

UNIVERSIDAD DE MÁLAGA

DOCTORAL THESIS

---

# Ubiquitous Distributed Intelligence at the Edge

Inteligencia Distribuida Ubicua  
en el Borde

---

*Author:*  
M.Sc. José Ángel MORELL

*Supervisor:*  
Prof. Dr. Enrique ALBA

*A thesis submitted in fulfillment of the requirements  
for the degree of Doctor of Philosophy*


*in the*

PhD degree in Computer Technologies  
Programa de Doctorado en Tecnologías Informáticas  
Departamento de Lenguajes y Ciencias de la Computación  
E.T.S.I. Informática



UNIVERSIDAD  
DE MÁLAGA

AUTOR: José Ángel Morell Martínez

 <https://orcid.org/0000-0002-6654-1171>

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga



Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional:

<http://creativecommons.org/licenses/by-nc-nd/4.0/legalcode>

Cualquier parte de esta obra se puede reproducir sin autorización pero con el reconocimiento y atribución de los autores.

No se puede hacer uso comercial de la obra y no se puede alterar, transformar o hacer obras derivadas.

Esta Tesis Doctoral está depositada en el Repositorio Institucional de la Universidad de Málaga (RIUMA): [riuma.uma.es](http://riuma.uma.es)



# Declaración de Autoría y Originalidad



UNIVERSIDAD DE MÁLAGA

**M.Sc. José Ángel Morell Martínez**, estudiante del programa de doctorado Tecnologías Informáticas de la Universidad de Málaga, autor de la tesis, presentada para la obtención del título de doctor por la Universidad de Málaga, titulada:

## **“Ubiquitous Distributed Intelligence at the Edge”**

Realizada bajo la tutorización y dirección del **Prof. Dr. Enrique ALBA**

DECLARO QUE:

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo.

Igualmente asumo, ante a la Universidad de Málaga y ante cualquier otra instancia, la responsabilidad que pudiera derivarse en caso de plagio de contenidos en la tesis presentada, conforme al ordenamiento jurídico vigente.

Fecha y firma:

---

This page is intentionally left blank.

# Declaración de Supervisión



UNIVERSIDAD DE MÁLAGA

El **Prof. Dr. Enrique Alba Torres**, perteneciente al Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga,

## **Certifica**

que, M.Sc. José Ángel Morell Martínez, Magíster en Ingeniería del Software e Inteligencia Artificial por la Universidad de Málaga, ha realizado en el Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga, bajo su dirección, el trabajo de investigación correspondiente a su Tesis Doctoral (por compendio de artículos) titulada:

**“Ubiquitous Distributed Intelligence at the Edge”**

**“Inteligencia Distribuida Ubicua en el Borde”**

Revisado el presente trabajo, estimo que puede ser presentado al tribunal que ha de juzgarlo. Y para que conste a efectos de lo establecido en la legislación vigente, autorizo la presentación de esta Tesis Doctoral en la Universidad de Málaga.

Fecha y firma:

---

This page is intentionally left blank.

*“It is not because things are difficult that we do not dare, it is because we do not dare that things are difficult.”*

Lucius Annaeus Seneca, 4 BC – 65 AD

This page is intentionally left blank.



UNIVERSIDAD DE MÁLAGA

# *Preface*

E.T.S.I. Informática  
Departamento de Lenguajes y Ciencias de la Computación

Doctor of Philosophy

## **Ubiquitous Distributed Intelligence at the Edge**

by M.Sc. José Ángel MORELL

So far, most distributed computing models have considered processing information in a powerful central cloud. At the same time, the edge devices (nearer the users) such as smartphones, tablets, and laptops have been used to gather or deliver information. However, the edge is changing, becoming more extensive and more sophisticated, causing the amount of data produced on edge devices to grow exponentially, outstripping the network's capabilities and making it impossible to send it all to the cloud for processing and analysis. Moreover, much of this data that remains on edge is private and would not be allowed to be transferred to external central processing centres, questioning the way to process the information, e.g. traditional big data processing. Still, learning from these data would be very useful for science and society. Other devices need to make decisions in real-time and cannot wait for the cloud to give them an answer before taking action, or even they can be temporary uncommunicated. All the mentioned scenarios advocate fault-tolerant distributed computing at the edge, running the AI algorithms on light devices, and learning global models using the devices' local data, i.e. federated edge learning. In this thesis, we propose to see federated edge learning as a type of volunteer computing where users donate their edge devices' computing resources to a project that trains a shared machine learning model. We first define and then address the challenges and desirable features to achieve volunteer computing for federated edge learning. We seek to adapt to the volatility of dis/connections and get fault tolerance. We propose two approaches, one synchronous and one asynchronous, to deal with this problem using a given number of constrained, unreliable, and heterogeneous (hardware and software) devices and then show their suitability for this purpose in dynamic environments. We get a numerical accuracy similar to today's configurations that use a static platform for learning. Also, we formulate and model the problem of communication overhead in federated learning, an important challenge for ubiquitously distributed intelligence and efficiently learning from users' data, as a multi-objective problem. We then propose tackling it using genetic algorithms for multi-objective function optimisation achieving higher model accuracy while reducing communications compared to the maximum communication setting. We got promising results opening the door to new lines of research in multi-objective optimisation of federated learning challenges. Finally, the results of this thesis prove that the deployment of ubiquitously distributed intelligence at edge devices is feasible and valuable, also providing a better understanding of how such ubiquitous distributed intelligence should be, its associated problems, and how these problems must be addressed.

This page is intentionally left blank.

## Acknowledgements

This thesis is the result of my PhD studies over the last few years. However, this work would not have been possible without the necessary support of my family and friends. First, I want to thank my supervisor Prof. Dr Enrique Alba, for proposing and allowing me to conduct a PhD under his supervision. I am grateful for his invaluable advice, corrections, and patience during this hard road that allowed me to go through it with better and better skills. Following this, I would like to mention Francisco Chicano and Gabriel Luque, two great researchers and even better people to whom I am very grateful.

I am also grateful for the time shared with my colleagues and former colleagues in the NEO group and other lab mates during these years. There are many but I would like to mention some of them: Andrés Camero, Zakaria Dahi, Rubén Saborido, Jamal Toutouh, Manuel López-Ibáñez, Christian Cintrano, Javier Ferrer, Amr Abdelhafez, Miguel Ángel Domínguez, Daniel Pandolfi, Andrea Villagra, Martin Salvachua, Rodrigo Gil-Merino, Housseem Ben-Smida, Ignacio Villalobos, Aitor Canca and Álvaro Gutiérrez. I would also like to mention some of my classmates during my Master's studies, such as Cristóbal González and Daniel Paul Pena, the latter of whom I am sure will be a great researcher when he finishes his PhD studies.

I would especially like to thank my loved girlfriend, Candy, who has always supported and walked me through all the good and bad times for more than a third of my life. Also, my parents, Pedro and Mari Carmen have always helped me unconditionally. To my siblings María del Mar and my late brother Pedro of whom I will always be very proud. To my in-laws, Eduardo and Juana, great people and always ready to help. To my grandparents, Angel, Pepa and Juan, may they rest in peace, particularly to my grandmother Julia, who is an example of strength and joy at 92. To my uncles Juan Francisco (a science enthusiast), Leo and my young and wise cousin Adrián. For the Christmas past, my cousins Marta, Amanda and Adrián and my uncles Pepe and Margari. I also would like to mention Alberto Pinzón, Carlos Pavón and Jorge Gutiérrez. And, to all those who accompanied me for part of the way and whom I have not named.

Finally, I would like to thank the Ministerio de Educación, Cultura y Deporte, Gobierno de España for funding me with the grant FPU16/02595. Also, to the Universidad de Málaga, Andalucía Tech, Consejería de Economía y Conocimiento de la Junta de Andalucía and FEDER under grant number UMA18-FEDERJA-003 (PRECOG); PID 2020-116727RB-I00 (HUmove) funded by MCIN/AEI/ 10.13039/501100011033; and TAILOR ICT-48 Network (No 952215) funded by EU Horizon 2020 research and innovation programme.



Horizon 2020  
European Union Funding  
for Research & Innovation

This page is intentionally left blank.

# Contents

<b>List of Figures</b>	<b>xv</b>
<b>List of Tables</b>	<b>xvii</b>
<b>List of Abbreviations</b>	<b>xix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Introduction and Motivation . . . . .	1
1.1.1 Origin and Meaning of Ubiquitous Computing . . . . .	1
1.1.2 Bringing Computation and AI Closer to the Edge . . . . .	2
1.2 Thesis Methodology . . . . .	4
1.2.1 Objectives and Phases . . . . .	4
1.2.2 Thesis Contributions . . . . .	5
<b>2 Fundamentals</b>	<b>11</b>
2.1 Decision Problems and Complexity Theory . . . . .	11
2.2 Optimisation Problems and Classes of Algorithms . . . . .	13
2.3 Metaheuristics: Problem-Independent Strategies for Complex Problems	15
2.3.1 Genetic Algorithms: A Population-Based Metaheuristic . . . . .	15
2.3.2 NSGA-II: A GA for Multi-Objective Function Optimisation . . . . .	17
2.4 The Process of Training an Artificial Neural Network . . . . .	18
2.5 Parallel and Distributed Deep Learning . . . . .	21
2.5.1 Synchronous vs. Asynchronous DDL . . . . .	23
<b>3 State-of-the-Art</b>	<b>25</b>
3.1 Federated Learning: Learning Across Decentralised Edge Devices While Keeping Data Locally . . . . .	25
3.2 Reviewing Volunteer Computing . . . . .	27
3.2.1 Volunteer Computing . . . . .	27
3.2.2 VC on the Web Browser: BBVC 1st and 2nd Generation . . . . .	28
3.2.3 VC on the Web Browser: BBVC 3rd Generation . . . . .	28
3.3 Challenges and Desirable Features of VC4FL . . . . .	31
<b>4 Summary of Results</b>	<b>37</b>
4.1 Analysing Suitability of Edge Devices . . . . .	38
4.2 Proposal for NN Training Using BBVC and the MapReduce Paradigm	41
4.3 Proposal for Asynchronous FL using Unreliable Volunteer Edge Devices	44
4.4 Proposal for Communication Overhead Reduction in FL . . . . .	51
<b>5 Conclusions and Future Work</b>	<b>55</b>

<b>Appendix A Data Sets, Problems, and Solvers Used in This Thesis</b>	<b>59</b>
A.1 MNIST Dataset . . . . .	59
A.2 OneMax Problem . . . . .	59
A.3 Minimum Tardy Task Problem (MTTP) . . . . .	59
A.4 Massively Multimodal Deceptive Problem (MMDP) . . . . .	59
A.5 Error Correcting Code Design (ECC) . . . . .	60
A.6 Capacitated Vehicle Routing Problem (CVRP) . . . . .	60
A.7 CVRP Solver . . . . .	61
<b>Appendix B Extending the Work of Analysing Edge Devices</b>	<b>63</b>
B.1 Choosing Programming Languages for Edge Computing . . . . .	63
B.2 Evaluation of Edge Devices and Programming Languages Running Genetic Algorithms . . . . .	63
<b>Appendix C Summary in Spanish</b>	<b>69</b>
C.1 Introducción . . . . .	69
C.2 Propuesta de trabajo . . . . .	71
C.3 Contribución . . . . .	72
C.3.1 Analizando la idoneidad de los dispositivos de borde . . . . .	77
C.3.2 Propuesta de entrenamiento de NN en el navegador web uti- lizando VC y el paradigma MapReduce . . . . .	78
C.3.3 Propuesta de FL asíncrono utilizando dispositivos de borde voluntarios no fiables . . . . .	79
C.3.4 Propuesta para reducir la sobrecarga de comunicación en FL . .	80
C.4 Conclusiones y trabajo futuro . . . . .	81
<b>Appendix D Publications Supporting This Thesis</b>	<b>85</b>
D.1 Publications in Journals . . . . .	85
D.2 Publications in the Proceedings of International and National Confer- ences . . . . .	85
D.3 Publications Indirectly Related to This Thesis . . . . .	86
<b>References</b>	<b>87</b>

# List of Figures

1.1	From left to right, we see the three eras of computing: the mainframe, the PC and mobile devices. . . . .	2
1.2	New types of edge devices connected to the cloud. . . . .	2
1.3	Ubiquitous distributed intelligence: Data from users and cities are used to train distributed ML models while associated problems are optimised with optimisation algorithms. . . . .	3
2.1	NP complexity. . . . .	13
2.2	Classification of optimisation methods. . . . .	14
2.3	Euler diagram of classifications of metaheuristics. . . . .	16
2.4	Graphical representation of our steady state GA. . . . .	16
2.5	AI overview. . . . .	18
2.6	A neural network with two hidden layers. . . . .	19
2.7	A perceptron sums inputs' multiplication by weights and applies an activation function to get the output. . . . .	19
2.8	Decision boundary for linear and non-linear activation functions. . . . .	19
2.9	Gradient descent. . . . .	20
2.10	Model and data parallelism. . . . .	22
2.11	Parallel mini-batch gradient descent. . . . .	22
3.1	Federated learning architecture. . . . .	25
3.2	The evolution of volunteer computing and browser-based volunteer computing. . . . .	30
3.3	Communication reduction approaches in FL. . . . .	32
4.1	Resources usage. . . . .	40
4.2	Our proposal for NN Training using the MapReduce paradigm. . . . .	43
4.3	High-level system architecture. . . . .	46
4.4	Initialisation execution flow. The worker obtains all the necessary information to start collaborating. . . . .	46
4.5	Execution flow of the collaborative learning process. . . . .	47
4.6	The FL-COP modelling levels. . . . .	52
4.7	A 3-layers model: (a) abstract and (b) concrete FL-COP solutions. . . . .	53
A.1	The Vehicle Routing Problem (VRP). . . . .	60
A.2	A representation of a solution for 8 cities and 4 vehicles. . . . .	61
A.3	Edge Recombination Crossover (ERX). . . . .	61
A.4	Mutation operators are applied with equal probability. . . . .	62
B.1	Device/language normalised scores per each problem (the less score, the better). . . . .	67
B.2	Device/problem normalised scores per each programming language (the less score, the better). . . . .	67

C.1	Nuevos tipos de dispositivos de borde conectados a la nube. . . . .	69
C.2	Inteligencia distribuida ubicua: Datos de usuarios y ciudades se usan para entrenar modelos distribuidos de ML mientras los problemas asociados se optimizan con algoritmos de optimización. . . . .	71
C.3	Propuesta para el entrenamiento de NN utilizando el paradigma MapReduce. . . . .	78
C.4	Arquitectura del sistema de alto nivel. . . . .	79
C.5	Las técnicas de reducción de la comunicación en FL-COP por niveles. .	80



# List of Tables

3.1	Desirable features of VC4FL. . . . .	35
4.1	Hardware in the edge: features. . . . .	38
4.2	Processor specifications. . . . .	38
4.3	Benchmarks. . . . .	38
4.4	Problem instances. . . . .	39
4.5	Problem results. . . . .	40
4.6	Distributed and sequential training. . . . .	43
4.7	Comparison of experiments in case studies 1 to 3. . . . .	50
4.8	Results of case study 4. Each worker has a different number of samples of each class and a maximum of five classes. The larger the number of workers, the better the accuracy. . . . .	50
B.1	OneMax runtime and normalised score. . . . .	64
B.2	MTTP runtime and normalised score. . . . .	64
B.3	MMDP runtime and normalised score. . . . .	65
B.4	ECC runtime and normalised score. . . . .	65
B.5	CVRP runtime and normalised score. . . . .	66
B.6	C++ normalised score. . . . .	66
B.7	Java normalised score. . . . .	66
B.8	WASM normalised score. . . . .	66

This page is intentionally left blank.

# List of Abbreviations

<b>AI</b>	Artificial Intelligence
<b>ANN</b>	Artificial Neural Network
<b>BBVC</b>	Browser-Based Volunteer Computing
<b>CK</b>	Cohen's Kappa
<b>CNN</b>	Convolutional Neural Network
<b>DDL</b>	Distributed Deep Learning
<b>DL</b>	Deep Learning
<b>EA</b>	Evolutionary Algorithm
<b>EC</b>	Edge Computing
<b>FEEL</b>	Federated Edge Learning
<b>FL</b>	Federated Learning
<b>FNN</b>	Feedforward Neural Network
<b>GA</b>	Genetic Algorithm
<b>HPC</b>	High-Performance Computing
<b>HW</b>	Hardware
<b>i.i.d.</b>	Independent and Identically Distributed
<b>IoT</b>	Internet of Things
<b>JCR</b>	Journal Citation Reports
<b>LSTM</b>	Long Short Term Memory
<b>MAE</b>	Mean Absolute Error
<b>ML</b>	Machine Learning
<b>MOEA</b>	Multi-Objective Evolutionary Algorithm
<b>MSE</b>	Mean Squared Error
<b>NaN</b>	Not a Number
<b>NN</b>	Neural Network
<b>PC</b>	Personal Computer
<b>RMSE</b>	Root Mean Squared Error
<b>RNN</b>	Recurrent Neural Network
<b>RP3</b>	Raspberry Pi 3
<b>SC</b>	Smart City
<b>SD</b>	Standard Deviation
<b>SGD</b>	Stochastic Gradient Descent
<b>SJR</b>	SCImago Journal Rank
<b>ssGA</b>	Steady State Genetic Algorithm
<b>SW</b>	Software
<b>UC</b>	Ubiquitous Computing
<b>VC</b>	Volunteer Computing
<b>VC4FL</b>	Volunteer Computing 4 (for) Federated Learning
<b>WASM</b>	WebAssembly

This page is intentionally left blank.

*Thanks to the people who have supported me*

This page is intentionally left blank.

## Chapter 1

# Introduction

THIS thesis is a scientific summary of work carried out in previous years addressing the challenge of distributed ubiquitous intelligence in edge devices. In particular, we propose to see Federated Edge Learning (FEEL) as a type of Volunteer Computing (VC) where users donate their edge devices' computing resources to a project that trains a shared Machine Learning (ML) model (VC4FL). We look at what such a platform should look like, why it is necessary to move computing from the cloud to the edge and how we can learn from users' local data without moving the data to the cloud. Such a platform must be designed to learn from users' data and optimise potential problems. As we detail in Section 3.3, that means dealing with constrained, unreliable, and heterogeneous devices in hardware (HW) and software (SW). This platform must adapt to the volatility of dis/connections and be fault-tolerant. Also, we propose using metaheuristics to optimise the problems that may arise.

In this section, we first describe the introduction and motivation of our work. Next, we explain the thesis methodology, where we define the objectives and phases of this work and present the contributions of the thesis.

### 1.1 Introduction and Motivation

In this section, we first introduce the origin and meaning of ubiquitous computing. Next, we explain why moving computation closer to the edge, i.e. nearer the user devices, is necessary.

#### 1.1.1 Origin and Meaning of Ubiquitous Computing

Mark Weiser is considered the father of the term “*ubiquitous computing*” (UbiComp / UC) [117] [118] also known as “*pervasive computing*”, “*calm computing*” or “*ambient intelligence*”. This term describes a reality where computers are embedded in all objects of our environment such that they might bend to our will and support us in our personal life and work. This idea was described in 1997 as follows [118]:

*“The ubiquitous computing era will have lots of computers sharing each of us. Some of these computers will be the hundreds we may access in the course of a few minutes of Internet browsing. Others will be imbedded in walls, chairs, clothing, light switches, cars - in everything. Ubiquitous computing is fundamentally characterized by the connection of things in the world with computation. This will take place at a many scales, including the microscopic.”*

UC is considered the third era of computing (Fig. 1.1). The first era was characterised by mainframe computers in which many people used one big centralised computer. The second era was composed of personal computers, one person used each. The third era, UC, which has just started, is the era of mobile and embedded devices in which many computers exist for each person.



FIGURE 1.1: From left to right, we see the three eras of computing: the mainframe, the PC and mobile devices.

Weiser's idea is becoming a reality. At the same time, the rise of edge devices adds new challenges that prevent computing from being done entirely in the cloud, as it has been done until now. We explain this in more detail in the next section.

### 1.1.2 Bringing Computation and AI Closer to the Edge

So far, most distributed computing models have considered processing information in a powerful central cloud. To some extent, edge devices [102] (nearer the users) such as smartphones, tablets, and laptops are terminals that show what is happening in the cloud. However, the edge is changing, being more extensive and more sophisticated. There are not just mobile devices but also self-driving cars, drones, robots, smart lights, smart refrigerators, wearable devices, and multiple sensors everywhere. Many of these devices collect a large amount of data, send it to the cloud, and wait for a response (see Fig. 1.2).



FIGURE 1.2: New types of edge devices connected to the cloud.

Statista<sup>1</sup> has declared that 23.8 billion interconnected computing devices are active worldwide and will produce 149 zettabytes of usable data by 2024<sup>2</sup>. Cisco<sup>3</sup> also estimated that at least 85 of the 850 zettabytes created in 2021 would be usable, while only seven zettabytes of it will be stored. Indeed, data production is far exceeding the network's capacity. Most of this data will not be able to be stored/processed on the cloud due to the exponential increase in data demand and the high speed at

<sup>1</sup>[www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide](http://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide)

<sup>2</sup>[www.statista.com/statistics/871513/worldwide-data-created](http://www.statista.com/statistics/871513/worldwide-data-created)

<sup>3</sup>[blogs.cisco.com/sp/five-things-that-are-bigger-than-the-internet-findings-from-this-years-global-cloud-index](https://blogs.cisco.com/sp/five-things-that-are-bigger-than-the-internet-findings-from-this-years-global-cloud-index)



which data is generated. For example, the data generated by a Boeing 787 or an autonomous vehicle is 1 and 5 GB per second, respectively [40, 112]. That is not just a problem of the bandwidth capacity, but we are also talking about devices that have to make real-time decisions, which may not always be able to communicate with the cloud. It is necessary to create new applications specialised and optimised for analytics at the edge [97]. Devices have to be smarter, safer and better-connected [53]. In addition, real-time data processing would need to occur at the edge where the data is being collected. Edge Computing (EC) [102] has been proposed to address the problem of exponential growth of data generation at edge devices. It proposes to increase the amount of processing on edge devices while reducing raw data exchange with the cloud.

Moreover, today's advances in Artificial Intelligence (AI) allow the training of ML models by exploiting the daily-generated data that was previously considered useless [67]. In addition, data privacy prevents sharing it with third parties (e.g. medical images). As a promising solution to these issues, Federated Learning (FL) appeared. FL originally comes from Distributed Deep Learning (DDL) [75]. It is a learning paradigm that trains a shared model in a distributed manner while keeping private the data locally on edge devices. FL is being actively investigated and widely applied [50, 69, 93] (e.g. medicine [101]). Both FL and EC have many things in common; therefore, several authors have proposed solutions that satisfy both, calling it Federated Edge Learning (FEEL) [109, 121].

In this thesis, we propose to see FEEL as a type of VC [6, 30, 57, 64, 86] (VC4FL) where users donate the computing resources of their edge devices to a project in which train a shared DL model. To this end, we must study how we should use optimisation algorithms on edge devices. The training of a neural network (NN) is indeed an optimisation problem. Moreover, in FL, new ones arise related to the challenges of the volatility of edge device connections, communication overhead, fault tolerance, device heterogeneity, and scalability, among many others. Many of these problems are NP-hard [23, 24], so exact algorithms [23] become unusable. Metaheuristics [3, 110], allow us to find acceptable solutions (non-optimal) to many of these complex problems. Both the training of NN and the use of optimisation techniques such as metaheuristics are processes that generally require a high computational load, making them difficult to use on edge devices. However, recent advances in computing power, memory capacity and communications speed of these devices [102] allow us to think that running small and medium optimisation algorithms in a distributed manner on these devices is feasible and valuable.

In the following section, we detail the methodology of the thesis, where we define the objectives and phases of this work and present the contributions.

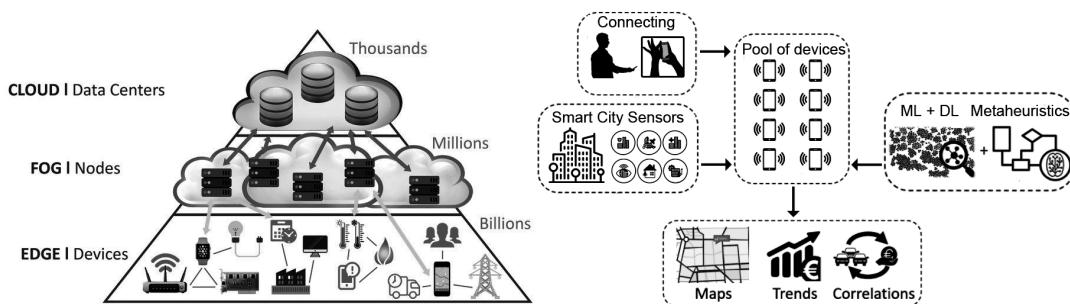


FIGURE 1.3: Ubiquitous distributed intelligence: Data from users and cities are used to train distributed ML models while associated problems are optimised with optimisation algorithms.

## 1.2 Thesis Methodology

In this section, we define the objectives and phases of this work and present the contributions of the thesis.

### 1.2.1 Objectives and Phases

This thesis addresses the challenge of getting distributed ubiquitous intelligence in edge devices (see Figure 1.3). Specifically, we propose to see FEEL as a VC type where users donate their edge devices' computing resources to a project that trains a shared DL model (VC4FL). As we detail in Section 3.3, that means dealing with constrained, unreliable, and heterogeneous (in HW and SW) devices. Such a platform must adapt to the volatility of dis/connections and be fault-tolerant. We look at what such a platform should look like, why it is necessary to move computing from the cloud to the edge and how we can learn from users' local data without moving it to the cloud. In addition, we propose using metaheuristics to solve problems that arise in this type of systems, such as optimising the right amount of communications needed without losing quality in the learned artificial intelligence model. We have four main objectives to achieve our purpose:

- G1** Analyse heterogeneous hardware and software of edge devices.
- G2** Investigate paradigms, protocols, algorithms, and challenges that may arise.
- G3** Design and implement techniques that allow us to perform volunteer distributed artificial intelligence running the computing on users' edge devices.
- G4** Disseminate the results and make publicly available the designed and implemented software.

Below we detail our main objectives following the phases of the scientific method [34]:

#### 1. Observation.

- (a) Analyse the heterogeneous HW we expect to use in the ubiquitous platform. Research the most suitable benchmarks to compare these devices. **G1.**
- (b) Analyse the heterogeneous SW we can find in these devices and the programming languages most appropriate to implement our proposal. **G1.**
- (c) Research the most suitable algorithms to achieve our purpose. **G2.**
- (d) Research and survey volunteer computing using edge devices with low computing capacity and communication and power constraints. **G2.**
- (e) Study the problems that may arise to fulfil our purpose. **G2.**

#### 2. Induction.

- (a) Identify paradigms and appropriate, efficient and safe network protocols to design and build the ubiquitously distributed computing architecture. **G2.**

### 3. In the context of VC4FL, our hypothesis is:

- (a) *“Deploying distributed intelligence in a ubiquitous and fault-tolerant manner using heterogeneous, unknown and unreliable edge devices is a need and can be competitive with traditional approaches that use a homogeneous, known and static cluster of computers.”*

### 4. Experimentation.

- (a) Design, implement, and evaluate fault tolerance techniques for unreliable devices that can fail at any time. **G3**.
- (b) Design, implement, and evaluate adaptive techniques for dynamic environments where devices are connected and disconnected at will. **G3**.
- (c) Design, implement, and evaluate the distributed computing architecture. **G3**.
- (d) Develop usable prototypes. **G3**.
- (e) Perform extreme cases of dynamic experiments using unreliable devices to test the fault-tolerant and adaptive of the distributed platform. **G3**.
- (f) Analyse numerical results of the volunteer distributed computing architecture and validate our proposal, demonstrating our hypothesis. **G3**.
- (g) Propose solutions to tackle the problems that arise to fulfil our purpose and test them. **G3**.

### 5. Conclusions.

- (a) Write a new body of knowledge about ubiquitous AI. **G4**.
- (b) Disseminate our results, benchmarks and software packages in conferences and impact journals, Internet, videos, and user interactions. Define future work and cross-fertilisation. **G4**.

## 1.2.2 Thesis Contributions

This doctoral thesis is presented as a compendium of four publications aligned with the objectives defined in Section 1.2.1 and seeking the common goal of improving state of art in distributed computing. Considering the relevance of the general objective of this thesis and its specific problems, we expect that our results will contribute to the research community and society. Section 4 is organised into four chapters, each referring to the work performed in one of the papers. Remarkably, two of these papers have been published in journals indexed in the JCR (Q1). Also, another one was published in the proceedings of a national conference, and one in the proceedings of an international conference in which the PhD candidate received the “Outstanding Student” award.

We organise the contribution of this thesis into four main contributions:

- LIT.** We have contributed to enlarging the literature about EC and FL (FEEL) by reviewing the most suitable benchmarks for comparing performance on edge devices (**G1**). By analysing the hardware and software of the most common edge devices and their performance when solving complex problems (**G1**). By reviewing the literature that uses the web browser as a VC platform (**G2**). By

defining the challenges and desirable features of VC4FL by combining the traditionally used criteria for VC and FL (**G2**) and showing that the web browser is the right platform for this purpose due to its sandboxing, ubiquity, and no software installation required (**G2**). Finally, we summarised the most common techniques used to reduce the communication overhead problem in FL (**G2** and **G4**).

- SYN.** We have proposed a synchronous approach for performing distributed volunteer computing using the MapReduce programming paradigm and the message queue pattern (**G2**) through web browsers without interrupting the site's user experience and installing additional software. We designed, implemented (**G3**), and evaluated our proposal showing that it has good scalability despite the constraints on the communication channel (**G4**).
- ASYN.** We have proposed an asynchronous algorithm for FEEL that adapts to constrained, unreliable, and heterogeneous (in HW and SW) devices (**G2**) when the number of workers is low and can even drop to zero during training. We have designed, implemented (**G3**), and empirically evaluated our proposal in highly dynamic and changing scenarios getting a numerical accuracy and Cohen's Kappa (CK) score similar to today's configurations that use a static distributed platform for learning (**G4**).
- MOD.** We formulated and modelled the problem of communication overhead in FL (**G2**), an important challenge for ubiquitously distributed intelligence and efficiently learning from users' data, as a multi-objective problem. We proposed solving it using genetic algorithms for multi-objective function optimisation (**G3**). Our proposal achieves higher accuracy while reducing communications from 10 to 2000 depending on the neural network topology compared to the maximum communication setting (**G4**).

We explain our contributions in more detail in the following. All our work pursues the goal **G4** of disseminating results. In our first work [85] (see Section 4.1), we address the objective **G1** and **G2** and the sub-objectives *1-a*, *1-b*, *1-c*, *5-a* and *5-b*. The contributions of this work are **LIT**:

1. We studied and collected the available benchmarks for analysing the performance of edge devices. **G1**.
  - (a) We showed that classic benchmarks are not reliable for evaluating the performance of edge devices, but they are helpful to get a rough idea. They are dependent on the hardware architecture and the operating system. Depending on the specific problem to be solved, the results may vary. For instance, Antutu is so common that hardware manufacturers have taken to cheating on the benchmark, which makes it unreliable. **G1**.
2. We analysed the performance of different edge devices while solving a representative set of optimisation problems with different features by running genetic algorithms. **G1**.
  - (a) We showed that Raspberry Pi 3 (RP3) is a perfectly suitable platform for executing algorithms on edge. RP3 is very cheap, uses less memory than a laptop and solves problems in a reasonable time. RP3 is commonly used in intelligent city sensors, making it suitable for performing edge calculations. **G1**.

- (b) We showed that edge devices running Android have problems running high-performance applications. This operating system seems more interested in keeping battery consumption low and using less memory (using the garbage collector more often) than performance. **G1**.
- (c) We showed that benchmarks like GeekBench 4 got good results on Android devices. Therefore, we believe that it is necessary to execute native code in Android devices to be able to solve the restrictions of this operating system. In appendix B, we show a subsequent unpublished study. We analysed the performance of other programming languages such as C++ and WASM (same devices, problems and algorithms). We showed that our hypothesis was correct, getting a performance closer to RP3 and the laptop on Android devices. **G1** and **G2**.
  - i. We implemented the algorithms using three programming languages (Java, C++ and WASM-JavaScript). We showed that we get good performance in most devices by using native code and compiling it into WebAssembly (WASM) with Emscripten. Moreover, when using WASM, the results of the best and worst devices are very similar, obtaining an excellent homogeneity which is very important when running algorithms in a parallel way. We proved that running these algorithms on the web browser is efficient and has the advantages of a lightweight virtualisation property and being multiplatform. **G1**.
  - ii. We demonstrated that devices with low processor capacity are perfectly appropriate for solving optimisation problems at the edge. Inexpensive devices such as RP3 and mobile devices can achieve laptop-like performance when solving complex optimisation problems. **G1**.

In our second work [86] (see Section 3.2 and 4.2), we address the objectives **G2** and **G3**, and the sub-objectives *1-a*, *1-d*, *1-e*, *2-a*, *4-a*, *4-b*, *4-c*, *4-d*, *4-e*, *4-f*, *4-g*, *5-a* and *5-b*. The contributions of this work are **LIT** and **SYN**:

3. We summarised the literature that uses the web browser as a volunteer computing (BBVC) platform and described recent improvements that increasingly make this possible (see Section 3.2). **G2**.
4. We proposed a BBVC framework for distributed volunteer computing using the MapReduce programming paradigm and the message queue pattern through web browsers without interrupting the site's user experience and installing additional software. We showed that our proposal has good scalability despite the constraints on the communication channel. **G3**.
5. We proved that web browser-based distributed neural network training is feasible and efficient. We conducted a proof-of-concept to show that distributed training of neural networks in the browser is possible. Using small and medium NN models is perfectly suitable for solving edge devices' problems. The results show that it is feasible, scalable, and an exciting area to explore. **G3**.

In our third work [84] (see Section 3.3 and 4.3), we address the objectives **G2**, **G3**, and **G4** and the sub-objectives *1-a*, *1-b*, *1-d*, *1-e*, *2-a*, *4-a*, *4-b*, *4-c*, *4-d*, *4-e*, *4-f*, *4-g*, *5-a* and *5-b*. The contributions of this work are **LIT** and **ASYN**:

6. We defined the challenges and desirable features of VC4FL (see Section 3.3). **G2**.

7. We proposed an algorithm for FEEL that adapts to asynchronous clients joining and leaving the computation when the number of workers is low and can even drop to zero during training. **G3**.
8. We proposed, implemented and evaluated a software platform for performing FL that meets the defined challenges and desirable features. We evaluated our proposal via extensive experimentation in a static configuration and highly dynamic and changing scenarios. We then showed that the platform using a given number of constrained, unreliable, and heterogeneous (in HW and SW) devices adapts well to this changing environment getting a numerical accuracy and CK score similar to today's configurations that use a static platform for learning. Next, we demonstrated the fault-tolerance of the platform that recovers from unexpected disconnections of volunteer devices. **G3**.
9. We released a modular open-source library covering most FEEL and VC desirable features. **G4**.

In our fourth work [87] (see Section 3.3 and 4.4), we address the objectives **G2** and **G3**, and the sub-objectives 1-e, 4-f, 5-a and 5-b. The contributions of this work are **LIT** and **MOD**:

10. We summarised the most common techniques in literature for solving the communication overhead problem in FL (see Section 3.3). **G2**.
11. We formulated and modelled Federated Learning Communication Overhead Problem as a multi-objective Problem (FL-COP). **G3**.
12. We proposed solving the FL-COP by using genetic algorithms for multi-objective function optimisation. Our proposal achieves higher accuracy while reducing communications from 10 to 2000 depending on the neural network topology compared to the maximum communication setting. **G3**.
13. The proposed algorithm provides a population of solutions that facilitates decision-making when configuring the parameters of FL. **G3**.

The publications used to support this thesis are listed below in chronological order. With the publications' details, we present each publication's available metrics, i.e. the journal impact factor (JIF).

- I) Morell, J. Á., & Alba, E. (2018). Running Genetic Algorithms in the Edge: A First Analysis. In Conference of the Spanish Association for Artificial Intelligence (pp. 251-261). Springer, Cham. [10.1007/978-3-030-00374-6\\_24](https://doi.org/10.1007/978-3-030-00374-6_24)
- II) Morell, J. Á., Camero, A., & Alba, E. (2019). JSDoop and TensorFlow.js: Volunteer Distributed Web Browser-based Neural Network Training. IEEE Access, 7, 158671-158684. DOI: [10.1109/ACCESS.2019.2950287](https://doi.org/10.1109/ACCESS.2019.2950287)
  - JCR COMPUTER SCIENCE, INFORMATION SYSTEMS - SCIE, **Q1**, ranking 35/156, 2019 JIF = 3.745
- III) Morell, J. Á., & Alba, E. (2022). Dynamic and Adaptive Fault-tolerant Asynchronous Federated Learning Using Volunteer Edge Devices. Future Generation Computer Systems. DOI: [10.1016/j.future.2022.02.024](https://doi.org/10.1016/j.future.2022.02.024)

- JCR COMPUTER SCIENCE, THEORY & METHODS - SCIE, **Q1**, ranking 7/110, 2020 JIF = 7.187

IV) Morell, J. Á., Dahi, ZA, Chicano, F, Luque, G & Alba, E. (2022). Optimising Communication Overhead in Federated Learning Using NSGA-II. International Conference on the Applications of Evolutionary Computation. Springer, Cham. [10.1007/978-3-031-02462-7\\_21](https://doi.org/10.1007/978-3-031-02462-7_21)

- Outstanding Student Award.

In summary, our contributions are in line with our objectives. We have analysed the HW and SW of a wide variety of edge devices. We have investigated the most suitable paradigms, protocols, and algorithms. We have proposed, designed, implemented, and evaluated techniques to address the more significant challenges we have encountered to achieve our primary goal. We have disseminated the results in high-impact journals and conferences, and the implemented SW is publicly available. Finally, the results of this thesis prove that the deployment of ubiquitously distributed intelligence at edge devices is feasible and valuable, also providing a better understanding of how such ubiquitous distributed intelligence should be, the associated problems that exist, and how these problems must be addressed.

In the next section, we explain the fundamentals of this thesis, which is the background necessary to understand the techniques we use. In Section 3, we introduce state-of-the-art. In Section 4, we present a summary of the results. Finally, in Section 5, we outline the conclusions and future work.

This page is intentionally left blank.



## Chapter 2

# Fundamentals

This thesis address the challenge of distributed ubiquitous intelligence in edge devices. We deal with constrained, unreliable, and heterogeneous (in HW and SW) devices and seek to adapt to the volatility of dis/connections and get fault tolerance. In line with that, we want to use users' data to train artificial intelligence models. It is possible to use different machine learning models to learn from users' data, but in this thesis, we focus on distributed neural networks. Training an artificial neural network (ANN) is indeed an optimisation problem [1]. Moreover, doing this on devices with so many constraints raises a multitude of optimisation problems, many of them NP-hard, that we need to address. Therefore, we must study how we should use optimisation algorithms on edge devices. Metaheuristics are particularly useful for addressing these problems [3, 110]. There are many different types, but we focus on evolutionary algorithms due to their robustness and well-known good behaviour in many optimisations, design, control, and machine learning applications [3, 110]. In this chapter, we present the fundamentals of this thesis. First, we introduce decision and optimisation problems. Then, we explain why metaheuristics are helpful when the complexity of optimisation problems is high. Subsequently, we summarise the training of artificial neural networks (ANNs). Finally, we describe parallel and distributed deep learning.

### 2.1 Decision Problems and Complexity Theory

A *decision problem* [110] is a yes-or-no question over an infinite set of inputs (instances) with a set of solutions for every instance which can be empty. An example is deciding whether a given natural number is prime.

It is common to define a decision problem as the possible set of inputs followed by the set of inputs for which the answer is yes. Inputs can be natural numbers, but they can also be values of other types. Decision problems are often defined as *formal languages*, i.e. the subset of values for which the problem returns yes. Using the Gödel encoding, we can encode any string as a natural number and, therefore, any decision problem can be defined as a subset of the natural numbers [106].

*Computability theory* [23] studies *decision problems* that can be solved with a Turing machine. The Church-Turing thesis states that if there is an *algorithm*, a procedure that terminates, then there is an equivalent Turing machine.

There are *decidable* and *undecidable* decision problems [104]. They are decidable if the input for which the answer is yes is a recursive set, i.e. if there is an *algorithm* that for any arbitrary input correctly decides whether or not it belongs to the set. A problem is partially decidable if the set of inputs for which the answer is yes is a recursively enumerable set, i.e. there is an algorithm that can answer positively when

it is in the language but runs indefinitely otherwise. Partially decidable and undecidable problems are called undecidable. Undecidable problems can never have an algorithm to solve them, even with unlimited time and space resources [110].

*Algorithm analysis* studies the complexity of algorithms, and *computational complexity theory* analyses the complexity of problems. The complexity of a problem is equivalent to the complexity of the best-known algorithm to solve it. Complexity is generally defined in terms of worst-case analysis. The *big O notation* is utilised to classify algorithms according to how execution time or space (memory) grows when the input size increases [104, 110].

Decision problems can be ordered according to many-one reducibility [104] and related to feasible reductions as polynomial-time reductions. A problem A is reducible to problem B if an algorithm for solving B can also be used as a subroutine to solve A. A decision problem P is said to be complete for a set of decision problems S if P belongs to S, and each problem in S can be reduced to P. Computational theory classifies problems into complexity classes. We can distinguish:

- **P class Problem (polynomial time).**
  - There is at least one algorithm that solves it in polynomial time.
  - Example of P problems are graph connectivity, primality testing, matrix determinant or linear programming.
- **NP class Problem (nondeterministic polynomial).**
  - It is not solvable in polynomial time by a deterministic Turing machine.
  - It is solvable in polynomial time by a nondeterministic Turing machine.
  - Its solution can be guessed and verified in polynomial time.
  - NP class contains P class as a subset.
  - Example of NP problems are factoring and graph isomorphism.
- **NP-hard class Problem.**
  - Not all NP-hard problems have to be in NP.
  - It is at least as hard as any NP-complete problem.
  - Example of NP-hard problems are matrix permanent and halting problem.
- **NP-complete class Problem.**
  - It is not solvable in polynomial time by a deterministic Turing machine.
  - It is solvable in polynomial time by a nondeterministic Turing machine.
  - It is contained in both NP and NP-hard class problems.
  - They are the hardest problem of the NP class problems.
  - Example of NP-complete problems are hamilton cycle, steiner tree, graph 3-coloring, satisfiability and maximum clique.

A problem is *NP-hard* if an algorithm for solving it can be translated into one for solving any other NP-problem [104]. An NP-hard problem does not have to be in the NP class. It is easier to prove that a problem is NP than to demonstrate that it is NP-hard. If a problem is both, it is called *NP-complete*. An example of decision problems that is NP-hard but not NP-complete is the halting problem.

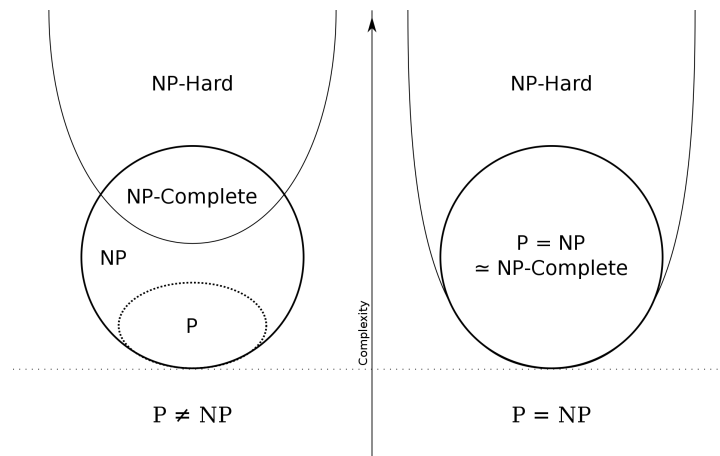


FIGURE 2.1: NP complexity.

There is no polynomial-time algorithm for any NP-complete problem yet, nor has it been proven that there is no polynomial-time algorithm for solving them. Suppose, in the future, a polynomial-time algorithm is discovered to solve an NP-complete problem. In that case, all NP-complete problems will be solved in polynomial time, proving that  $P = NP$  [42] (see Figure 2.1).

Unlike decision problems, where there is only one correct answer for each input, *optimisation problems* deal with finding the best answer to a particular input. Complexity theory has typically focused on decision problems. Optimisation problems remain of interest in computability theory and operations research.

It is easier to compare the complexity of decision problems than optimisation problems, which is why NP-completeness is used for decision problems. However, solving a decision problem in polynomial time usually allows us to solve optimisation problems in polynomial time by making several polynomial calls to the decision problem. Therefore, it is equivalent to talking about the complexity of decision and optimisation problems in practice.

## 2.2 Optimisation Problems and Classes of Algorithms

In society, individuals, companies and governments must make decisions every time and everywhere. In science, we define a structured way for the decision-making process consisting of the following steps [110]: (I) formulate a problem, (II) model the problem, (III) optimise the problem, and (IV) implement a solution. *Optimising a problem* is the process to find a globally optimal solution  $s^*$ , or in the case that more than one exists, to find all globally optimal solutions. However, finding optimal solutions to a given problem can be intractable, so in practice, we may settle for finding a good enough solution to that problem without it being the best possible one. We use heuristic and metaheuristic algorithms [3] to obtain acceptable solutions. They do not guarantee to find optimal solutions (exact algorithms), nor do we know how close we are to the optimal solution (approximation algorithms).

**Definition 2.2.1** (Optimisation problem). An optimisation problem is formalised as a pair  $(S, f)$ , where  $S \neq \emptyset$  represents the set of feasible solutions of the problem, while  $f : S \rightarrow \mathbb{R}$  is the objective function or fitness function.

The *objective function* assigns a real value to a solution representing how good it is at solving the optimisation problem.

**Definition 2.2.2** (Global optimum). A solution  $s^* \in S$  is a global optimum if its objective function is better than all solutions in the search space. If minimising  $\forall s \in S, f(s^*) \leq f(s)$ .

Depending on the domain to which  $S$  belongs, we can define binary  $S \subseteq \mathbb{B}^*$ , integer  $S \subseteq \mathbb{N}^*$ , continuous  $S \subseteq \mathbb{R}^*$  or heterogeneous  $S \subseteq (\mathbb{B} \cup \mathbb{N} \cup \mathbb{R})^*$  optimisation problems.

However, in many real-world problems, we must optimise several objectives that often conflict with each other. In this type of problem, we do not have a global optimum solution but a set of solutions that dominate the others in one or the other objective. To achieve this, we use *multi-objective optimisation* [110]. A *multi-objective optimisation problem (MOP)* has multiple objective functions. It can be formulated as:

**Definition 2.2.3** (Multi-objective optimisation problem). A multi-objective optimisation problem is formalised as a pair  $(S, F)$ , where  $S \neq \emptyset$  represents the set of feasible solutions of the problem, while  $F : S \rightarrow \mathbb{R}$  is a vector of objective functions or fitness function and  $F = f_1, f_2, \dots, f_k, k \geq 2$ .

There is usually no feasible solution in multi-objective optimisation that simultaneously minimises all objective functions but solutions that cannot improve any objectives without degrading at least one of the other objectives, i.e. dominating other solutions. They are called *Pareto optimal solutions*. The *Pareto optimal front* is the set of solutions that dominate all other solutions [110].

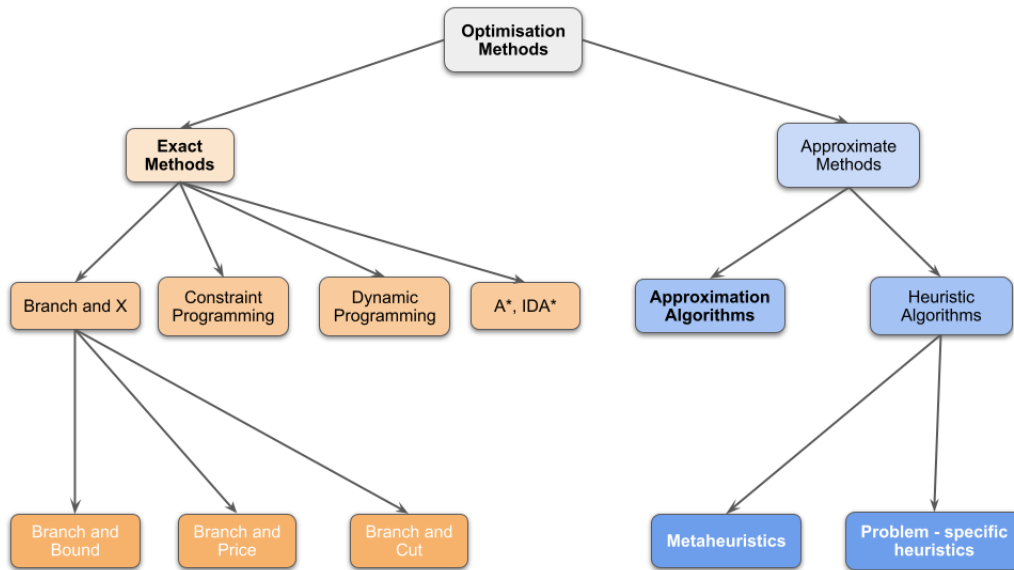


FIGURE 2.2: Classification of optimisation methods.

We can divide *optimisation algorithms* into four main categories [14, 110] (see Figure 2.2): exact algorithms, approximation algorithms, problem-specific heuristics, and metaheuristics. *Exact methods* achieves optimal solutions and guarantee their optimality [110]. However, when the problem is NP-Hard, the time needed to solve it increases exponentially concerning the dimensions of the problem. Examples of exact methods are the Branch-and-X family (X is the name of the variant), Integer Linear Programming (ILP), Mixed Integer Linear Programming (MILP), and dynamic programming. In *approximation algorithms*, there is a guarantee on the bound of the solution got concerning the global optimum. These approximations are, in many cases, too far from the optimal global solution, making them not very useful

for many real-world problems. *Problem-specific heuristics* attempt to use knowledge of the problem to obtain acceptable solutions but do not guarantee finding the optimal one. They tend to be too greedy and often get trapped in a local optimum. *Metaheuristics* are techniques that are independent of the problem to be solved and can be used as black boxes. They are generally not greedy and may even temporarily accept solutions that worsen the best solution obtained to increase exploration.

## 2.3 Metaheuristics: Problem-Independent Strategies for Complex Problems

Metaheuristics are problem-independent strategies that guide the search process. They know nothing about the problem in which they are used and can therefore treat functions as black boxes. There are numerous metaheuristics [13] (see Figure 2.3<sup>12</sup>) such as Tabu Search, Simulated Annealing, Particle Swarm, Evolutionary Algorithms, etc. Each of them can be divided into subgroups. We can use different classification criteria depending on nature vs. non-nature inspired, memory usage vs. memory-less usage, deterministic vs. stochastic, population-based vs. single-solution-based search, and iterative vs. greedy, among others. In the works carried out using metaheuristics in this thesis, we decided to use a type of evolutionary algorithm called genetic algorithm [3, 45, 110]. We chose it due to its robustness, well-known good behaviour in many optimisations, design, control, and machine learning applications, and high exploratory capacity. We performed multi-objective optimisation in one of our works using an algorithm called NSGA-II [32]. There are other more recent algorithms for multi-objective optimisation that are giving good results [123]. However, we chose NSGA-II as the first approach to that problem as it is a well-known algorithm with good results in many experiments and has more than 40 thousand citations. In the following subsections, we introduce the genetic algorithms for single-objective function problems and the NSGA-II algorithm for multi-objective function problems.

### 2.3.1 Genetic Algorithms: A Population-Based Metaheuristic

A *Genetic Algorithm (GA)* [3, 45, 110] is a type of evolutionary algorithm, a population-based metaheuristic inspired by the process of natural selection. The GA works in a stochastic and iterative way. They are used to get high-quality solutions to optimisation and search problems. They are especially helpful for solving NP-hard problems. This algorithm uses a population of candidate solutions, called individuals, to evolve iteratively over generations towards better solutions. Each individual has a set of properties (the chromosome) which can be mutated and recombined with other individuals, as in nature.

The GA (Fig. 2.4) uses three main types of rules at each step to create the next generation from the current population (see Algorithm 1). First, selection rules select the individuals, called parents, that will be recombined. Second, recombination rules combine the parents to form offspring for the next generation. Third, mutation rules apply random changes to the recombined individual to obtain the final offspring. Next, we replace the individuals of the population with the offspring.

<sup>1</sup>[metah.nojhan.net/post/2007/10/12/Classification-of-metaheuristics](http://metah.nojhan.net/post/2007/10/12/Classification-of-metaheuristics)

<sup>2</sup>[en.wikipedia.org/wiki/Metaheuristic](http://en.wikipedia.org/wiki/Metaheuristic)

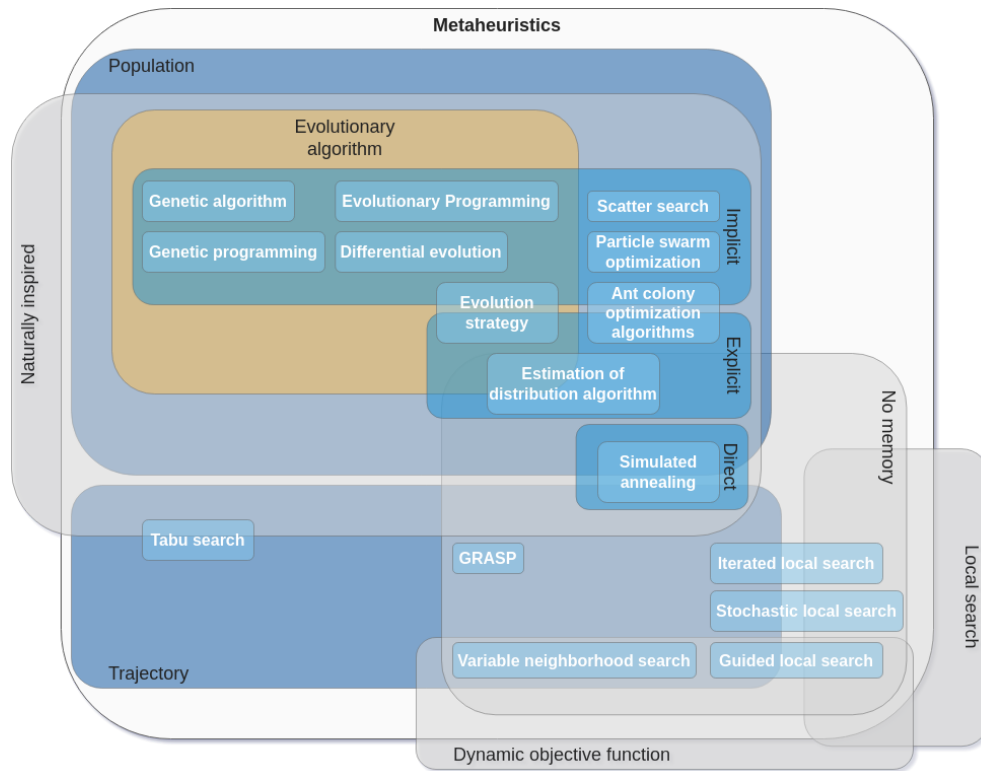


FIGURE 2.3: Euler diagram of classifications of metaheuristics.

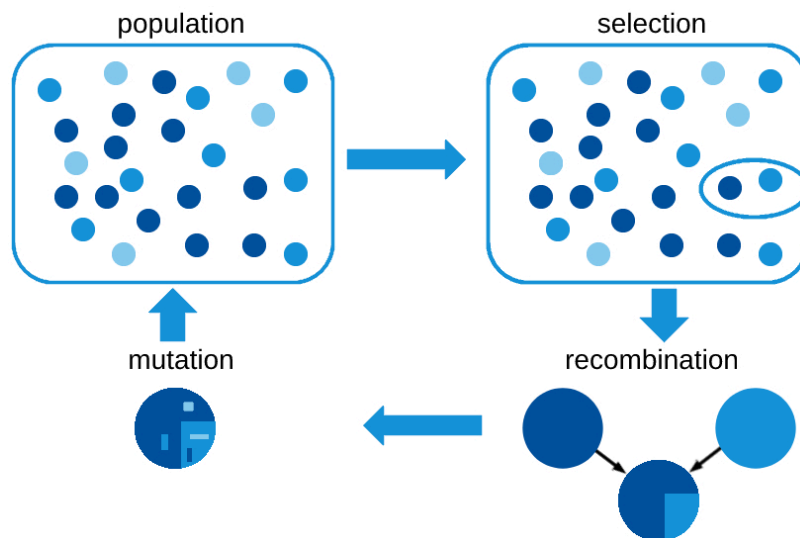


FIGURE 2.4: Graphical representation of our steady state GA.

Depending on the replacement strategy, we can distinguish [110]:

- *Generational replacement.* The offspring replace the entire population of size  $\mu$ .
- *Steady-state replacement.* Only one offspring is generated in each generation, and therefore only one individual from the population is replaced. If we replace the worst individual from the population, we use elitism replacement.
- *Intermediate replacement.* Many different strategies for replacing a given number of  $1 < \lambda < \mu$  individuals in the population can be applied. Replacements can be stochastic or deterministic.

**Algorithm 1** Genetic Algorithm

---

```

 $g \leftarrow 0$ 
 $P_g \leftarrow \text{initialisePopulation}()$ 
 $P_g \leftarrow \text{evaluate}(P_g)$ 
 $P_g \leftarrow \text{calculateStatistics}(P_g)$ 
while !terminationCriterion() do
   $\text{parents} \leftarrow \text{selection}(P_g)$  //Using selection criteria.
   $\text{offspring} \leftarrow \text{recombination}(\text{parents})$ 
   $\text{offspring} \leftarrow \text{mutation}(\text{offspring})$ 
   $\text{offspring} \leftarrow \text{evaluate}(\text{offspring})$ 
   $\text{population} \leftarrow \text{replace}(\text{offspring}, P_g)$  //Using replacement criteria.
   $P_g \leftarrow \text{calculateStatistics}(P_g)$ 
   $g \leftarrow g + 1$ 
end while

```

---

The process is repeated until a termination criterion is achieved, such as getting a good enough solution, a specified number of evaluations or iterations, etc.

**2.3.2 NSGA-II: A GA for Multi-Objective Function Optimisation**

NSGA-II is a generational *multi-objective evolutionary algorithm* (MOEA) proposed by Deb et al. [32]. Its main contributions are the non-dominated sorting and the diversity-preservation heuristics with a computational complexity of  $\mathcal{O}(MN^2)$  and  $\mathcal{O}(MN \log N)$ . Having a problem with  $M$  objectives, NSGA-II starts by randomly initialising a population  $P$  of  $U$  individuals  $\vec{X} = \{x_i, \dots, x_d\}$ , where  $i \in [1, d]$  and  $d$  is the size of the problem to be solved. Then it performs a loop applying binary tournament selection, crossover and mutation to generate a population  $Q$  of  $U$  offspring. Both parent and offspring populations,  $P \cup Q$ , are used as input for a replacement operator to decide the solutions that will survive to the next iteration  $P'$  (see Algorithm 2). The loop ends when some termination criterion is reached.

**Algorithm 2** NSGA-II

---

```

1: Set the objective functions  $F$  and  $x$  as a typical solution.
2: Set  $R = \{r_1, \dots, r_K\}$  the non-dominated fronts;
3:  $P \leftarrow \text{Random\_Initialisation}(H)$ ;
4: while termination criterion not reached do
5:    $A \leftarrow \text{Selection}(P, \text{Crowding\_Comparison})$ ;
6:    $B \leftarrow \text{Crossover}(A)$ ;
7:    $Q \leftarrow \text{Mutation}(B)$ ;
8:    $R \leftarrow \text{Non\_Dominated\_Sorting}(P \cup Q)$ ;
9:    $P' \leftarrow \emptyset$ ;
10:   $i \leftarrow 1$ ;
11:  while ( $|P' \cup r_i| \leq H$  and  $i \leq K$ ) do
12:     $P' \leftarrow P' \cup r_i$ ;
13:     $i \leftarrow i + 1$ ;
14:  end while
15:   $r_i \leftarrow \text{Sort\_Crowding\_Comparison}(r_i)$ ;
16:   $P \leftarrow P' \cup r_i[1 : (H - |P'|)]$ ;
17: end while

```

---

The replacement is performed using the non-dominated sorting heuristic and the crowding-comparison operator to decide the composition of the population  $P'$  that will undergo the next iteration [32]. The non-dominated sorting results in a set  $R = \{r_1, \dots, r_K\}$  of  $K$  non-dominated fronts of increasing rank  $i$ , where  $i \in [1, K]$ . Having  $r_1$  the front that is not dominated by any other one, while the remaining fronts are dominated by all the ones with a lower rank. The algorithm selects fronts in order of dominance, that is, favouring solutions of fronts of lower rank. Then, to select the remaining survivors of the last front, the algorithm performs the crowding-comparison operator, selecting solutions with a higher crowding distance.

## 2.4 The Process of Training an Artificial Neural Network

*Artificial neural networks* (ANNs) [48, 67], also known as neural networks (NNs), are computer systems with interconnected nodes inspired by biological neural networks that conform to the brains of animals. We use the term deep learning (DL) when referring to a NN with multiple non-linear layers. However, in practice, we interchangeably use both terms. They are a subset of ML (see Figure 2.5). They are composed of an input layer, one or more hidden layers and an output layer (see Figure 2.6). The connections between neurons are called *weights*. The output of one neuron acts as input to other neurons. A neuron's inputs are multiplied by their respective weights and summed together with the used bias. Subsequently, the above result fed an *activation function* [7] that calculates the final output (see Figure 2.7). The neuron is said to be activated if its output is above a certain threshold. In that case, the result pass to the next layer of the NN. Otherwise, no data is transmitted to the next network layer. When the outputs of neurons in one layer are passed on to neurons in the next layer in order without a cycle, we call this a *Feedforward Neural Network* (FNN). Other more complex configurations are possible such as *Convolutional Neural Network* (CNN) or *Recurrent Neural Network* (RNN) [48, 67]. This thesis uses these three types of neural networks: FNN, CNN and RNN.

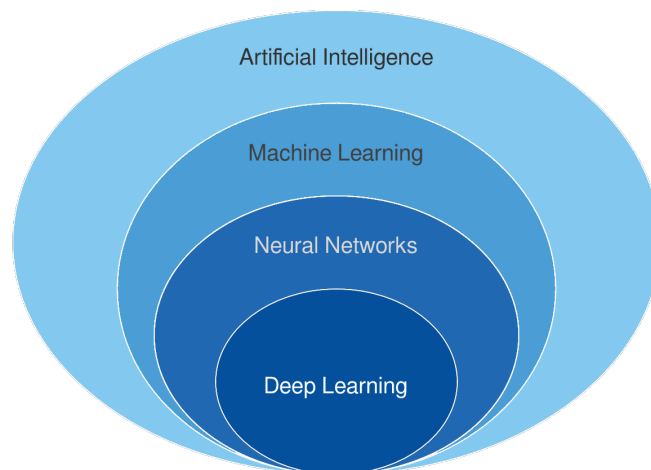


FIGURE 2.5: AI overview.

The purpose of the *activation function* is to introduce non-linearity (see Figure 2.8) into the output of a neuron allowing the NN to learn and perform more complex tasks. The NN becomes a simple linear regression model if we do not use a non-linear activation function. No matter how many hidden layers we use, the composition of two or more linear functions results in a linear function. We can visualise



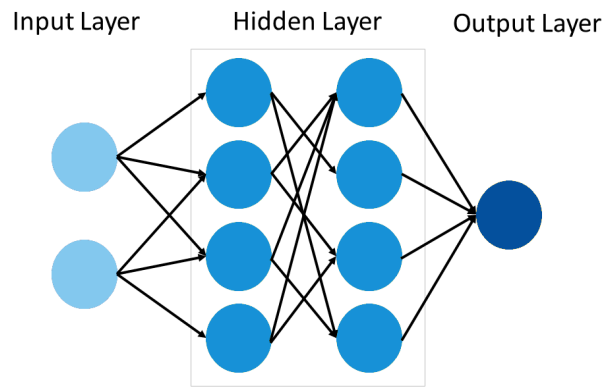


FIGURE 2.6: A neural network with two hidden layers.

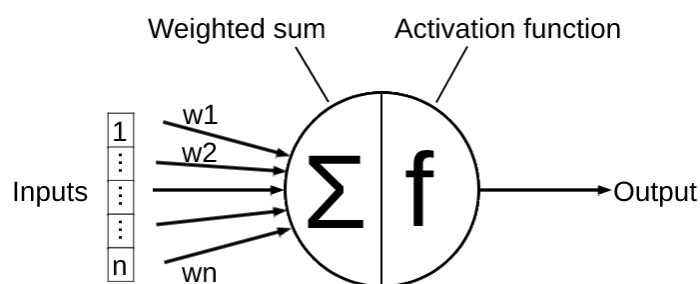


FIGURE 2.7: A perceptron sums inputs' multiplication by weights and applies an activation function to get the output.

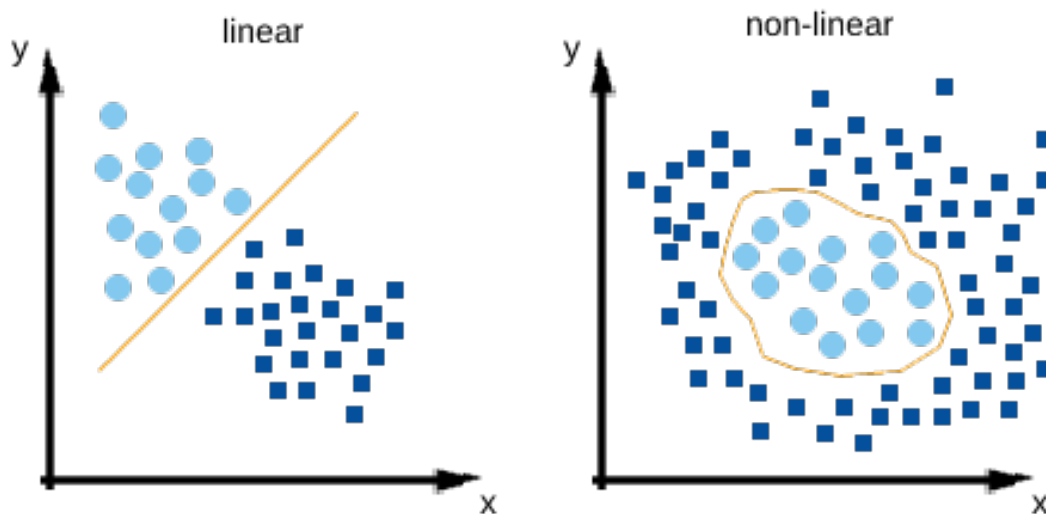


FIGURE 2.8: Decision boundary for linear and non-linear activation functions.

each NN neuron without activation function as a linear regression model composed of input data, weights, a bias and an output (see Algorithm 2.1). For example, if a specific neuron has three inputs:

$$y = \sum_{i=1}^m w_j \cdot x_j + \text{bias} = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + \text{bias}, \text{ where } m = 3 \quad (2.1)$$

Training a NN model is an optimisation problem consisting in minimising the cost/loss function by adjusting the values of the NN weights. For this, it is necessary to move the weights in the negative direction of the derivative of the loss function. Remember that the derivative of a function shows the direction in which the function increases (i.e. its negative is the direction we want to move in).

*Cost functions* are the tool that helps us to understand the difference between the predicted value and the actual value. Usually, it is said *loss function* when referring to the error for a single training example and *cost function* when referring to the average over the entire training dataset. A cost function is necessary because different models may have similar accuracy, for example, in classifying classes. Still, the line dividing two categories may be strictly between them or closer to one. The cost function helps to measure the performance of our model, in this case, how well it separates classes. In addition to accuracy and loss, other metrics measure model performance. For example, in multi-class classification, metrics such as accuracy do not provide enough information about the performance of our classifier. In contrast, the Cohen's Kappa score (CK score) [103] is a valuable metric that can handle multi-class and imbalanced class problems well. We use these metrics in this thesis.

*Backpropagation* is a widely used algorithm for training NNs [51, 95, 99]. In a NN, the path from the input layer to the output layer is basically a composition of functions. Therefore, partial derivatives and the chain rule can be used to define the relationship between any weight and the loss function. Then, this insight can be used to update the weights by gradient descent. The goal of backpropagation is to minimise the difference between the actual output and the predicted output of the NN (i.e. the error). The loss function computes this error (e.g. mean square error) and propagates it to previous layers. The optimisation function (e.g. gradient descent) calculates the gradients, i.e. the partial derivative of the loss function with respect to the weights. Then, the weights are modified in the opposite direction of the calculated gradient.

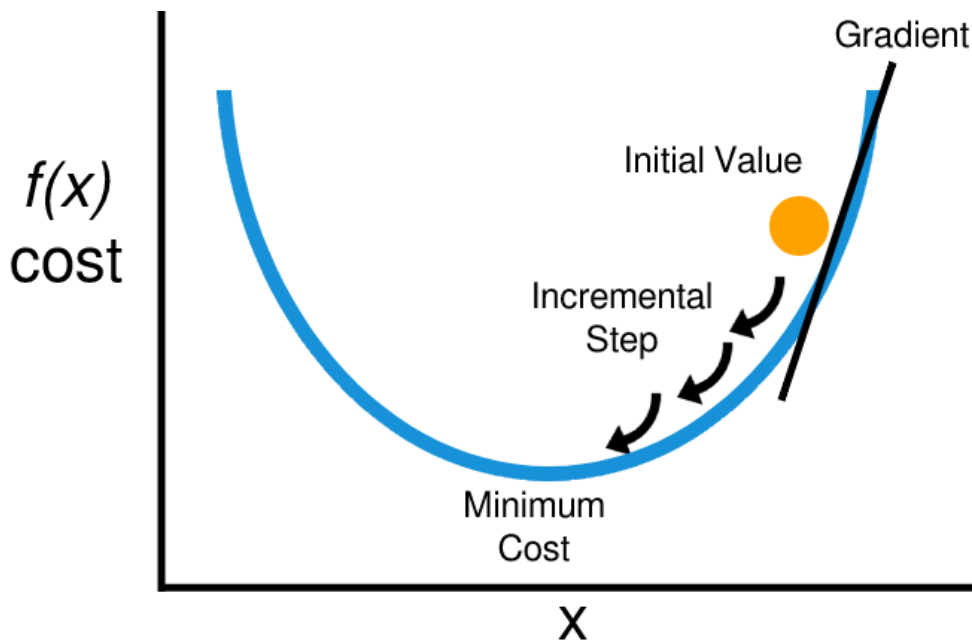


FIGURE 2.9: Gradient descent.

*Gradient descent (GD)* (see Figure 2.9) is the most common method to optimise NNs [51, 95, 99], i.e. adjusting the model's parameters to go down through the loss function. There are three variants of GD. Vanilla gradient descent (i.e. batch gradient descent) computes the gradient using the entire training dataset. Stochastic gradient descent (SGD) performs a parameter update for each training example. Mini-batch gradient descent approximates the partial derivative over the loss function using a randomly sampled subset of the data (i.e. a mini-batch). The last is the standard for training a NN, and the term SGD is also often used when we use mini-batches. Averaging gradients reduce the effect of noise. Otherwise, the gradient decrease may oscillate around the optimum without converging. We can use different techniques to adjust the parameters, such as subtracting their derivative multiplied by a learning rate  $\eta$ , regulating how much we want to move in the gradient direction. However, many algorithms improve on the basic SGD algorithm, such as Adagrad, RMSprop, Adam, and more [95].

Here we present a canonical SGD. We define  $D$  as a dataset. For each data example  $j$ , we define the loss function [58] as  $f(w, x_j, y_j)$ , which we write as  $f_j(w)$  in short. For each mini-batch of data  $B$  where  $B \subseteq D$ , we define the loss function obtained after applying the mini-batch  $B_k$  in iteration  $t$  as:

$$F_{k,t}(w_t) = \frac{1}{|B_k|} \sum_{j \in B_k} f_j(w_t) \quad (2.2)$$

Then, we update the weights of the model using an optimisation function, e.g. if we use canonical SGD:

$$w_{t+1} = w_t - \eta \nabla F_{k,t}(w_t) \quad (2.3)$$

where  $\eta > 0$  is the step size (i.e. the learning rate).

## 2.5 Parallel and Distributed Deep Learning

To achieve the objectives of this thesis, we need to divide the learning process among the available edge devices. For this purpose, in this section, we summarise the main types of distributed deep learning.

There are many ways to parallelise or distribute DL computation [10, 52, 75]. It is possible to perform single-machine parallelism, e.g. by using shared memory and a GPU. Also, it is possible to distribute the training between multiple machines using multi-machine parallelism. Both parallelisms can be combined. When we cannot store the model or data on a single machine, then distributed training across multiple machines is used. There are two main categories (see Figure 2.10):

1. *Data Parallelism*: Data is split between multiple machines while each worker stores a copy of the model. When a worker performs training, loads a mini-batch of data, applies a specific learning algorithm and computes the gradients. Then, using a synchronisation mechanism, the parameter server aggregates all gradients and updates the global shared model. SGD is applied in parallel using mini-batches (see Fig. 2.11).
2. *Model Parallelism*: Split the model layers between multiple machines. That is helpful only when the model does not fit on a single machine because it slows down the training.

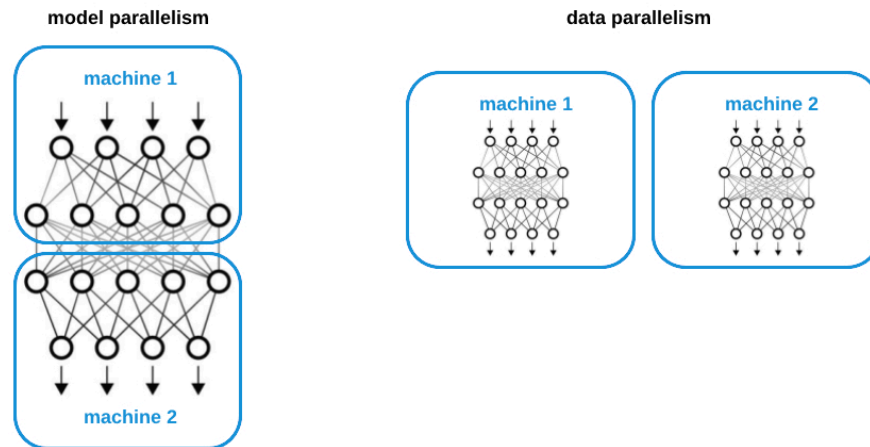


FIGURE 2.10: Model and data parallelism.

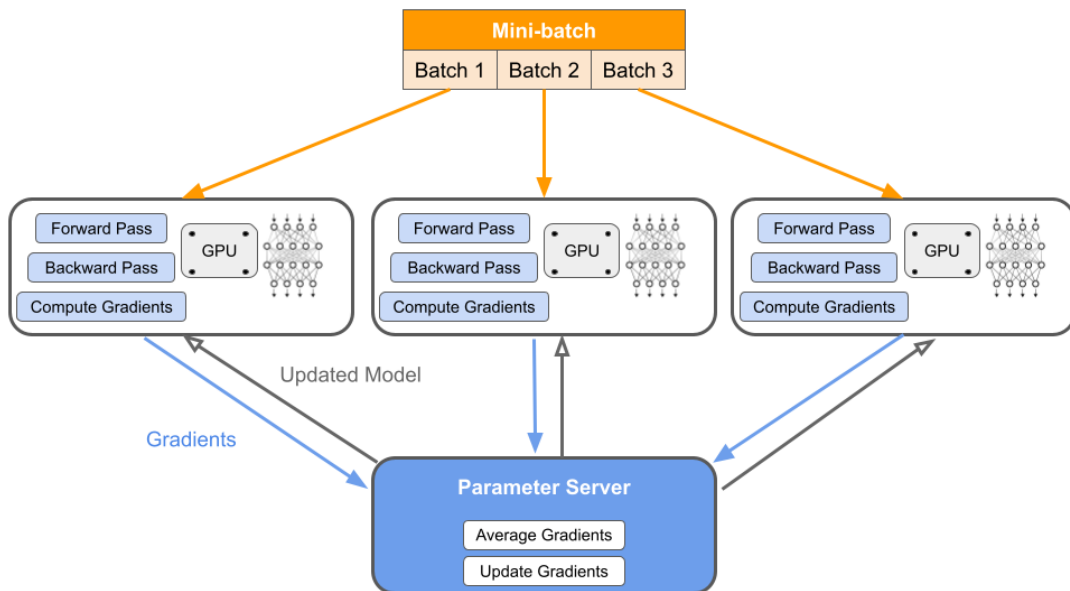


FIGURE 2.11: Parallel mini-batch gradient descent.

The *parameter server architecture* is a type of architecture of data parallelism which has two types of nodes [10, 52, 75]: *server* and *worker*. The server store the globally shared model. In each training step, workers download synchronised global weights, perform backpropagation to calculate the gradients, and send them to the server, which aggregates the gradients calculated in client devices (workers) and updates the globally shared model.

The size of the mini-batch of data used influences the final accuracy of the model. Increasing the size of the minibatch makes learning less noisy. However, large mini-batch sizes reduce the learning noise too much, not allowing it to diverge, favour convergence to local optima. Therefore, there is an upper limit to distributed learning through data parallelism.

In this thesis, we used single-machine parallelism and multi-machine data parallelism. Also, we used the parameter server architecture.

### 2.5.1 Synchronous vs. Asynchronous DDL

There are two main methods of SGD with parameter server [10, 52, 75]: *synchronous* and *asynchronous*. Synchronous methods are easier to implement and analyse but have bottleneck problems and are less robust to machine failures. They use all gradients calculated in all machines to update the globally shared model in the parameter server. The limited bandwidth may force queueing of these gradients. Also, we have to wait until the last device calculates the gradients and sends them before performing a new iteration. Therefore, the final computation time is determined by the slowest node, i.e. *straggler effects*, causing faster nodes to wait, wasting computational resources [10, 75]. When all workers must synchronise their weights before processing the next minibatch of data we say we have a *consistent model*.

It is possible to relax the synchronisation constraint by allowing *inconsistent models* [10, 52, 75], making the learning more robust to failures and faster. (i) *Asynchronous training* allows workers to update their model entirely independently. However, it has the drawback of lower stability, lower accuracy and convergence is not ensured. One solution to this is to use (ii) *bounded asynchronous training* in which out-of-date (*stale*) gradients or weights are used to update the globally shared model as long as they are not older than a threshold [124]. Depending on how outdated they are, we can use a different learning rate. Minor deviations and non-determinism during the training process do not necessarily have to harm the model accuracy [75].

We used both approaches in this thesis, first synchronous [86] and later bounded asynchronous [84] training.

In this section, we have seen the background necessary to understand the techniques used in this thesis. This thesis addresses the challenge of distributed ubiquitous intelligence in edge devices. We perform training of distributed neural networks using constrained, unreliable, and heterogeneous (in HW and SW) devices and seek to adapt to the volatility of dis/connections and get fault tolerance. We optimise the associated complex problems using metaheuristics. In this section, we have first explained what optimisation problems are. We have seen that training neural networks is an optimisation problem, and as we will explain in the next section, doing it on constrained devices [85] in a distributed manner gives rise to many others [87]. We have seen that it is usually impossible to find the optimal solution to complex optimisation problems but that we can be satisfied with an acceptable solution on many occasions. We have seen how metaheuristics are suitable for achieving acceptable solutions to complex optimisation problems. We summarised the training of neural networks and how to parallelise them on one or several machines. Finally, we have seen that distributed neural network training can be synchronous [86] or asynchronous [84] and the advantages and disadvantages of each approach.

This page is intentionally left blank.

## Chapter 3

# State-of-the-Art

In this chapter, we briefly present the state-of-the-art of this thesis. We propose to see FEEL as a type of VC (VC4FL) where users donate their edge devices' computing resources to a project that trains a shared DL model. First, we introduce federated learning (FL). Later, we review volunteer computing (VC) and browser-based volunteer computing (BBVC). Finally, we collect the challenges and desirable features of VC4FL.

### 3.1 Federated Learning: Learning Across Decentralised Edge Devices While Keeping Data Locally

FL originally comes from DDL [75]. It is a learning paradigm that trains a shared model in a distributed manner while keeping private the data locally on edge devices (see Figure 3.1). FL is being actively investigated and widely applied [50, 69, 93], e.g. in medicine [101]. Its working mechanism induces a substantial communication overload that limits its applicability. It has been proven that this overhead is generated by several factors such as the number of devices participating in the learning process, the complexity of the model (e.g. number of layers, neurons, etc.), number of communication rounds, etc. [75].

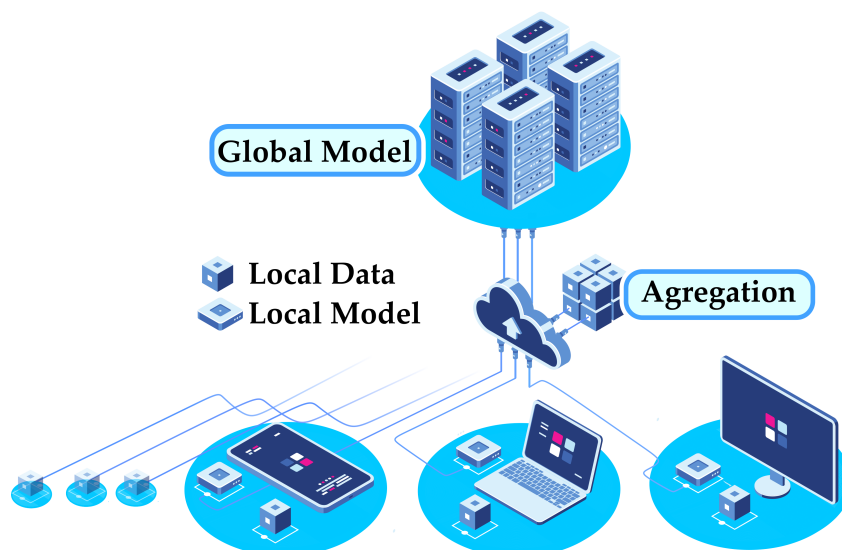


FIGURE 3.1: Federated learning architecture.

In Algorithm 3, we see the Federated Averaging algorithm (FedAvg) [76]. The vanilla FedAVG is synchronous. In the beginning, the server initialises a shared ML model, typically a NN, with random weights. There are  $N$  connected devices that can be thousands. Each iteration round  $m$  devices are selected randomly. Therefore,

**Algorithm 3** FedAvg algorithm

---

```

1: Initialise( $w_0^*$ );
2: for  $t = 1, \dots, T$  do
3:    $m \leftarrow \max(C \cdot N, 1)$ ;
4:    $S_t \leftarrow \text{random\_pick}(S, m)$ ;
5:   for all clients  $k \in S_t$  in parallel do
6:      $w_{t_0}^k \leftarrow w_t^*$ 
7:     for  $e \in 1, \dots, E$  do
8:        $w_{t_e}^k \leftarrow w_{t_{e-1}}^k - \eta \nabla F(w_{t_{e-1}}^k)$ ;
9:     end for
10:     $w_{(t+1)_0}^k \leftarrow w_{t_E}^k$ ;
11:   end for
12:    $w_{t+1}^* \leftarrow \sum_{k \in S_t} P^k \cdot w_{t+1}^k / m$ ;
13: end for

```

---

not all connected devices cooperate in all communication rounds. Next, the chosen devices  $S_t$  receive the updated weights from the server and use them to replace the local model. Then, they train for  $E$  local steps, i.e. each update of the local weights using a mini-batch of data. Later, these selected devices send the weights back to the server. Finally, the server aggregates all the received weights (typically averaging) and updates the shared model. The loop is executed repeatedly for  $T$  iterations.

In Google's original paper [76], they propose to keep data private on clients' mobile devices and learn a shared model by aggregating locally computed updates. They suggest a central server, millions of clients (mobile devices) and private data on clients' devices. Data is not independently or identically distributed (non-i.i.d.), is generated locally, and each device cannot read the data of other clients. However, since then, different categories of FL have been proposed. Some of these categories [77, 122] are:

- **Model-centric vs. data-centric FL.** The first has the goal of getting better centrally shared models. Each device stores its data locally and cannot read other clients' data. Data remains decentralised, is unbalanced and non-i.i.d. In the second one, users (researchers and organisations) send their models to train against a centred and private dataset.
- **Cross-device vs. cross-silo FL.** In the first case, there are potentially thousands or millions of user edge devices connected, which may connect and disconnect during the learning. The second one deals with private distributed databases of large organisations such as hospitals or banks.
- **Horizontal vs. vertical FL.** In the first one, there are different data samples in each node, but all of them have the same features. In the second one, data samples are the same in the dataset of each node, but each one stores different features for the same data sample ID.
- **Federated transfer learning.** This technique involves reusing the knowledge gained from solving a problem to solve a new one. It replaces the last few layers of the NN trained on big data on the cloud and then trains again with a smaller dataset, in this case using FL.

This thesis focuses on the model-centric, cross-device, horizontal FL, similar to Google's original paper.



## 3.2 Reviewing Volunteer Computing

As we explain in Section 3.3, we must deal with unreliable devices to achieve our goal of VC4FL. These devices behave similarly to devices on a VC platform, so we face similar challenges. In this section, we summarise the evolution of the VC and BBVC. First, we present VC systems. Next, we introduce the first and second generations of BBVC. Finally, we describe the third generation of BBVC. Figure 3.2 summarises the evolution of VC and BBVC. Also, we highlight the most important technological advances made in the last decade to web browsers and relevant works in the field. The web browser seems to be the right platform for achieving accessibility, which is essential in VC.

### 3.2.1 Volunteer Computing

The first references to the VC date back to the mid-1990s. The original idea was to use volatile idle resources spread worldwide to solve complex problems. Notably, in 1996, the project GIMPS (Great Internet Mersenne Prime Search<sup>1</sup>) released the first known VC platform. Its goal was (and is, it is running still) to find Mersenne prime numbers, and to encourage volunteers, they proposed a \$50,000 prize for the volunteer who found a prime number having at least 100,000,000 decimal digits.

Following this, in 1997, the *distributed.net* project arose, intending to break the RC5-56 part of the RSA Secret-Key Challenge. Next, in 1999, the University of Berkeley began the SETI@home project [64], which later evolved as a platform for general volunteer and grid computing, BOINC (Berkeley Open Infrastructure for Network Computing) [6]. At that time, the Folding@home project<sup>2</sup> was launched for disease research simulating computational drug design, protein folding, and other molecular dynamics. One year on, XtremeWeb [16, 39] emerged as an open-source software to create lightweight desktop grids.

VC systems have shown to be particularly suitable for CPU-intensive applications, which can be divided into many independent and autonomous tasks but are rather unsuitable for data-intensive tasks. Typically, VC requires data to be served by a group of centralised servers. From its beginnings until now, VC has improved by using dedicated protocols or stealing cycles during CPU idle time. However, the lack of accessibility and usability of VC remains a major challenge [28].

There are different ways of attracting volunteers [96, 98] in VC platforms. We can distinguish between true volunteers (i.e. altruists), paid volunteers, forced volunteers, or use *gamification*. Anyway, VC platforms require users to run specific software, i.e. installation is required, and it is well-known that is something many people do not want or know how to do. They do not trust on installing unfamiliar software on their machines [38], because they are not sure what the software can do or what information it can access, which would compromise the user's privacy [27].

Nevertheless, VC has matured to achieve remarkable processing capacity. For instance, BOINC has 27 PetaFLOPS average computing power available, more than 300 thousand active users and almost 850 thousand active computers [6]. For comparison, Summit or OLCF-4 (one of the fastest supercomputers in the world) get 143.5 PetaFLOPS in the LINPACK benchmark<sup>3</sup>. It is stunning that a distributed platform composed of devices with limited processing capacity can perform close to that of one of the world's fastest supercomputers.

<sup>1</sup><http://www.mersenne.org>

<sup>2</sup><https://foldingathome.org>

<sup>3</sup><https://www.top500.org/lists/2018/11/>

### 3.2.2 VC on the Web Browser: BBVC 1st and 2nd Generation

In the earliest days of VC, researchers began to explore the use of web browsers to perform distributed computation tasks resulting in BBVC. The first attempts were to develop Java applets that ran in the browser (e.g. Charlotte [59], Javelin [21], Bayanihan [96], Popcorn [15], SuperWeb [4], among others). Its usage consists of clicking a link to download an applet and allowing it to run in the background. Based on the expectations created, several authors forecasted in 1997 that by 2007 supercomputing in the web browser would be a reality [44]. However, the need for user interaction to start the application and installing a Java plugin (which was very slow to start) were drawbacks in this generation of BBVC. Moreover, Java and Flash plugins became obsolete due to serious security issues. Since 2015, many browser vendors have removed plugin support, such as Flash, Silverlight, Java and other plugin-based technologies.

Due to accessibility problems, BBVC was forgotten until 2007, when a second-generation based on JavaScript appeared. However, we should note that at that time, Javascript did not have modern features (e.g. JIT compiler, WebSocket, etc.). Some of the most important works include, *RABC*, one by Konishi et al. [63], which suggests a large scale distributed system using AJAX, and one of Merelo et al. [81], that develops a distributed evolutionary algorithm using P2P in a volunteer computing environment.

This second-generation had the advantage of not requiring user interaction. Users started collaborating by simply opening a webpage on a browser. However, in 2007, the major drawback they had was the performance. JavaScript was from 9.8 to 23.2 times slower than Java [63] and between 20 to 200 than C [61]. Poor performance, the lack of support for multi-thread, and the inability for direct communications between browsers made second-generation BBVC fail.

### 3.2.3 VC on the Web Browser: BBVC 3rd Generation

In 2008, significant advances were made to improve the browser's computing power [49], including the release of Chrome V8<sup>4</sup>, a high-performance JavaScript. Though it was not until 2010 that several developments emerged that made a difference in this respect. Google released a new compiling infrastructure called Crankshaft. Also, workers appeared (multi-thread Javascript<sup>5</sup>). Thanks to this, the browser shifted significantly towards a competitive performance.

In 2011, two relevant technologies for HPC in the browser were launched: WebCL<sup>6</sup> and WebGL<sup>7</sup>. WebCL is a JavaScript link to OpenCL, a heterogeneous parallel computing framework that leverages CPUs and multicore GPUs within the web browser without plugins. There are works that used WebCL (e.g. WeevilScout [28], CrowdCL [73]). Yet, WebCL is still under development, no browser supports it natively, and the only way to use it is via browser extensions. In counterpart, WebGL allows GPU-accelerated usage of physics and image processing in the web browser without plugins. Although WebGL was initially developed for graphics rendering, it is also used for other applications like machine learning (ML) in the browser, e.g. TensorFlow.js<sup>8</sup> (note that Google previously released *deeplearn.js*, but

<sup>4</sup><https://v8.dev/>

<sup>5</sup><https://html.spec.whatwg.org/multipage/workers.html>

<sup>6</sup><https://www.khronos.org/webcl/>

<sup>7</sup><https://www.khronos.org/webgl/>

<sup>8</sup><https://github.com/tensorflow/tfjs>

now it is integrated into the core of TensorFlow.js). In the same year, WebSocket<sup>9</sup> and WebRTC [11, 33], two important web-browser communication technologies, were launched. The first provides full-duplex communication channels over a single TCP connection which is more suitable than HTTP when low latency is required. WebRTC is a browser-based real-time peer-to-peer communication without plugins. WebSockets have better performance than HTTP, although HTTP is easier to scale.

In 2013, Mozilla released *asm.js*<sup>10</sup> with the goal of running native code (e.g. C, C++) in the web browser. In 2015, WebAssembly (WASM) [49] did something similar. Both approaches use a source-to-source compiler (e.g. Emscripten<sup>11</sup>) to translate the original source code to the desired format, i.e. *asm.js* or WASM code. Thanks to these technologies, several desktop applications have been migrated successfully to the web such as SQLite<sup>12</sup>, Unreal Engine 3<sup>13</sup>, and AutoCAD<sup>14</sup> presenting a radical shift in the possibilities of web browsers as ubiquitous platforms.

In addition to performance, another critical issue for users is security. In this generation, several advances were made in this respect, like the introduction of the sandbox [56] which allows web applications to run isolated on modern web browsers accessing only a limited set of resources. As a result, users can run applications on the web browsers, ensuring they won't access their privacy as long as they do not have security vulnerabilities and without needing installation.

Thanks to all these improvements, applications running in the browser have reached a new potential. Interesting generalist BBVC platforms have been launched, such as QMachine [119] and one proposed by Chorazyk [20]. Also others specific to ML, such as MLitB [79] and OpenML [113].

However, frameworks that take advantage of all these improvements have not yet been realised despite these advances. For example, since the release of TensorFlow.js [105] in 2018, the implementation of Google's popular ML framework in Javascript, there has been no evidence of work using it in BBVC systems. Currently, BBVC systems stand out for their portability, extraordinary computing power potential, and being more secure than typical desktop applications. Therefore, in this thesis, we planned to explore the possibilities of distributed volunteer computing in web browsers. In this thesis, we use Emscripten and WASM in Section B, TensorFlow.js using the WASM and WebGL backend in sections 4.2 and 4.3, WebSockets in Section 4.2, and REST API over HTTP in Section 4.3.

---

<sup>9</sup><https://www.w3.org/TR/websockets/>

<sup>10</sup><http://asmjs.org>

<sup>11</sup><https://emscripten.org>

<sup>12</sup><https://www.sqlite.org>

<sup>13</sup><https://www.unrealengine.com>

<sup>14</sup><https://web.autocad.com>

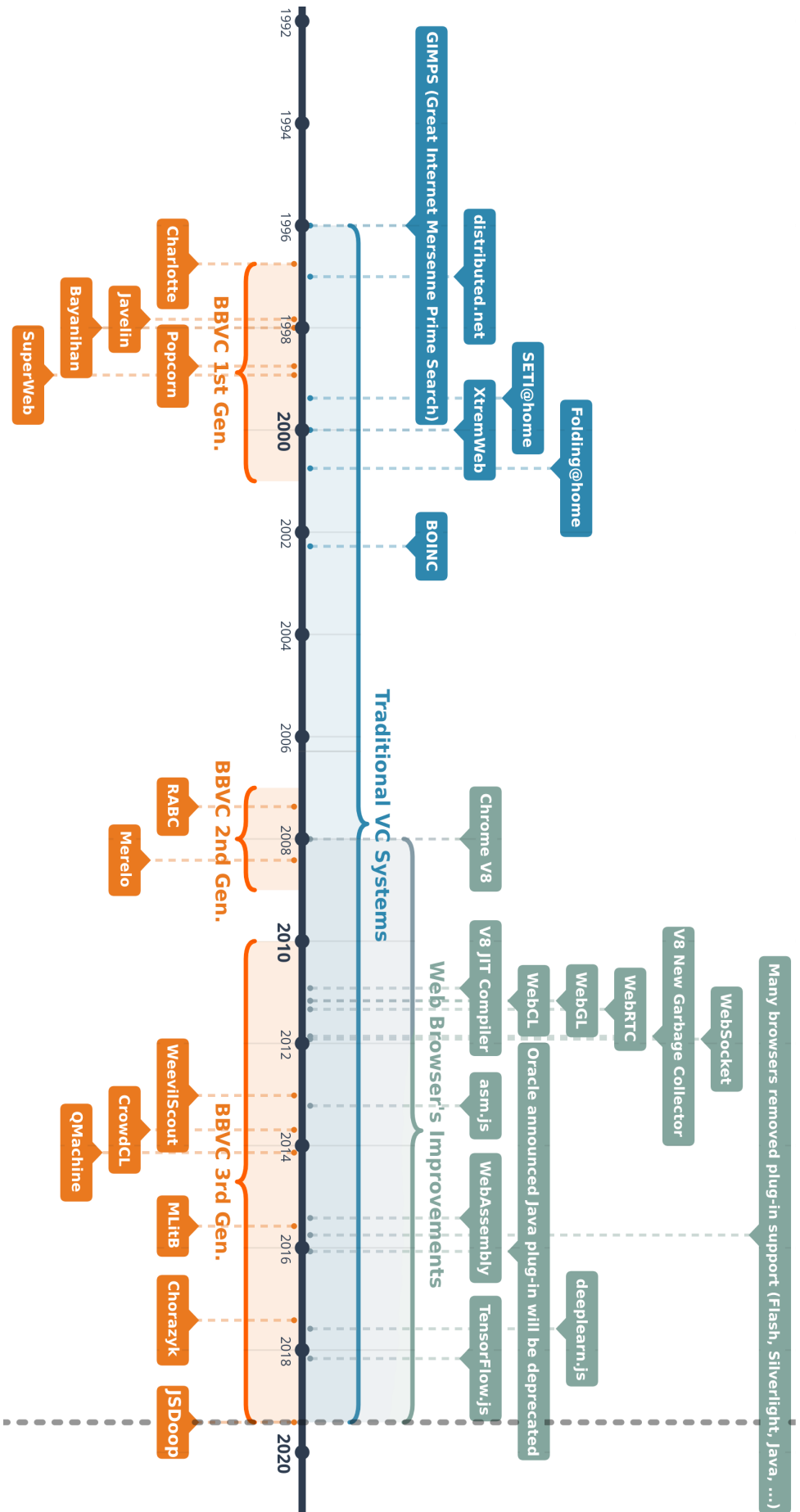


FIGURE 3.2: The evolution of volunteer computing and browser-based volunteer computing.

### 3.3 Challenges and Desirable Features of VC4FL

This section studies the challenges and the desirable features of VC4FL (see Table 3.1). We seek a volunteer distributed computing platform for collaboratively learning a shared ML model while keeping all training data locally on volunteer edge devices. FL and VC share many similarities, such as heterogeneous hardware and software, fault tolerance, and devices arbitrarily joining and leaving the system, among other things. Some of the most important challenges are [36, 77, 108]:

1. **Unreliable devices.** Devices are highly unreliable, can connect and disconnect at any time. We cannot be sure that volunteers will always be available for two reasons. The first one is when they are volunteers, and therefore, they can connect and disconnect at will. The second one is because of connections failures or errors as they are devices not thought for high-performance computing (HPC). In this thesis, we address this problem in Sections 4.2 and 4.3
  - (a) **Addressability.** In cross-device FL, we don't know who or when will be connected, so clients cannot be indexed directly. In this thesis, we address this problem in Sections 4.2 and 4.3.
  - (b) **Stateless.** Clients may connect during learning and participate in only one task before disconnecting again. In this thesis, we address this problem in Sections 4.2 and 4.3.
    - i. **Minimising loss of computation.** A possible solution to minimise the loss of computation when a device disconnects before finishing a task is to assign a customised number of local updates to each device depending on its performance and probability of failure. We plan to investigate this further in future work.
  - (c) **Unbalanced and *non-i.i.d.* data.** In FL, the data stored on each device is different, each device may have a different number of samples (unbalanced), and data is not identically distributed. That is, datasets are divided between nodes in an unbalanced and non-identically distributed manner (*non-i.i.d.*) [77]. Traditional distributed NN training does not have this problem because data is divided into *i.i.d.* (i.e. sufficiently randomly disordered to make the order of the data unrelated) balanced datasets [62]. However, when performing FEEL, data is always private and stored in local devices, so we can expect to be facing unbalanced and not identically distributed data (e.g. *non-i.i.d.*) [75, 77]. In this thesis, we analyse this problem in Section 4.3.
  - (d) **Data availability.** The connections and disconnections of devices make the available data for training varies during the learning process. Fluctuations in connected devices during learning mean that the dataset available for training at any given time also fluctuates. Therefore, the global dataset (the dataset of all available devices) is dynamic. That is related to the previous point, but it is not the same. In this thesis, we analyse this problem in Section 4.3.
2. **Scalability.** Scalability is critical on a platform like this that could have thousands of concurrent connections at a time. In this thesis, we address this problem in sections 4.3 and 4.4

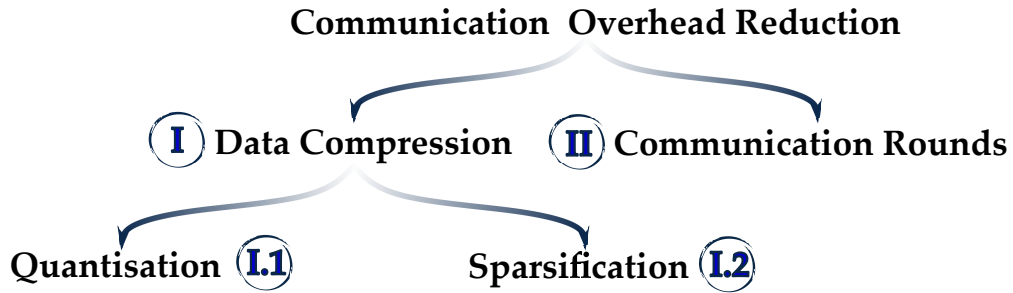


FIGURE 3.3: Communication reduction approaches in FL.

- (a) **Bottleneck accessing data.** Another common problem is the bottleneck when accessing data. In traditional distributed NN training, the data is shuffled and divided among the participating nodes at the beginning of each epoch. In the case of FEEL, the data remains locally and private. We would think that there is no bottleneck to access them in this case. However, the fact that the data remains private does not necessarily mean that it is on a single device. Data can be stored in a shared database between several local devices keeping privacy in the local network while collaborating with other remote devices. Furthermore, this data may be dynamic, i.e. the content of this database may be changing during the learning process. One way to access shared data during a learning process by reducing the bottleneck as much as possible is through random access to the data, which is equivalent to sampling with replacement [77, 80, 100] allowing accessing data simultaneously, without the need of distributing data or waiting for each other. We propose random access to the data in Section 4.3.
- (b) **Bottleneck in communications.** Similarly to DDL, the communications bottleneck is also a key feature in FL that can compromise both the efficacy and applicability [10, 75, 109, 120]. In FL, sending and receiving model weights during learning through slow connections is necessary, generating communication bottlenecks. The literature that studied the communication reduction in DDL can be classified into three categories according to the approach it uses: (I) *data compression*, (II) *decreasing communication rounds*, and (III) *reducing the number of participating devices (sampling)* (see Fig. 3.3).
- i. **Data Compression.** Regarding the data compression approach, two approaches can be mentioned: (I.1) *quantisation* [5] and (I.2) *sparsification* [115]. The first approach is related to compressing gradients/weights before, during or after the training representing the data using small-sized low-precision data (i.e. using fewer bits to represent a value). The second approach transmits indispensable values of each communication, i.e. not all weights are sent, but only the most representative values of gradients/weights above a specific threshold. Quantisation tends to slow the convergence of the model, and the maximum data compression rate is limited to 1/32, because DDL generally uses 32-bits-coded data. Sparsification attains a 1/100 compression rate without major changes in the model's convergence speed and accuracy. Sparsification introduces additional steps such as sampling, (de)compression and (de)coding during the

training process. These additional steps can affect the overall training efficiency, particularly in low-performance (e.g. netbook) and battery-sensitive (e.g. smartphone) devices. We use quantisation and sparsification in Section 4.4.

- ii. **Reducing Communication Rounds.** Considering now the second reduction technique, decreasing communication rounds [125], the client-server communication occurs after  $E$  local steps. Training a deep neural network requires hundreds of thousands of iterations. Thus using sparse communication intervals allows for reducing the communication overhead. So, the FedAvg and its variants permit clients to execute several iterations of local training before updating the global model [125]. Research shows that communication rounds' reduction increases the convergence speed. Concretely, the communication intervals in FedAvg are controlled by the hyperparameter  $E$ . The former affects the model's accuracy-vs-training-efficiency tradeoff. Higher values of  $E$  can lead to slow convergence. Therefore, fine-tuning  $E$  require advanced knowledge (e.g. expertise) to allow the model to provide the best accuracy. We optimise the number of local steps in Section 4.4.
  - iii. **Reducing the number of participating devices (sampling).** Not all devices must participate in each communication round. It is possible to use intelligent node selection algorithms [108] to deal with HW heterogeneity and with non-i.i.d. datasets. We optimise the number of participating devices in Section 4.4, but we plan to design more sophisticated optimisation algorithms for node selection based on HW heterogeneity and non-i.i.d. datasets in future work.
3. **HW and SW Heterogeneity.** When using volunteer devices, it is expected to deal with different hardware and software, leading to various performances. We analyse the heterogeneity of edge devices in Section 4.1 and deal with them in sections 4.2 and 4.3. In the associated literature [122], there are several techniques to address this problem:

- **SW Heterogeneity.** In the case of volunteer distributed NN training, each node may use different operating systems and different software (e.g. ML library). During training, it is necessary to exchange gradients, weights and model topologies between the edge devices and the central server. There are currently several approaches to this problem, like the Open Neural Network (ONNX)<sup>15</sup> which tries to achieve interoperability of the various NN libraries. However, it is not yet fully compatible with all of them. Additionally, software containers make it easier to deploy applications across different platforms. For example, using Docker<sup>16</sup> we can package and deploy algorithms into standardised services and applications. In Section 4.3, we interchange NN information between different programming languages and NN libraries, TensorFlow in Python and TensorFlow.js in JavaScript and NodeJS. Both NN libraries have similar names and similar APIs but have different implementations. For this purpose, we used the HDF5 format<sup>17</sup> for sharing the model topology, the

<sup>15</sup>[onnx.ai](https://onnx.ai)

<sup>16</sup>[www.docker.com](https://www.docker.com)

<sup>17</sup>[www.hdfgroup.org/solutions/hdf5/](https://www.hdfgroup.org/solutions/hdf5/)

NPY file format<sup>18</sup> for sharing the gradients/weights, and Docker containers for easy deployment.

- **HW Heterogeneity.** When using volunteer devices, the hardware of these devices can be very diverse, so some will be much faster than others when computing a given task. Some researchers [71, 114] propose algorithms to adapt learning to this heterogeneity. Proposals are diverse; for instance, we can use customised local steps based on the device's capability. There are also proposals suggesting the use of intelligent selection algorithms [108] for the selection of nodes participating in a communication round instead of using random node selection as vanilla FedAVG [76] does. These algorithms could choose participants not only by their performance but also by the distribution of their local datasets. Another solution is to use asynchronous strategies for learning. However, most of the current work uses synchronous approaches because these algorithms are often more accurate, simpler to apply and make the results easier to analyse. Most existing research papers use constant and preconfigured workers without dynamic connections and disconnections during learning. We believe this does not fit a real scenario where all devices are not always available but dynamic, so the algorithm must adapt to these changes and continue learning. Therefore, we consider that the asynchronous algorithms approach is more appropriate to address this problem despite the risks and difficulties of this scenario. Section 4.2 presents a synchronous method for DDL using web browsers and NodeJS processes. Section 4.3 proposes an asynchronous method for FL using web browsers and python processes. Some ways to deal with HW heterogeneity are:

- (a) **Asynchronous Communication.** Each device process and communicate with the server without waiting for the other workers. Asynchronous training allows the server to aggregate the results of devices that performed different numbers of local steps, and it can receive the results at different times. When this happens, some researchers suggest using a threshold  $Z$  [124] to discard too old results, which is called bounded asynchronous training or/and using a different learning rate. We use these techniques in Section 4.3.
  - i. **Model heterogeneity.** Bounded async training allows each device to train with a different model version for a given time.

From a system design point of view. VC4FL must meet the following desirable features (see Table 3.1) which address the challenges of VC and FEEL:

1. **Accesibility.** Users need to be able to connect to the platform in a simple way and without complex pre-configuration to achieve the best accessibility. In addition, we must be aware that users give up their resources when performing VC. When we conduct FL, they also voluntarily give up access to their data (always keeping it private). Therefore, to be willing to do so, we must ensure that their effort is minimal and use reward techniques such as gamification [19] to retain users for more extended and engage them in a global project for the good of science and society. In this thesis, we propose and analyse the use of web browsers for this purpose. We believe that web browser computing is an appropriate direction for this type of volunteer computing, thanks to its

<sup>18</sup>[numpy.org/devdocs/reference/generated/numpy.lib.format.html](https://numpy.org/devdocs/reference/generated/numpy.lib.format.html)



TABLE 3.1: Desirable features of VC4FL.

Desirable feature	Description
Accessibility	Users must connect to the platform easily
Adaptability / Dynamicity	The platform must adapt to an ever-changing environment
Availability	The platform must have minimal service interruption
Data privacy	The access to customer's data must be restricted and remain private
Fault-Tolerance	The platform should be tolerant of failures and disconnections
Heterogeneity	The platform must enable interoperability between heterogeneous HW and SW
Programmability	Developers should be able to add new features to the platform quickly
Scalability	The platform must handle a growing number of connections
Security	The machines of the volunteers should not be compromised
Usability	The platform has to be easy to deploy and use

ubiquity, sandboxing, and no need for software installation [82]. In the works presented in sections 4.2 and 4.3, we also use gamification techniques, such as dynamic graphs that display information about the training (training speed, ranking of users who have contributed the most, among others.), and a game embedded in the web browser to keep the user engaged.

2. **Adaptability/Dynamicity.** The platform must tolerate device connections and disconnections and adapt to these changes if necessary without stopping its work. We propose using the message queue pattern in sections 4.2 and 4.3. Also, the use of REST APIs in Section 4.3. We must design adaptive algorithms to adjust to the different performances of devices, variability of device connections and disconnections and thus to the dynamicity of the available data for learning. We propose an adaptive algorithm in Section 4.3.
3. **Fault-Tolerance.** In addition to voluntary disconnections, the platform must be tolerant to connection failures, errors or any other unexpected situation, such as if devices freeze, thus paralysing the training. The learning must continue despite any obstacles. Fault-tolerant is usually tackled using techniques such as checkpointing, redundant processing, and heartbeat monitoring [108]. Usually, checkpointing and heartbeat are more useful when the processing time before communication is high. Concerning redundancy, some authors argued that it has to deal with a considerable amount of redundant computing and therefore, researchers must study new strategies to improve the efficiency of big data processing on VC [72]. Other authors suggest using redundancy techniques with minimal impact on performance [108]. Still, we must consider that distributed training of a NN is slow, so redundancy may not be the best way to speed up processing. Also, when performing FL, data is stored locally on edge devices and is private. In this case, it is impossible to guarantee redundancy because each device will compute using different local data. We propose assigning a maximum time for performing a task in section 4.3 as a possible solution. We address fault-tolerance in sections 4.2 and 4.3.
4. **Availability.** Getting this requires that the system is as decoupled as possible, i.e. each system element is as independent of the others as possible. In addition, the platform should replace the system elements in case of failures or disconnections. In short, it is necessary to follow a decentralised and decoupled approach. We address this problem in Section 4.3.

5. **Scalability.** This thesis studies the techniques presented earlier to address this problem in Section 4.4.
6. **Heterogeneity.** The platform must deal with the HW and SW heterogeneity of edge devices. We address this problem in sections 4.1, 4.2 and 4.3.
7. **Programmability.** For this platform to be successful, developers need to be able to add new features quickly and easily. Therefore, it must be modular and easily extensible to add new techniques and algorithms incrementally. Finally, this volunteer platform must be open-source so that there is no doubt about what it does when users use it. This thesis's designed and implemented software follow these guidelines and are publicly available in a GIT repository.
8. **Usability.** Users should be able to deploy and use the platform quickly and easily. For this purpose, it is appropriate to use software containers such as Docker that simplify this task by minimising compatibility problems between different platforms. We used this in Section 4.3.
9. **Data privacy.** A volunteer platform for FL must take special care always to keep users' data private as this is one of FL's primary purposes.
10. **Security.** This type of volunteer platform for FL has plenty of challenges [89] related to the security and privacy of user data, but these aspects are outside the scope of this thesis.

Despite the increase in FEEL research in recent years and its many real-world applications, no open-source software platform still meets all the challenges and desirable features of VC4FL using real resource-constrained edge devices. Most published works use preconfigured parameters in controlled laboratory environments [12, 17, 18, 76, 114, 121]. They use a fixed number of workers or always take a constant subset of all available ones at each iteration, needing a pre-configuration identifying the participant devices before learning starts (devices do not connect and disconnect during learning). They assume there are always thousands or millions of devices connected, choosing some of them in each iteration. However, they do not consider the possibility that the number of devices may decrease to just a few (or even zero) and how they must adapt the algorithm accordingly. They usually use synchronous training algorithms having to wait for the slowest node to continue learning in each iteration. Nevertheless, this may not be practical in a realistic environment where devices are unreliable (can disconnect at any time) and heterogeneous with diverse performances. We think, in real-world problems, we must use asynchronous training allowing collaborator's devices to join or leave during learning. Also, these approaches do not consider the interoperability [90] of the participating devices. Although some deal with multiple performances, none with diverse software like various learning libraries. Furthermore, they need a centralised node to manage the whole process and take care of the partial results mergers. Some works address some of these points, but none cover them. Regarding BOINC, its long computation cycle [72] and the software installation need are not the most suitable approaches for VC4FL. Alternatively, Ray [88] is used in a static computer cluster and does not address the possibility of not knowing which nodes and when will participate in learning. Moreover, in this thesis, we decided to design and implement the whole platform, always oriented to real resource-constrained devices to better identify the problems and needs.

## Chapter 4

# Summary of Results

In this chapter, we review the works undertaken by the PhD student to support this thesis. These publications result from the study carried out to achieve the thesis objectives. These works have been published in [85], [86], [84] and [87]. They are four contributions, two of them published in JCR journals, one in an international conference and one in a national conference. All our work pursues the goal **G4** of disseminating results. Our first article is focused on **G1**, which analyses the heterogeneity of hardware and software in edge devices. The second work, which focuses on the goals **G2** and **G3**, examines the best paradigms and protocols for the volunteer distributed platform. A synchronous but decoupled version is suggested, enabling the connection and disengagement of volunteers. Goals **G2** and **G3** are pursued in the third article as well, but this time we also address the issue of data privacy on edge devices and an asynchronous approach is suggested. The fourth paper pursues goals **G2** and **G3**, tackling the FL's communication overhead issue. They are presented in chronological order:

1. In [85], we analyse the suitability of edge devices as a platform for running genetic algorithms (GAs). **G1**, **G2**, and **G4**. **LIT**.
2. In [86], we summarise the literature that uses the web browser as a volunteer computing (BBVC) platform and describes recent improvements that increasingly make this possible. We propose a BBVC framework for distributed volunteer computing using the MapReduce programming paradigm and the message queue pattern through web browsers. **G2**, **G3**, and **G4**. **LIT** and **SYN**.
3. In [84], we define the challenges and desirable features of VC4FL. We propose an algorithm, implement and evaluate a software platform, and evaluate our proposal via extensive experimentation. **G2**, **G3**, and **G4**. **LIT** and **ASYN**.
4. In [87], we summarised the most common techniques in literature for solving the communication overhead problem in FL. Next, we formulate and model the Federated Learning Communication Overhead Problem as a multi-objective Problem (FL-COP), and we propose to solve it using NSGA-II. **G2**, **G3**, and **G4**. **LIT** and **MOD**.

For a complete list of publications and a description of the journal or conference concerned, see Appendix [D](#).

## 4.1 Analysing Suitability of Edge Devices

In this work [85], we address the goals **G1**, **G2**, and **G4** and sub-goals *1-a*, *1-b*, *1-c*, *5-a* and *5-b*. The contributions of this work are related to **LIT** (see Section 1.2.2). To achieve the objectives of the thesis, we must analyse the most common edge devices. For this purpose, we perform a set of experiments in which we measure the speed, memory usage, and battery consumption of these devices, solving a set of well-known optimisation problems using GAs. Specifically, we wanted to analyse if these devices are suitable to solve complex problems and study their differences in resource usage.

TABLE 4.1: Hardware in the edge: features.

	laptop	RP3	m4	tablet	m10
OS	Ubuntu 16.04 LTS (64-bit)	Raspbian 9 (64-bit)	Android 5.1.1 ARMv7 (32-bit)	Android 5.0.1 x86 (32-bit)	Android 6.0 AArch64 (64-bit)
Java VM	1.8.0_161	1.8.0_65	-	-	-
Model	Toshiba Satellite L50-B	Raspberry Pi 3	Zuk Z1	Lenovo TAB S8-50F	LeMobile LEX653
Memory	7893 MB 1600 MHz	745 MB 900 MHz	2871 MB 933 MHz	1870 MB 778 MHz	3759 MB 800 MHz
Factory Battery	14.8 V 2800 mAh	3.7 V 3800 mAh	3.7 V 4000 mAh	3.8 V 4290 mAh	4.2 V 4000 mAh

TABLE 4.2: Processor specifications.

	laptop	RP3	m4	tablet	m10
Name	Intel Core i7-4510U (64-bit)	ARM v7 BCM2709 (64-bit)	Qualcomm Krait 400 (32-bit)	Intel Atom Z3745 (32-bit)	ARM Cortex-A72 ARM Cortex-A53 MT6797D (64-bit)
Topology	1 Processor, 2 Cores, 4 Threads	1 Processor, 4 Cores	1 Processor, 4 Cores	1 Processor, 4 Cores	1 Processor, 10 Cores
Frequency	2.0 - 3.1 GHz	1.2 GHz	2.5 GHz	1.33 - 1.86 GHz	2.3 GHz × 2 1.85 GHz × 4 1.4 GHz × 4
L1 Inst. Cache	32.0 KB × 2	16.0 KB × 1	16.0 KB × 2	32.0 KB × 4	-
L1 Data Cache	32.0 KB × 2	16.0 KB × 1	16.0 KB × 2	24.0 KB × 4	-
L2 Cache	256 KB × 2	512 KB × 1	2.00 MB × 1	1.00 MB × 1	-
L3 Cache	4.00 MB × 1	-	-	-	-

We selected a good representation of edge devices with a wide variety of performances such as two different types of smartphones (*m4* and *m10*), a tablet (*tab*), a laptop (*lap*) and a raspberry pi 3 (*RP3*) which is one of the most used devices for edge deployments [47, 83, 92] (see Table 4.1 and 4.2).

TABLE 4.3: Benchmarks.

	laptop	RP3	m4	tablet	m10
GeekBench 4	1	-	2.15	2.43	1.38
Antutu CPU	-	-	1.12	1.26	1
Antutu Maths	-	-	1.06	1.90	1
Antutu Mem.	-	-	1	1.39	1.13
Whetstone	1	1.38	1.44	1.56	3.55
Dhrystone 2	1	4.42	2.65	4.13	2.18
Linpack	1	9.71	3.95	12.16	40.89
Livermore	1	13.53	6.89	17.90	5.91

We performed some well-known benchmarks (see Table 4.3) on these devices to get a baseline idea of their performance. Most of these benchmarks are not multiplatform. However, we can obtain certain information using them on our edge devices. The used benchmarks are the classical Whetstone [26], Dhrystone 2 [116], Linpack

TABLE 4.4: Problem instances.

Problem	Chrom. Size	Pop. Size	Recomb.	Probabilities		Type
				Mutation	Local Search	
OneMax	5000	100	0.8	$\frac{1}{\text{chromSize}}$	-	Binary
MTTP	200	100	0.8	$\frac{1}{\text{chromSize}}$	-	Binary
ECC	144	5000	0.8	$\frac{1}{\text{chromSize}}$	-	Binary
MMDP	240	5000	0.8	$\frac{1}{\text{chromSize}}$	-	Binary
CVRP	54	500	0.4	$\frac{1.2}{\text{chromSize}}$	1	Integer

[35] and Livermore [78]. We also use the Antutu benchmark<sup>1</sup>, which is specific for Android OS, and we focus on their CPU (1 single-core) and memory score. And finally, we use one of the benchmarks most used today for multi-platform devices, Geekbench 4 (1 single-core)<sup>2</sup>. The results of the different benchmarks have been normalised, getting a score. A score of 1 represents the best result, and the rest of the scores are proportional to this one. The results obtained in Antutu and GeekBench 4 are closer to what we expected. The problem is that Antutu is just for Android OS, and GeekBench 4 does not have an implementation for Raspbian OS. Based on the benchmark results and the features of the devices (Table 4.1 and 4.2) we can expect that the order from the faster to the slower device is as follow: *lap*, *m10*, *m4*, *tab*, *RP3*. These benchmarks help us have a first approach to these platforms as a first step before our experiments begin. Although, we should know that these benchmarks are just a number and do not have to be similar to the final results.

We chose a variety of combinatorial optimisation problems for the evaluation (see Table 4.4 and Appendix A). We use binary and integer representation problems of different dimensions, one NP-Hard problem, various constraints, and varied fitness functions. These problems have interesting features in optimisation, such as epistasis, multimodality, and deceptiveness. The binary representation problems chosen are the OneMax problem [37], the Minimum Tardy Task Problem (MTTP) [107], the Massively Multimodal Deceptive Problem (MMDP) [46], and the Error Correcting Code Design (ECC) [74]. Also, as an integer representation problem, we have selected the well-known Capacitated Vehicle Routing Problem (CVRP) [29] which can scale and characterise problems in smart mobility in cities and is NP-hard. We selected one instance of each problem (Table 4.4). We choose four binary representation instances from JCell Library<sup>3</sup>, and one using integer representation called CMT1, a CVRP instance proposed by Christofides [22] for this problem. These instances are not so large because we want to solve them using constrained devices with low processing capacity. We are interested in analysing the performance of solving the instance problems using heterogeneous edge devices and different programming languages and not solving huge problems. Therefore, we have chosen a varied set that all the evaluated devices can solve in a reasonable time.

We use the canonical steady-state GA to solve the above problems. We use binary tournament selection, one-point crossover, bit-flip mutation and elitism replacement for binary representation problems. The algorithm used to solve the CVRP was implemented similarly to that used by Dorronsoro, and Alba [2] (see Appendix A). In this case, we use Edge Recombination Crossover (ERX) as recombination method, and three mutation operators with equal probability: Insertion [41], Swap [8], and Inversion [55]. We also use a local search operator as an extra step to the canonical

<sup>1</sup><http://www.antutu.com>

<sup>2</sup><https://www.geekbench.com>

<sup>3</sup><http://neo.lcc.uma.es/software/jcell/>

TABLE 4.5: Problem results.

Device	Prob.	Fitness		Evaluations			Time (s)			score
		avg	hit	avg $\pm$ sd	max	min	avg $\pm$ sd	max	min	
laptop	OneMax	$5 \times 10^3$	100 %	$18(1) \times 10^4$	$23 \times 10^4$	$16 \times 10^4$	$1.95 \pm 0.14$	<b>2.38</b>	<b>1.76</b>	<b>1.00</b>
RP3	OneMax	$5 \times 10^3$	100 %	$18(1) \times 10^4$	$23 \times 10^4$	$16 \times 10^4$	$21.47 \pm 1.46$	26.28	19.43	11.01
m10	OneMax	$5 \times 10^3$	100 %	$18(1) \times 10^4$	$23 \times 10^4$	$16 \times 10^4$	$98.79 \pm 7.65$	117.13	87.46	50.66
m4	OneMax	$5 \times 10^3$	100 %	$18(1) \times 10^4$	$23 \times 10^4$	$16 \times 10^4$	$99.91 \pm 6.85$	119.85	90.33	51.23
tablet	OneMax	$5 \times 10^3$	100 %	$18(1) \times 10^4$	$23 \times 10^4$	$16 \times 10^4$	$176.32 \pm 12.34$	215.39	158.40	90.41
laptop	MTTP	$25 \times 10^{-4}$	100 %	$48(48) \times 10^4$	$24 \times 10^5$	$39 \times 10^3$	$0.52 \pm 0.53$	<b>2.80</b>	<b>0.05</b>	<b>1.00</b>
RP3	MTTP	$25 \times 10^{-4}$	100 %	$48(48) \times 10^4$	$24 \times 10^5$	$39 \times 10^3$	$5.86 \pm 5.82$	29.65	0.50	11.23
m10	MTTP	$25 \times 10^{-4}$	100 %	$48(48) \times 10^4$	$24 \times 10^5$	$39 \times 10^3$	$55.67 \pm 54.97$	278.69	4.42	106.67
tablet	MTTP	$25 \times 10^{-4}$	100 %	$48(48) \times 10^4$	$24 \times 10^5$	$39 \times 10^3$	$66.01 \pm 65.44$	334.04	5.63	126.49
m4	MTTP	$25 \times 10^{-4}$	100 %	$48(48) \times 10^4$	$24 \times 10^5$	$39 \times 10^3$	$71.75 \pm 71.93$	369.39	6.48	137.49
laptop	MMDP	40	100 %	$37(46) \times 10^4$	$22 \times 10^5$	$19 \times 10^4$	$2.12 \pm 0.66$	<b>4.68</b>	<b>1.69</b>	<b>1.00</b>
RP3	MMDP	40	100 %	$37(46) \times 10^4$	$22 \times 10^5$	$19 \times 10^4$	$17.94 \pm 5.66$	40.09	14.29	8.47
m10	MMDP	40	100 %	$37(46) \times 10^4$	$22 \times 10^5$	$19 \times 10^4$	$331.35 \pm 195.25$	1180.94	199.50	156.46
tablet	MMDP	40	100 %	$37(46) \times 10^4$	$22 \times 10^5$	$19 \times 10^4$	$416.22 \pm 74.83$	683.50	349.13	196.53
m4	MMDP	40	100 %	$37(46) \times 10^4$	$22 \times 10^5$	$19 \times 10^4$	$476.01 \pm 122.47$	983.44	379.61	224.76
laptop	ECC	$674 \times 10^{-4}$	100 %	$33(6) \times 10^4$	$45 \times 10^4$	$23 \times 10^4$	$2.68 \pm 0.34$	<b>3.41</b>	<b>1.99</b>	<b>1.00</b>
RP3	ECC	$674 \times 10^{-4}$	100 %	$33(6) \times 10^4$	$45 \times 10^4$	$23 \times 10^4$	$25.42 \pm 3.45$	32.85	18.94	9.48
m10	ECC	$674 \times 10^{-4}$	100 %	$33(6) \times 10^4$	$45 \times 10^4$	$23 \times 10^4$	$470.22 \pm 75.97$	621.57	310.18	175.30
tablet	ECC	$674 \times 10^{-4}$	100 %	$33(6) \times 10^4$	$45 \times 10^4$	$23 \times 10^4$	$533.14 \pm 66.72$	672.33	403.31	198.76
m4	ECC	$674 \times 10^{-4}$	100 %	$33(6) \times 10^4$	$45 \times 10^4$	$23 \times 10^4$	$577.81 \pm 71.82$	725.69	438.77	215.41
laptop	CVRP	524.61	100 %	$45(11) \times 10^6$	$80 \times 10^6$	$24 \times 10^6$	$9.85 \pm 2.45$	<b>17.29</b>	<b>5.09</b>	<b>1.00</b>
RP3	CVRP	524.61	100 %	$45(11) \times 10^6$	$80 \times 10^6$	$24 \times 10^6$	$76.39 \pm 19.54$	136.23	39.89	7.76
m10	CVRP	524.61	100 %	$45(11) \times 10^6$	$80 \times 10^6$	$24 \times 10^6$	$1637.20 \pm 553.04$	3389.71	771.45	166.28
tablet	CVRP	524.61	100 %	$45(11) \times 10^6$	$80 \times 10^6$	$24 \times 10^6$	$2376.80 \pm 601.78$	4218.04	1247.25	241.40
m4	CVRP	524.61	100 %	$45(11) \times 10^6$	$80 \times 10^6$	$24 \times 10^6$	$2440.79 \pm 589.31$	4150.62	1332.99	247.90

GA. Results have been confirmed using a Wilcoxon test with Bonferroni correction and a significance level of 0.025.

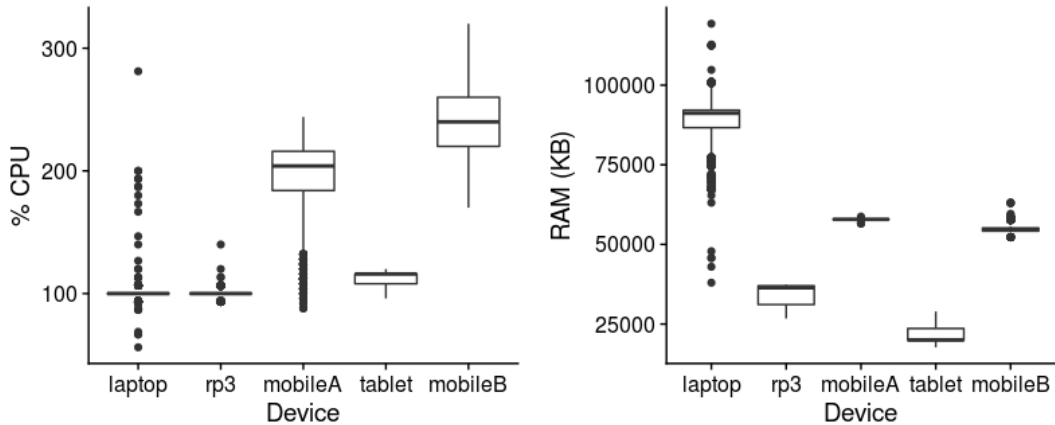


FIGURE 4.1: Resources usage.

Table 4.5 shows the results of the experiments. In Figure 4.1 we can see the CPU and memory usage.

The contributions of this work are related to **LIT**:

1. We studied and collected the available benchmarks for analysing the performance of edge devices. **G1**.
  - (a) We showed that classic benchmarks are not reliable for evaluating the performance of edge devices, but they are helpful to get a rough idea. They are dependent on the hardware architecture and the operating system. Depending on the specific problem to be solved, the results may vary. For instance, Antutu is so common that hardware manufacturers have taken to cheating on the benchmark, which makes it unreliable. **G1**.

2. We analysed the performance of different edge devices while solving a representative set of optimisation problems with different features by running genetic algorithms. **G1**.
  - (a) We showed that Raspberry Pi 3 (RP3) is a perfectly suitable platform for executing algorithms on edge. RP3 is very cheap, uses less memory than a laptop and solves problems in a reasonable time. RP3 is commonly used in intelligent city sensors, making it suitable for performing edge calculations. **G1**.
  - (b) We showed that edge devices running Android have problems running high-performance applications. This operating system seems more interested in keeping battery consumption low and using less memory (using the garbage collector more often) than performance. **G1**.
  - (c) We showed that benchmarks like GeekBench 4 got good results on Android devices. Therefore, we believe that it is necessary to execute native code in Android devices to be able to solve the restrictions of this operating system. In appendix B, we show a subsequent unpublished study. We analysed the performance of other programming languages such as C++ and WASM (same devices, problems and algorithms). We showed that our hypothesis was correct, getting a performance closer to RP3 and the laptop on Android devices. **G1** and **G2**.
    - i. We implemented the algorithms using three programming languages (Java, C++ and WASM-JavaScript). We showed that we get good performance in most devices by using native code and compiling it into WebAssembly (WASM) with Emscripten. Moreover, when using WASM, the results of the best and worst devices are very similar, obtaining an excellent homogeneity which is very important when running algorithms in a parallel way. We proved that running these algorithms on the web browser is efficient and has the advantages of a lightweight virtualisation property and being multiplatform. **G1**.
    - ii. We demonstrated that devices with low processor capacity are perfectly appropriate for solving optimisation problems at the edge. Inexpensive devices such as RP3 and mobile devices can achieve laptop-like performance when solving complex optimisation problems. **G1**.

## 4.2 Proposal for NN Training Using BBVC and the MapReduce Paradigm

Our previous work showed that edge devices with low processing capacity were perfectly suited to solve optimisation problems and that the web browser could be a suitable platform to achieve our goal. In this paper, we propose an approach to use the web browser to train neural networks in a distributed manner using volunteers. We propose a decoupled design so volunteers can connect and disconnect anytime without stopping the training.

In this work [86], we address the goals **G2**, **G3**, and **G4**, and sub-goals *1-a*, *1-d*, *1-e*, *2-a*, *4-a*, *4-b*, *4-c*, *4-d*, *4-e*, *5-a* and *5-b*. The contributions of this work are related to **LIT** and **SYN** (see Section 1.2.2). We first summarise the literature that uses the web browser as a volunteer computing (BBVC) platform and describes recent improvements that increasingly make this possible (see also Section 3.2). Next, we

propose a BBVC framework, JSDoop<sup>4</sup>, for distributed collaborative HPC using the MapReduce programming paradigm [65] and the message queue pattern through web browsers. As we have seen in sections 3.2 and 3.3 (see Table 3.1), the biggest obstacles for VC platforms are accessibility, usability and security. Installing applications is often a significant barrier for users because of difficulty, laziness or fear of installing unfamiliar software on their devices. We propose to use web browsers as runtime software mainly because it solves most of these problems. Moreover, as explained in section 3.2, the improvements that have been made to web browsers in recent years bring them closer and closer to the performance of native programs running on the terminal. Finally, we use this framework for training an LSTM-based RNN [54] in a distributed and collaborative manner.

An RNN is a type of NN in which the connections between nodes form a directed or undirected graph over a time sequence. MapReduce [31] is a well-known technology used in popular systems like Hadoop and Spark. It is composed of two primary operators: map and reduce. The first one gets the data and distributes it for parallel calculation. The second one aggregates the result of the calculation.

The design of our proposal was guided by the desirable features (see Table 3.1) of such platforms. However, our implementation is a proof of concept and aspects such as security should be more important in a final version. In our design, we can distinguish the following actors: **Initiator**, **WebServer**, **QueueServer**, **DataServer**, and **Volunteers** (see Fig. 4.2). The WebServer stores the web code that will run the volunteer on the web browser. The Initiator is the person who creates the job and divides the NN training into map and reduce tasks. The QueueServer stores the map and reduce tasks. The DataServer stores the current saved model and the dataset. And the Volunteers are the devices that connect and disconnect at will and process the tasks on their web browser.

The training process is divided into map-reduce tasks. Volunteers can connect to the platform using the web browser by clicking on a link. Then the program is executed in the background transparently, getting tasks from a queue on a server, solving them one by one, and accessing the DataServer when necessary to take the data. We use the message queue pattern. Therefore, tasks are not removed from the queue until an ACK is received. That allows volunteer devices to connect and disconnect anytime. If they disconnect before finishing a task, that task returns to the queue achieving fault-tolerance.

Map tasks mean computing gradients. Reduce tasks mean aggregate gradients and update the shared model. Before calculating a map task, the program executed in the volunteer web browser downloads the updated model and the mini-batch of data corresponding to that task, calculates gradients and stores the result in a results queue in the QueueServer. Before calculating a reduce task, the program executed in the volunteer web browser downloads the gradients identified with specific IDs, aggregates them, updates the model, and finally stores the updated model in the database.

Our proposal was implemented using NodeJS (version 10.15)<sup>5</sup>. Therefore, we can run a program developed using our library in a native application (e.g. in the console) and a web browser. For communications, on the client side, we use STOMP over WebSocket. The QueueServer is implemented using RabbitMQ (AMQP protocol)<sup>6</sup> for handling the queues. Also, the in-memory DataServer use Redis<sup>7</sup> to store

<sup>4</sup>Code available in <https://github.com/jsdoop/>

<sup>5</sup>[nodejs.org](https://nodejs.org)

<sup>6</sup>[www.rabbitmq.com](https://www.rabbitmq.com)

<sup>7</sup>[redis.io](https://redis.io)



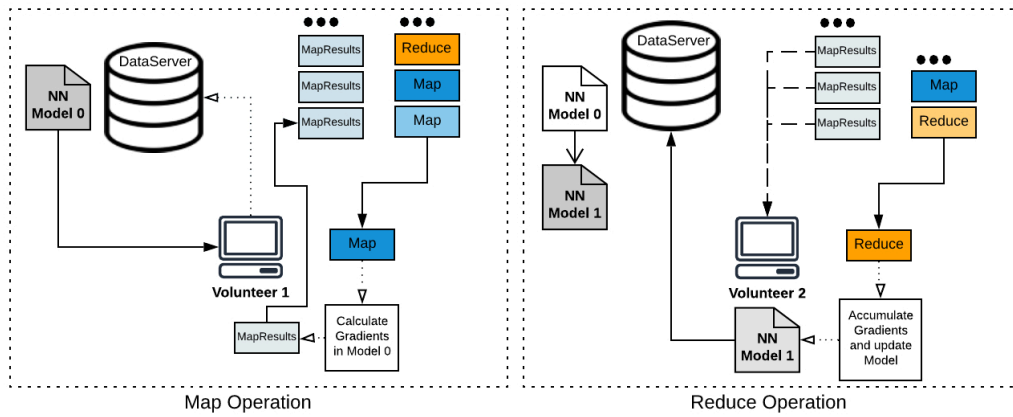


FIGURE 4.2: Our proposal for NN Training using the MapReduce paradigm.

TABLE 4.6: Distributed and sequential training.

System	Workers	Runtime	Loss
<b>TFJS-Sequential-128</b>	<b>1</b>	<b>0.9</b>	<b>4.6</b>
<b>JSDoop-classroom-sync-start</b>	<b>32</b>	<b>2.5</b>	<b>4.6</b>
JSDoop-classroom-async-start	32	2.7	4.6
JSDoop-classroom-sync-start	16	5.4	4.6
<b>JSDoop-cluster</b>	<b>32</b>	<b>8.4</b>	<b>4.6</b>
JSDoop-cluster	16	8.8	4.6
JSDoop-cluster	8	12.0	4.6
JSDoop-cluster	4	16.7	4.6
TFJS-Sequential-8	1	21.7	12.7
JSDoop-cluster	2	37.0	4.6
JSDoop-cluster	1	177.1	4.6

the data used in the experimentation. Finally, we use TensorFlow.js as a library for training neural networks.

We perform three groups of experiments: JSDoop-cluster, JSDoop-classroom and TFJS-Sequential. First, we tested six cases: 1, 2, 4, 8, 16, and 32 workers in a cluster of computers HTCondor [111]. Later, we tested our proposal on desktop web browsers by opening a hyperlink in a university classroom. In this case, we performed three types of experiments. In the first one, the 32 web browsers opened the hyperlink gradually (i.e. async-start). In the second one, we repeated the experiments using 32 web browsers with the webpage already opened (i.e. sync-start). In the third one, we closed 16 web browsers and repeated the experiment with the other already connected 16 web browsers (i.e. sync-start). We compare these results with TFJS-Sequential, in which we trained the RNN using the sequential version of the training algorithm with a batch size of 8 and a batch size of 128. In this sense, the sequential and the distributed algorithms compute the gradient the same number of times. Therefore, we compare both approaches under *similar* conditions, i.e. computing and accumulating the same number of times the gradient. In Table 4.6 we show the results of the experiments. The runtime is presented in minutes. The best time of each experiment is highlighted in bold.

The results show that web browser-based distributed neural network training is feasible. Our proposal allow volunteers to collaborate by simply accessing a URL, and proved to be an adequate implementation of BBVC achieving high scalability and allowing to add/delete volunteers dynamically during execution without losing information (tasks).

The main advantages of our proposal are:

- *It is independent of the number of connected devices.* The system sends tasks to whoever requests them and only removes a task from the queue when an ACK is received informing that the task has been completed.
- *It recovers from failures easily.* If a volunteer disconnects while solving a task, the task is added back to the queue.
- *The number of volunteers changes dynamically.* It allows a dynamic number of devices to join or leave the system at will.
- *It is cross-platform.* Any device can collaborate by simply connecting to a URL through a browser.

The main contributions of this work (**LIT** and **SYN**) are as follows:

1. We summarised the literature that uses the web browser as a volunteer computing (BBVC) platform and described recent improvements that increasingly make this possible (see Section 3.2). **G2**.
2. We proposed a BBVC framework for distributed volunteer computing using the MapReduce programming paradigm and the message queue pattern through web browsers without interrupting the site's user experience and installing additional software. We showed that our proposal has good scalability despite the constraints on the communication channel. **G3**.
3. We proved that web browser-based distributed neural network training is feasible and efficient. We conducted a proof-of-concept to show that distributed training of neural networks in the browser is possible. Using small and medium NN models is perfectly suitable for solving edge devices' problems. The results show that it is feasible, scalable, and an exciting area to explore. **G3**.

### 4.3 Proposal for Asynchronous FL using Unreliable Volunteer Edge Devices

In our previous work, we proposed to use the web browser to train NNs in a distributed volunteer way. However, we used a synchronous approach which has some disadvantages, as we saw in Section 2.5.1, such as *straggler effects* and devices that can freeze paralysing the training. In this paper, we propose an asynchronous method to tackle these problems. In particular, we focus on the situation when the number of volunteers is low and may even drop to zero during part of the training.

In this work [84] (see Section 3.3 and 4.3), we address the goals **G2**, **G3**, and **G4**, and sub-goals 1-a, 1-b, 1-d, 1-e, 2-a, 4-a, 4-b, 4-c, 4-d, 4-e, 4-f, 5-a and 5-b. The contributions of this work are related to **LIT** and **ASYN** (see Section 1.2.2). We first define the challenges and desirable features of VC4FL (see Section 3.3). We propose an algorithm for FEEL that adapts to asynchronous heterogeneous clients joining and leaving the computation. The aim is to continue the learning process and avoid waiting for slower devices. We propose, implement and evaluate a new software platform (JSDoop version 2.0<sup>8</sup>, redesigned and reimplemented from scratch) for DDL on volunteer edge devices.

<sup>8</sup>Code available in <https://github.com/jsdoop/>

In adaptive distributed mini-batch gradient descent, the number of connected edge nodes is dynamic and varies over time. We assume that the maximum number of edge nodes is  $N$  (connected or disconnected), that each local dataset  $D_i$  is unbalanced and *non-i.i.d.*,  $\cup_i^N D_i = D$  and  $\forall_i \forall_j |D_i \cap D_j| \geq 0$ . We define the current set of connected edge nodes as  $E$  where  $|E| \leq N$ . We dynamically adapt the *global aggregation* to the current connected edge node devices using:

$$w_t^{global} = \frac{1}{P} \sum_{y=1}^P w_{local,y} \quad , \quad P \simeq |E| \quad (4.1)$$

---

**Algorithm 4** Async-Adapt Distributed Gradient Descent
 

---

**Input:**  $\tau, Z$

**Output:** Final model parameter  $w_{t=T}^{global}$

```

1: Initialise:
   // Each worker loads global weights when connected.
    $w_{i,t} \leftarrow w_t^{global}$ ;
   // Aggregator initialise list of weights for aggregation
    $W \leftarrow []$ ;
2: while termination criterion is not met (each worker async) do
3:   for  $u = 1, 2, \dots, \tau$  do
4:     // Each node  $i$  compute local update (2.3)
5:   end for
6:   // Each node sends local weights  $w_{i,t}$ 
7:   // Aggregator receives the local weights async
   // Aggregator receives info of connected workers  $|E|$  and update  $P$ 
8:    $P = |E|$ 
9:   // Aggregator checks if received weights are too old
10:  if  $t_i + Z \geq t_{global}$  then
11:    // Aggregator adds weights to list for aggregation
     $W.insert(w_{i,t})$ 
12:  end if
13:  if  $W.size() \geq P$  then
14:    // Global aggregation (4.1) using  $W$ 
15:     $W = []$ 
16:  end if
17: end while

```

---

We define  $|E|$  as the number of connected edge node devices (workers) and  $P$  as the number of gradients the aggregator accumulates each aggregation. The adaptive algorithm dynamically adjusts  $P \simeq |E|$ . We refer to connected nodes  $|E|$  as the nodes that have communicated with the logical server(s) in the last  $S$  seconds. Also, we use asynchronous training, so devices do not have to wait for each other. Therefore, a new threshold variable  $Z$  has been defined to accept or discard old gradients depending on how old they are (see Algorithm 4).

We use six types of actors (see Fig. 4.3).

1. *The initiator* is the user that creates a job and uploads it to the platform. If the NN topology is not yet uploaded to the platform, he/she must upload it first.
2. *The workers* are the volunteers, i.e. edge devices. They collaborate solving tasks by yielding their computing resources and private data. They are unreliable

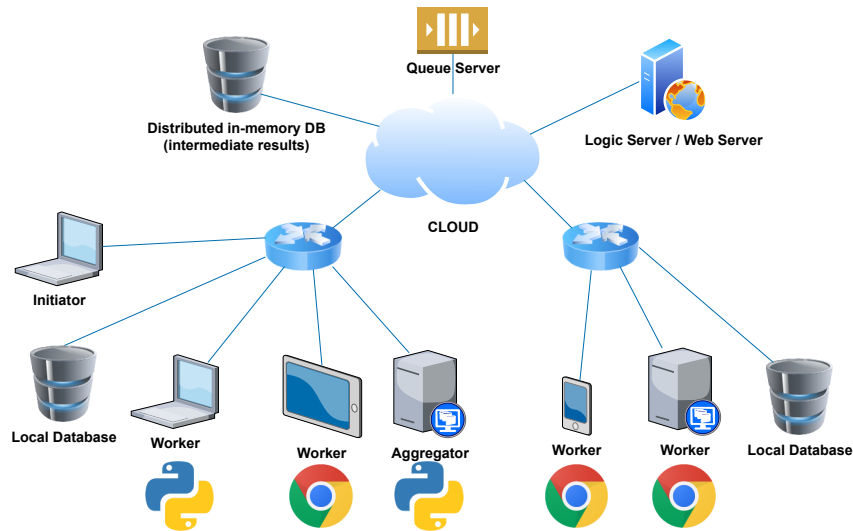


FIGURE 4.3: High-level system architecture.

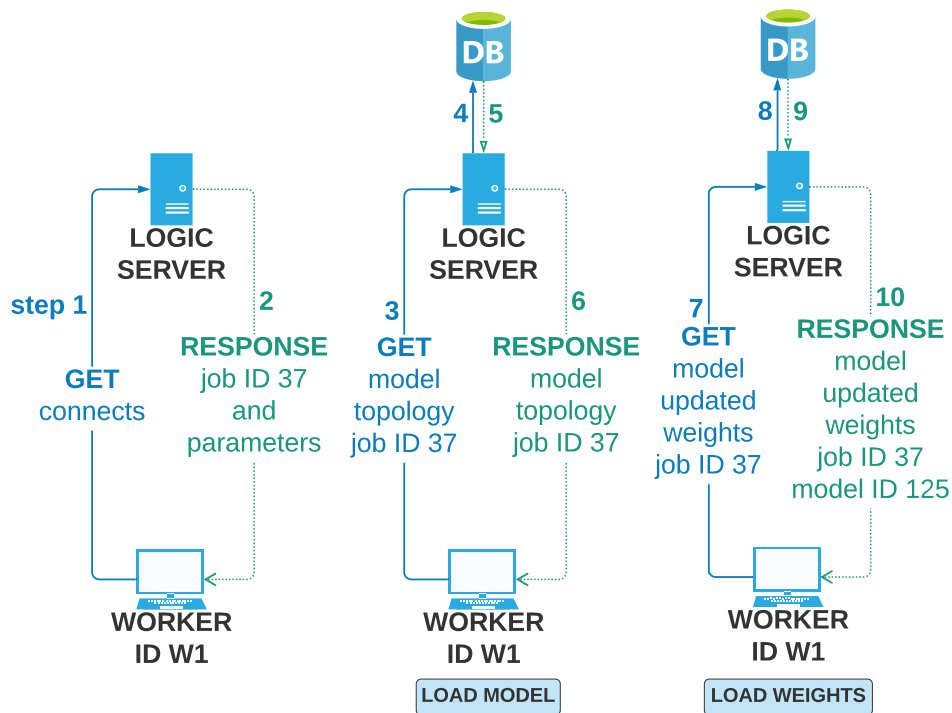


FIGURE 4.4: Initialisation execution flow. The worker obtains all the necessary information to start collaborating.

devices that can connect and disconnect anytime for many reasons, such as errors, connection problems, or because the user decides to stop collaborating.

- When connecting they download the *job* information associated with a job ID (see steps 1 and 2 in Fig. 4.4). Then, they download the NN model topology used in that job ID (see steps 3 to 6). Next, they download the job's current state, i.e. the updated weights (see steps 7 to 10), and replace the local model. The ID of the updated weights means the number of global aggregations performed. Now they are ready to collaborate.

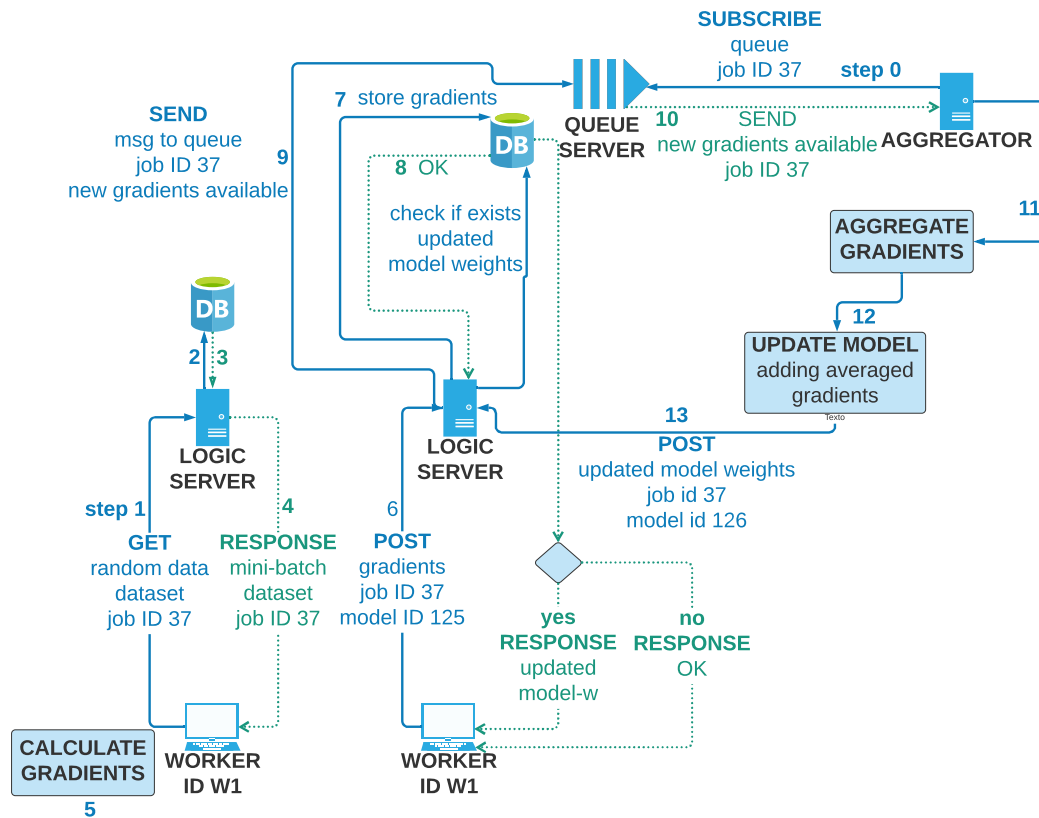


FIGURE 4.5: Execution flow of the collaborative learning process.

- Then, they repeat the following loop until a termination criterion is done or a disconnection occurs. The worker trains its local model for the local steps indicated in the job information (see steps 1 to 5 in Fig. 4.5), next send the local gradients/weights back to the server (see steps 6) and finally download the newly updated weights (if they exist).
3. *The logical server* is responsible for the orchestration and scheduling of jobs, and although it is usually located in the cloud, it can also be at the edge.
    - When receiving gradients/weights from workers (see step 6 in Fig. 4.5), it stores these intermediate results in the *DIMDB* (see steps 7 and 8). It then sends a message to the *queue server* to inform it that new results associated with a job ID are now available (see step 9).
    - Then, the *logical server* checks for updated weights with a newer weights ID than the worker weights have in the remote *DIMDB* before responding to the worker. If so, the *logical server* sends a buffer containing those weights to the worker. If not, it sends an empty buffer.
    - When the *logical server* stops receiving messages from a worker for  $S$  seconds, it considers that it has been disconnected and then deletes the information stored about it, i.e. its worker ID. Later, if using the adaptive aggregation algorithm, the aggregator will adapt  $P$  according to the number of connected workers. This technique allows learning to be robust and fault-tolerant.

4. The *distributed in-memory database (DIMDB)* is used by the *logical server* for storing intermediate results. It is located in the cloud.
5. The *aggregator* can be in the cloud or at the edge. It aggregate (averaging) gradients/weights calculated by the workers and updates the shared model, i.e. weights.
  - It is subscribed to a message queue of the queue server and waits for results associated with a specific job ID (see step 0 in Fig. 4.5). When receiving a message informing that new results are available (see step 10), it downloads them from the *DIMDB*. Then, it checks if it has the required number of gradients to perform the aggregation. If so, it performs aggregation (see step 11 and 12 in Fig. 4.5). Later, it sends the updated weights back to the *logical server* who stores them in the *DIMDB* (see step 13).
  - The aggregator needs to know the number of connected workers to aggregate the corresponding gradients/weights when using the adaptive aggregation algorithm (see Algorithm 4). With this goal, the logical server sends to the aggregator together with the intermediate results the *workers* IDs of whom have communicated with it in the last  $S$  seconds. The aggregator uses this information to adapt the parameter  $P$ .
  - Each worker trains asynchronously, sending processing results as soon as they finish without waiting for the slower workers. The aggregator discards the results that are older than a specified threshold  $Z$ .
6. The *queue server* is in the cloud and is used by the *logical server* to inform the *aggregator* that new results are available.

Participants interact through the *logical server* using REST API protocol. Workers can be running on web browsers (TensorFlow.js) or in a Python process (TensorFlow), maximising the number of collaborators while requesting a minimum effort (just a click). We believe that web browser computing is an appropriate direction for this type of volunteer computing, thanks to its ubiquity, sandboxing, and no need for software installation [82]. The aggregator runs in a Python process using the TensorFlow library. The logical server that provides access to the database run in a Java process. The *DIMDB* is implemented in Redis<sup>9</sup>, the *queue server* uses RabbitMQ<sup>10</sup> (AMQP protocol). The model topology is share using the HDF5 format<sup>11</sup>. The gradients/weights are share using the NPY file format<sup>12</sup>. All code is available in a public Git repository, and it is packaged in Docker containers to allow for reproducibility<sup>13</sup> (programmability and usability).

Finally, we conducted an exhaustive empirical analysis to evaluate the proposal. We analysed the main features of the platform mentioned above. We performed twenty experiments organised into four case studies and used from 1 to 64 workers. We use two types of workers: Python processes (TensorFlow) limited to one core and 2 GB RAM on HTCondor [111] as constrained devices and up to 24 desktops collaborating from web browsers (TensorFlow.js). We analyse how asynchrony, dynamic connections, and disconnections of devices affect learning and how we can

<sup>9</sup>[redis.io](https://redis.io)

<sup>10</sup>[www.rabbitmq.com](https://www.rabbitmq.com)

<sup>11</sup>[www.hdfgroup.org/solutions/hdf5/](https://www.hdfgroup.org/solutions/hdf5/)

<sup>12</sup>[numpy.org/devdocs/reference/generated/numpy.lib.format.html](https://numpy.org/devdocs/reference/generated/numpy.lib.format.html)

<sup>13</sup>[github.com/jsdoop/jsdoop](https://github.com/jsdoop/jsdoop)

adapt learning to keep high accuracy. The devices were connected and disconnected stochastically during learning following two exponential distributions [66]. Exponential distributions describe the time between events in a Poisson process, i.e. a process in which events occur continuously and independently at a constant average rate ( $\lambda$ ). We show that the proposed decentralised and adaptive system architecture for asynchronous learning allows volunteer users to yield their device resources and local data to train a shared ML model. Devices can join and leave the system at any time without stopping the learning process. This architecture does not need to know where or when the participating devices will connect. This system is fault-tolerant, so unexpected disconnections of workers do not interrupt the learning. The open-source implementation enables collaboration between devices with heterogeneous hardware and software. User devices can join the learning in different ways, such as from a web browser or running a Python process on the client device, allowing almost any device to participate. We organised the analysis into four case studies:

- **Case study 1:** We analyse the robustness and interoperability of the platform when using heterogeneous hardware and software. We demonstrate that the platform achieves high accuracy despite heterogeneity.
- **Case study 2:** We examine the most suitable setting when using asynchronous FEEL. We show that it is necessary to adapt the aggregation parameter when the number of workers varies to get high accuracy.
- **Case study 3:** We test the fault-tolerance of the software platform to variations in the number of collaborating devices. We also evaluate the self-adaptation of the algorithm to these changes. We prove that the platform is fault-tolerant, can continue its learning, and keep offering a high accuracy despite connections and disconnections of volunteer devices.
- **Case study 4:** We evaluate the self-adaptation and fault-tolerance of the platform using a more extreme *non-i.i.d* data. Also, we perform an analysis of the scalability and random drops. We confirm that when the system uses the adaptive algorithm and random drops, we can get similar results as when the system uses a constant number of workers during learning.

Table 4.7 and 4.8 show the results of the experiments. We show that the platform can adapt to changes and continue learning in a changing environment where volunteer devices connect and disconnect at any time. We test the system using different data distributions. Also, the platform enables interoperability and obtains high accuracy when using heterogeneous devices in hardware and software with different NN training libraries collaborating (i.e. web browsers using TensorFlow.js and Python processes using TensorFlow). Next, we release a modular open-source library publicly available in a Git repository that enables VC4FL in a decentralised, asynchronous, and fault-tolerant way. There is no need for the previous configuration of participating workers. Instead, the platform allows workers to join and leave at any time. That opens up an exciting avenue for research allowing researchers to experiment with new VC4FL techniques on real edge devices and real users easily.

Experimentation results are remarkable, showing that our adaptive approach is effective for VC4FL in very diverse scenarios, using unreliable dis/connecting volunteers, extreme *non-i.i.d* data and the challenges mentioned in section 3.3.

TABLE 4.7: Comparison of experiments in case studies 1 to 3.

Job	Workers	$P$	Min Loss	Max Acc.	CK score
<b>W64P64</b>	<b>64py</b>	<b>64</b>	<b>0.0016</b>	<b>0.9894</b>	<b>0.9878</b>
W64P32	64py	32	0.0017	0.9889	0.9867
mixW64P64	24js40py	64	0.0017	0.9887	0.9860
adaptive	dynamic	adaptive	0.0017	0.9884	0.9859
W32P32	32py	32	0.0018	0.9878	0.9842
W16P16	16py	16	0.0019	0.9876	0.9843
W16P32	16py	32	0.0020	0.9865	0.9846
W8P8	8py	8	0.0025	0.9844	0.9817
W4P4	4py	4	0.0029	0.9811	0.9770
W2P2	2py	2	0.0043	0.9741	0.9724
W1P1	1py	1	0.0055	0.9663	0.9626
W8P32	8py	32	0.0722	0.6706	0.5116

TABLE 4.8: Results of case study 4. Each worker has a different number of samples of each class and a maximum of five classes. The larger the number of workers, the better the accuracy.

Job	Workers	$P$	Min Loss	Max Acc.	CK score
<b>W64P64#5</b>	<b>64py</b>	<b>64</b>	<b>0.0020</b>	<b>0.9866</b>	<b>0.9833</b>
W16P16#5	16py	16	0.0022	0.9859	0.9840
W32P32#5	32py	32	0.0023	0.9856	0.9834
adaptive#5	dynamic	adaptive	0.0022	0.9854	0.9822
W8P8#5	8py	8	0.0035	0.9782	0.9728
W4P4#5	4py	4	0.0047	0.9701	0.9618
W2P2#5	2py	2	0.0088	0.9431	0.9228
W1P1#5	1py	1	0.0758	0.4776	0.2001

In summary, the main contributions of this work (**LIT** and **ASYN**) are:

1. We defined the challenges and desirable features of VC4FL (see Section 3.3). **G2.**
2. We proposed an algorithm for FEEL that adapts to asynchronous clients joining and leaving the computation when the number of workers is low and can even drop to zero during training. **G3.**
3. We proposed, implemented and evaluated a software platform for performing VC4FL that meets the defined challenges and desirable features. We evaluated our proposal via extensive experimentation in a static configuration and highly dynamic and changing scenarios. We then showed that the platform using a given number of constrained, unreliable, and heterogeneous (in HW and SW) devices adapts well to this changing environment getting a numerical accuracy and CK score similar to today's configurations that use a static platform for learning. Next, we demonstrated the fault-tolerance of the platform that recovers from unexpected disconnections of volunteer devices. **G3.**
4. We released a modular open-source library covering most VC4FL desirable features. **G4.**



## 5. Other main findings:

- (a) Modifying the adaptive parameter  $P$  according to the available workers  $|E|$  together with using a threshold  $Z$  for discarding old results is effective for getting good learning results in highly dynamic scenarios of VC4FL with a crowd of unreliable non-pre-configured workers. However, we need further research to analyse the best trade-off between both parameters and find the most suitable aggregation technique depending on the distribution of the local datasets of the workers.
- (b) If the noise generated by the previous techniques is not too big not only does not worsen the learning results but can improve them. Experiments show that the most accurate value for  $P$  lies between  $\frac{|E|}{2}$  and  $E \cdot 2$ . Therefore, the adaptive algorithm must always keep  $P$  within that range to obtain good learning results. **G3**.
- (c) If the number of connected workers is too low and does not change for too long, some classes' little or no data available limits learning. We should investigate new techniques to deal with this problem, such as decreasing the learning or even temporarily stopping training if this situation is prolonged. That is not the case if the connected workers shift or temporarily decrease and later increase again because, in this case, the model learns using different data, obtaining good learning results. **G3**.
- (d) This work is the first, to the authors' knowledge, on distributed NN training in which web browsers and python processes collaborate, then proving the interoperability [70, 82] of our proposal. We believe web browser computing is the right path forward for FL thanks to its sandboxing, ubiquity, and no software installation required [86]. **G2**.

#### 4.4 Proposal for Communication Overhead Reduction in FL

In the two previous sections, we have focused on the distributed learning process. However, we have not paid attention to some of the challenges associated with FL. In particular, one of the biggest problems of FL is a large amount of information (weights or gradients) that needs to be communicated in each iteration. Therefore, we focus in this work on reducing the amount of data sent during the learning process while maintaining and even improving the accuracy level of the model.

In this work [87], we address the goals **G2**, **G3**, and **G4**, and sub-goals 1-e, 2-a, 4-f, 5-a and 5-b. The contributions of this work are related to **LIT** and **MOD** (see Section 1.2.2). Achieving high-quality results in FL (see Section 3.1) requires a large amount of communication where information is exchanged between the edge devices and the server [75]. Finding the minimum amount of communication that achieves the same or higher accuracy is a multi-objective problem where we want to reduce communications and increase the model's accuracy. For such a problem's class, stochastic algorithms such as metaheuristics and, in particular, the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) are a promising alternative that provides a good trade-off between finding an optimal solution and the time required [32]. In this research, (I) we model and formulate the Federated Learning Communication Overhead Problem as a multi-objective Problem (FL-COP), (II) and we apply the NSGA-II to solve it. Other authors have proposed a multi-objective approach in which they evolve the neural network architecture [126]. We instead

investigate the main parameters triggering communication overhead that the literature usually tackles separately within the same work. This work is our first step in this direction. In future work, we plan to research adaptive algorithms that adjust these parameters during learning. Our proposal has been assessed by simulating a server/client architecture of 4 devices, tested with both convolutional and fully connected neural networks with 12 and 3 layers, 887,530 and 33,400 weights, respectively. The validation has been done on MNIST dataset containing 70,000 images of handwritten digits.

Our formulation of the FL-COP is a bi-objective optimisation problem, where the two conflictual objectives consist of (I) minimising the communication overhead while (II) maximising the model's accuracy. It also assumed that each client in the architecture has a similar model (i.e. nodes, connections, layers, activation functions, etc.) as the one on the server. When mentioning the *local* and *global* models, we refer to the client's and server's models, respectively. Let us assume an architecture of one server connected to  $N$  clients.

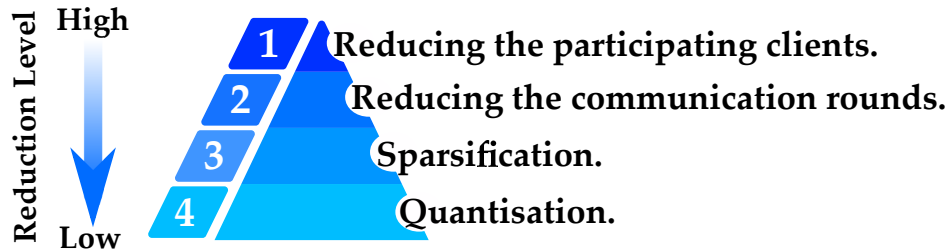


FIGURE 4.6: The FL-COP modelling levels.

We refer to the model as a tuple  $L = (l_1, \dots, l_{|L|})$  containing vectors of weights, where vector  $l_i$  ( $i = 1, \dots, |L|$ ) represents the weights of connections between the layer  $i$  and  $i - 1$  of the NN. We can find different techniques to reduce the communication overhead problem in bibliographic. We have ordered some of these techniques from a high to a low level. The FL-COP modelling is thought as a 4-levels communication-reduction scheme (see Figure 4.6), where each layer represents when a given communication-reduction approach is applied. At the highest level, we identify the number of clients that will participate in training the global model. The second level reduces the number of communication rounds by incrementing the number of local steps performed in each device. The third and fourth levels refer to the sparsification and quantisation performed before sending the local weights back to the server to be aggregated.

The overall amount of communications happening during the learning process is proportional to the number of clients  $m \in [1, N]$  that participate in training the global model. So, the first part of the FL-COP modelling stands in finding the number  $m$  of clients, selected randomly among the complete set  $S$ , and which will be the only ones sending their local models to the server at iteration  $t$ . A second part of the FL-COP modelling consists in finding the number of training iterations  $E \in [1, 1000]$  after which all the clients send their local models to the server. This variable determines the number of training steps that the clients perform before sending their local models (e.g. weights, gradients, etc.). It is important to note that for each client, the maximum number of training iterations allowed on overall is  $(E \cdot T)$ , where  $T$  is the maximum number of times the clients can send their local models to the server. The third part of the problem modelling consists in selecting, for each layer  $L_i$  having  $n_i$  weights, a percentage  $\mu \in [0\%, 50\%]$  of the weights that will not be sent to the server.

Using the classical FedAvg, the weights are encoded with full precision (i.e. all their decimals) using 32 bits. Thus, the fourth, and final part of our FL-COP modelling consists in finding the optimal number of bits  $b_i$  allocated to encode the weights of each layer  $L_i$  in the model, where  $i = 1, \dots, l$ . We also assume that  $\omega_i$  and  $q_i$  represent, respectively, the maximum and minimum values of the weights in the  $i^{\text{th}}$  layer. Having  $b_i$  bits means that  $2^{b_i}$  binary combinations can be created. We assign the all-ones and all-zeros combinations to encode the  $\omega_i$  and  $q_i$  values, respectively. The  $(2^{b_i} - 2)$  remaining combinations will encode  $(2^{b_i} - 2)$  values that are equally drawn from the interval  $[\omega_i, q_i]$ . Technically, the data that will be sent to the server will be the series of combinations that encodes each weight, as well as  $\omega_i$  and  $q_i$ . The server will perform the reverse mechanism to retrieve the full-precision weights. Each client will send to the server  $\sum_{i=1}^l (n_i \cdot b_i) + 64$  bits instead of  $\Theta = \sum_{i=1}^l (n_i \cdot 32)$  original bits.

Our formulation of the FL-COP is described using Equations (4.2)-(4.5). The first objective function  $f_1(\vec{X})$  defined by Equation (4.2) calculates the percentage of data reduction that the solution  $\vec{X}$  achieves. Concretely, it is the sum of the percentage  $\alpha$  and  $\beta \in [0, 1]$  of data sent and received, respectively, by all the clients together from and to the server. These percentages are expressed with regard to the original data that would have been sent or received when no communication reduction is applied ( $T \cdot N \cdot \Theta$ ). The second objective function  $f_2(\vec{X})$  defined by Equation (4.3) evaluates the accuracy of the global model  $w_T^*$  at communication  $T$  (i.e. the last iteration) achieved via the solution  $\vec{X}$ . The server's model  $w_T^* = \sum_{k=0}^m w_T^k / m$  is computed as the mean of the  $m$  local models obtained after  $T$  communications, while the accuracy is computed as the division of  $\lambda$  by  $\nu$ , where  $\lambda$  and  $\nu$  are the number of correct and total predictions made using the model  $w_T^*$ , respectively.

$$\underset{\vec{X}=\{x_1, \dots, x_d\}}{\text{Min}} \quad f_1(\vec{X}) = \frac{\alpha + \beta}{2} \quad (4.2)$$

$$\underset{\vec{X}=\{x_1, \dots, x_d\}}{\text{Max}} \quad f_2(\vec{X}) = \frac{\lambda}{\nu} \quad (4.3)$$

Where:

$$\alpha = \frac{1}{E} \cdot \frac{m}{N} \quad (4.4)$$

$$\beta = \frac{m}{N} \cdot \frac{1}{E} \cdot \sum_{i=1}^l \frac{b_i}{32} \cdot \frac{100 - \mu_i}{100} \cdot \frac{n_i}{\sum_{j=1}^l n_j} \quad (4.5)$$

Subject to:

$$m, E, \mu_i, b_i \in \mathcal{N}, 1 \leq m \leq N, 1 \leq E \leq 1000, 0 \leq \mu_i \leq 50, 1 \leq b_i \leq 32$$



FIGURE 4.7: A 3-layers model: (a) abstract and (b) concrete FL-COP solutions.

The Fig. 4.7(a) sketches a typical solution  $\vec{X}$  of an FL-COP that trains a  $l = 3$  layers model. On the other hand, Fig. 4.7(b) represents a concrete solution  $\vec{X}$  for the same configuration using 20 training iterations, 2 rounds of client-server communications. During each round, only 90% of the weights of the 1<sup>st</sup> layer, 55% of the 2<sup>nd</sup>, and 98% from the 3<sup>rd</sup> are sent to the server. The weights sent from the 1<sup>st</sup>, 2<sup>nd</sup> and 3<sup>rd</sup> layers are encoded using 2, 20 and 15 bits, respectively. The code is publicly available on Github<sup>14</sup>. Results have been confirmed using a Wilcoxon test with Bonferroni correction and a significance level of 0.025.

The contributions of this work are:

1. We summarised the most common techniques in literature for solving the communication overhead problem in FL (see Section 3.3). **G2**.
2. We formulated and modelled Federated Learning Communication Overhead Problem as a multi-objective Problem (FL-COP). **G3**.
3. We proposed solving the FL-COP by using genetic algorithms for multi-objective function optimisation. Our proposal achieves higher accuracy while reducing communications from 10 to 2000 depending on the neural network topology compared to the maximum communication setting. **G3**.
4. Our proposal provides a population of solutions that facilitates decision-making when configuring the parameters of FL. **G3**.

---

<sup>14</sup>Code available in <https://github.com/NEO-Research-Group/flcop>

## Chapter 5

# Conclusions and Future Work

This thesis has addressed the challenge of distributed ubiquitous intelligence in edge devices. Specifically, we proposed to see FEEL as a VC type where users donate their edge devices' computing resources to a project that trains a shared DL model (VC4FL). We dealt with constrained, unreliable, and heterogeneous (in HW and SW) devices and adapted the learning to the volatility of dis/connections getting fault tolerance. We also have formulated and modelled the problem of communication overhead in FL, which is an important challenge for ubiquitously distributed intelligence and efficiently learning from users' data, as a multi-objective problem. And we tackled it using genetic algorithms for multi-objective function optimisation achieving higher model accuracy while reducing communications compared to the maximum communication setting.

We have divided the contributions of this thesis into four main contributions (**LIT**, **SYN**, **ASYN** and **MOD**) to meet the four objectives we specified (**G1-G4**) (see Section 1.2.1 and 1.2.2 for more details about goals and contributions). In all our works we pursued the goal **G4** of dissemination of the results. Following this goal, the software implemented in works 2 and 3 is publicly available on Github<sup>1</sup>. Also, the software implemented in our fourth work<sup>2</sup>.

In our first work [85] (see Section 4.1), we pursued our first goal **G1** of analysing heterogeneous hardware and software of edge devices. The main contribution of this work is **LIT**. We explained that traditional benchmarks are unreliable for assessing edge device performance, but they help gain a general understanding. We showed that the RP3 platform is suited for running algorithms on edge and that Android devices (smartphones and tablets) have problems running high-performance applications. This operating system appears more concerned with memory efficiency and low battery consumption (often running the trash collector) than performance. We observed that using native code in Android allowed us to avoid the issue mentioned above. Additionally, we demonstrated that using native code and compiling it into WebAssembly (WASM) with Emscripten results in a good performance on most devices. WASM is a portable binary code that can run in modern web browsers. We confirmed the effectiveness of executing these algorithms on the web browser using WASM, which also has the benefits of being multiplatform and having a lightweight virtualisation property.

In our second work [86] (see Section 3.2 and 4.2), we focus on goals **G2** and **G3**. The main contributions of this work are **LIT** and **SYN**. We reviewed the literature on using the web browser as a platform for volunteer computing (BBVC) and discussed recent advancements that are making this more and more viable (see Section 3.2). We presented a BBVC framework for distributed volunteer computing utilising the message queue pattern and the MapReduce programming paradigm via web browsers

<sup>1</sup>Code available in <https://github.com/jsdoop/>

<sup>2</sup>Code available in <https://github.com/NEO-Research-Group/flcop>

without interfering with the site’s user experience or adding extra software installation. The topic of data privacy at the edge is not yet covered in this study. Each task indicates which age of the model and which portion of the data to use. It allows us reproducibility of the training regardless of the number of workers. Despite the communication channel’s constraints, we showed our approach’s viability and scalability.

In our third work [84] (see Section 3.3 and 4.3), we focus on goals **G2** and **G3**. The main contributions of this work are **LIT** and **ASYN**. We outlined the challenges and desirable features of VC4FL (see Section 3.3). We addressed the problem of data remaining locally on edge devices. Due to the volatile connections and disconnections of the volunteers, the dataset available during training changes dealing with unbalanced and *non-i.i.d.* datasets. We proposed an asynchronous algorithm for FEEL, which adapts the training to unreliable clients joining and departing the computation when there are few workers—or even drop to none—available. A software platform that satisfies the specified challenges and desirable features was designed, put into practice, and evaluated. We then showed that the platform using a given number of constrained, unreliable, and heterogeneous (in HW and SW) devices adapts well to this changing environment getting a numerical accuracy and CK score similar to today’s configurations that use a static platform for learning. We showed how the platform recovers from unexpected device disconnections from volunteers. Also, we showed some interesting findings about the parameters  $P$  and  $Z$  setting when the number of available workers  $|E|$  varies. However, we must investigate new techniques to deal with different casuistries in data distribution and connections and disconnections of volunteers. To the best of the authors’ knowledge, this work is the first on distributed neural network training in which web browsers and python processes collaborate, proving the interoperability [70, 82] of our proposal.

In our fourth work [87] (see Section 3.3 and 4.4), we focus on goals **G2** and **G3**. The main contributions of this work are **LIT** and **MOD**. One of FL’s biggest challenges is communications overload. That is why we tried to address this problem in this work, i.e. reducing the communication overhead without reducing or even increasing the accuracy of the trained model. We summarised the most common techniques in literature for solving the communication overhead problem in FL (see Section 3.3). We formulated and modelled Federated Learning Communication Overhead Problem as a multi-objective Problem (FL-COP). We suggested employing EAs for multi-objective function optimisation to solve the FL-COP. Our method improves accuracy compared to the maximum communication setting while reducing communications from 10 to 2000 depending on the neural network architecture. To the authors’ knowledge, this work is the first in which this problem is formulated and modelled by combining all these approaches within the same study proposing multi-objective genetic algorithms and probing their suitability for this purpose. It opens the door to a wide range of future work, such as designing algorithms that dynamically adapt these hyperparameters during training.

Finally, the results of this thesis prove that the deployment of ubiquitously distributed intelligence at edge devices is feasible and valuable, also providing a better understanding of how such ubiquitous distributed intelligence should be, the associated problems that exist, and how these problems must be addressed.

The work on this thesis has been arduous and laborious. During the development of this thesis, we have used a wide variety of programming languages and technologies that have taken time to learn. Several algorithms and problems have

been implemented on real distributed edge devices with limited resources not designed for this purpose. We have done a lot of research into the most appropriate technologies to make these devices work together and achieve remarkable results. In the end, this work has borne fruit with **two publications in JCR-indexed journals (Q1)** with a high impact factor. Moreover, two more publications at conferences, **one at a national conference** and **another at an international conference** in which the PhD student also obtained the **outstanding student award**.

However, many challenges and questions remain for future work. Based on these results, many research perspectives that we intend to explore in future work came to light. Some of these research lines can be summarised as follows:

- Research in more depth the task assignment management based on the performance of heterogeneous devices and the distribution of the data used by each device.
  - Design intelligent selection algorithms for selecting the devices participating in the following communication round depending on their data distribution and performance.
  - Design optimisation algorithms to set a customised number of local updates to each device based on its performance and probability of failure.
  - Design adaptive aggregation algorithms that aggregate models from heterogeneous devices depending on the number of local steps performed and the distribution of the data used by each device.
- Further investigate the reduction of ML model size before, during and after training without reducing the accuracy.
  - Further investigate the real-time optimisation of federated learning parameters during the learning process.
  - Optimise federated learning parameters by adding new objectives, such as energy consumption, comparing a more significant number of multi-objective algorithms, and proposing new operators.
  - Design new adaptive algorithms that optimise communication overhead in FL during learning, e.g. by using quantisation and sparsification techniques.
- Design a multi-level learning platform with several learning layers, i.e. edge, fog and cloud.
- Investigate new distributed optimisation techniques using users' local and private data for complex problems.
- Investigate new FL techniques integrated with reinforcement learning with applications [94] such as resource allocation, communication networks, control optimisation, and attack detection, among others.

This page is intentionally left blank.



## Appendix A

# Data Sets, Problems, and Solvers Used in This Thesis

This appendix briefly introduces the problems, datasets and solvers used in this doctoral thesis.

### A.1 MNIST Dataset

The MNIST [68]<sup>1</sup> database of handwritten digits has a training set of 60,000 examples and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been normalised in size and centred on a fixed-size image.

### A.2 OneMax Problem

The *OneMax* problem [37], also known as *BitCounting* consist in maximising the number of ones of a string of bits. It is defined formally as finding a string  $\vec{x} = \{x_1, x_2, \dots, x_N\}$ , where  $x_i \in \{0, 1\}$ , that maximises the following equation:

$$f_{OneMax}(\vec{x}) = \sum_{i=1}^N x_i \quad (\text{A.1})$$

### A.3 Minimum Tardy Task Problem (MTTP)

The *MTTP* [107] is an NP-hard task-scheduling problem wherein each task  $i$  from the set of tasks  $T = 1, 2, \dots, n$  has a time length for execution  $l_i$ , a deadline before which has to be scheduled  $d_i$ , and a penalty  $w_i$ , where  $l_i, d_i, w_i \in \mathbb{N}$ . If the task remains unscheduled, the penalty  $w_i$  is added to the objective function. Scheduling a subset of tasks  $S$  of  $T$  is to find the start time, ensuring that at most one of the tasks is performed at a time and each one finishes before its deadline. The objective function of this problem is to minimise the sum of the weights of the unscheduled tasks.

$$f_{MTTP}(\vec{x}) = \sum_{i \in T-S} w_i \quad (\text{A.2})$$

### A.4 Massively Multimodal Deceptive Problem (MMDP)

In the *MMDP* [46], bipolar deceptive functions with two global optima and with several deceptive optima are designed. In MMDP, each subproblem  $s_i$  contributes

<sup>1</sup>[yann.lecun.com/exdb/mnist](http://yann.lecun.com/exdb/mnist)

to the total fitness value according to the number of ones it has. The local optima number is considerable ( $n^k$ ), while only  $2^k$  global optima solutions are available. The  $k$  parameter regulates the degree of multimodality.

$$f_{MMDP(s)} = \sum_{i=1}^k fitness_{s_i} \quad (A.3)$$

## A.5 Error Correcting Code Design (ECC)

In the ECC problem [74] messages are sent over noisy channels and we have to assign codewords to an alphabet that minimises the length of sent messages while maximising correction of single uncorrelated bit errors. A code is represented using a three-tuple  $(n, M, d)$ , where  $n$  is the length of a codeword,  $M$  defines the amount of codewords, and  $d$  represents the minimum Hamming distance between every pair of codewords. The goal is to construct a code of  $M$  binary codewords, each of length  $n$ , such that  $d$  is maximised.

$$f_{ECC} = \frac{1}{\sum_{i=1}^M \sum_{j=1, i \neq j}^M \frac{1}{d_{ij}^2}} \quad (A.4)$$

## A.6 Capacitated Vehicle Routing Problem (CVRP)

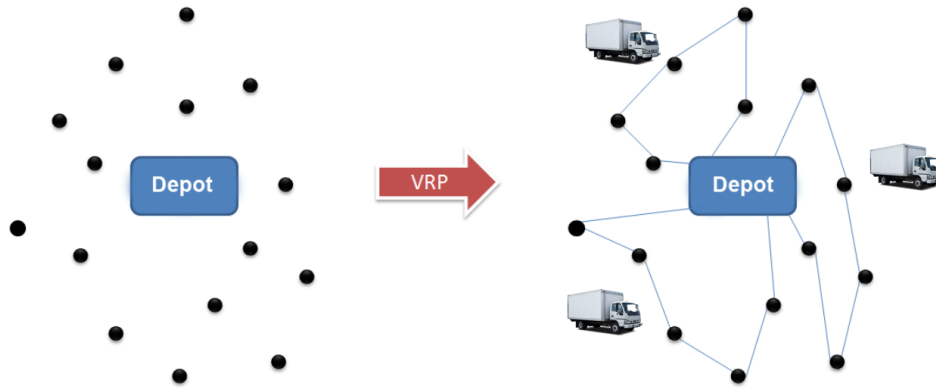


FIGURE A.1: The Vehicle Routing Problem (VRP).

In the CVRP [29] (see Figure A.1) a constant fleet of distribution vehicles of identical capacity serve the expected client demands for a given product from a shared depot while minimising total transport costs. We define CVRP using an undirected graph  $G = (V, E)$ .  $V = \{v_0, v_1, \dots, v_n\}$  is a vertex set, where  $v_0$  is the depot and  $v_1, \dots, v_n$  is a set of destinations. The  $m$  identical vehicles of capacity  $Q$  start their trips in the same shared depot  $v_0$  and must serve all the clients  $v_1, \dots, v_n$ .  $E = \{(v_i, v_j) | v_i, v_j \in V, i < j\}$  is an edge set representing non-negative costs (i.e. distances) between clients. We also have a matrix  $C = (c_{i,j})$  where we store the non-negative cost between every customer  $v_i$  and  $v_j$  defined on  $E$ . A solution of the CVRP is a split  $R_1, R_2, \dots, R_m$  of  $V$  that represents each route of each identical vehicle. The fitness function is calculated as the sum of the costs of each route  $R_i = \{v_0, v_1, \dots, v_{k+1}\}$  where  $v_j \in V, v_0 = v_{k+1}$  is the depot, and satisfying  $\forall R_i, \sum_{j=1}^k q_j \leq Q$ . We represent the cost function as:

$$Cost = \sum_{i=1}^m Cost(R_i) = \sum_{j=0}^k c_{j,j+1} \tag{A.5}$$

### A.7 CVRP Solver

The algorithm used to solve the CVRP was implemented similarly to that used by Dorronsoro and Alba [2].

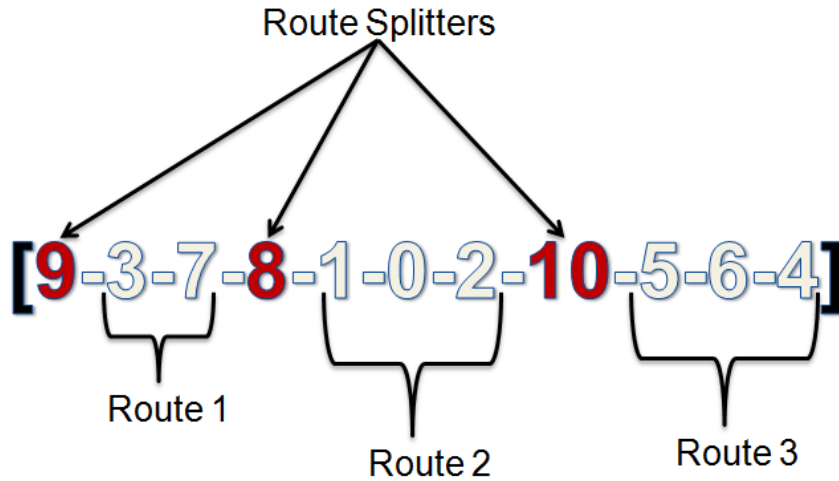


FIGURE A.2: A representation of a solution for 8 cities and 4 vehicles.

In the CVRP, we represent a solution as a vector that contains a permutation of customers and route splitters (see Figure A.2). Two contiguous route splitters represent an empty route. The vector’s beginning and end act as a virtual route splitter.

As a selection method, we use binary tournament selection. It consists of randomly selecting two individuals and choosing the best one between them. This process is repeated once again, obtaining the second parent.

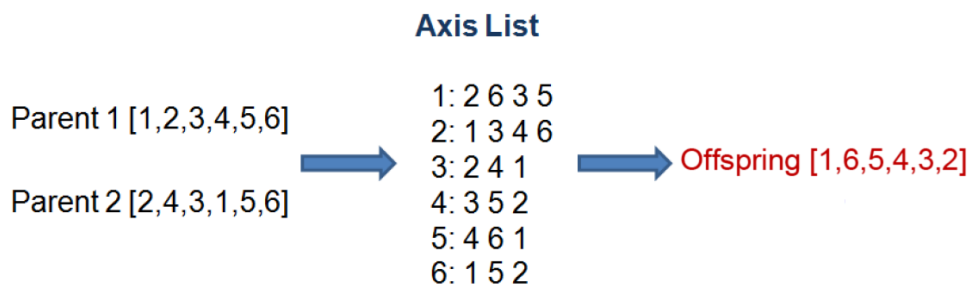


FIGURE A.3: Edge Recombination Crossover (ERX).

As a recombination method, we use the Edge Recombination Crossover (ERX) (Fig. A.3). It is a crossover technique for permutation chromosomes. It strives to introduce the fewest paths possible while preserving the links between customers.

As mutation method, three mutation operators are used with an equal probability to operate on every gene (see Figure A.4): Insertion [41], Swap [8], and Inversion [55]. All of them have proven to be useful for this problem, and the reason for using all three is to cover more search space with different types of changes in the chromosome.

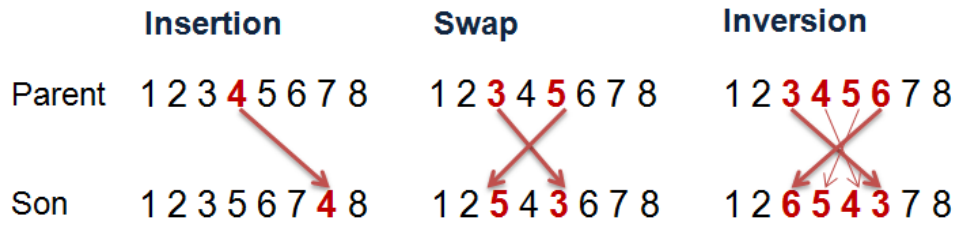


FIGURE A.4: Mutation operators are applied with equal probability.

Finally, the local search technique consists in applying 2-Opt [25], and 1-Interchange [91] to every individual.

Next, we show the function for fitness computation of a candidate solution  $S$ :

$$f(\vec{S}) = F_{CVRP}(\vec{S}) + \lambda \cdot \text{overcap}(\vec{S}) + \mu \cdot \text{overtm}(\vec{S}) \quad (\text{A.6})$$

Overcap( $S$ ) means the overcapacity of the truck, and overtm( $S$ ) means the overtime of the truck. Both are multiplied by constant values  $\lambda = \mu = 1000$ . Thus, the candidate solutions that do not meet the restrictions are penalised. It is a minimisation problem. Therefore, candidate solutions with lower fitness values will have a better chance of surviving.

## Appendix B

# Extending the Work of Analysing Edge Devices

This appendix is an extension of the work [85] presented in Section 4.1. We perform a comparative analysis of three common programming languages for edge devices: Java, C++, and WASM running GAs on different types of edge computing devices. The devices, problems and algorithms used are the same as the mentioned work. We address the objective **G1**, **G2**, and **G4** and the sub-objectives 1-a, 1-b, 1-c, 5-a and 5-b. The contributions of this work are **LIT**.

### B.1 Choosing Programming Languages for Edge Computing

In addition to comparing the performance of algorithms on edge devices, we also want to compare the programming languages best suited to implement algorithms for these devices. Based on the 2019 IoT Developer Survey<sup>1,2</sup>, the most widely used programming languages for IoT constrained devices are Java, C, C++ and JavaScript. Among these possibilities we chose Java, C++ and WebAssembly (WASM) with JavaScript. Java because it allows us to natively implement these algorithms in Android applications, which is the most common operating system on mobile devices. We selected C++ because of its well-known outstanding good performance and low memory consumption in computationally intensive tasks. Finally, we decided WASM because it allows us to compile the algorithms implemented in C++ code to portable binary code that can be executed in web browsers and JavaScript code to show the results and the user interface logic. Finally, despite its rising popularity, we discarded Python because when these experiments were carried out (2019), there were no stable frameworks for developing Android applications in Python, nor were they widely used in IoT devices.

### B.2 Evaluation of Edge Devices and Programming Languages Running Genetic Algorithms

We evaluate the performance of heterogeneous edge devices by solving problem instances using different programming languages. We compare these results with the benchmarks we have previously performed. We do not use multithreading, all devices use the same seeds and the same number of evaluations. Therefore, we use runtime as a metric to compare the results. We implement all the algorithms to solve

<sup>1</sup><https://outreach.eclipse.foundation/download-the-eclipse-iot-developer-survey-results>

<sup>2</sup><https://fossbytes.com/top-programming-languages-for-iot-development-in-2019/>

the problem instances explained in Section 4.1 in Java, C++ and WASM and compare their performance using the heterogeneous edge devices: *lap*, *m10*, *m4*, *tab*, *RP3*.

TABLE B.1: OneMax runtime and normalised score.

Device	Language	Time (s)			Normalised Score
		avg $\pm$ sd	max	min	
<b>lap</b>	<b>C++</b>	<b>0.49 <math>\pm</math> 0.05</b>	<b>0.62</b>	<b>0.40</b>	<b>1.00</b>
<b>lap</b>	<b>Java</b>	<b>1.34 <math>\pm</math> 0.12</b>	<b>1.59</b>	<b>1.15</b>	<b>2.75</b>
<b>lap</b>	<b>WASM</b>	<b>1.48 <math>\pm</math> 0.14</b>	<b>1.82</b>	<b>1.25</b>	<b>3.02</b>
rp3	C++	6.48 $\pm$ 0.50	7.87	5.73	13.28
m4	WASM	7.87 $\pm$ 0.71	9.95	6.92	16.06
tab	WASM	12.91 $\pm$ 1.09	15.80	11.38	26.35
m10	WASM	15.39 $\pm$ 1.30	18.98	13.59	31.41
rp3	WASM	16.62 $\pm$ 1.37	20.53	14.59	33.92
rp3	Java	21.58 $\pm$ 1.70	26.09	18.84	44.19
m4	C++	25.93 $\pm$ 2.15	31.85	22.94	53.09
tab	C++	36.96 $\pm$ 3.12	45.52	32.58	75.67
m10	C++	89.46 $\pm$ 17.58	122.09	53.20	183.16
m4	Java	106.36 $\pm$ 8.31	129.09	94.60	217.76
m10	Java	153.67 $\pm$ 30.95	203.74	91.02	314.61
tab	Java	177.40 $\pm$ 14.69	217.36	156.61	363.19

TABLE B.2: MTTP runtime and normalised score.

Device	Language	Time (s)			Normalised Score
		avg $\pm$ sd	max	min	
<b>lap</b>	<b>C++</b>	<b>0.42 <math>\pm</math> 0.45</b>	<b>2.02</b>	<b>0.04</b>	<b>1.00</b>
<b>lap</b>	<b>Java</b>	<b>0.51 <math>\pm</math> 0.39</b>	<b>1.81</b>	<b>0.14</b>	<b>1.21</b>
<b>lap</b>	<b>WASM</b>	<b>0.70 <math>\pm</math> 0.75</b>	<b>3.21</b>	<b>0.10</b>	<b>1.67</b>
rp3	C++	2.98 $\pm$ 3.24	13.63	0.27	7.12
tab	WASM	4.44 $\pm$ 4.77	19.50	0.42	10.57
m4	WASM	4.70 $\pm$ 4.99	20.71	0.46	11.19
m10	WASM	5.33 $\pm$ 5.71	23.26	0.49	12.69
m4	C++	5.81 $\pm$ 6.33	26.46	0.53	13.88
tab	C++	6.43 $\pm$ 7.00	29.34	0.59	15.35
rp3	Java	6.60 $\pm$ 6.99	28.27	0.68	15.77
rp3	WASM	8.03 $\pm$ 8.71	35.76	0.73	19.12
m10	C++	18.06 $\pm$ 23.37	104.30	0.82	43.13
tab	Java	88.50 $\pm$ 95.35	389.25	8.07	211.38
m10	Java	89.38 $\pm$ 96.47	392.76	8.36	213.49
m4	Java	94.31 $\pm$ 101.64	412.38	8.67	225.25

In Tables B.1, B.2, B.3, B.4, B.5 we can see the results of the experiments per problem. The first table shows the result for the OneMax problem, the second for MTTP, the third for MMDP, the fourth for ECC and the last one for CVRP. Each row has a different colour to visualise the different programming languages easily. The row with the darkest colour is C++, the medium colour is Java, and the lightest colour is WASM. We normalised times getting a score. The device with the shorter time in each problem gets a score of 1; the others get a score proportional to this incrementally. The device that has obtained the best score in each programming language is shown in bold. Figure B.1 shows these results using a heatmap per problem. In this case, the darkest colour is the worst score, and the lightest colour is a score of 1 (the less score, the better).

TABLE B.3: MMDP runtime and normalised score.

Device	Language	Time (s)			Normalised Score
		avg $\pm$ sd	max	min	
lap	Java	<b>1.27 <math>\pm</math> 0.12</b>	<b>1.74</b>	<b>1.07</b>	<b>1.00</b>
lap	C++	<b>1.51 <math>\pm</math> 0.17</b>	<b>2.08</b>	<b>1.26</b>	<b>1.19</b>
lap	WASM	<b>1.78 <math>\pm</math> 0.22</b>	<b>2.61</b>	<b>1.52</b>	<b>1.40</b>
tab	WASM	8.27 $\pm$ 1.08	12.85	6.80	6.51
rp3	C++	9.21 $\pm$ 0.94	12.69	7.67	7.24
m4	WASM	10.23 $\pm$ 1.24	14.85	8.30	8.06
m10	WASM	13.79 $\pm$ 1.46	19.52	11.55	10.86
rp3	Java	16.36 $\pm$ 1.74	23.30	13.83	12.87
rp3	WASM	19.16 $\pm$ 2.02	27.03	16.04	15.09
m4	C++	27.08 $\pm$ 2.39	35.01	22.99	21.30
tab	C++	30.17 $\pm$ 2.59	38.60	25.65	23.73
m10	C++	33.17 $\pm$ 5.21	50.43	26.54	26.09
tab	Java	394.35 $\pm$ 34.97	510.81	333.09	310.25
m4	Java	437.65 $\pm$ 37.94	560.09	363.95	344.31
m10	Java	479.27 $\pm$ 56.65	639.43	400.50	377.06

TABLE B.4: ECC runtime and normalised score.

Device	Language	Time (s)			Normalised Score
		avg $\pm$ sd	max	min	
lap	C++	<b>1.81 <math>\pm</math> 0.25</b>	<b>2.38</b>	<b>1.17</b>	<b>1.00</b>
lap	Java	<b>1.83 <math>\pm</math> 0.25</b>	<b>2.43</b>	<b>1.22</b>	<b>1.01</b>
lap	WASM	<b>2.91 <math>\pm</math> 0.53</b>	<b>4.36</b>	<b>1.60</b>	<b>1.61</b>
rp3	C++	12.72 $\pm$ 1.71	16.77	8.26	7.03
m4	WASM	12.82 $\pm$ 1.71	16.85	8.22	7.08
tab	WASM	14.03 $\pm$ 2.07	18.98	8.77	7.75
m10	WASM	19.93 $\pm$ 2.70	26.25	12.89	11.01
rp3	Java	25.61 $\pm$ 3.58	34.03	16.45	14.17
rp3	WASM	26.94 $\pm$ 3.69	35.76	17.49	14.88
m4	C++	33.11 $\pm$ 4.13	42.85	22.29	18.32
tab	C++	38.92 $\pm$ 4.96	50.53	25.92	21.53
m10	C++	72.35 $\pm$ 32.32	139.64	32.43	40.03
m10	Java	460.69 $\pm$ 136.12	693.89	248.15	254.90
tab	Java	536.56 $\pm$ 70.54	702.96	353.10	296.88
m4	Java	572.27 $\pm$ 73.79	746.86	379.29	316.64

We can see that the laptop gets the best score for all programming languages. After it, *RP3* using C++ gets the best score in all problems except in the MMDP where *tab* using WASM gets a better result. On the one hand, Java is clearly the language with which devices get the worst score. On the other hand, it is interesting that the two mobiles and the tablet score better when they use WASM than when they use C++ or Java. However, when we observe the laptop and *RP3*, they get a different order of programming languages: C++, Java and WASM.

When we visualise Figure B.1, we can see something interesting. The difference between the best and worst devices in each programming language is more homogeneous in WASM and C++ than in Java. Java results are very heterogeneous because when mobile devices and tablet use Java, they get a poor score. It is clear that something is not working well in optimising the Java VM on Android. Android does not use Oracle Java VM but uses Android Runtime (ART) (replaced by Dalvik Virtual

TABLE B.5: CVRP runtime and normalised score.

Device	Language	Time (s)			Normalised Score
		avg $\pm$ sd	max	min	
lap	Java	<b>6.76 <math>\pm</math> 1.71</b>	<b>11.64</b>	<b>4.59</b>	<b>1.00</b>
lap	C++	<b>11.49 <math>\pm</math> 3.07</b>	<b>19.52</b>	<b>7.66</b>	<b>1.70</b>
lap	WASM	<b>19.56 <math>\pm</math> 4.80</b>	<b>32.07</b>	<b>12.97</b>	<b>2.89</b>
rp3	C++	77.72 $\pm$ 20.60	131.91	52.08	11.49
rp3	Java	81.60 $\pm$ 21.74	139.54	54.29	12.07
tab	WASM	92.07 $\pm$ 24.13	155.62	61.08	13.62
m4	WASM	109.04 $\pm$ 28.71	183.74	73.95	16.13
m4	C++	111.09 $\pm$ 29.61	189.99	73.75	16.43
m10	WASM	145.35 $\pm$ 38.21	245.17	98.01	21.5
m10	C++	148.57 $\pm$ 40.51	251.20	84.62	21.97
tab	C++	149.05 $\pm$ 39.80	255.41	98.70	22.04
rp3	WASM	244.27 $\pm$ 63.81	409.58	163.26	36.13
tab	Java	2532.09 $\pm$ 667.46	4296.34	1681.83	374.43
m4	Java	2568.01 $\pm$ 647.58	4288.11	1689.50	379.74
m10	Java	2695.03 $\pm$ 721.57	4080.26	1615.00	398.53

TABLE B.6: C++ normalised score.

Problem	Device				
	lap	rp3	m4	tab	m10
onemax	<b>1.00</b>	13.28	53.09	75.67	183.16
mttp	<b>1.00</b>	7.12	13.88	15.35	43.13
mmdp	<b>1.00</b>	6.08	17.88	19.92	21.91
ecc	<b>1.00</b>	7.03	18.32	21.53	40.03
cvrp	<b>1.00</b>	6.76	9.67	12.97	12.93

TABLE B.7: Java normalised score.

Problem	Device				
	lap	rp3	m4	tab	m10
onemax	<b>1.00</b>	16.06	79.16	132.02	114.36
mttp	<b>1.00</b>	12.98	185.45	174.03	175.76
mmdp	<b>1.00</b>	12.87	344.31	310.25	377.06
ecc	<b>1.00</b>	14.03	313.48	293.92	252.36
cvrp	<b>1.00</b>	12.07	379.74	374.43	398.53

TABLE B.8: WASM normalised score.

Problem	Device				
	lap	rp3	m4	tab	m10
onemax	<b>1.00</b>	11.23	5.32	8.72	10.40
mttp	<b>1.00</b>	11.47	6.71	6.34	7.61
mmdp	<b>1.00</b>	10.76	5.75	4.65	7.75
ecc	<b>1.00</b>	9.26	4.41	4.82	6.85
cvrp	<b>1.00</b>	12.49	5.57	4.71	7.43



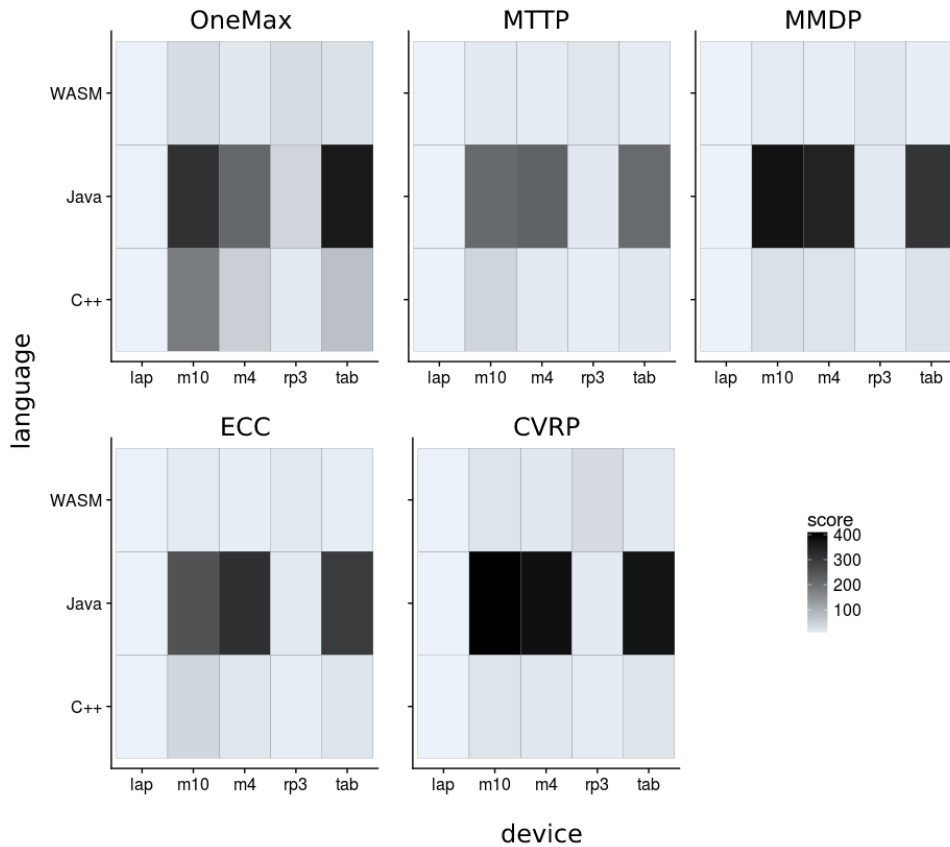


FIGURE B.1: Device/language normalised scores per each problem (the less score, the better).

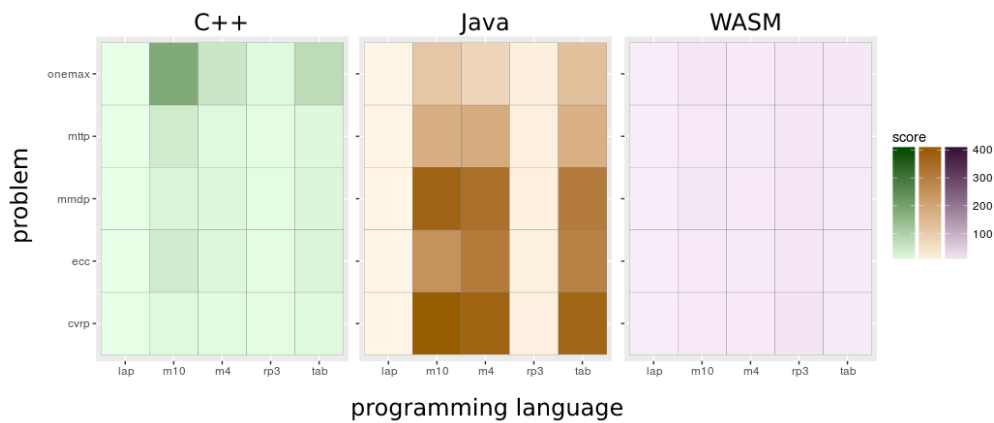


FIGURE B.2: Device/problem normalised scores per each programming language (the less score, the better).

Machine after Android KitKat). DVM was created to avoid JVM copyright and better use memory and power in more limited devices. There are many studies [9] that show that DVM was much slower than JVM; we have to say that some years later, ART did not improve that results.

In Tables B.6, B.7, and B.8, we can see the experiment results per programming language. These are the same results explained above but represented differently. The first table shows the result for C++, the second for Java, and the third for WASM. Each row is a different problem, and each column is a different device. Each cell has

a different colour to easily visualise the position of a device on a problem (sorted by score). The cell with the lightest colour is the best score in one problem, and the cell with the darkest colour is the worst score in the same problem (the less score, the better). Figure B.2 shows these results using a heatmap per programming language in a similar way that explained above. We can easily visualise the homogeneity in the results of the experiments in WASM and C++ explained above in contrast with what we observed in the results in Java. Regarding the Tables B.6, B.7, and B.8, what we see more highlighted is that both in C++ and Java, the order of the devices is: laptop, RP3, and then mobile devices. While in the WASM results, we can see how mobile devices get better scores than RP3, with RP3 occupying the last position. RP3 and the laptop get 2 to 3 times slower using web browsers (WASM) than using Java or C++, which is an acceptable result. In contrast, mobile devices perform better when they run algorithms using WASM, which is interesting as it makes WASM an appropriate programming language for edge computing.

It is interesting to see that RP3, a device of 30 € cost, can obtain a result not far off that of a laptop even when it has a much worst CPU. The RP3 takes about ten times longer than the laptop to solve the same problem. That is not much because of the difference in cost of the RP3 compared to the laptop. Moreover, when devices solve algorithms using the WASM programming language, the results of the best and worst devices are very similar, obtaining an excellent homogeneity. Another remarkable result is that mobile devices and RP3 can solve complex problems in a time comparable to that of a laptop. Therefore, despite their constraints, we think these devices are suitable for solving complex optimisation problems.

In conclusion, Java performed poorly on mobile devices. We showed that we get a performance closer to RP3 and the laptop on Android devices when using native code probing the hypothesis we had in Section 4.1. WASM and C++ achieved very close results on smartphones. In the case of *lap* and *RP3*, C++ and Java the results obtained are similar. The WASM result is only 2 to 3 times slower in these two cases. Also, WASM got the most homogeneous scores, which is very important when running algorithms in a parallel way [43, 60]. Therefore, WASM and C++ seem more suitable for implementing parallel algorithms on edge than Java. Results have been confirmed using a Wilcoxon test with Bonferroni correction and a significance level of 0.025.

The contributions of this work are related to **LIT**:

1. We implemented the algorithms using three programming languages (Java, C++ and WASM-JavaScript). We showed that we get good performance in most devices by using native code and compiling it into WebAssembly (WASM) with Emscripten. Moreover, when using WASM, the results of the best and worst devices are very similar, obtaining an excellent homogeneity which is very important when running algorithms in a parallel way. We proved that running these algorithms on the web browser is efficient and has the advantages of a lightweight virtualisation property and being multiplatform. **G1**.
2. We demonstrated that devices with low processor capacity are perfectly appropriate for solving optimisation problems at the edge. Inexpensive devices such as RP3 and mobile devices can achieve laptop-like performance when solving complex optimisation problems. **G1**.

## Appendix C

# Summary in Spanish

### C.1 Introducción

Hasta el momento presente, la mayor parte de los modelos de computación distribuida han procesado la información en potentes computadores alojados en la nube y, hasta cierto punto, los dispositivos de borde [102] como los teléfonos inteligentes, las tabletas y los ordenadores portátiles han actuado como terminales que muestran lo que ocurre en la nube. Sin embargo, el borde está cambiando, siendo más extenso y más sofisticado. No solo hay dispositivos móviles, sino también coches autónomos, drones, robots limpiadores, luces inteligentes, frigoríficos inteligentes, relojes inteligentes y una amplia variedad de sensores que están instalados por todas partes. Muchos de estos dispositivos recogen una gran cantidad de datos, los envían a la nube y esperan una respuesta (ver Fig. C.1).



FIGURE C.1: Nuevos tipos de dispositivos de borde conectados a la nube.

Según Statista <sup>1</sup> existen alrededor de 23.800 millones de dispositivos informáticos interconectados que están activos en el mundo y que producirán 149 zettabytes de datos útiles en 2024 <sup>2</sup>. Cisco <sup>3</sup> estimó que alrededor de 85 de los 850 zettabytes que se crearían en 2021 serían útiles mientras que solo serían almacenados unos siete zettabytes y el resto no serían analizados. En efecto, la producción de datos está

<sup>1</sup>[www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide](http://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide)

<sup>2</sup>[www.statista.com/statistics/871513/worldwide-data-created](http://www.statista.com/statistics/871513/worldwide-data-created)

<sup>3</sup>[blogs.cisco.com/sp/five-things-that-are-bigger-than-the-internet-findings-from-this-years-global-cloud-index](https://blogs.cisco.com/sp/five-things-that-are-bigger-than-the-internet-findings-from-this-years-global-cloud-index)

superando con creces la capacidad de la red de manera que no es posible almacenar o procesar la mayoría de estos datos en la nube debido al aumento exponencial de la demanda de datos y la velocidad de generación de los mismos. Por ejemplo, los datos generados por un Boeing 787 o un vehículo autónomo son de 1 y 5 GB por segundo, respectivamente [40, 112]. No solo se trata de un problema de capacidad de ancho de banda, sino que también estamos hablando de dispositivos que tienen que tomar decisiones en tiempo real, y que además no siempre pueden comunicarse con la nube. Esto hace que sea necesario diseñar nuevas aplicaciones especializadas y optimizadas para la analítica en el borde [97]. Los dispositivos tienen que ser más inteligentes, más seguros y estar mejor conectados [53]. Además, la toma de decisiones en tiempo real obliga a que el procesamiento de los datos se produzca en el borde que es donde estos se recogen. La computación de borde (EC) [102] trata el problema del crecimiento exponencial de la generación de datos en los dispositivos de borde proponiendo aumentar la cantidad de procesamiento en estos dispositivos y reducir el intercambio de datos en bruto con la nube.

Por otro lado, los grandes avances que se han producido en los últimos años en la Inteligencia Artificial (IA) nos permiten entrenar modelos de aprendizaje automático (ML) utilizando una gran cantidad de datos que hasta ahora se habían deshechado por considerarlos inútiles [67]. Además, las leyes y normas que protegen la privacidad de los ciudadanos impiden que muchos datos puedan ser compartidos con terceros (por ejemplo, las imágenes médicas). Como una solución prometedora a estos problemas, se ha propuesto el aprendizaje federado (FL), el cual proviene originalmente del aprendizaje profundo distribuido (DDL) [75]. Se trata de un paradigma de aprendizaje que entrena un modelo compartido de forma distribuida, mientras que los datos permanecen almacenados en los dispositivos de borde manteniendo su privacidad. FL se está investigando activamente y se aplica ampliamente [50, 69, 93] (por ejemplo, en medicina [101]). Tanto FL como EC tienen muchas cosas en común; por ello, varios autores han propuesto soluciones que satisfacen a ambos, llamándolo aprendizaje federado en el borde (FEEL) [109, 121].

En esta tesis, proponemos ver FEEL como un tipo de computación voluntaria (VC) [6, 30, 57, 64, 86] donde los usuarios donan los recursos de computación de sus dispositivos de borde a un proyecto en el que se entrena un modelo de DL compartido (VC4FL). Para lograr este objetivo debemos estudiar primero cómo utilizar los algoritmos de optimización en los dispositivos de borde. El entrenamiento de una NN es un problema de optimización pero además, en FL, surgen otros nuevos relacionados con los retos de la volatilidad de las conexiones de los dispositivos de borde, la sobrecarga de comunicación, la tolerancia a fallos, la heterogeneidad de los dispositivos y la escalabilidad, entre otros muchos. La mayoría de estos problemas son NP-hard [23, 24], por lo que los algoritmos exactos [23] no son efectivos. Por otro lado, las metaheurísticas [3, 110] permiten encontrar soluciones aceptables (no óptimas) a muchos de estos problemas complejos. El entrenamiento de NN y el uso de técnicas de optimización como la metaheurística son procesos que generalmente requieren una alta carga computacional, lo que dificulta su uso en dispositivos de borde. Sin embargo, los recientes avances en la potencia de cálculo, la capacidad de memoria y la velocidad de las comunicaciones de estos dispositivos [102] nos hacen pensar que la ejecución de pequeños y medianos algoritmos de optimización de forma distribuida en estos dispositivos es factible y útil para la ciencia y la sociedad.

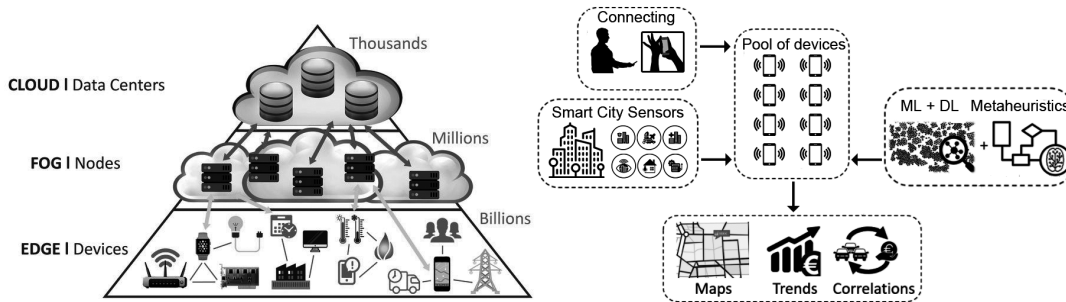


FIGURE C.2: Inteligencia distribuida ubicua: Datos de usuarios y ciudades se usan para entrenar modelos distribuidos de ML mientras los problemas asociados se optimizan con algoritmos de optimización.

## C.2 Propuesta de trabajo

Esta tesis aborda el reto de la inteligencia distribuida ubicua en los dispositivos de borde (ver Figura 1.3). En concreto, proponemos ver FEEL como un tipo de VC (VC4FL) en el que los usuarios donan los recursos informáticos de sus dispositivos de borde a un proyecto que entrena un modelo de DL compartido. Como detallamos en la sección 3.3, esto significa tratar con dispositivos con restricciones, poco fiables y heterogéneos (en HW y SW). Esta plataforma debe adaptarse a la volatilidad de las desconexiones y ser tolerante a los fallos. Examinamos cómo debe ser una plataforma de este tipo, por qué es necesario trasladar la computación desde la nube al borde y cómo podemos aprender de los datos locales de los usuarios sin enviarlos a la nube. Además, proponemos el uso de metaheurísticas para resolver los problemas que surgen en este tipo de sistemas, como la optimización de la cantidad adecuada de comunicación necesaria sin perder calidad en el modelo de inteligencia artificial aprendido. Tenemos cuatro objetivos principales para lograr nuestro propósito:

- G1** Analizar el hardware y el software heterogéneos de los dispositivos de borde.
- G2** Investigar los paradigmas, protocolos, algoritmos y desafíos que puedan surgir.
- G3** Diseñar e implementar técnicas que nos permitan realizar un aprendizaje automático distribuido voluntario ejecutando la computación en los dispositivos de borde de los usuarios.
- G4** Difundir los resultados y poner a disposición pública el software diseñado e implementado.

A continuación detallamos nuestros principales objetivos siguiendo las fases del método científico [34]:

### 1. Observación.

- (a) Analizar el HW heterogéneo que vamos a utilizar en la plataforma ubicua (rendimiento, batería y memoria de los dispositivos más comunes que utilizaremos). Investigar los puntos de referencia más adecuados para comparar estos dispositivos. **G1**.
- (b) Analizar el SW heterogéneo que podemos encontrar en estos dispositivos y los lenguajes de programación más adecuados para implementar nuestra propuesta. **G1**.

- (c) Investigar los algoritmos más adecuados para conseguir nuestro propósito. **G2.**
- (d) Investigar la literatura sobre la computación voluntaria mediante dispositivos de borde con baja capacidad de computación y limitaciones de comunicación y energía. **G2.**
- (e) Estudiar los problemas que puedan surgir para cumplir nuestro propósito. **G2.**

## 2. Inducción.

- (a) Identificar paradigmas y protocolos de red apropiados, eficientes y seguros para diseñar y construir la arquitectura de computación distribuida de forma ubicua. **G2.**

## 3. En el contexto de VC4FL, nuestra hipótesis es:

- (a) *“El despliegue de la inteligencia distribuida de forma ubicua y tolerante a los fallos utilizando dispositivos de borde heterogéneos, desconocidos y poco fiables es una necesidad y puede competir con los enfoques tradicionales que utilizan un clúster homogéneo, conocido y estático de computadores.”*

## 4. Experimentación.

- (a) Diseñar, implementar y evaluar técnicas de tolerancia a fallos para dispositivos poco fiables que pueden fallar en cualquier momento. **G3.**
- (b) Diseñar, implementar y evaluar técnicas de adaptación en entornos dinámicos en los que los dispositivos se conectan y desconectan a voluntad. **G3.**
- (c) Diseñar, implementar y evaluar la arquitectura de computación distribuida. **G3.**
- (d) Desarrollar prototipos utilizables. **G3.**
- (e) Realizar experimentos con casos extremos de dinamicidad de conexiones y desconexiones utilizando dispositivos no fiables para probar la tolerancia a fallos y la adaptación de la plataforma distribuida. **G3.**
- (f) Analizar los resultados numéricos de la arquitectura de computación distribuida voluntaria y validar nuestra propuesta, demostrando nuestra hipótesis. **G3.**
- (g) Proponer soluciones para hacer frente a los problemas que surjan para cumplir nuestro propósito y ponerlas a prueba. **G3.**

## 5. Conclusiones.

- (a) Escribir un nuevo cuerpo de conocimientos sobre la IA ubicua. **G4.**
- (b) Difundir nuestros resultados en conferencias y revistas de impacto, Internet, vídeos e interacciones con los usuarios. Definir el trabajo futuro y la fertilización cruzada. **G4.**

## C.3 Contribución

Esta tesis doctoral se presenta como un compendio de cuatro publicaciones alineadas con los objetivos definidos en la Sección C.2 y que buscan el objetivo común

de mejorar el estado del arte en computación distribuida. Considerando la relevancia del objetivo general de esta tesis y sus problemas específicos, esperamos que nuestros resultados contribuyan a la comunidad investigadora y a la sociedad. Esta sección está organizada en cuatro subapartados, cada uno se refiere al trabajo realizado en uno de los trabajos. Cabe destacar que dos de estos trabajos han sido publicados en revistas indexadas por JCR (Q1). Además, otro fue publicado en las actas de un congreso nacional, y otro en las actas de un congreso internacional en el que el doctorando recibió el premio “Outstanding Student”.

Organizamos la aportación de esta tesis en cuatro contribuciones principales:

- LIT.** Hemos contribuido a ampliar la literatura sobre EC y FL (FEEL) revisando los puntos de referencia más adecuados para comparar el rendimiento en dispositivos de borde (**G1**). Analizando el hardware y el software de los dispositivos de borde más comunes y su rendimiento al resolver problemas complejos (**G1**). Revisando la literatura que utiliza el navegador web como plataforma de VC (**G2**). Definiendo los retos y las características deseables de VC4FL combinando los criterios tradicionalmente utilizados para VC y FL (**G2**), y mostrando que el navegador web es la plataforma adecuada para este propósito debido al sandboxing, su ubicuidad y a que no requiere instalación de software (**G2**). Por último, resumimos las técnicas más comunes utilizadas para reducir el problema de la sobrecarga de comunicación en FL (**G2** y **G4**).
- SYN.** Hemos propuesto un enfoque síncrono para realizar computación voluntaria distribuida utilizando el paradigma de programación MapReduce y el patrón de cola de mensajes (**G2**) a través de los navegadores web sin interrumpir la experiencia del usuario del sitio ni instalar software adicional. Diseñamos, implementamos (**G3**), y evaluamos nuestra propuesta mostrando que tiene una buena escalabilidad a pesar de las restricciones en el canal de comunicación (**G4**).
- ASYN.** Hemos propuesto un algoritmo asíncrono para FEEL que se adapta a la entrada y salida de clientes en el cómputo (**G2**) cuando el número de trabajadores es bajo e incluso puede llegar a cero durante el entrenamiento. Hemos diseñado, implementado (**G3**), y evaluado empíricamente nuestra propuesta en escenarios altamente dinámicos y cambiantes obteniendo una precisión numérica y una puntuación CK similares a las configuraciones actuales que utilizan una plataforma distribuida estática para el aprendizaje (**G4**).
- MOD.** Formulamos y modelamos el problema de la sobrecarga de comunicación en FL (**G2**), uno de los retos más importantes para realizar FL de forma eficiente, como un problema multiobjetivo. Propusimos resolverlo utilizando algoritmos genéticos para la optimización de funciones multiobjetivo (**G3**). Nuestra propuesta consigue una mayor precisión a la vez que reduce las comunicaciones de 10 a 2000 dependiendo de la topología de la red neuronal en comparación con la configuración de comunicación máxima (**G4**).

A continuación explicamos con más detalle nuestras aportaciones. Todo nuestro trabajo persigue el objetivo **G4** de difundir los resultados. En nuestro primer trabajo [85] (ver Sección C.3.1), abordamos el objetivo **G1** y **G2** y los subobjetivos 1-a, 1-b, 1-c, 5-a y 5-b. Las aportaciones de este trabajo son **LIT**:

1. Seleccionamos y estudiamos los puntos de referencia más utilizados para analizar el rendimiento de los dispositivos de borde. **G1**.

- (a) Demostramos que los puntos de referencia clásicos no son fiables para evaluar el rendimiento de los dispositivos de borde, pero son útiles para hacerse una idea aproximada. Dependen de la arquitectura del hardware y del sistema operativo. Además, según el problema específico que se quiera resolver los resultados pueden variar. Por ejemplo, Antutu es tan común que los fabricantes de hardware han empezado a hacer trampas en el punto de referencia, lo que lo hace poco fiable. **G1**.
2. Analizamos el rendimiento de diferentes dispositivos de borde mientras se resuelve un conjunto representativo de problemas de optimización con diferentes características mediante la ejecución de algoritmos genéticos. **G1**.
- (a) Demostramos que la RP3 es una plataforma perfectamente adecuada para ejecutar algoritmos en el borde. La RP3 es muy barata, utiliza menos memoria que un portátil y resuelve los problemas en un tiempo razonable. La RP3 se utiliza habitualmente en los sensores de las ciudades inteligentes, lo que lo hace idóneo para realizar cálculos en el borde. **G1**.
- (b) Demostramos que los dispositivos de borde que ejecutan Android tienen problemas para ejecutar aplicaciones de alto rendimiento. Este sistema operativo parece estar más interesado en mantener un bajo consumo de batería y utilizar menos memoria (usando el recolector de basura más a menudo) que en el rendimiento. **G1**.
- (c) Demostramos que los puntos de referencia como GeekBench 4 obtienen buenos resultados en dispositivos Android. Por lo tanto, creemos que es necesario ejecutar código nativo en dispositivos Android para poder resolver las restricciones de este sistema operativo. En el Apéndice B, mostramos un estudio posterior no publicado. Analizamos el rendimiento de otros lenguajes de programación como C++ y WASM (mismos dispositivos, problemas y algoritmos). Demostramos que nuestra hipótesis era correcta, obteniendo un rendimiento más cercano a la RP3 y al portátil en dispositivos Android. **G1 y G2**.
- i. Implementamos los algoritmos utilizando tres lenguajes de programación (Java, C++ y WASM-JavaScript). Demostramos que obtenemos un buen rendimiento en la mayoría de los dispositivos utilizando código nativo y compilándolo en WebAssembly (WASM) con Emscripten. Además, al utilizar WASM, la diferencia entre los mejores y peores resultados en los dispositivos de borde es muy pequeña, obteniendo una excelente homogeneidad lo que es muy importante al ejecutar algoritmos de manera distribuida. Probamos que la ejecución de estos algoritmos en el navegador web es eficiente y tiene las ventajas de la virtualización ligera y de ser multiplataforma. **G1**.
- ii. Demostramos que los dispositivos con poca capacidad de procesamiento son perfectamente adecuados para resolver problemas de optimización en el borde. Los dispositivos económicos, como la RP3 y los dispositivos móviles, pueden alcanzar un rendimiento similar al de los ordenadores portátiles cuando resuelven problemas de optimización complejos. **G1**.

En nuestro segundo trabajo [86] (ver Sección 3.2 y 4.2), abordamos los objetivos **G2** y **G3**, y los subobjetivos *1-a*, *1-d*, *1-e*, *2-a*, *4-a*, *4-b*, *4-c*, *4-d*, *4-e*, *4-f*, *4-g*, *5-a* y *5-b*. Las aportaciones de este trabajo son **LIT** y **SYN**:



3. Resumimos la literatura que utiliza el navegador web como plataforma de computación voluntaria (BBVC) y describimos las recientes mejoras que lo hacen cada vez más posible (ver Sección 3.2). **G2**.
4. Proponemos un framework BBVC para la computación voluntaria distribuida utilizando el paradigma de programación MapReduce y el patrón de cola de mensajes a través de los navegadores web sin interrumpir la experiencia del usuario del sitio ni instalar software adicional. Demostramos que nuestra propuesta tiene una buena escalabilidad a pesar de las restricciones del canal de comunicación. **G3**.
5. Realizamos una prueba de concepto en la que demostramos que el entrenamiento distribuido de redes neuronales basado en el navegador web es factible y eficiente. El uso de modelos de NN pequeños y medianos es perfectamente adecuado para resolver los problemas de los dispositivos de borde. Los resultados demuestran que es factible y escalable, además de ser un área apasionante para explorar. **G3**.

En nuestro tercer trabajo [84] (ver Sección 3.3 y 4.3), abordamos los objetivos **G2**, **G3**, y **G4** y los subobjetivos 1-a, 1-b, 1-d, 1-e, 2-a, 4-a, 4-b, 4-c, 4-d, 4-e, 4-f, 4-g, 5-a y 5-b. Las aportaciones de este trabajo son **LIT** y **ASYN**:

6. Definimos los retos y las características deseables de VC4FL (ver Sección 3.3). **G2**.
7. Proponemos un algoritmo para FEEL que se adapta a los clientes asíncronos que se unen y abandonan el cómputo de manera dinámica y nos centramos en el caso en el que el número de trabajadores es bajo e incluso puede llegar a cero durante el entrenamiento. **G3**.
8. Proponemos, implementamos y evaluamos una plataforma de software para realizar FL que cumple con los desafíos y las características deseables definidas. Evaluamos nuestra propuesta mediante una amplia experimentación en una configuración estática y en escenarios altamente dinámicos y cambiantes. A continuación, demostramos que la plataforma se adapta bien a este entorno cambiante utilizando un número determinado de dispositivos poco fiables y heterogéneos (hardware y software) obteniendo una precisión numérica similar a las configuraciones actuales que utilizan una plataforma estática para el aprendizaje. A continuación, demostramos la tolerancia a fallos de la plataforma la cual se recupera de desconexiones inesperadas de los dispositivos voluntarios. **G3**.
9. Hemos lanzado una biblioteca modular de código abierto que cubre la mayoría de las características deseables de FEEL y VC. **G4**.

En nuestro cuarto trabajo [87] (ver secciones 3.3 y 4.4), abordamos los objetivos **G2** y **G3**, y los subobjetivos 1-e, 4-f, 5-a y 5-b. Las aportaciones de este trabajo son **LIT** y **MOD**:

10. Resumimos las técnicas más comunes en la literatura para resolver el problema de la sobrecarga de comunicación en FL (ver Sección 3.3). **G2**.
11. Formulamos y modelamos el problema de la sobrecarga de comunicación en el aprendizaje federado como un problema multiobjetivo (FL-COP). **G3**.

12. Proponemos resolver el FL-COP utilizando algoritmos genéticos para la optimización de funciones multiobjetivo. Nuestra propuesta consigue una mayor precisión a la vez que reduce las comunicaciones de 10 a 2000 veces dependiendo de la topología de la red neuronal en comparación con la configuración de comunicación máxima. **G3**.
13. Nuestra propuesta proporciona una población de soluciones que facilita la toma de decisiones a la hora de configurar los parámetros de FL. **G3**.

Los resultados de esta tesis demuestran que el despliegue de inteligencia distribuida de manera ubicua en los dispositivos de borde es factible y útil, proporcionando también una mejor comprensión de cómo debe ser dicha inteligencia distribuida, los problemas asociados y cómo deben abordarse estos problemas.

En resumen, nuestras contribuciones están en consonancia con los objetivos inicialmente marcados. En las siguientes secciones se resumen los trabajos realizados por el doctorando para apoyar esta tesis los cuales han sido publicados en [85], [86], [84] y [87]. Estas publicaciones son el resultado del estudio realizado para alcanzar los objetivos de la misma. Son cuatro contribuciones, dos de ellas publicadas en revistas JCR, una en un congreso internacional y otra en un congreso nacional. Todos nuestros trabajos persiguen el objetivo **G4** de difundir los resultados. Nuestro primer artículo se centra en **G1** y **G2**, que analiza la heterogeneidad del hardware y el software de los dispositivos de borde. El segundo trabajo se centra en los objetivos **G2** y **G3**, examinando los mejores paradigmas y protocolos para la plataforma distribuida voluntaria. Se sugiere una versión sincrónica pero desacoplada, que permita la conexión y desconexión de los voluntarios. Los objetivos **G2** y **G3** se persiguen también en el tercer artículo, pero esta vez también se aborda la cuestión de la privacidad de los datos en los dispositivos de borde y se sugiere un enfoque asíncrono. El cuarto artículo persigue los objetivos **G2** y **G3**, abordando el problema de la sobrecarga de comunicación de FL. Se presentan en orden cronológico:

1. En [85] analizamos la idoneidad de los dispositivos de borde como plataforma para ejecutar algoritmos genéticos (GAs). **G1, G2, y G4. LIT**.
2. En [86] resumimos la literatura que utiliza el navegador web como plataforma de computación voluntaria (BBVC) y describimos las recientes mejoras que lo hacen cada vez más posible. Proponemos un framework BBVC para la computación voluntaria distribuida utilizando el paradigma de programación MapReduce y el patrón de cola de mensajes a través de los navegadores web. **G2, G3, and G4. LIT y SYN**.
3. En [84] definimos los retos y las características deseables para VC4FL. Proponemos un algoritmo e implementamos y evaluamos una plataforma de software. Evaluamos nuestra propuesta mediante una amplia experimentación. **G2, G3, and G4. LIT y ASYN**.
4. En [87], resumimos las técnicas más comunes en la literatura para resolver el problema de la sobrecarga de comunicación en FL. Formulamos y modelamos el problema de sobrecarga de la comunicación en el aprendizaje federado como un problema multiobjetivo (FL-COP) y proponemos resolverlo utilizando NSGA-II. **G2, G3, y G4. LIT y MOD**.

Para obtener una lista completa de publicaciones y una descripción de la revista o conferencia en cuestión, ver el Apéndice D.

### C.3.1 Analizando la idoneidad de los dispositivos de borde

En este trabajo [85], abordamos los objetivos **G1**, **G2**, y **G4** y los subobjetivos *1-a*, *1-b*, *1-c*, *5-a* y *5-b*. Las contribuciones de este trabajo están relacionadas con **LIT** (ver Sección 1.2.2). Para alcanzar los objetivos de la tesis, debemos analizar los dispositivos de borde más comunes. Para ello, realizamos un conjunto de experimentos en los que medimos el rendimiento, el uso de la memoria y el consumo de batería de estos dispositivos, resolviendo un conjunto de problemas de optimización bien conocidos mediante GAs. En concreto, queremos analizar si estos dispositivos son adecuados para resolver problemas complejos y estudiar sus diferencias en el uso de recursos. Para ello seleccionamos una buena representación de dispositivos de borde con una amplia variedad de prestaciones (ver Tabla 4.1 y 4.2). A continuación analizamos los resultados de un conjunto de puntos de referencia bien conocidos (ver Tabla 4.3) para obtener un indicador de su rendimiento.

Elegimos una amplia variedad de problemas para la evaluación (ver Tabla 4.4 y Apéndice A). Utilizamos problemas de representación binaria y entera de diferentes dimensiones, un problema NP-Hard, diversas restricciones y variadas funciones de fitness. Estos problemas tienen características interesantes en la optimización, como la epistasis, la multimodalidad y el engaño. Los problemas de representación binaria elegidos son el problema OneMax [37], el MTTP [107], el MMDP [46], y el ECC [74]. Además, como problema de representación entera, seleccionamos el problema bien conocido CVRP [29] que puede escalar y caracterizar los problemas de movilidad inteligente en las ciudades y es NP-hard. Seleccionamos una instancia de cada problema (Tabla 4.4). Elegimos cuatro instancias de representación binaria de la biblioteca JCell, y una utilizando representación entera llamada CMT1, una instancia CVRP propuesta por Christofides [22] para este problema. Estas instancias no son muy grandes porque queremos resolverlas utilizando dispositivos restringidos con baja capacidad de procesamiento. En este análisis no tenemos el objetivo de resolver problemas enormes si no que queremos analizar el rendimiento al resolver estos problemas en dispositivos de borde heterogéneos con diferentes lenguajes de programación. Por lo tanto, hemos elegido un conjunto variado de instancias que todos los dispositivos evaluados pueden resolver en un tiempo razonable. Utilizamos el GA canónico de estado estacionario para resolver los problemas anteriores.

Los resultados de este trabajo muestran que los puntos de referencia clásicos no son fiables para evaluar el rendimiento de los dispositivos de borde, pero son útiles para hacerse una idea aproximada. Por otro lado, la RP3 se muestra como una plataforma perfectamente adecuada para ejecutar algoritmos en el borde. También vemos que los dispositivos de borde que ejecutan Android (móviles y tabletas) tienen problemas para ejecutar aplicaciones de alto rendimiento. Este sistema operativo parece estar más interesado en mantener un bajo consumo de batería y utilizar menos memoria (usando el recolector de basura más a menudo) que en el rendimiento. Por último, en un estudio posterior no publicado que extiende este trabajo (ver Apéndice B), analizamos el rendimiento utilizando otros lenguajes de programación como C++ y WASM-JavaScript (navegador web) observando que con ellos podemos obtener un rendimiento más cercano a RP3 y al portátil en dispositivos Android. Por último, demostramos que obtenemos un buen rendimiento en la mayoría de los dispositivos utilizando código nativo y compilándolo en WebAssembly (WASM) con Emscripten. Demostramos que la ejecución de estos algoritmos en el navegador web es eficiente y tiene las ventajas de una virtualización ligera y de ser multiplataforma.

### C.3.2 Propuesta de entrenamiento de NN en el navegador web utilizando VC y el paradigma MapReduce

Nuestro trabajo anterior demostró que los dispositivos de borde con baja capacidad de procesamiento eran perfectamente adecuados para resolver problemas de optimización y que el navegador web podía ser una plataforma adecuada para lograr nuestro objetivo. En este trabajo, proponemos un enfoque para utilizar el navegador web para entrenar redes neuronales de forma distribuida utilizando voluntarios. Además, realizamos un diseño desacoplado para que los voluntarios puedan conectarse y desconectarse en cualquier momento sin detener el entrenamiento.

En este trabajo [86], abordamos los objetivos **G2**, **G3**, y **G4**, y los subobjetivos *1-a*, *1-d*, *1-e*, *2-a*, *4-a*, *4-b*, *4-c*, *4-d*, *4-e*, *5-a* y *5-b*. Las contribuciones de este trabajo están relacionadas con **LIT** y **SYN** (ver Sección 1.2.2). En primer lugar, resumimos la literatura que utiliza el navegador web como plataforma de computación voluntaria (BBVC) y describimos las recientes mejoras que lo hacen cada vez más posible (véase también la Sección 3.2). A continuación, proponemos un framework BBVC, JSDoop<sup>4</sup>, para la HPC colaborativa distribuida utilizando el paradigma de programación MapReduce [65] y el patrón de cola de mensajes a través de los navegadores web. Como mostramos en las secciones 3.2 y 3.3 (ver Tabla 3.1), los mayores obstáculos para las plataformas de VC son la accesibilidad, la usabilidad y la seguridad. La instalación de aplicaciones suele ser una barrera importante para los usuarios por la dificultad, la pereza o el miedo a instalar software desconocido en sus dispositivos. Proponemos utilizar navegadores web como software de ejecución principalmente porque resuelve la mayoría de estos problemas. Además, como se explica en la sección 3.2, las mejoras que se han introducido en los navegadores web en los últimos años los acercan cada vez más al rendimiento de los programas nativos que se ejecutan en una consola de comandos. Finalmente, utilizamos este framework para entrenar una RNN basada en LSTM [54] de forma distribuida y colaborativa.

El diseño de nuestra propuesta se ha guiado por las características deseables (ver Tabla 3.1) de dichas plataformas. Sin embargo, nuestra implementación es una prueba de concepto y aspectos como la seguridad deberían ser más importantes en una versión final. En nuestro diseño, podemos distinguir los siguientes actores: **iniciador**, **servidor web**, **servidor de cola**, **servidor de datos**, y **voluntarios** (ver Fig. C.3).

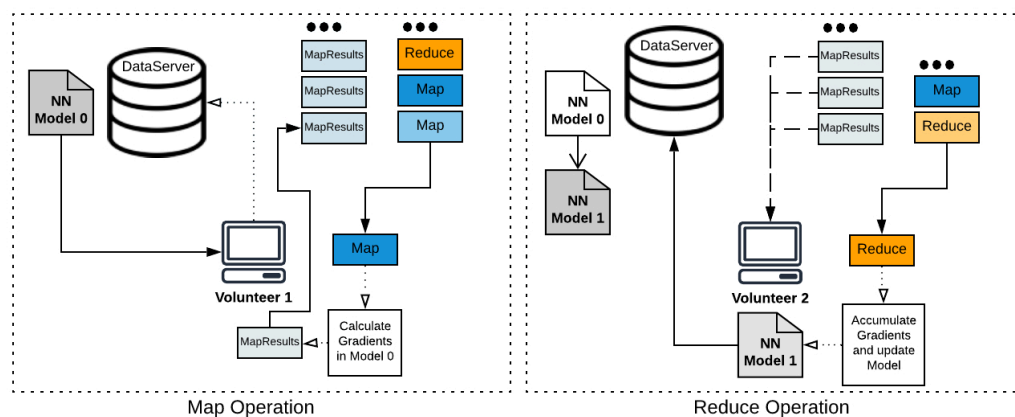


FIGURE C.3: Propuesta para el entrenamiento de NN utilizando el paradigma MapReduce.

<sup>4</sup>Código disponible en <https://github.com/jsdoop/>

Los resultados muestran que el entrenamiento de redes neuronales distribuidas utilizando navegadores web es factible y eficiente. Nuestra propuesta permite que los voluntarios colaboren simplemente accediendo a una URL, y demostró ser una implementación adecuada de BBVC logrando una buena escalabilidad y permitiendo añadir/eliminar voluntarios dinámicamente durante la ejecución sin perder información.

Nuestra propuesta es independiente del número de dispositivos conectados, es tolerante a fallos, permite que el número de voluntarios cambie de manera dinámica durante el aprendizaje, es multiplataforma y permite una sencilla conexión al sistema mediante un simple click en un enlace web.

### C.3.3 Propuesta de FL asíncrono utilizando dispositivos de borde voluntarios no fiables

En nuestro trabajo anterior, propusimos utilizar el navegador web para entrenar redes neuronales de forma voluntaria distribuida. Sin embargo, utilizamos un enfoque síncrono que tiene algunas desventajas, como vimos en la Sección 2.5.1, como tener que esperar a los dispositivos más lentos y dispositivos que pueden congelarse y paralizar el entrenamiento. En este trabajo, proponemos un método asíncrono para resolver estos problemas. En particular, nos centramos en la situación en la que el número de voluntarios es bajo y puede incluso llegar a cero durante parte del entrenamiento.

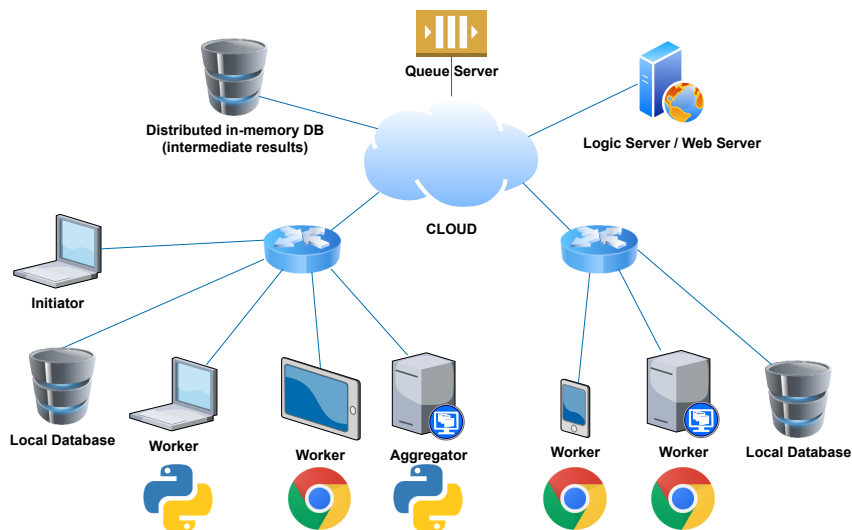


FIGURE C.4: Arquitectura del sistema de alto nivel.

En este trabajo [84] (ver secciones 3.3 y 4.3), abordamos los objetivos G2, G3, y G4, y los subobjetivos 1-a, 1-b, 1-d, 1-e, 2-a, 4-a, 4-b, 4-c, 4-d, 4-e, 4-f, 5-a y 5-b. Las contribuciones de este trabajo están relacionadas con LIT y ASYN (ver Sección 1.2.2). En primer lugar definimos los retos y las características deseables de VC4FL (ver Sección 3.3). A continuación, proponemos un algoritmo para FEEL que se adapta a los clientes heterogéneos asíncronos que se conectan y desconectan de manera dinámica durante el aprendizaje. El objetivo es continuar el proceso de aprendizaje y evitar la espera de los dispositivos más lentos. Proponemos, implementamos y evaluamos una nueva plataforma de software (JSDoop versión 2.0<sup>5</sup>, rediseñada y reimplementada desde cero) para DDL en dispositivos de borde de manera voluntaria

<sup>5</sup>Código disponible en <https://github.com/jsdoop/>

(ver Fig. C.4). A continuación, evaluamos sus resultados en problemas relevantes para FEEL (ver Sección 3.3).

Por último, realizamos un exhaustivo análisis empírico para evaluar la propuesta. Estudiamos las principales características de la plataforma mencionadas anteriormente. Analizamos cómo la asincronía, las conexiones dinámicas y las desconexiones de los dispositivos afectan al aprendizaje y cómo podemos adaptar el aprendizaje para mantener una alta precisión. Demostramos que la arquitectura del sistema descentralizado y adaptativo propuesto para el aprendizaje asíncrono permite a los usuarios voluntarios ceder los recursos de sus dispositivos y los datos locales para entrenar un modelo ML compartido. Los dispositivos pueden unirse y abandonar el sistema en cualquier momento sin detener el proceso de aprendizaje. Esta arquitectura no necesita saber dónde o cuándo se conectarán los dispositivos participantes. Este sistema es tolerante a los fallos, por lo que las desconexiones inesperadas de los trabajadores no interrumpen el aprendizaje. La implementación de código abierto permite la colaboración entre dispositivos con hardware y software heterogéneos. Los dispositivos de los usuarios pueden unirse al aprendizaje de diferentes maneras, como desde un navegador web o ejecutando un proceso de Python en el dispositivo cliente, permitiendo así que unirse a la plataforma casi cualquier dispositivo.

Los resultados demuestran que nuestra propuesta se adapta bien a este entorno cambiante utilizando un número determinado de dispositivos poco fiables y heterogéneos (hardware y software) obteniendo una precisión numérica similar a las configuraciones actuales que utilizan una plataforma estática para el aprendizaje. También, demostramos la tolerancia a fallos de la plataforma la cual se recupera de desconexiones inesperadas de los dispositivos voluntarios (ver Sección C.3 para una explicación más detallada de las contribuciones de este trabajo).

### C.3.4 Propuesta para reducir la sobrecarga de comunicación en FL

En las dos secciones anteriores nos hemos centrado en el proceso de aprendizaje distribuido. Sin embargo, no hemos prestado atención a algunos de los retos asociados a FL. En particular, uno de los mayores problemas de FL es la gran cantidad de información (pesos o gradientes) que hay que comunicar en cada iteración. Por lo tanto, en este trabajo nos centramos en reducir la cantidad de datos enviados durante el proceso de aprendizaje, manteniendo e incluso mejorando el nivel de precisión del modelo.

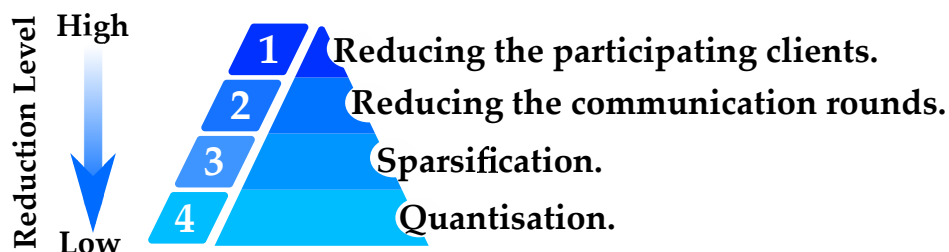


FIGURE C.5: Las técnicas de reducción de la comunicación en FL-COP por niveles.

En este trabajo [87], abordamos los objetivos G2, G3, y G4, y los subobjetivos 1-e, 2-a, 4-f, 5-a y 5-b. Las contribuciones de este trabajo están relacionadas con LIT y MOD (ver Sección 1.2.2). Conseguir resultados de alta calidad en FL requiere una gran cantidad de comunicación en la que se intercambia información entre los

dispositivos de borde y el servidor [75]. Encontrar la cantidad mínima de comunicación que logre la misma o mayor precisión es un problema multiobjetivo en el que queremos reducir las comunicaciones y aumentar la precisión del modelo. Para esta clase de problemas, los algoritmos estocásticos como la metaheurística y, en particular, los algoritmos genéticos multiobjetivo como el NSGA-II son una alternativa prometedora que proporciona un buen equilibrio entre la búsqueda de una solución óptima y el tiempo requerido [32]. En esta investigación, (I) modelamos y formulamos el problema de sobrecarga de comunicación en el aprendizaje federado como un problema multiobjetivo (FL-COP), (II) y aplicamos el NSGA-II para resolverlo. Asimismo, investigamos y detallamos los principales parámetros desencadenantes de la sobrecarga de comunicación que la literatura suele abordar por separado dentro de un mismo trabajo (ver Fig. C.5).

Nuestra propuesta consigue una mayor precisión a la vez que reduce las comunicaciones de 10 a 2000 veces dependiendo de la topología de la red neuronal en comparación con la configuración de comunicación máxima. Además, nos aporta una población de soluciones que facilita la toma de decisiones a la hora de configurar los hiperparámetros.

## C.4 Conclusiones y trabajo futuro

Esta tesis ha abordado el reto de la inteligencia ubicua distribuida en dispositivos de borde. En concreto, propusimos ver FEEL como un tipo de VC (VC4FL) en el que los usuarios donan los recursos informáticos de sus dispositivos de borde a un proyecto que entrena un modelo de DL compartido. Hemos tratado con dispositivos restringidos, poco fiables y heterogéneos (en HW y SW) y hemos adaptado el aprendizaje a la volatilidad de las desconexiones consiguiendo tolerancia a fallos. También hemos formulado y modelado el problema de la sobrecarga de comunicación en FL, que es un reto importante para la inteligencia distribuida de forma ubicua y el aprendizaje eficiente de los datos de los usuarios, como un problema multiobjetivo. Y lo abordamos utilizando algoritmos genéticos para la optimización de funciones multiobjetivo, logrando una mayor precisión del modelo y reduciendo las comunicaciones en comparación con la configuración de comunicación máxima.

Hemos dividido las aportaciones de esta tesis en cuatro contribuciones principales (**LIT**, **SYN**, **ASYN** y **MOD**) para cumplir con los cuatro objetivos principales que hemos especificado (**G1-G4**) (ver Sección 1.2.1 y 1.2.2 para más detalles sobre los objetivos y las contribuciones). En todos nuestros trabajos perseguimos el objetivo **G4** de la difusión de los resultados. Siguiendo este objetivo, el software implementado en los trabajos 2, 3<sup>6</sup> y 4<sup>7</sup> está disponible públicamente en Github.

En nuestro primer trabajo [85] (ver Sección 4.1), perseguimos nuestro primer objetivo **G1** de analizar el hardware y el software heterogéneos de los dispositivos de borde. La principal aportación de este trabajo es **LIT**. Explicamos que los puntos de referencia tradicionales no son fiables para evaluar el rendimiento de los dispositivos de borde, pero ayudan a obtener una comprensión general. Demostramos que la plataforma RP3 es adecuada para ejecutar algoritmos en el borde y que los dispositivos Android de borde (smartphones y tablets) tienen problemas para ejecutar aplicaciones de alto rendimiento. Este sistema operativo parece más preocupado por la eficiencia de la memoria y el bajo consumo de batería (a menudo ejecutando el recolector de basura) que por el rendimiento. Observamos que el uso

<sup>6</sup>Código disponible en <https://github.com/jsdoop/>

<sup>7</sup>Código disponible en <https://github.com/NEO-Research-Group/flcop>

de código nativo en Android nos permitía evitar el problema mencionado anteriormente. Además, demostramos que utilizar código nativo y compilarlo en WebAssembly (WASM) con Emscripten da como resultado un buen rendimiento en la mayoría de los dispositivos. WASM es un código binario portátil que puede ejecutarse en los navegadores web modernos. Confirmamos la eficacia de la ejecución de estos algoritmos en el navegador web utilizando WASM, que además tiene las ventajas de ser multiplataforma y tener una propiedad de virtualización ligera.

En nuestro segundo trabajo [86] (ver Sección 3.2 y 4.2), nos centramos en los objetivos **G2** y **G3**. La principal contribución de este trabajo es **LIT** y **SYN**. Revisamos la literatura sobre el uso del navegador web como plataforma para la computación voluntaria (BBVC) y discutimos los recientes avances que están haciendo que esto sea cada vez más viable (ver Sección 3.2). Presentamos un framework BBVC para la computación voluntaria distribuida que utiliza el patrón de cola de mensajes y el paradigma de programación MapReduce a través de los navegadores web sin interferir con la experiencia del usuario del sitio ni añadir una instalación de software adicional. En este estudio todavía no se aborda la privacidad de los datos en el borde. Cada tarea indica la edad del modelo y qué parte de los datos debe utilizar lo que permite la reproducibilidad del entrenamiento independientemente del número de trabajadores. A pesar de las limitaciones del canal de comunicación, demostramos la viabilidad y escalabilidad de nuestro enfoque.

En nuestro tercer trabajo [84] (ver secciones 3.3 y 4.3), nos centramos en los objetivos **G2** y **G3**. Las principales aportaciones de este trabajo son **LIT** y **ASYN**. Definimos los retos y las características deseables de VC4FL (véase la sección 3.3). Abordamos el problema de los datos que permanecen localmente en los dispositivos de borde. Debido a la volatilidad de las conexiones y desconexiones de los voluntarios, el conjunto de datos disponible durante el entrenamiento cambia, tratándose de conjuntos de datos desbalanceados y *non-i.i.d.* Propusimos un algoritmo asíncrono para FEEL, que adapta el entrenamiento a los clientes poco fiables que se incorporan y abandonan el cómputo cuando hay pocos trabajadores -o incluso pueden reducirse a cero- disponibles. Se diseñó, puso en práctica y evaluó una plataforma de software que satisface los retos especificados y las características deseables. A continuación, demostramos que la plataforma que utiliza un número determinado de dispositivos poco fiables y heterogéneos (HW y SW) se adapta bien a este entorno cambiante obteniendo una precisión numérica y una puntuación CK similares a las configuraciones actuales que utilizan una plataforma estática para el aprendizaje. Mostramos cómo la plataforma se recupera de las desconexiones inesperadas de los dispositivos de los voluntarios. Además, mostramos algunos hallazgos interesantes sobre la configuración de los parámetros  $P$  y  $Z$  cuando varía el número de trabajadores disponibles  $|E|$ . Sin embargo, debemos investigar nuevas técnicas para hacer frente a diferentes casuísticas en la distribución de los datos y las conexiones y desconexiones de los voluntarios. Hasta donde los autores saben, este trabajo es el primero sobre entrenamiento distribuido de NN en el que colaboran navegadores web y procesos de python, demostrando la interoperabilidad [70, 82] de nuestra propuesta.

En nuestro cuarto trabajo [87] (ver secciones 3.3 y 4.4), nos centramos en los objetivos **G2** y **G3**. Las principales contribuciones son **LIT** y **MOD**. Uno de los mayores retos de FL es la sobrecarga de comunicaciones. Por ello, en este trabajo intentamos abordar este problema, es decir, reducir la sobrecarga de comunicaciones sin reducir o incluso aumentar la precisión del modelo entrenado. Resumimos las técnicas más comunes en la literatura para resolver el problema de la sobrecarga de comunicaciones en FL (ver Sección 3.3). Formulamos y modelamos el problema de la



sobrecarga de comunicación en el aprendizaje federado como un problema multiobjetivo (FL-COP). Sugerimos emplear EAs para la optimización de funciones multiobjetivo para resolver el FL-COP. Nuestro método mejora la precisión en comparación con la configuración de comunicación máxima, al tiempo que reduce las comunicaciones de 10 a 2000 dependiendo de la arquitectura de la red neuronal. Hasta donde saben los autores, este es el primer trabajo en el que se formula y modela este problema combinando estas técnicas dentro del mismo estudio, proponiendo algoritmos genéticos multiobjetivo y probando su idoneidad para este fin. Abre la puerta a un amplio abanico de trabajos futuros, como el diseño de algoritmos que adapten dinámicamente estos hiperparámetros durante el entrenamiento.

Por último, los resultados de esta tesis demuestran que el despliegue de la inteligencia ubicua distribuida en los dispositivos de borde es factible y útil, ofreciendo una mejor comprensión de cómo debe ser dicha inteligencia distribuida, los problemas asociados que existen y cómo deben abordarse estos problemas.

El trabajo de esta tesis ha sido arduo y laborioso. Durante su desarrollo, hemos utilizado una gran variedad de lenguajes de programación y tecnologías que nos ha llevado tiempo aprender. Varios algoritmos y problemas se han implementado en dispositivos reales de borde distribuidos con recursos limitados no diseñados para este fin. Se ha realizado una profunda investigación sobre las tecnologías más apropiadas para hacer que estos dispositivos trabajen juntos y logren resultados notables. Al final, este trabajo ha dado sus frutos con **dos publicaciones en revistas indexadas en JCR (Q1)** con un alto factor de impacto. Además, dos publicaciones más en congresos, **uno en un congreso nacional** y **otro en un congreso internacional** en el que el doctorando también obtuvo el **premio al estudiante destacado**.

Sin embargo, aún quedan muchos retos y preguntas para el trabajo futuro. A partir de estos resultados, salieron a la luz muchas perspectivas de investigación que pretendemos explorar. Algunas de estas líneas de investigación pueden resumirse como sigue:

- Investigar en mayor profundidad la gestión de la asignación de tareas en función del rendimiento de los dispositivos heterogéneos y la distribución de los datos utilizados por cada dispositivo.
  - Diseñar algoritmos de selección inteligente para seleccionar los dispositivos que participan en la siguiente ronda de comunicación en función de su distribución de datos y rendimiento.
  - Diseñar algoritmos de optimización para asignar un número personalizado de pasos locales a cada dispositivo en función de su rendimiento y probabilidad de fallo.
  - Diseñar algoritmos de agregación adaptativos que agreguen modelos de dispositivos heterogéneos en función del número de pasos locales realizados y de la distribución de los datos utilizados por cada dispositivo.
- Investigar más a fondo la reducción del tamaño del modelo ML antes, durante y después del entrenamiento sin reducir la precisión.
  - Investigar la optimización en tiempo real de los parámetros de aprendizaje federado durante el proceso de aprendizaje.
  - Optimizar los parámetros de aprendizaje federado añadiendo nuevos objetivos, como el consumo de energía, comparando un número más significativo de algoritmos multiobjetivo, y proponiendo nuevos operadores

- Diseñar nuevos algoritmos adaptativos que optimicen la sobrecarga de comunicación en FL durante el aprendizaje, por ejemplo, utilizando técnicas de cuantificación y esparsificación.
- Diseñar una plataforma de aprendizaje multinivel con varias capas de aprendizaje, es decir, en el borde, niebla y nube.
- Investigar nuevas técnicas de optimización distribuida utilizando los datos locales y privados de los usuarios para problemas complejos.
- Investigar nuevas técnicas de FL integradas con el aprendizaje por refuerzo con aplicaciones [94] como la asignación de recursos, las redes de comunicación, la optimización y control, y la detección de ataques, entre otras.

## Appendix D

# Publications Supporting This Thesis

This appendix presents a list of publications that we have produced during our doctoral research. These publications are the result of the study carried out to achieve the objectives of the thesis. First, we present the papers published in journals that support this thesis. Second, we show the studies published in international and national conference proceedings. We list the publications in chronological order. With the publication details, we present the available metrics of each publication, i.e. the journal impact factor (JIF).

### D.1 Publications in Journals

1. Morell, J. Á., Camero, A., & Alba, E. (2019). JSDoop and TensorFlow.js: Volunteer Distributed Web Browser-based Neural Network Training. *IEEE Access*, 7, 158671-158684. DOI: [10.1109/ACCESS.2019.2950287](https://doi.org/10.1109/ACCESS.2019.2950287)
  - JCR COMPUTER SCIENCE, INFORMATION SYSTEMS - SCIE, **Q1**, ranking 35/156, 2019 JIF = 3.745
2. Morell, J. Á., & Alba, E. (2022). Dynamic and Adaptive Fault-tolerant Asynchronous Federated Learning Using Volunteer Edge Devices. *Future Generation Computer Systems*. DOI: [10.1016/j.future.2022.02.024](https://doi.org/10.1016/j.future.2022.02.024)
  - JCR COMPUTER SCIENCE, THEORY & METHODS - SCIE, **Q1**, ranking 7/110, 2020 JIF = 7.187

### D.2 Publications in the Proceedings of International and National Conferences

1. Morell, J. Á., & Alba, E. (2018). Running Genetic Algorithms in the Edge: A First Analysis. In *Conference of the Spanish Association for Artificial Intelligence* (pp. 251-261). Springer, Cham. [10.1007/978-3-030-00374-6\\_24](https://doi.org/10.1007/978-3-030-00374-6_24)
2. Morell, J. Á., Dahi, ZA, Chicano, F, Luque, G & Alba, E. (2022). Optimising Communication Overhead in Federated Learning Using NSGA-II. *International Conference on the Applications of Evolutionary Computation*. Springer, Cham. [10.1007/978-3-031-02462-7\\_21](https://doi.org/10.1007/978-3-031-02462-7_21)
  - Outstanding Student Award.

### D.3 Publications Indirectly Related to This Thesis

1. Morell, J. A., & Alba, E. (2017). Distributed Genetic Algorithms on Portable Devices for Smart Cities. In International Conference on Smart Cities (pp. 51-62). Springer, Cham. DOI: [10.1007/978-3-319-59513-9\\_6](https://doi.org/10.1007/978-3-319-59513-9_6)
2. Dahi, Z. A., Morell, J. Á. (2022). Models for Coverage Optimisation in Cellular Networks: Review, Analysis and Perspectives. IEEE 9th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT) (pp. 312-319). IEEE. DOI: [10.1109/SETIT54465.2022.9875463](https://doi.org/10.1109/SETIT54465.2022.9875463)

# References

- [1] E Alba, JF Aldana, and JM Troya. "Genetic algorithms as heuristics for optimizing ANN design". In: *Artificial Neural Nets and Genetic Algorithms*. Springer. 1993, pp. 683–690.
- [2] Enrique Alba and Bernabé Dorronsoro. *Cellular genetic algorithms*. Vol. 42. Springer Science & Business Media, 2009.
- [3] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow. "Parallel metaheuristics: recent advances and new trends". In: *International Transactions in Operational Research* 20.1 (2013), pp. 1–48.
- [4] Albert D Alexandrov et al. "Superweb: Towards a global web-based parallel computing infrastructure". In: *Proceedings 11th International Parallel Processing Symposium*. IEEE. 1997, pp. 100–106.
- [5] Dan Alistarh et al. "QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA, 2017, 1707–1718. ISBN: 9781510860964.
- [6] David P Anderson. "Boinc: A platform for volunteer computing". In: *Journal of Grid Computing* 18.1 (2020), pp. 99–122.
- [7] Andrea Apicella et al. "A survey on modern trainable activation functions". In: *Neural Networks* 138 (2021), pp. 14–32.
- [8] Wolfgang Banzhaf. "The "molecular" traveling salesman". In: *Biological Cybernetics* 64.1 (1990), pp. 7–14.
- [9] Leonid Batyuk et al. "Developing and benchmarking native linux applications on android". In: *International Conference on Mobile Wireless Middleware, Operating Systems, and Applications*. Springer. 2009, pp. 381–392.
- [10] Tal Ben-Nun and Torsten Hoefler. "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis". In: *ACM Computing Surveys (CSUR)* 52.4 (2019), pp. 1–43.
- [11] Adam Bergkvist et al. "WebRTC 1.0: Real-time communication between browsers". In: *Working draft, W3C* 91 (2012).
- [12] Keith Bonawitz et al. "Towards federated learning at scale: System design". In: *Proceedings of Machine Learning and Systems* 1 (2019), pp. 374–388.
- [13] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. "A survey on optimization metaheuristics". In: *Information sciences* 237 (2013), pp. 82–117.
- [14] Jose Caceres-Cruz et al. "Rich vehicle routing problem: Survey". In: *ACM Computing Surveys (CSUR)* 47.2 (2014), pp. 1–28.
- [15] Noam Camiel. "The POPCORN project: Distributed computation over the Internet in Java". In: *6th International World Wide Web Conference*. 1997.
- [16] Christophe Cérin and Gilles Fedak. *Desktop grid computing*. Chapman and Hall/CRC, 2012.

- [17] Yang Chen, Xiaoyan Sun, and Yaochu Jin. "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation". In: *IEEE transactions on neural networks and learning systems* 31.10 (2019), pp. 4229–4238.
- [18] Yujing Chen et al. "Asynchronous online federated learning for edge devices with non-iid data". In: *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 15–24.
- [19] Zichen Chen et al. "A Gamified Research Tool for Incentive Mechanism Design in Federated Learning". In: *Federated Learning*. Springer, 2020, pp. 168–175.
- [20] Pawel Chorazyk et al. "Lightweight volunteer computing platform using web workers". In: *Procedia Computer Science* 108 (2017), pp. 948–957.
- [21] Bernd O Christiansen et al. "Javelin: Internet-based parallel computing using Java". In: *Concurrency: Practice and Experience* 9.11 (1997), pp. 1139–1160.
- [22] N Christofides, A Mingozzi, and P Toth. "Loading problems". In: *N. Christofides and al., editors, Combinatorial Optimization* (1979), pp. 339–369.
- [23] S Barry Cooper. *Computability theory*. Chapman and Hall/CRC, 2017.
- [24] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2022.
- [25] Georges A Croes. "A method for solving traveling-salesman problems". In: *Operations research* 6.6 (1958), pp. 791–812.
- [26] Harold J Curnow and Brian A. Wichmann. "A synthetic benchmark". In: *The Computer Journal* 19.1 (1976), pp. 43–49.
- [27] Charles Cusack, Chris Martens, and Priyanshu Mutreja. "Volunteer computing using casual games". In: *Proceedings of Future Play 2006 International Conference on the Future of Game Design and Technology*. 2006, pp. 1–8.
- [28] Reginald Cushing et al. "Distributed computing on an ensemble of browsers". In: *IEEE Internet Computing* 17.5 (2013), pp. 54–61.
- [29] George B Dantzig and John H Ramser. "The truck dispatching problem". In: *Management science* 6.1 (1959), pp. 80–91.
- [30] Rhiju Das et al. "Structure prediction for CASP7 targets using extensive all-atom refinement with Rosetta@ home". In: *Proteins: Structure, Function, and Bioinformatics* 69.S8 (2007), pp. 118–128.
- [31] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.
- [32] K. Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Trans. on Ev. Comp.* 6.2 (2002), pp. 182–197.
- [33] David Dias and Luís Veiga. "BrowserCloud.js-A federated community cloud served by a P2P overlay network on top of the web platform". In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC*. Vol. 18. Cite-seer. 2015, pp. 2175–2184.
- [34] Gordana Dodig-Crnkovic. "Scientific methods in computer science". In: *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia*. 2002, pp. 126–130.
- [35] Jack J Dongarra et al. *LINPACK users' guide*. Siam, 1979.

- [36] Muhammad Nouman Durrani and Jawwad A Shamsi. "Volunteer computing: requirements, challenges, and solutions". In: *Journal of Network and Computer Applications* 39 (2014), pp. 369–380.
- [37] L Eshelman. "On crossover as an evolutionarily viable strategy". In: *Proceedings of the Fourth International Conference on Genetic Algorithms*. 1991, pp. 61–68.
- [38] Tomasz Fabisiak and Arkadiusz Danilecki. "Browser-based harnessing of voluntary computational power". In: *Foundations of Computing and Decision Sciences* 42.1 (2017), pp. 3–42.
- [39] Gilles Fedak et al. "Xtremweb: A generic global computing system". In: *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE. 2001, pp. 582–587.
- [40] Matthew Finnegan. "Boeing 787s to create half a terabyte of data per flight, says Virgin Atlantic". In: *Computerworld UK* 6 (2013).
- [41] David B Fogel. "An evolutionary approach to the traveling salesman problem". In: *Biological Cybernetics* 60.2 (1988), pp. 139–144.
- [42] Lance Fortnow. "The status of the P versus NP problem". In: *Communications of the ACM* 52.9 (2009), pp. 78–86.
- [43] Ian Foster. *Designing and building parallel programs: concepts and tools for parallel software engineering*. Addison-Wesley Longman Publishing Co., Inc., 1995.
- [44] Geoffrey C Fox and Wojtek Furmanski. "PETAOPS and EXAOPS: Supercomputing on the Web". In: *IEEE Internet Computing* 1.2 (1997), pp. 38–46.
- [45] David E Golberg. "Genetic algorithms in search, optimization, and machine learning". In: *Addion wesley* 1989.102 (1989), p. 36.
- [46] David E Goldberg, Kalyanmoy Deb, and Jeffrey Horn. "Massive multimodality, deception, and genetic algorithms". In: *Urbana* 51 (1992), p. 61801.
- [47] Chao Gong et al. "Intelligent Cooperative Edge Computing in Internet of Things". In: *IEEE Internet of Things Journal* 7.10 (2020), pp. 9372–9382.
- [48] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [49] Andreas Haas et al. "Bringing the web up to speed with WebAssembly". In: *ACM SIGPLAN Notices*. Vol. 52. 6. ACM. 2017, pp. 185–200.
- [50] Andrew Hard et al. "Federated learning for mobile keyboard prediction". In: *arXiv preprint arXiv:1811.03604* (2018).
- [51] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.
- [52] Vishakh Hegde and Sheema Usmani. "Parallel and distributed deep learning". In: *Tech. report, Stanford University*. 2016.
- [53] Attila Hegyi et al. "Application orchestration in mobile edge cloud: placing of iot applications to the edge". In: *Foundations and Applications of Self\* Systems, IEEE International Workshops on*. IEEE. 2016, pp. 230–235.
- [54] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

- [55] John H Holland. "Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence". In: *Ann Arbor, MI: University of Michigan Press* (1975), pp. 439–444.
- [56] Tyler Hunt et al. "Ryoan: A distributed sandbox for untrusted computation on secret data". In: *ACM Transactions on Computer Systems (TOCS)* 35.4 (2018), p. 13.
- [57] IBM. *World Community Grid*. <https://www.worldcommunitygrid.org/>. Online; accessed 28 May 2019.
- [58] Katarzyna Janocha and Wojciech Marian Czarnecki. "On loss functions for deep neural networks in classification". In: *arXiv preprint arXiv:1702.05659* (2017).
- [59] Arash Baratloo Mehmet Karaul Zvi Kedem and Peter Wyckoff. "Charlotte: Metacomputing on the web". In: *In The 9th ICPDCS International Conference on Parallel and Distributed Computing and Systems*. Citeseer. 1996.
- [60] Ashfaq A. Khokhar et al. "Heterogeneous computing: Challenges and opportunities". In: *Computer* 26.6 (1993), pp. 18–27.
- [61] Jon Klein and Lee Spector. "Unwitting distributed genetic programming via asynchronous JavaScript and XML". In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM. 2007, pp. 1628–1635.
- [62] Jakub Konečný et al. "Federated learning: Strategies for improving communication efficiency". In: *arXiv preprint arXiv:1610.05492* (2016).
- [63] Fumikazu Konishi et al. "Rabc: A conceptual design of pervasive infrastructure for browser computing based on ajax technologies". In: *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*. IEEE. 2007, pp. 661–672.
- [64] Eric Korpela et al. "SETI@home—massively distributed computing for SETI". In: *Computing in science & engineering* 3.1 (2001), p. 78.
- [65] Ralf Lämmel. "Google's MapReduce programming model—Revisited". In: *Science of computer programming* 70.1 (2008), pp. 1–30.
- [66] Averill M Law, W David Kelton, and W David Kelton. *Simulation modeling and analysis*. Vol. 3. McGraw-Hill New York, 2000.
- [67] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521 (2015), pp. 436–444.
- [68] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [69] Yi Liu et al. "Federated learning in the sky: Aerial-ground air quality sensing framework with UAV swarms". In: *IEEE Internet of Things Journal* (2020).
- [70] Yu Liu et al. "Enhancing the interoperability between deep learning frameworks by model conversion". In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2020, pp. 1320–1330.
- [71] Yunlong Lu et al. "Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles". In: *IEEE Transactions on Vehicular Technology* 69.4 (2020), pp. 4298–4311.



- [72] Zhihan Lv, Dongliang Chen, and Amit Kumar Singh. “Big data processing on volunteer computing”. In: *ACM Transactions on Internet Technology* 21.4 (2021), pp. 1–20.
- [73] Tommy MacWilliam and Cris Cecka. “CrowdCL: Web-based volunteer computing with WebCL”. In: *2013 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE. 2013, pp. 1–6.
- [74] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*. Elsevier, 1977.
- [75] Ruben Mayer and Hans-Arno Jacobsen. “Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools”. In: *ACM Computing Surveys (CSUR)* 53.1 (2020), pp. 1–37.
- [76] Brendan McMahan et al. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282.
- [77] H Brendan McMahan et al. “Advances and Open Problems in Federated Learning”. In: *Foundations and Trends® in Machine Learning* 14.1 (2021).
- [78] Frank H McMahon. *The Livermore Fortran Kernels: A computer test of the numerical performance range*. Tech. rep. Lawrence Livermore National Lab., CA (USA), 1986.
- [79] Edward Meeds et al. “MLitB: machine learning in the browser”. In: *PeerJ Computer Science* 1 (2015), e11.
- [80] Qi Meng et al. “Convergence analysis of distributed stochastic gradient descent with shuffling”. In: *Neurocomputing* 337 (2019), pp. 46–57.
- [81] Juan Julián Merelo-Guervós et al. “Asynchronous distributed genetic algorithms with Javascript and JSON”. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, pp. 1372–1379.
- [82] Tommi Mikkonen, Cesare Pautasso, and Antero Taivalsaari. “Isomorphic Internet of Things Architectures With Web Technologies”. In: *Computer* 54.7 (2021), pp. 69–78.
- [83] Roberto Morabito et al. “LEGIoT: A lightweight edge gateway for the Internet of Things”. In: *Future Generation Computer Systems* 81 (2018), pp. 1–15.
- [84] José Ángel Morell and Enrique Alba. “Dynamic and adaptive fault-tolerant asynchronous federated learning using volunteer edge devices”. In: *Future Generation Computer Systems* 133 (2022), pp. 53–67.
- [85] José Ángel Morell and Enrique Alba. “Running genetic algorithms in the edge: A first analysis”. In: *Conference of the Spanish Association for Artificial Intelligence*. Springer. 2018, pp. 251–261.
- [86] José Ángel Morell, Andrés Camero, and Enrique Alba. “JSDoop and TensorFlow.js: Volunteer Distributed Web Browser-Based Neural Network Training”. In: *IEEE Access* 7 (2019), pp. 158671–158684.
- [87] José Ángel Morell et al. “Optimising Communication Overhead in Federated Learning Using NSGA-II”. In: *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer. 2022, pp. 317–333.
- [88] Philipp Moritz et al. “Ray: A distributed framework for emerging AI applications”. In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 561–577.

- [89] Virraji Mothukuri et al. "A survey on security and privacy of federated learning". In: *Future Generation Computer Systems* 115 (2021), pp. 619–640.
- [90] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. "Interoperability in internet of things: Taxonomies and open challenges". In: *Mobile Networks and Applications* 24.3 (2019), pp. 796–809.
- [91] Ibrahim Hassan Osman. "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem". In: *Annals of operations research* 41.4 (1993), pp. 421–451.
- [92] Claus Pahl et al. "A container-based edge cloud paas architecture based on raspberry pi clusters". In: *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), Aug 22-24, Vienna, Austria*. IEEE. 2016, pp. 117–124.
- [93] Jason Posner et al. "Federated Learning in Vehicular Networks: Opportunities and Solutions". In: *IEEE Network* (2021).
- [94] Jiaju Qi et al. "Federated reinforcement learning: techniques, applications, and open challenges". In: *arXiv preprint arXiv:2108.11887* (2021).
- [95] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).
- [96] Luis FG Sarmenta. "Bayanihan: Web-based volunteer computing using Java". In: *International Conference on Worldwide Computing and Its Applications*. Springer. 1998, pp. 444–461.
- [97] Mahadev Satyanarayanan et al. "Edge analytics in the internet of things". In: *IEEE Pervasive Computing* 14.2 (2015), pp. 24–31.
- [98] Alimohammad Shahri et al. "Gamification for volunteer cloud computing". In: *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE. 2014, pp. 616–617.
- [99] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- [100] Ohad Shamir. "Without-replacement sampling for stochastic gradient methods". In: *Advances in neural information processing systems*. 2016, pp. 46–54.
- [101] Micah J Sheller et al. "Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data". In: *Scientific reports* 10.1 (2020), pp. 1–12.
- [102] Weisong Shi et al. "Edge computing: Vision and challenges". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646.
- [103] Julius Sim and Chris C Wright. "The kappa statistic in reliability studies: use, interpretation, and sample size requirements". In: *Physical therapy* 85.3 (2005), pp. 257–268.
- [104] Michael Sipser. *Introduction to the Theory of Computation*. Cengage, 2021.
- [105] Daniel Smilkov et al. "Tensorflow.js: Machine learning for the web and beyond". In: *Proceedings of Machine Learning and Systems* 1 (2019), pp. 309–321.
- [106] Jason W Steinmetz. "An Intuitively Complete Analysis of Godel's Incompleteness". In: *arXiv preprint arXiv:1512.03667* (2015).
- [107] D Stinson. *An introduction to the design and analysis of algorithms*. The Charles Babbage Research Centre, St. Pierre, 1985.

- [108] Jaspal Subhlok et al. "Resilient parallel computing on volunteer PC grids". In: *Concurrency and Computation: Practice and Experience* 30.18 (2018), e4478.
- [109] Afaf Tak and Soumaya Cherkaoui. "Federated Edge Learning: Design Issues and Challenges". In: *IEEE Network* (2020).
- [110] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. Vol. 74. John Wiley & Sons, 2009.
- [111] Douglas Thain, Todd Tannenbaum, and Miron Livny. "Distributed computing in practice: the Condor experience". In: *Concurrency and computation: practice and experience* 17.2-4 (2005), pp. 323–356.
- [112] Mark Van Rijmenam. *Self-driving cars will create 2 petabytes of data, what are the big data opportunities for the car industry*. 2017.
- [113] Joaquin Vanschoren et al. "OpenML: networked science in machine learning". In: *ACM SIGKDD Explorations Newsletter* 15.2 (2014), pp. 49–60.
- [114] Shiqiang Wang et al. "Adaptive federated learning in resource constrained edge computing systems". In: *IEEE Journal on Selected Areas in Communications* 37.6 (2019), pp. 1205–1221.
- [115] Jianqiao Wangni et al. "Gradient Sparsification for Communication-Efficient Distributed Optimization". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada, 2018, 1306–1316.
- [116] Reinhold P Weicker. "Dhrystone: a synthetic systems programming benchmark". In: *Communications of the ACM* 27.10 (1984), pp. 1013–1030.
- [117] Mark Weiser. "The Computer for the 21 st Century". In: *Scientific american* 265.3 (1991), pp. 94–105.
- [118] Mark Weiser and John Seely Brown. "The coming age of calm technology". In: *Beyond calculation*. Springer, 1997, pp. 75–85.
- [119] Sean R Wilkinson and Jonas S Almeida. "QMachine: commodity supercomputing in web browsers". In: *BMC bioinformatics* 15.1 (2014), p. 176.
- [120] Jinjin Xu et al. "Ternary compression for communication-efficient federated learning". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).
- [121] Yunfan Ye et al. "Edgefed: optimized federated learning based on edge computing". In: *IEEE Access* 8 (2020), pp. 209191–209198.
- [122] Chen Zhang et al. "A survey on federated learning". In: *Knowledge-Based Systems* 216 (2021), p. 106775.
- [123] Qingfu Zhang and Hui Li. "MOEA/D: A multiobjective evolutionary algorithm based on decomposition". In: *IEEE Transactions on evolutionary computation* 11.6 (2007), pp. 712–731.
- [124] Ruiliang Zhang and James Kwok. "Asynchronous distributed ADMM for consensus optimization". In: *International conference on machine learning*. 2014, pp. 1701–1709.
- [125] Yuhao Zhou, Qing Ye, and Jian Cheng Lv. "Communication-Efficient Federated Learning with Compensated Overlap-FedAvg". In: *IEEE Transactions on Parallel and Distributed Systems* (2021).
- [126] Hangyu Zhu and Yaochu Jin. "Multi-objective evolutionary federated learning". In: *IEEE transactions on neural networks and learning systems* 31.4 (2019), pp. 1310–1322.