, Italy, July 18-23, 2022*. IEEE, 2022, pp. 1–8. DOI: 10.1109/IJCNN55064.2022.9891914. URL: https://doi.org/10.1109/IJCNN55064.2022.9891914.

[151] John Shalf, Sudip S. Dosanjh, and John Morrison. "Exascale Computing Technology Challenges." In: *High Performance Computing for Computational Science - VECPAR 2010 - 9th International conference, Berkeley, CA, USA, June 22-25, 2010, Revised Selected Papers*. Ed. by José M. Laginha M. Palma et al. Vol. 6449. Lecture Notes in Computer Science. Springer, 2010, pp. 1–25. DOI: 10.1007/978-3-642-19328-6_1. URL: https://doi.org/10.1007/978-3-642-19328-6%5C_1.

[152] Nir Shlezinger et al. "UVeQFed: Universal Vector Quantization for Federated Learning." In: *IEEE Trans. Signal Process.* 69 (2021), pp. 500–514. DOI: 10.1109/tsp.2020.3046971. URL: https://doi.org/10.1109/TSP.2020.3046971.

[153] Hanan Shukur et al. "Cloud Computing Virtualization of Resources Allocation for Distributed Systems." In: *Journal of Applied Science and Technology Trends* 1.2 (June 2020), pp. 98–105. DOI: 10.38094/jastt1331. URL: https://www.jastt.org/index.php/jasttpath/article/view/31.

[154] Shashi Kant Singh, Shubham Kumar, and Pawan Singh Mehra. "Chat GPT & Google Bard AI: A Review." In: *2023 International Conference on IoT, Communication and Automation Technology (ICICAT)*. 2023, pp. 1–6. DOI: 10.1109/icicat57735.2023.10263706. URL: https://doi.org/10.1109/ICICAT57735.2023.10263706.

[155] Kamal Srivastava and Atul Kumar. "A new approach of cloud: Computing infrastructure on demand." In: *Trends in Information Management* 7.2 (2011), p. 145. ISSN: 0973-4163.

[156] Maarten van Steen and Andrew S. Tanenbaum. "A brief introduction to distributed systems." In: *Computing* 98.10 (2016), pp. 967–1009. DOI: 10.1007/s00607-016-0508-7. URL: https://doi.org/10.1007/s00607-016-0508-7.

[157] Hang Su et al. "Multi-view Convolutional Neural Networks for 3D Shape Recognition." In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 945–953. DOI: 10.1109/ICCV.2015.114. URL: https://doi.org/10.1109/ICCV.2015.114.

[158] Ziteng Sun et al. "Can You Really Backdoor Federated Learning?" In: *CoRR* abs/1911.07963 (2019). arXiv: 1911.07963. URL: http://arxiv.org/abs/1911.07963.

[159] Toyotaro Suzumura et al. "Towards Federated Graph Learning for Collaborative Financial Crimes Detection." In: *CoRR* abs/1909.12946 (2019). arXiv: 1909.12946. URL: http://arxiv.org/abs/1909.12946.

[160] Tianxiang Tan and Guohong Cao. "FastVA: Deep Learning Video Analytics Through Edge Processing and NPU in Mobile." In: *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*. IEEE, 2020, pp. 1947–1956. DOI: 10.1109/INFOCOM41043.2020.9155476. URL: https://doi.org/10.1109/INFOCOM41043.2020.9155476.

[161] Nicolò Tonci et al. "Distributed-Memory FastFlow Building Blocks." In: *Int. J. Parallel Program.* 51.1 (2023), pp. 1–21. DOI: 10.1007/s10766-022-00750-5. URL: https://doi.org/10.1007/s10766-022-00750-5.

[162] Liang Tong, Yong Li, and Wei Gao. "A hierarchical edge cloud architecture for mobile computing." In: *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. Ieee, 2016, pp. 1–9. DOI: 10.1109/infocom.2016.7524340. URL: https://doi.org/10.1109/INFOCOM.2016.7524340.

[163] Massimo Torquati. "Harnessing Parallelism in Multi/Many-Cores with Streams and Parallel Patterns." PhD thesis. University of Pisa, Italy, 2019. URL: https://etd.adm.unipi.it/theses/available/etd-02132019-111752/.

[164] Hugo Touvron et al. "Llama 2: Open Foundation and Fine-Tuned Chat Models." In: *CoRR* abs/2307.09288 (2023). DOI: 10.48550/arxiv.2307.09288. arXiv: 2307.09288. URL: https://doi.org/10.48550/arXiv.2307.09288.

[165] Hugo Touvron et al. "LLaMA: Open and Efficient Foundation Language Models." In: *CoRR* abs/2302.13971 (2023). DOI: 10.48550/arxiv.2302.13971. arXiv: 2302.13971. URL: https://doi.org/10.48550/arXiv.2302.13971.

[166] Stephen M Trimberger, ed. *Field-Programmable Gate Array Technology*. 1. Springer, 2012. ISBN: 978-0-7923-9419-8. DOI: 10.1007/978-1-4615-2742-8. URL: https://doi.org/10.1007/978-1-4615-2742-8.

[167] Blesson Varghese et al. "Challenges and Opportunities in Edge Computing." In: *2016 IEEE International Conference on Smart Cloud, SmartCloud 2016, New York, NY, USA, November 18-20, 2016*. IEEE Computer Society, 2016, pp. 20–26. DOI: 10.1109/smartcloud.2016.18. URL: https://doi.org/10.1109/SmartCloud.2016.18.

[168] Hao Wang et al. "Optimizing Federated Learning on Non-IID Data with Reinforcement Learning." In: *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*. Ieee, 2020, pp. 1698–1707. DOI: 10.1109/infocom41043.2020.9155494. URL: https://doi.org/10.1109/INFOCOM41043.2020.9155494.

[169] Wei Wang, Ben Liang, and Baochun Li. "Multi-Resource Fair Allocation in Heterogeneous Cloud Computing Systems." In: *IEEE Trans. Parallel Distributed Syst.* 26.10 (2015), pp. 2822–2835. DOI: 10.1109/tpds.2014.2362139. URL: https://doi.org/10.1109/TPDS.2014.2362139.

[170] Kang Wei et al. "Federated Learning With Differential Privacy: Algorithms and Performance Analysis." In: *IEEE Trans. Inf. Forensics Secur.* 15 (2020), pp. 3454–3469. DOI: 10.1109/tifs.2020.2988575. URL: https://doi.org/10.1109/TIFS.2020.2988575.

[171] Leon Witt et al. "Decentral and Incentivized Federated Learning Frameworks: A Systematic Literature Review." In: *IEEE Internet Things J.* 10.4 (2023), pp. 3642–3663. DOI: 10.1109/jiot.2022.3231363. URL: https://doi.org/10.1109/JIOT.2022.3231363.

[172] David H. Wolpert. "Stacked generalization." In: *Neural Networks* 5.2 (1992), pp. 241–259. DOI: 10.1016/s0893-6080(05)80023-1. URL: https://doi.org/10.1016/S0893-6080(05)80023-1.

[173] David H. Wolpert and William G. Macready. "No free lunch theorems for optimization." In: *IEEE Trans. Evol. Comput.* 1.1 (1997), pp. 67–82. DOI: 10.1109/4235.585893. URL: https://doi.org/10.1109/4235.585893.

[174] Chuhan Wu et al. "Communication-efficient federated learning via knowledge distillation." In: *Nature communications* 13.1 (2022), p. 2032.

[175] Chuhan Wu et al. "FedKD: Communication Efficient Federated Learning via Knowledge Distillation." In: *CoRR* abs/2108.13323 (2021). arXiv: 2108.13323. URL: https://arxiv.org/abs/2108.13323.

[176] Hongda Wu and Ping Wang. "Fast-Convergent Federated Learning With Adaptive Weighting." In: *IEEE Trans. Cogn. Commun. Netw.* 7.4 (2021), pp. 1078–1088. DOI: 10.1109/tccn.2021.3084406. URL: https://doi.org/10.1109/TCCN.2021.3084406.

[177] Zhaomin Wu, Qinbin Li, and Bingsheng He. "Practical Vertical Federated Learning with Unsupervised Representation Learning." In: *CoRR* abs/2208.10278 (2022). DOI: 10.48550/arxiv.2208.10278. arXiv: 2208.10278. URL: https://doi.org/10.48550/arXiv.2208.10278.

[178] Saining Xie et al. "Aggregated Residual Transformations for Deep Neural Networks." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 5987–5995. DOI: 10.1109/cvpr.2017.634. URL: https://doi.org/10.1109/CVPR.2017.634.

[179] Yuexiang Xie et al. "FederatedScope: A Flexible Federated Learning Platform for Heterogeneity." In: *Proc. VLDB Endow.* 16.5 (2023), pp. 1059–1072. DOI: 10.14778/3579075.3579081. URL: https://www.vldb.org/pvldb/vol16/p1059-li.pdf.

[180] Mengwei Xu et al. "FwdLLM: Efficient Federated Finetuning of Large Language Models with Perturbed Inferences." In: *Proceedings of the 2024 USENIX Annual Technical Conference, USENIX ATC 2024, Santa Clara, CA, USA, July 10-12, 2024*. Ed. by Saurabh Bagchi and Yiying Zhang. USENIX Association, 2024, pp. 579–596. URL: https://www.usenix.org/conference/atc24/presentation/xu-mengwei.

[181] Xiaowei Xu et al. "Scaling for edge inference of deep neural networks." In: *Nature Electronics* 1.4 (2018), pp. 216–222. DOI: https://doi.org/10.1038/s41928-018-0059-3. URL: https://doi.org/10.1038/s41928-018-0059-3.

[182] Linting Xue et al. "mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer." In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, June 2021, pp. 483–498. DOI: 10.18653/v1/2021.naacl-main.41. URL: https://aclanthology.org/2021.naacl-main.41.

[183] Qiang Yang et al. "Federated Machine Learning: Concept and Applications." In: *ACM Trans. Intell. Syst. Technol.* 10.2 (2019), 12:1–12:19. DOI: 10.1145/3298981. URL: https://doi.org/10.1145/3298981.

[184] Xun Yi, Russell Paulet, and Elisa Bertino. *Homomorphic Encryption and Applications*. Springer Briefs in Computer Science. Springer, 2014. ISBN: 978-3-319-12228-1. DOI: 10.1007/978-3-319-12229-8. URL: https://doi.org/10.1007/978-3-319-12229-8.

[185] Andy B. Yoo, Morris A. Jette, and Mark Grondona. "SLURM: Simple Linux Utility for Resource Management." In: *Job Scheduling Strategies for Parallel Processing, 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003, Revised Papers*. Ed. by Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Vol. 2862. Lecture Notes in Computer Science. Springer, 2003, pp. 44–60. DOI: 10.1007/10968987_3. URL: https://doi.org/10.1007/10968987%5C_3.

[186] Yong Yu et al. "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures." In: *Neural Comput.* 31.7 (2019), pp. 1235–1270. DOI: 10.1162/neco_a_01199. URL: https://doi.org/10.1162/neco%5C_a%5C_01199.

[187] Chengliang Zhang et al. "BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning." In: *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*. Ed. by Ada Gavrilovska and Erez Zadok. USENIX Association, 2020, pp. 493–506. URL: https://www.usenix.org/conference/atc20/presentation/zhang-chengliang.

[188] Shifeng Zhang et al. "Occlusion-Aware R-CNN: Detecting Pedestrians in a Crowd." In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part III*. Ed. by Vittorio Ferrari et al. Vol. 11207. Lecture Notes in Computer Science. Springer, 2018, pp. 657–674. DOI: 10.1007/978-3-030-01219-9_39. URL: https://doi.org/10.1007/978-3-030-01219-9%5C_39.

[189] Xinwei Zhang et al. "Hybrid Federated Learning: Algorithms and Implementation." In: *CoRR* abs/2012.12420 (2020). arXiv: 2012.12420. URL: https://arxiv.org/abs/2012.12420.

[190] Wenbo Zheng et al. "Federated Meta-Learning for Fraudulent Credit Card Detection." In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. Ed. by Christian Bessiere. ijcai.org, 2020, pp. 4654–4660. DOI: 10.24963/ijcai.2020/642. URL: https://doi.org/10.24963/ijcai.2020/642.

[191] Jike Zhong, Hong-You Chen, and Wei-Lun Chao. "Making Batch Normalization Great in Federated Deep Learning." In: *CoRR* abs/2303.06530 (2023). DOI: 10.48550/arxiv.2303.06530. arXiv: 2303.06530. URL: https://doi.org/10.48550/arXiv.2303.06530.

[192] Menglong Zhu et al. "Single image 3D object detection and pose estimation for grasping." In: *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*. IEEE, 2014, pp. 3936–3943. DOI: 10.1109/ICRA.2014.6907430. URL: https://doi.org/10.1109/ICRA.2014.6907430.

[193] Weiming Zhuang, Chen Chen, and Lingjuan Lyu. "When Foundation Model Meets Federated Learning: Motivations, Challenges, and Future Directions." In: *CoRR* abs/2306.15546 (2023). DOI: 10.48550/ARXIV.2306.15546. arXiv: 2306.15546. URL: https://doi.org/10.48550/arXiv.2306.15546.

[194] Alexander Ziller et al. "PySyft: A Library for Easy Federated Learning." In: *Federated Learning Systems: Towards Next-Generation AI*. Ed. by Muhammad Habib ur Rehman and Mohamed Medhat Gaber. Cham: Springer International Publishing, 2021, pp. 111–139. ISBN: 978-3-030-70604-3. DOI: 10.1007/978-3-030-70604-3_5. URL: https://doi.org/10.1007/978-3-030-70604-3%5C_5.