Developing a wireless distributed embedded machine learning application for the Arduino Portenta H7

23rd January 2023

David Ramos Domínguez

Computing specialization

Director: Felix Freitag

Contents

List of tables

List of figures

1. Introduction

1.1. Context

1.1.1. Introduction

This project is my Degree Final Project for my Degree in Informatics Engineering in the Computing specialization, which I am studying at the Facultat d'Informàtica de Barcelona (FIB). I will be supervised by Felix Freitag, who is part of the Distributed Systems Group.

In this project I will develop a machine learning application for an embedded device, specifically the Arduino Portenta H7, which is a board with a built in microcontroller and the necessary circuits to be able to add many accessories, like cameras, microphones or LEDs.

1.1.2. Terms and Concepts

TinyML: also called embedded machine learning, is a branch of machine learning based on using the microcontrollers of embedded systems, processing data on-device. The main benefits are that microcontrollers consume very little power, and they reduce the computing time to make an inference because they don't have to communicate with any servers, the inference is done locally [1].

Microcontroller: small integrated circuit designed for a specific task in an embedded system. Microcontrollers are designed for embedded systems, while microprocessors are designed for computer systems [2].

Arduino: the company that developed the Arduino boards. They are an open-source hardware and software company, whose main objective is making products that are accessible and easy to use [3]. Their main product is the Arduino boards and accessories for the boards like sensors or tools. In their web they offer a free IDE to program their boards [4]. They also have their own forum where users can ask questions to professionals and other users [5].

Arduino Software: also known as Arduino IDE. It's an IDE that Arduino made to let users write programs for their boards. It has a simple design and comes with the libraries necessary to use many features of the boards. It's also used to load the sketches into the boards [4].

Sketch: a program written in the Arduino programming language to be executed by a board. Sketches are saved using the extension .ino, and they must have the functions setup() and loop(). Setup() is called only once when the sketch starts, and is used mostly for initializations. Loop() executes after setup finishes and repeats its code until the board is powered off.

1.1.3. Problem to be resolved

The main problem that we want to address is the characterization of the Portenta H7 for an embedded machine learning application, specifically:

- Options for interaction between the two cores of the Portenta H7
- Evaluating the machine learning task distribution between the two cores
- Usage of Portenta H7 communication interfaces as part of the machine learning application
- Understanding the performance of an embedded machine learning application
- Assessing the Portenta H7 as a Smart IOT device

1.1.4. Stakeholders

Since this project will be in an open source repository it will benefit anyone who is interested in this topic, but specially:

- The Distributed Systems Group in which the supervisor of this project is participating, since they study Federated learning in low capacity devices.
- The TinyML community, since they will obtain a complete application to experiment with.
- University students, since it will provide an example and the files necessary to test it for anybody who wants to do a similar project.

1.2. Justification

First I will explain why it is worth it to use TinyML, and then I will explain the benefits of Portenta H7 over other microcontroller boards.

1.2.1. TinyML

There is two main ways to implement machine learning on hardware, the Edge computing method and the Cloud computing method.

The Edge computing method is also called embedded machine learning or TinyML, which is based on running the machine learning models on the same embedded system as the one that obtains the data.

The Cloud computing method is based on having the devices communicate with a server which is in the cloud and has the machine learning model, while the devices only obtain the data with sensors, send it to the server and receive the inference from the server. Since the devices are embedded they are in the edge, separated from the cloud [6].

Since embedded machine learning doesn't have to send data to a server it won't have transmission costs and lower latency for each inference, and the data will be at lower risk of being compromised. They can also be used in isolated locations without internet connection. Additionally, TinyML is based on small microcontrollers, which means that they are cheaper to produce than computer systems and have low energy consumption, normally in the order of milliwatts [7].

However, since the model has to be in the embedded system they will be limited by the memory space of the device compared to a remote server, and it will be harder to fix

any issues and make modifications because you can't do it remotely like a server, you need to access the embedded system physically [8].

Since TinyML is faster, cheaper and can be used anywhere I believe that it's worth developing, even if making models that fit the limited space is more complicated.

1.2.2. Portenta H7

The main alternative board that Arduino offers is the Nano 33 BLE. Before Portenta it was the model with the biggest memory, having 32 times the size of the previous model, the Arduino Uno. It has 1MB of program memory and 256 KB of RAM, and an ARM® Cortex®-M4 core running at 64 MHz. It's the smallest board available, with only 45x18mm. It's also one of the cheapest boards in the Arduino store, costing 22,80 euros [9].

The main selling point of Potenta H7 is the fact that it has two cores, allowing the board to do two different tasks in parallel. It has 2MB of program memory and 1MB of RAM. It has an Arm® Cortex®-M7 core running at up to 480 MHz and an Arm® 32-bit Cortex®M4 core running at up to 240 MHz. It measures 66x25mm. However, it's one of the most expensive boards, costing 99,00 euros [10].

Although the price increase of the Portenta H7 is very noticeable, I believe that the bigger memory and the dual core system make an interesting board for experimentation, and it hasn't been as explored as the Nano board, so there's room for innovation.

1.3. Scope

1.3.1. Objectives and sub objectives

There will be three main objectives:

- Learn how to work with a microcontroller board: since this is a new field of study for me I will have to research the steps to develop any application with a microcontroller. This will have the following sub objectives:
    - Find the IDEs that are compatible with microcontrollers and install the ones that seem appropriate for my project.
    - Research how to write code that will be executed by a microcontroller board.
    - Learn how to use the previously mentioned IDEs to write programs into my Portenta H7 board.
    - Find what kind of accessories can be added to my board, like a microphone, buttons, etc.
    - Learn how to connect the previously mentioned accessories to the Portenta H7.

- Research existing TinyML applications: Seeing finished TinyML applications will show me different ways to use microcontrollers, and it could help me decide what kind of application I will make. This will have the following sub objective:
    - Search for applications compatible with Portenta H7 so I can do practical tests with the board that will be used in the project and see the code of complete applications.
- Implement my own machine learning application with Portenta H7: once I have completed the previous objectives I will be able to create a complete machine learning application that will be executed by a Portenta H7 board and address the problems to be solved defined in section 1.3.

1.3.2. Requirements

The main functional requirement will be that any user who wants to use my application must have Arduino Portenta H7 boards, since it will use two cores which is a feature exclusive to this model of board.

A non-functional requirement is that the application should be easy to use by any user with the necessary tools. To add to the usability of the application it should have a low execution time to facilitate testing.

1.3.3. Obstacles and risks

Unknown errors when programming: since Portenta H7 is relatively new it can be hard to find specific information about it. I would have to search in the Arduino documentation and its forum. I could also ask my tutor or make my own post in the forum.

Obsolete documentation: since TinyMl is constantly being developed there might be documentation that hasn't been updated with the most recent changes, which could lead to having incorrect information for the current version.

Library updates and changes during the development of the project: library modifications could lead to compilation errors if they change any names, and most likely the documentation won't be updated immediately, as mentioned previously.

Hardware problems with the board: without a working board the applications can't be tested. The director has offered additional boards if necessary, so this wouldn't cause much delay.

2. Project management

2.1. Methodology and rigor

2.1.1. Work methods

The procedure that I will follow for this project is making a basic TinyML application to begin, and once it's finished add elements to iteratively make increasingly complex

models. For each completed version of the model I will show my supervisor the results and discuss the different ways to proceed into the next version of the model.

The methodology that fits best this iterative work process is Agile, where making each version of my application is a sprint, plus there will be two additional sprints at the start for the first two main objectives explained in section 3.1.

2.1.2. Monitoring tools

During the development of this project I will be regularly informing my project director of my progress using Gmail, and if deemed necessary we will hold a videoconference using Google Meet or Jitsi to speak and show my code using Github repositories or documents using screen share.

I have pre-selected Trello to manage the progress of my project

2.2. Task description

This project has a start date of September 5, 2022, and should be finished before the oral defense, which will take place during the week of January 23, 2023.

I will spend 4 hours per day on average working on the project, and I will spend a total of 630 hours for the whole project.

For each task I will define a code, a description, the estimated time required, and any dependencies or specific resources if necessary.

2.2.1. T1: Project management

Tasks focused on the organization and management of the project:

- T1.1 Context and scope: defining the concepts necessary to understand the project, giving a justification of the project, its scope and the methodology that will be used. I will dedicate 30h to this task.
- T1.2 Planning: defining a plan for the different phases of the project. I will dedicate 15h to this task.
- T1.3 Budget: making a budget with the economic costs of the project. It will take 15h.
- T1.4 Sustainability report: studying other kinds of cost, like the environmental impact. I will dedicate 10h.
- T1.5 Meetings: The scheduled meetings with the director of the project to inform about the state of the project and to define goals to reach before the next meeting. Each meeting will last between 1 and 2h, so for the whole project we will spend 20h. The director of the project is a necessary resource.

2.2.2. T2: Learn to work with Portenta H7

The tasks necessary to learn how to use and program a microcontroller board without any previous knowledge. For these tasks the Portenta H7 board is necessary.

- T2.1 Install IDEs: finding the most used programs to make microcontroller applications and installing them. It will take 5h.
- T2.2 Research programming language: finding the language and format of microcontroller programs, researching the libraries exclusive to Arduino and learning about the usage of their functions. I will dedicate 10h to this task.
- T2.3 Experiment programming with the board: reading tutorials, trying examples and writing simple programs for the Portenta H7, using the IDE to upload them to the board, and learning the different capabilities of the Portenta H7 board. This is the most important part of the learning process, so I will dedicate 40h to this task. It will depend on having completed the two previous tasks.
- T2.4 Explore board accessories: researching what kind of accessories, like microphones or cameras, can be used with the Portenta H7 board. It will take 15h.
- T2.5 Research accessory applications: after finding the compatible accessories, search for tutorials and examples to learn what can be done with them and see how to write code to interact with them from the board. I will dedicate 20h to this task. It will depend on the previous task.

2.2.3. T3: Research existing TinyML applications

In order to see what can be done with TinyML I will search for complete TinyML applications, which will be helpful as examples and also to prevent making an application that already exists.

- T3.1 Explore different programs: Obtaining programs from the Arduino website, from Github or provided by the project director. I will spend 20h on this task

I will divide the applications in two groups: the ones that can be executed with the Portenta H7 board and the ones that can't. Some of the reasons why they don't execute are: libraries specifically designed for one kind of board, outdated code that isn't compatible with current libraries, using specific devices that aren't compatible with Portenta…

- T3.2 Test the compatible programs with the Portenta board: Since these work with the board of this project they can be tested and they can be used as a reference of code that executes, which could be useful in case of unknown errors. I will dedicate more time to this task: 30h. For testing the board is necessary.
- T3.3 Investigate the code of applications incompatible with Portenta: although they can't be tested and their codes can't be used as reference, they might have interesting ideas that could be adapted to fit in this project. It will take 20h.

T3.2 and T3.3 both depend on the programs obtained in T3.1.

2.2.4. T4: Implement my own machine learning application with Portenta H7

These tasks define what will be done in each iteration or sprint of this project:

- T4.1 Define the program to implement: 5h.
- T4.2 Connect the necessary hardware: 5h. Depends on T4.1.
- T4.3 Design the machine learning model: 20h. Depends on T4.1.
- T4.4 Write the code: 30h. Depends on T4.2 and 14.3.
- T4.5 Generate datasets: 10h. Depends on T4.2.
- T4.6 Test with Portenta H7: 20h. Depends on T4.4 and T4.5.

For each sprint all of these tasks will have to be done again. The Portenta H7 is a necessary resource.

2.2.5. T5: Documentation

Task related to writing the project documents. It will probably take over 80h.

2.2.6. T6: Oral defense preparation

Preparing the material of the presentation, rehearsing, etc. I will dedicate 30h.

2.3. Resources

I will define the resources necessary to complete this project, dividing them by human resources and material resources.

2.3.1. Human resources

The people involved in the development of the project are:

- Researcher: responsible for the development of the project.
- Project director: will guide the researcher during the development of the project and evaluate its progress.
- Project management tutor: will teach the researcher the process needed to complete the project management tasks.

2.3.2. Material resources

The material resources are the hardware and software that will be used to implement the project.

The hardware needed is:

- A computer: to do most of the work of the project: documentation, researching, programming… I will use a MSI laptop.
- Arduino Portenta H7: the microcontroller board that will execute the code developed in the project.

The software needed is:

- IDE: an integrated development environment to help write programs for microcontrollers. I will use the official Arduino Software IDE and PlatformIO.

Resource	Code
Human resources	
Researcher	R
Project director	PD
Project management tutor	PMT
Material resources	
MSI laptop	L
Arduino Portenta H7	PH7
IDE	IDE

Table 1: Resources and their codes

2.4. Estimates and the Gantt

In the table 2 there is a summary of all tasks with their times and dependencies, and in figure 1 there is the Gantt diagram of this project.

ID	Name	Time (h)	Dependencies	Human resources	Material resources
T1	Project management	90		R, PD, PMT	L
T1.1	Context and scope	30		R, PD, PMT	L
T1.2	Planning	15	T1.1	R, PD, PMT	L
T1.3	Budget	15	T1.2	R, PD, PMT	L
T1.4	Sustainability report	10	T1.3	R, PD, PMT	L
T1.5	Meetings	20		R, PD	L
T2	Learn to work with Portenta	90		R	L, PH7, IDE
T2.1	Install IDEs	5		R	L, IDE
T2.2	Research programming language	10		R	L
T2.3	Experiment programming with the board	40	T2.1, T2.2	R	L, PH7, IDE
T2.4	Explore Portenta H7 accessories	15		R	L, PH7, IDE
T2.5	Research accessory applications	20	T2.4	R	L, PH7, IDE
T3	Research existing TinyML applications	70		R, PD	L, PH7, IDE
T3.1	Explore different programs	20		R, PD	L
T3.2	Test the compatible programs with the Portenta board	30	T3.1	R	L, PH7, IDE
T3.3	Investigate the code of applications incompatible with Portenta	20	T3.1	R	L, IDE
T4	Implement my own machine learning application with Portenta H7 (Sprint)	90	T2, T3	R, PD	L, PH7, IDE

T4.1	Define the program to implement	5		R, PD	L
T4.2	Connect the necessary hardware	5	T4.1	R	L, PH7
T4.3	Design the machine learning model	20	T4.1	R	L
T4.4	Write the code	30	T4.2, T4.3	R	L, IDE
T4.5	Generate datasets	10	T4.2	R	L, PH7, IDE
T4.6	Test with Portenta H7	20	T4.4, T4.5	R	L, PH7, IDE
T5	Documentation	80		R	L
T6	Oral defense preparation	30		R	L
	Total (1 Sprint)	450			
	Total (2 Sprint)	540			
	Total (3 Sprint)	630			

Table 2: Summary of the tasks

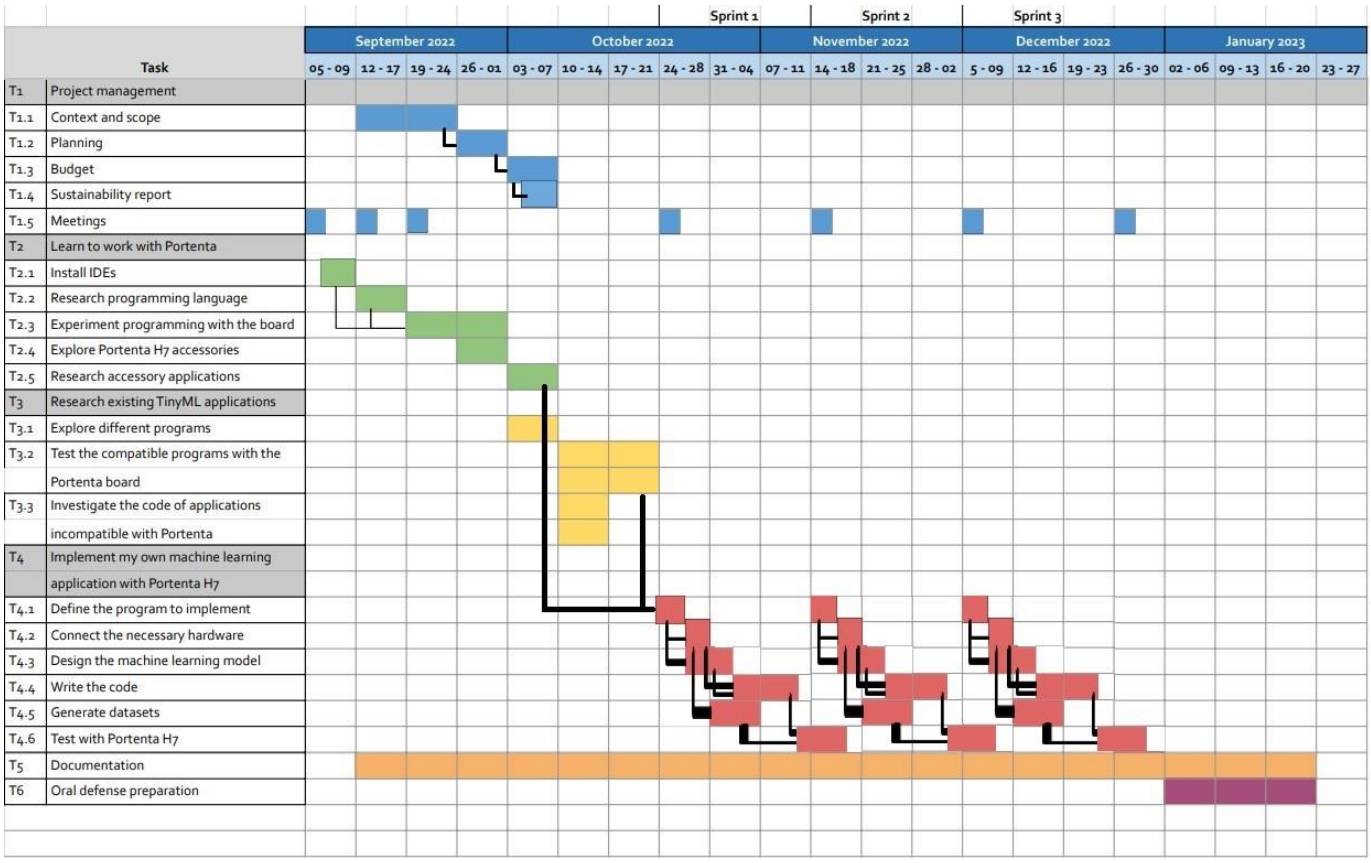

Figure 1: Gantt diagram. Own elaboration

2.5. Risk management: alternative plans and obstacles

In this section I will develop the obstacles defined in the section Obstacles and risks of this project, defining alternative tasks, the delay that they will cause and if any additional resources are needed.

For unknown errors when programming there would be a new task to find a solution for the error. The solution will probably be in the official Arduino forum, but I could need to search in other sources. Searching could take from 2h to 8h if the error is rare, and until it's fixed all programming tasks can't continue. It's a risk of medium level.

For obsolete documentation the consequence will probably be an unknown error, so it will have the same alternative task. However, since the documentation change should be recent it might be more difficult to find a solution, making the delay take up to 10h. It's a risk of medium level.

For library changes during the project the first step would also be search for a solution in the Arduino forum, but an alternative solution could be searching for the old version of the library that was used previously. The delay would be from 4h to 10h, and all programming is stopped. It's a risk of medium level.

For the hardware problems there would be a new task to replace the board with a new one. Thankfully the director has offered additional boards so the delay would only be the travel time to the university, so 3h, although the project would be using an additional resource, the new board. It's a risk of low level.

2.6. Budget

2.6.1. Identification of costs and cost estimates

In this section I will define the economic budget of this project. In order to do so, first I will calculate the staff costs of the people that will work on each of the tasks defined previously, then the general costs that aren't specific to any task, a contingency to cover unexpected obstacles and incidental costs to cover the obstacles that were defined previously in this project.

2.6.1.1. Staff costs

The staff costs is the cost of hiring the workers necessary to complete this project, using the tasks defined previously to see how much time each worker will need and their salaries to calculate the total cost.

The staff necessary to do this project can be defined with the following roles: a project manager, who will be responsible of the planning of the project and that the project is developed correctly, a machine learning engineer, to define machine learning models for the tinyML part of the application, and a programmer, to implement the application that will be executed on Portenta H7 using the models defined previously.

I will be doing the work of all the defined roles, and the project director will do the role of a project manager too.

The cost of social security payments must be included in the final budget, so the gross salaries will be multiplied by 1.35.

Role	Annual salary (€)	Annual salary including SS (€)	Salary per hour (€)	Played by
Project manager	33.685 [11]	45.474,75	21,86	Director, me
Machine learning engineer	32.512 [12]	43.891,20	21,10	Me
Programmer	30.244 [13]	40.829,40	19,63	Me

Table 3: Summary of salaries by role

After obtaining the salary per hour of each of the roles we can obtain the full staff cost of the project assigning the roles necessary to each of the tasks and calculating the price of each of the tasks.

ID	Name	Time(h)	Necessary roles	Cost (€)
T1	Project management	90	Project manager, Machine learning engineer, Programmer	2.782
T1.1	Context and scope	30	Project manager	655,80
T1.2	Planning	15	Project manager	327,90
T1.3	Budget	15	Project manager	327,90
T1.4	Sustainability report	10	Project manager	218,60
T1.5	Meetings	20	Project manager, Machine learning engineer, Programmer	437,20 + 422 + 392,60 = 1.251,80
T2	Learn to work with Portenta	90	Programmer	1.766,70
T2.1	Install IDEs	5	Programmer	98,15
T2.2	Research programming language	10	Programmer	196,30
T2.3	Experiment programming with the board	40	Programmer	785,20
T2.4	Explore Portenta H7 accessories	15	Programmer	294,45
T2.5	Research accessory applications	20	Programmer	392,60
T3	Research existing TinyML applications	70	Programmer, Machine learning engineer	2.851,10
T3.1	Explore different programs	20	Programmer, Machine learning engineer	392,60 + 422 = 814,60

T3.2	Test the compatible programs with the Portenta board	30	Programmer, Machine learning engineer	588,90 + 633 = 1.221,90
T3.3	Investigate the code of applications incompatible with Portenta	20	Programmer, Machine learning engineer	392,60 + 422 = 814,60
T4	Implement my own machine learning application with Portenta H7 (Sprint)	90	Project manager, Machine learning engineer, Programmer	2.025,60
T4.1	Define the program to implement	5	Project manager, Machine learning engineer, Programmer	109,30 + 105,50 + 98,15 = 312,95
T4.2	Connect the necessary hardware	5	Programmer	98,15
T4.3	Design the machine learning model	20	Machine learning engineer	422
T4.4	Write the code	30	Programmer	588,90
T4.5	Generate datasets	10	Machine learning engineer	211
T4.6	Test with Portenta H7	20	Programmer	392,60
T5	Documentation	80	Project manager	1.748,80
T6	Oral defense preparation	30	Project manager	655,80
	Total (1 Sprint)	450		11.830
	Total (2 Sprint)	540		13.855,60
	Total (3 Sprint)	630		15.881,20

Table 4: Estimated cost per task and necessary roles

The total staff cost (CPA) is 15.881,20 €.

2.6.1.2. General costs

The general costs that will be included in the budget are: the amortization of the resources, the electricity cost and the internet cost.

- Amortization

The hardware that will be used is a MSI laptop and the Portenta H7 board.

The software that will be used is the Arduino IDE and the PlatformIO IDE, but since they are free programs downloaded from their official websites there won't be an amortization cost.

To calculate the amortization cost of the hardware I will use the following formula:

$$Amortization = \frac{Price}{Useful\ life} x\ (Project\ time)$$

Considering that the useful life of hardware tends to be around 4 years and the project will last around 5 months, the amortization of each hardware is in the following table:

Hardware	Price (€)	Amortization (€)
MSI laptop	779,00	81,14
Portenta H7	99,00	10,31

Table 5: Amortization of the hardware

- Electricity

Electricity will be necessary to use the computer and be able to work on the project, so the cost will be computed for the full 630 hours of the project. On average it costs 0,35 €/kWh [14] and a laptop consumes 0,3 kWh, so the full cost is 66,15 €.

- Internet

Internet will be used for the meetings, to share files and to find information for the documentation and the development of the project.

On average it costs 30 € per month [15], and since this project will last around 5 months the total cost is 150 €.

Description	Cost (€)
Hardware amortization	81,14+10,31 = 91,45
Software amortization	0
Electricity	66,15
Internet	150
Total	307,60

Table 6: General costs

The total general cost (CG) is 307,60 €.

2.6.1.3. Contingency

In order to be prepared for unexpected obstacles I will define a contingency margin. Following the average rates of the IT sector I will define a rate of 15%, which means a contingency cost of (15.881,20+307,60) * 15% = 2.428,32 €.

2.6.1.4. Incidentals

For each of the obstacles defined in previous sections I will assign a probability of occurrence and which roles does it involve, to be able to calculate a cost based on the salaries.

Obstacle	Probability	Time(h)	Roles affected	Salary cost(€)	Final cost(€)
Unknown errors	40%	8	Programmer	157,04	62,81
Obsolete documentation	30%	10	Programmer	196,30	58,89

Library changes	20%	10	Programmer	196,30	39,26
Hardware problems	15%	3	Programmer, Project manager	58,89 + 65,58 = 124,47	18,67
Total					179,63

Table 7: Incidental costs

The total incidental cost is 179,63 €.

2.6.1.5. Final budget

Concept	Cost(€)
Staff costs	15.881,20
General costs	307,60
Contingency	2.428,32
Incidentals	179,63
Total	18.796,75

Table 8: Total cost of the project

2.6.2. Management control

In order to control any differences between the budget and the real cost of the project during its development I will be calculating the cost deviation for each of the tasks as they are completed. To do so I will define:

- Estimated cost: estimated cost of the task as defined in the table "Estimated cost per task and necessary roles".
- Real cost: real cost of the task recalculating the four kinds of cost as in the previous section.
- Deviation: difference between the Estimated cost and the Real cost.

With the deviation we can see if a task has been completed according to the budget, if it needs extra funding from the contingency or if it was completed with less cost than budgeted.

3. Sustainability aspects

3.1. Sustainability report

3.1.1. Self-assessment

In order to check my knowledge about sustainability I have answered the anonymous survey of EDINSOST2-ODS.

I thought I had a general knowledge about sustainability in practical projects, but after the survey I realized that I had a very limited vision about sustainability, for instance I don't know any metrics to measure the social impact of a project.

Additionally, I have been focused on the economic dimension of this project and haven't given much thought to the social and environmental dimensions.

Now I will analyze the sustainability of this project in its three dimensions: economic, environmental and social.

3.1.2. Economic dimension

Have you estimated the cost of undertaking the project (human and material resources)?

In the section dedicated to the budget I defined the human roles and hardware necessary to develop this project, including their prices, and the full cost of the project is also calculated.

In reality the cost of the project will be much lower than the budget because I will be doing all the roles, without hiring professionals, and the Portenta H7 board is borrowed from the director.

Have you quantified the cost (human and material resources) of undertaking the project? What decisions have you taken to reduce the cost? Have you quantified these savings?

As mentioned earlier, since all the human roles will be done by a single person the cost of the project will be lower than the budget.

The project took longer than calculated to be finished, approximately 650 hours. Assigning a salary of approximately 20 euros per hour, the human resource cost is 13.000 euros.

By not hiring workers the cost of the project is reduced by 15.881,20 – 13.000 = 2.881,2 euros.

The material resources used are the same as the ones planned, so the cost is 307,60 euros as mentioned in section 2.6.1.2.

Is the expected cost similar to the final cost? Have you justified any differences (lessons learnt)?

As mentioned earlier, there is a difference of 2.881,2 euros compared to the expected cost because all the roles are done by a single person.

How is the problem that you wish to address resolved currently (state of the art)? In what ways will your solution economically improve existing solutions?

As I discussed in the section of justification, this project would solve problems that are currently addressed using cloud computing. The microcontrollers that are used with TinyML are cheaper to produce than the computer systems that are used with cloud computing, and they have a lower energy consumption.

What cost do you estimate the project will have during its useful life? Could this cost be reduced to increase viability?

An Arduino microcontroller has an expected life of 20 years in good conditions, and on average it consumes 500 mW, so with 175.200 hours in 20 years it will use 87.600.000 mW, or 87,6 kW. With a price of 0,35 €/kWh the cost of a single board during its life is 30,66 €. However, the project needs at least 2 boards to work, so the cost will be at least 61,32 €.

The cost could be reduced if the energy consumed by the boards was reduced, for example keeping the LED light of the board always off.

Have you considered the cost of adaptations/updates/repairs during the useful life of the project?

The cost of replacing a board will mainly be the cost of buying a new one, it doesn't need to be programmed from scratch since all of them use the same code and it will receive models from the federated learning server. So the cost of replacing a board is 99,00 €.

Could situations occur that are detrimental to the project's viability?

If the cost of the original hardware is high the consumers might be reluctant to buy this project to replace it, even if in the long run it would be less expensive.

3.1.3. Environmental dimension

Have you estimated the environmental impact of undertaking the project? Have you considered how to minimize the impact, for example by reusing resources?

Since I will develop the project and power the Portenta H7 with the laptop as defined in the general costs of the budget, most of the environmental impact of the project will be the power consumed by the laptop, which is approximately 0,3 kWh.

The hardware will be reused, the Portenta H7 board is borrowed from the project director and it will be returned at the end of the project, so it can be reused for other projects.

Have you quantified the environmental impact of undertaking the project? What measures have you taken to reduce the impact? Have you quantified this reduction?

As mentioned earlier, the environmental impact of undertaking the project is the power consumed by the laptop used for all the developing and testing of the project.

The project took longer than calculated to be finished, approximately 650 hours. Therefore the environmental impact is 650 h * 0,3 kWh = 195 kW.

Since the meetings with the director were done using videoconferences, the transport cost to go to the university and return was saved. It takes approximately 1 hour of transport time to go and return, and we had 18 meetings, which means 18 hours of vehicle usage prevented.

If you carried out the project again, could you use fewer resources?

The project already uses very few resources, and we need to have at least 2 Portenta boards to be able to test if the federated learning process works correctly.

How is the problem that you wish to address resolved currently (state of the art)? In what ways will your solution environmentally improve existing solutions?

As explained in the economic dimension, currently computer systems are being used for machine learning, where using microcontrollers with TinyML would lower the power consumption to the order of milliwatts.

What resources do you estimate will be used during the useful life of the project? What will be the environmental impact of these resources?

As mentioned earlier, an Arduino microcontroller has an expected life of 20 years in good conditions, and on average it consumes 500 mW, so with 175.200 hours in 20 years it will use 87.600.000 mW, or 87,6 kW. Although to work properly this project needs multiple Portenta boards, so the impact will be multiplied by the amount of boards.

Will the project enable a reduction in the use of other resources? Overall, does the use of the project improve or worsen the ecological footprint?

With this project large devices could be replaced by microcontrollers, reducing the power consumption of the system and reducing fabrication costs, since they need less materials to be made. However, replacing existing systems means that the existing machines have to be removed, which could generate an environmental impact if not done properly.

Could situations occur that could increase the project's ecological footprint?

As mentioned earlier, although replacing existing systems with microcontrollers would reduce the energy consumption and material costs, the existing machines should be reused or recycled, otherwise they could generate contamination if they aren't properly disposed.

3.1.4. Social dimension

What do you think undertaking the project has contributed to you personally?

With this project I will be able to improve my knowledge on machine learning, which is a field that I find very interesting since it appeared in multiple subjects during the degree, and I believe that it is a field that is constantly evolving and reaching new applications.

I had never worked with microcontrollers before this project, but since they are used everywhere as part of machines I believe that learning how to program them can be a useful skill to have for the future.

Has undertaking this project led to meaningful reflections at the personal, professional or ethical level among the people involved?

Developing this project has helped me realize the difficulties that I could face when working in this field, and it has shown me my shortcomings that I should try to improve.

How is the problem that you wish to address resolved currently (state of the art)? In what ways will your solution socially improve (quality of life) existing solutions?

TinyMl is improving the quality of life by being used in hospitals and in homes for its fast inference and the fact that it doesn't need connections make it useful for people that live in rural areas.

With this project I could find uses for the Portenta H7 board that could be expanded for new applications to improve the quality of life.

Is there a real need for the project?

Although a lot of machine learning applications can work using cloud computing, since TinyML is faster, cheaper and can be used anywhere I believe that it's worth updating them to use TinyML.

Who will benefit from the use of the project? Could any group be adversely affected by the project? To what extent?

As mentioned earlier a big beneficiary of this project would be people living in remote areas.

As any machine learning application, the results obtained from the project depend on what kind of data is used to train the model. If the data of a specific group is intentionally modified or excluded the project will generate wrong results when trying to infer new data of this group of people.

To what extent does the project solve the problem that was established initially?

The project successfully implements a federated learning application, however the speed at which it executed isn't suitable for critical real world applications, like for a hospital.

Could situations occur in which the project adversely affects a specific population segment?

As mentioned earlier, it could marginalize populations if the input data is manipulated.

Could the project create any kind of dependency that puts users in a weak position?

Probably not.

4. Laws and regulations

A law that could affect future usage of this project is the Artificial Intelligence Act (AI Act) [16], which is a proposed European law on artificial intelligence, which assigns levels of risk to AI applications and applies restrictions according to their level of risk.

Low risk applications are left unregulated. Medium risk applications have to undertake conformance testing before being put in the market. High risk applications are completely prohibited.

Considering the scope of this project I doubt that it could be considered an application of medium or higher risk, so this law shouldn't affect the project.

5. Background

5.1. What is TinyML / embedded learning?

TinyML: also called embedded machine learning, is a branch of machine learning based on running the machine learning models on the microcontrollers of embedded systems, processing data on-device. The main benefits are that microcontrollers consume very little power, and they reduce the computing time to make an inference because they don't have to communicate with any servers, the inference is done locally

Since embedded machine learning doesn't have to send data to a server it won't have transmission costs and lower latency for each inference, and the data will be at lower risk of being compromised. They can also be used in isolated locations without internet connection. Additionally, TinyML is based on small microcontrollers, which means that they are cheaper to produce than computer systems and have low energy consumption.

However, since the model has to be in the embedded system they will be limited by the memory space of the device compared to a remote server, and it will be harder to fix any issues and make modifications because it has to be done in the embedded system physically.

5.2. Workflow for building a TinyML application

The most common way of developing a TinyML application is using TensorFlow Lite, an extension of TensorFlow.

TensorFlow is a free open-source library for machine learning developed by Google. It can be used with many programming languages, including Python, JavaScript, C++, and Java, which allows it to be used in a wide variety of sectors.

TensorFlow Lite converts the models developed with TensorFlow to be able to be used in embedded machines, reducing the space that they occupy in memory.

It addresses the following aspects of embedded machine learning: [17]

- Latency: low because it doesn't communicate with a server.

- Privacy: no personal data leaves the device.

- Connectivity: it doesn't need to be connected to internet.

- Size: the model occupies less memory space.

- Power consumption: low because of efficient inference and a lack of network connections.

The workflow of developing a TinyML application using TensorFlow Lite is defined with the following steps: [18]

1. Train the model: first we must train a regular TensorFlow model. We must be careful with the size of the model, if it's too large it won't fit in the embedded device after the conversion. It's recommended to use a Google Colab notebook to train the model, to execute the python code on Google cloud servers.
   Then we define and prepare a dataset, create a model and train it.
2. Convert the model: converting the trained TensorFlow model into a TensorFlow Lite model. TensorFlow Lite models use a special format known as FlatBuffers. Compared to regular TensorFlow models they have reduced memory size and faster inference because data is directly accessed without an extra parsing/unpacking step.
3. Optimize the model: in order to reduce even more the space occupied by the model. Some ways to optimize are Quantization and Weight Pruning.
   a. Quantization: reducing the precision of the numeric values of the parameters of the model. It can be done reducing the number of floating point bits, for example from 32 bits to 16 bits, or by converting the values to integers.
   b. Weight Pruning: removing values of the model that have low impact on the performance of the model. By doing so the model becomes sparser and can be compressed more efficiently.
4. Deploy the model: load the optimized model into the embedded device.
5. Make inferences: run the model on the embedded device.

The first three steps normally aren't done on the embedded device, since they need space to store the regular TensorFlow model and the dataset to train.

Figure 2 has a representation of the workflow of a tinyML application using Tensorflow Lite:

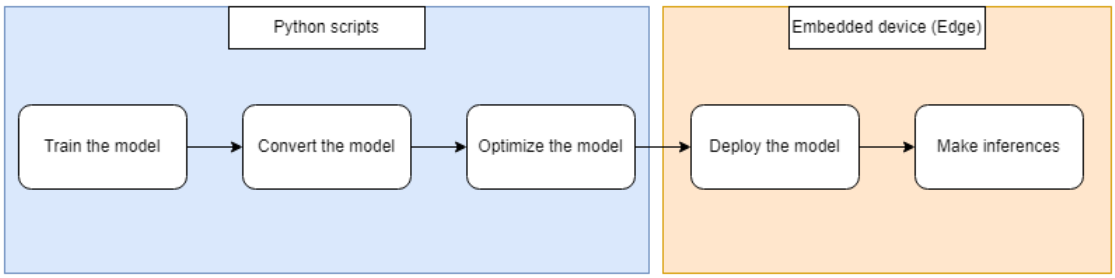

Figure 2: Workflow of a tinyML application using Tensorflow Lite. Own elaboration

What we will be doing in this project is building and training the model on the embedded device itself. This way we can do machine learning with only the device, without needing

a main computer or a server to provide the trained model. Although it loses a bit of space because it needs the code to train the model, it gains a lot of flexibility.

This way there are only two steps in the workflow:

1. Train the model: obtain the dataset using the sensors of the device. Initialize a model and train it with the obtained dataset.
2. Make inferences: run the model on the device.

Figure 3 has a representation of the workflow of the tinyML application of this project:

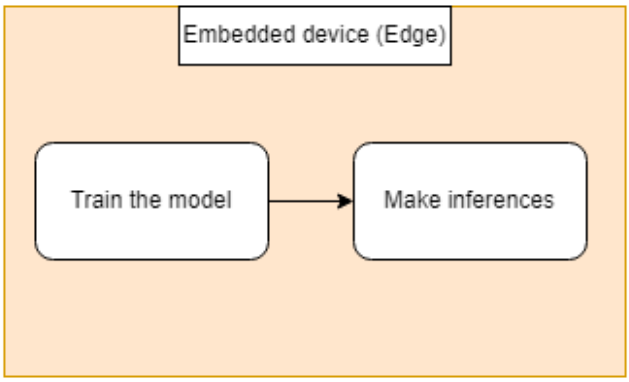

Figure 3: Workflow of the tinyML application of this project. Own elaboration

5.3. Software tools

The main programs that we will use to develop this project are Arduino IDE and PlatformIO.

5.3.1. Arduino IDE

The official programming IDE to develop sketches to run in Arduino boards, developed and distributed by Arduino.

It can downloaded for free in the Arduino website [4].

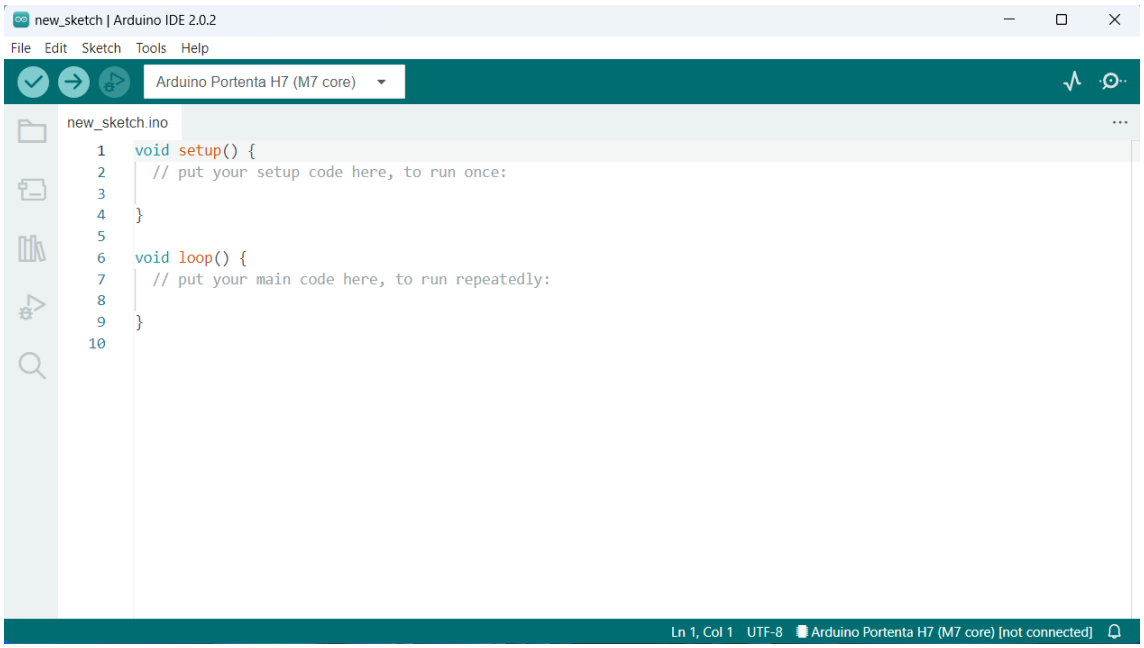

Figure 4: Screenshot of the Arduino IDE with a new sketch. Own elaboration

When a new sketch is created it gives the user the basic structures necessary to have a working Arduino sketch: a setup() and a loop(), as seen in Figure 4.

The setup will only be executed once during the execution of the sketch, it will be the first code that will run. It tends to be used to initialize and setup all the variables and functions that will be necessary for the execution of the sketch, hence its name.

The loop will be executed right after the setup, and it will repeat its code for as long as the board is connected. It will contain the main code of the program.

Once a sketch is uploaded into a board, every time the board is connected to a computer it will be executed.

At the top left there are the main buttons needed to run a sketch on a board:

- Checkmark: compile the currently open sketch.
- Arrow: compile the current sketch and upload it to the selected board.
- "Board model", in this case Arduino Portenta H7 (M7 core): the board that will receive and run the sketch. It has a list of currently connected boards and which ports they are using. Click one of them to select which one will be used.

In the Tools menu in the top left corner there are some useful options, which can be seen in Figure 5.

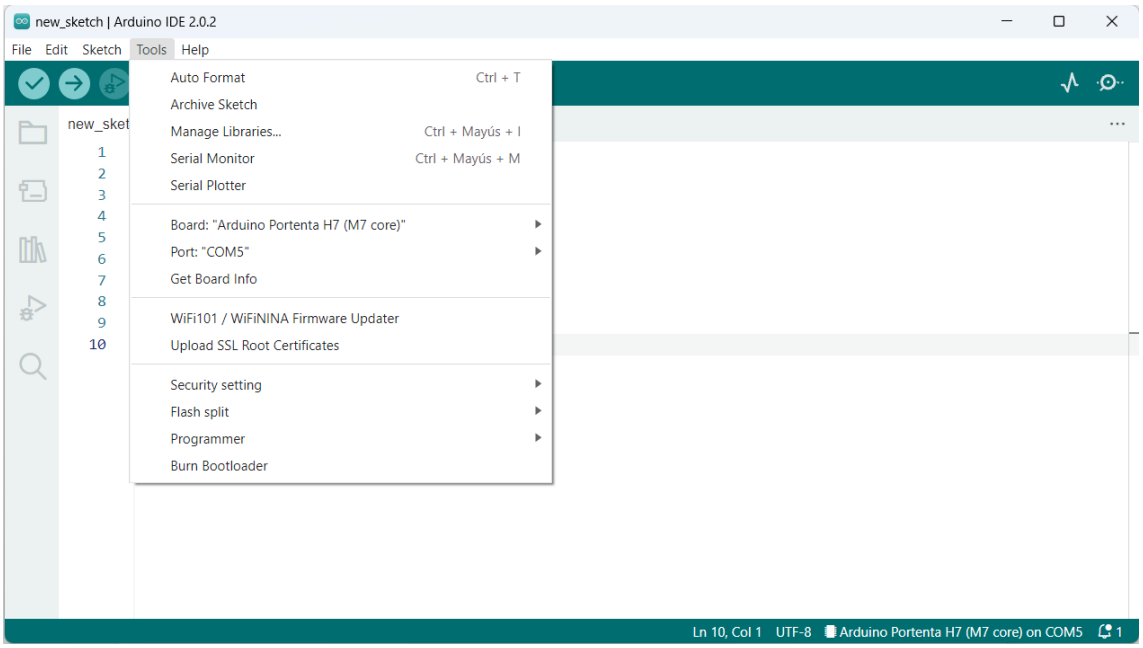

Figure 5: Screenshot of the Tools menu. Own elaboration

Some useful options are:

- Manage libraries: Arduino has a list of officially approved libraries with a wide range of utilities, which can be downloaded from the IDE using this menu. When this option is clicked the left bar will expand and show the list of libraries, and also a search bar and some filters to look for specific libraries. Once a library is installed it can be included in any sketch using the Include library option in the Sketch menu.
- Serial monitor: opens a new tab that prints everything that the sketch writes to the serial port.
- Board “board model”: in this menu the board model that the user will be using can be selected from a list. With this the IDE uses the compatible files to be able to compile and upload the sketch to the connected board. In the case of Portenta, it’s also used to select to which core the sketch will be uploaded to. If the user’s board model isn’t in the list it can be downloaded using the Boards manager option at the top of the list, which will open a list of boards on the left with a search bar to find the user’s model. Once it’s downloaded, it will show in the main model list.

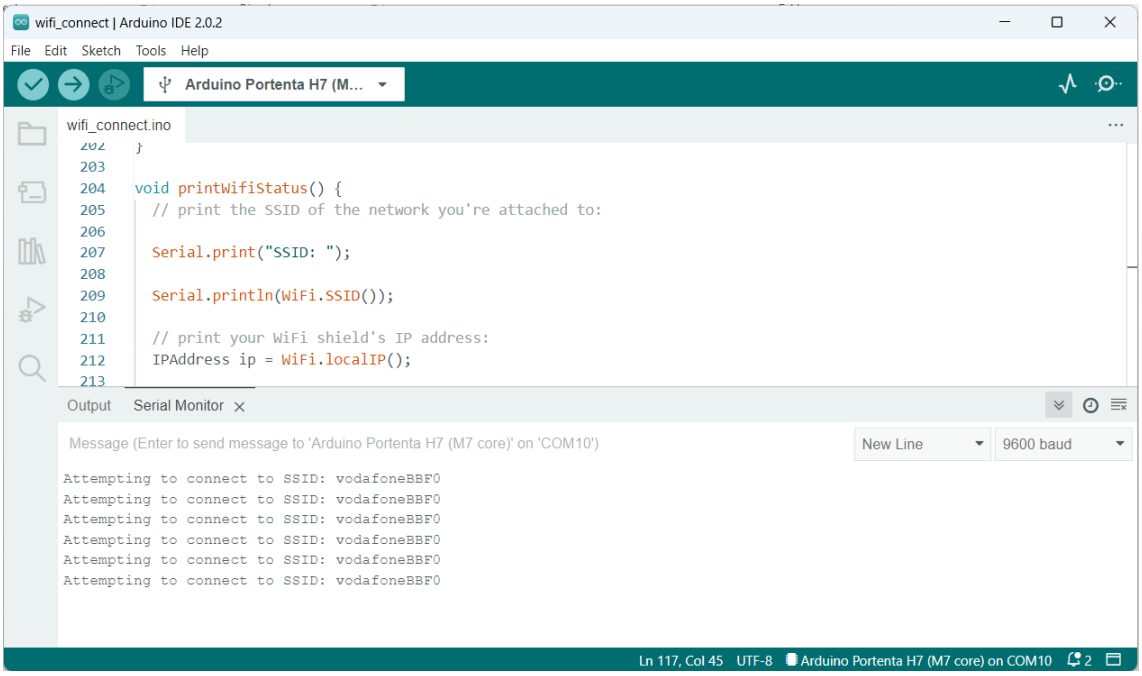

Figure 6: Screenshot of the Serial Monitor. Own elaboration

An example of the serial monitor printing the data sent to the Serial port can be seen in Figure 6.

Although this program is very good for testing because of its simplicity, it tends to be slow when it compiles more complex sketches, and it is difficult to navigate when there are more files than just the main file. For these reasons we looked for an alternative and used PlatformIO to program this project.

5.3.2. PlatformIO

It is an IDE built on top of Microsoft's Visual Studio Code, so it has all the functionalities of the basic Visual Studio Code in addition to functionalities to develop Arduino projects.

Some of the improvements that it has over the Arduino IDE are:

- Faster compilation and uploading to the board.
- Easier file access during programming: the files are shown in the left side of the screen and folders can be expanded or collapsed individually to show only the files that the user wants. It also enables opening files from multiple projects.
- Any variable or function can be right-clicked on to find its definition, even if it is in another file or a library, which is very useful for debugging.
- Since it's built from Visual Studio Code it's compatible with many programming languages. For example it can be used to write Python scripts inside an Arduino project and execute them with its internal Windows PowerShell terminal.
- It enables uploading projects to Github repositories, notifying the user if there are modifications on the user's files that haven't been pushed yet.

The installation instructions are in their website [19].

When PlatformIO is executed it will open the PIO home menu as seen in Figure 7:

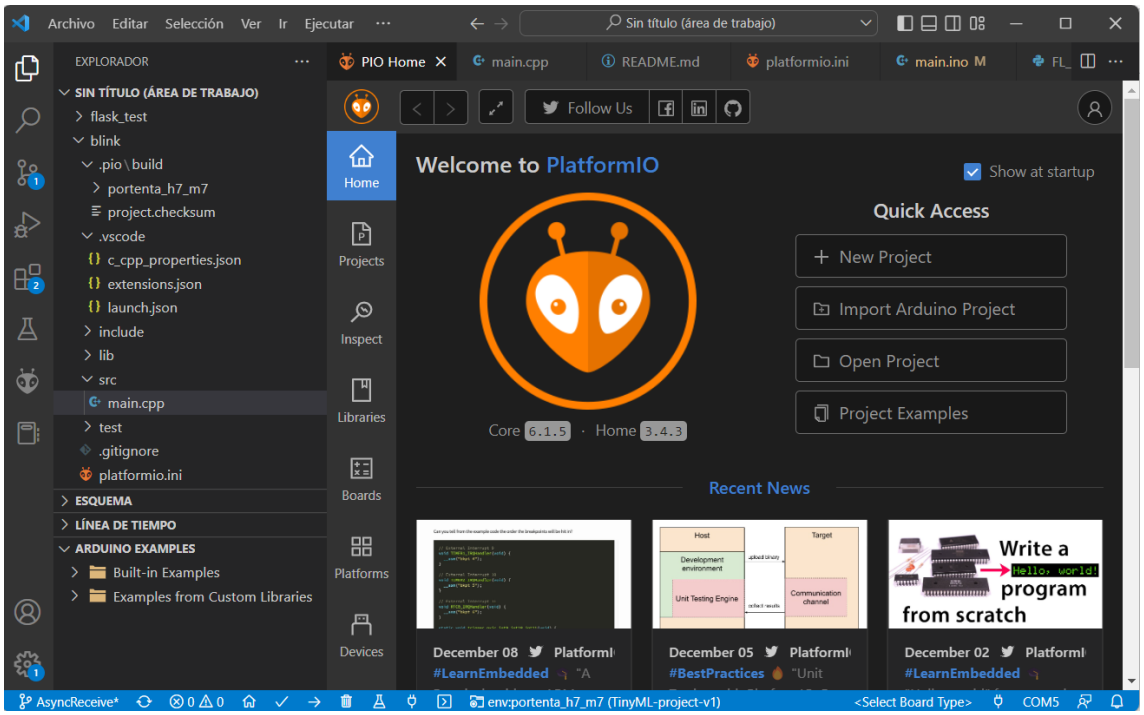

Figure 7: Screenshot of the PIO home menu. Own elaboration

On the left of the page there is a list of icons with some options for programming with Arduino. The most relevant are:

- Projects: a list of projects that have been opened with the IDE. From this tab the user can also create a new project or import a PlatformIO project from existing file folders. When creating a new project the board model which will run the files has to be specified.
- Libraries: to find and download Arduino libraries, similar to the menu in the Arduino IDE. It has a search bar to find the libraries, a tab with the currently installed libraries that shows what libraries are included in each of the user's projects, and a tab that shows any libraries that need to be updated.
- Devices: in the Serial tab it will show any boards that are currently connected to the computer, with the port number that it's using.

All PlatformIO projects must have a src folder with a main file and a platformio.ini file. When a new project is started these files are automatically generated.

The main file can be in .cpp format, which is the default, or it can be an .ino sketch, which will be automatically converted into a temporal .cpp file when it's compiled. It is the code that will be uploaded to the board. An example of a simple main file can be seen in Figure 8.

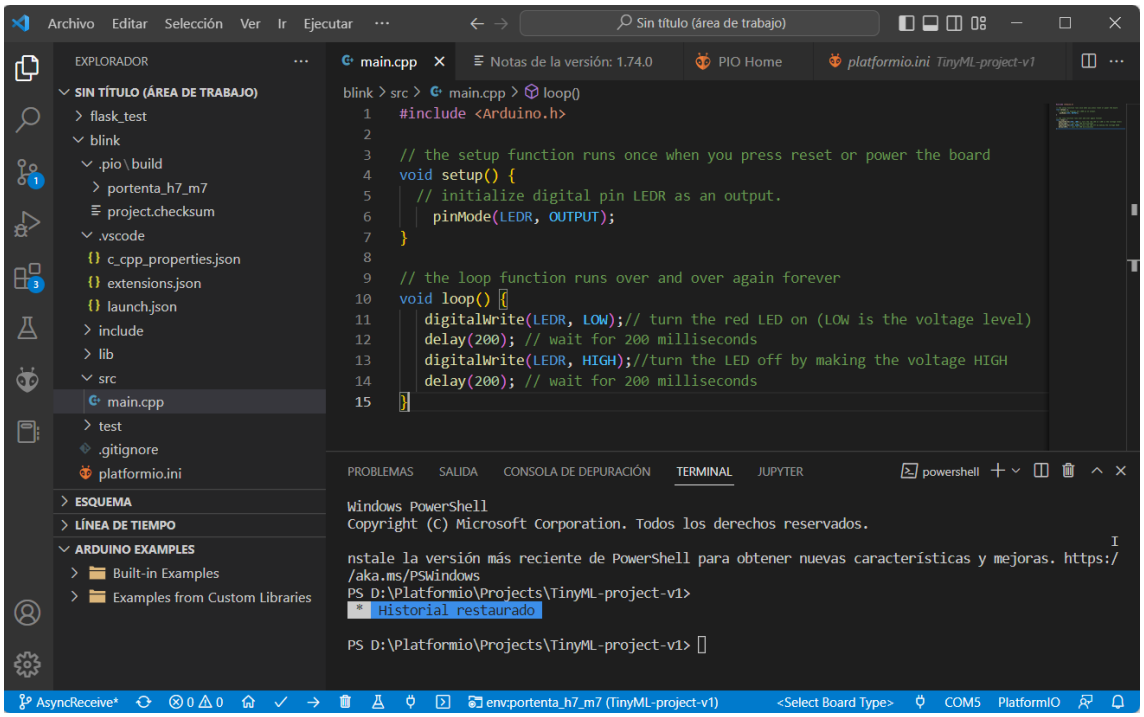

Figure 8: Screenshot of a main.cpp file. Own elaboration

The platformio.ini file is a configuration file that is used to be able to execute the PlatformIO project in different computers. At the start of the project it will just have the board information that was introduced when the project was created. If any Arduino library was imported it will show in a lib_deps parameter. An example of a platformio.ini file can be seen in Figure 9.

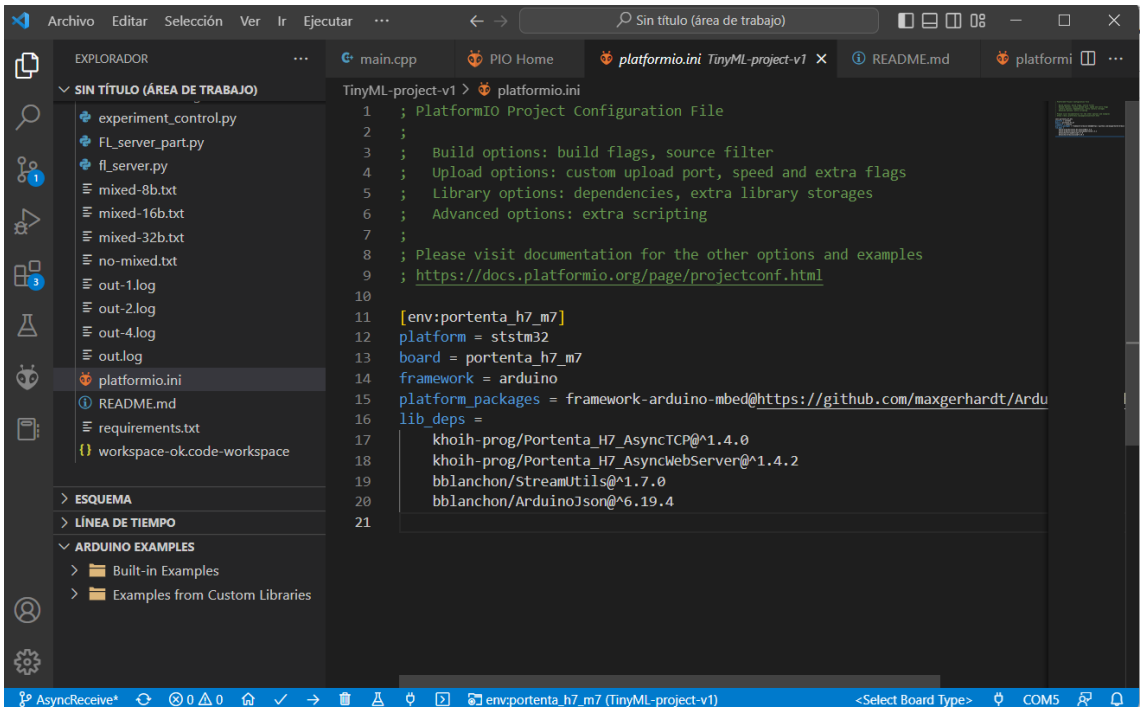

Figure 9: Screenshot of a platformio.ini file. Own elaboration

In the two previous figures it can be seen that the files of the project are shown in the left part of the screen, inside the Workspace (Area de trabajo). From there the user can open as many files as they want and they will be in separated tabs in the main window.

At the bottom of the screen, in the blue bar, there are buttons to open the PIO home menu, to compile the project, to compile and upload, to open a serial terminal, to open a powershell terminal, to select which project to use and to select which board to use.

5.4. Hardware

Portenta H7 is a microprocessor board developed by Arduino. It simultaneously runs high level code along with real time tasks. It include two processors that can run tasks in parallel, each core can execute a different code and they can communicate with each other during the execution. An image of the board can be seen in Figure 10.

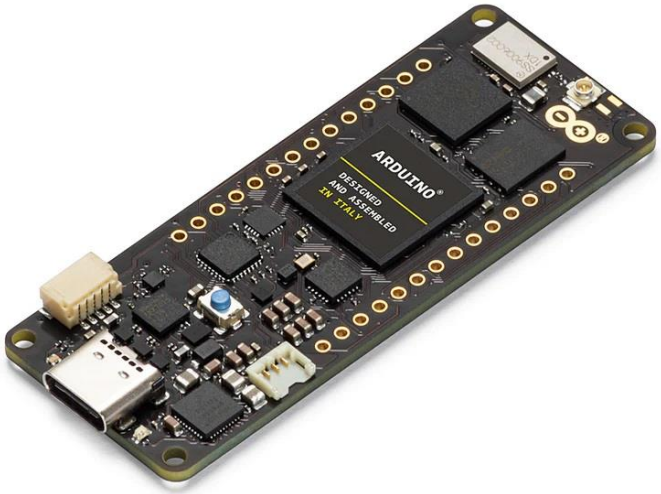

Figure 10: Portenta H7 board. Source: [10]

The main selling point of Potenta H7 is the fact that it has two cores. It has a Cortex®-M7 core running at up to 480 MHz and a Cortex®M4 core running at up to 240 MHz. the cores communicate with each other using a Remote Procedure Call mechanism, which allows them to call functions on the other core. Both processors share the chip peripherals and can run: Arduino sketches, Native Mbed applications, MicroPython / JavaScript via an interpreter and TensorFlow Lite. [10]

Portenta H7 can be used to build an embedded computer with a user interface, by connecting an external monito to the board. This is possible because of its on-chip GPU, the Chrom-ART Accelerator.

It has two 80 pin high density connectors at the bottom of the board, which allows it to be upgraded to be able to do additional functions, using accessories.

Portenta H7 can simultaneously manage WiFi and Bluetooth® connectivity. They can connect to internet by default, without adding any accessories, although their reach is

limited. Included in its package is an antenna that can be used to increase the range of WiFi and Bluetooth® connectivity. The WiFi interface can handle up to 65 Mbps transfer rate.

The board's programming connector is a USB-C port that can also be used to power the board.

5.5. Neural networks basics

Neural networks are a set of algorithms that aim to solve problems and recognize patterns mimicking the behavior of a human brain. When used in the field of machine learning they can also be called artificial neural network (ANN) or simulated neural network (SNN).

The basic element of a natural brain is a specialized type of cell called neuron. Neurons connect with other neurons forming a vast network. A neuron behaves like a switch, when it's excited by a stimulus it becomes active. The connections between neurons will have different strengths, a strong connection will provide a lot of stimulus.

In an artificial neural network the neurons and their connections are computed mathematically. The input is fed into some neurons, which calculate an output for each neuron, and then those outputs will be used as input by the connected neurons.

The strength of the connection between neurons is called weight.

The neurons in a network are divided in layers. There will always be an input layer, which receives the external input, and an output layer, which produces the result of the network. Normally there is also a hidden layer between the input and output layers. All the neurons in one layer are connected to all the neurons of the next layer. If there is more than one hidden layer in the network it can be called a deep neural network.

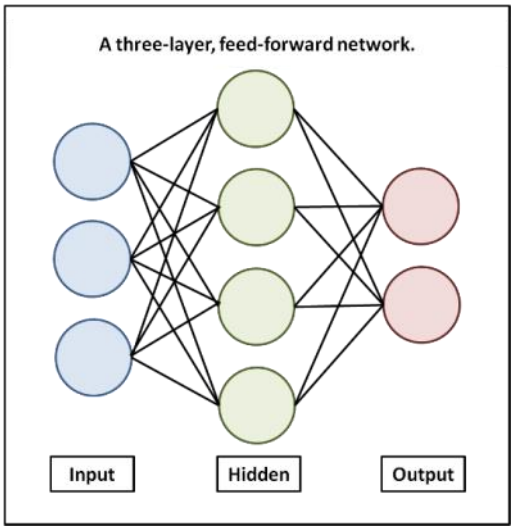

Figure 11: Representation of a three layer neural network. Source: [20]

Figure 11 contains a graphical representation of a three layer neural network, with the input layer, the hidden layer and the output layer.

Since a neural network can be represented as a graph, sometimes the neurons are called nodes and the connections are called edges.

There are many types of neural networks [21], for example:

- Feed-Forward Neural Networks: They process the information in one direction, from the input nodes until it reaches an output node. It's one of the most common neural network type. An example of real life use is facial recognition.
- Recurrent Neural Networks: They take the output of a neuron and transmit the information back into the network. Each neuron contains previous results that will be used in future iterations. If there is an error these results can be used to find what is the cause. An example of use is text-to-speech applications
- Convolutional Neural Networks: They have some layers where data is sorted into categories. These layers are called convolutional layers and will be between the input layer and the output layer. The convolutional layers have arear of images that are broken down. An example of use is image recognition applications.

In this project we will be using Feed-Forward Neural Networks.

5.6. Federated learning

Federated learning (FL) is a machine learning technique that is based on training a machine learning model across multiple edge devices. It can also be referred to as collaborative learning.

It is an alternative to Centralized machine learning, which is a traditional way of doing machine learning with multiple devices. It consists in sending the obtained data from the edge devices to a centralized server, where all the data is used to train a machine learning model. Once the model is trained it can be run in the server itself.

The main issues of centralized machine learning are:

- Connectivity: since there will be a lot of devices connected to the same server there must be a stable connection to exchange data correctly between the devices and the server.
- Network latency: to be used in real scenarios the edge devices should receive the results from the model as fast as possible, ideally instantaneously. Since the devices have to communicate with the server there might be delays when transmitting data.
- Data privacy: sometimes the data that will be used to train the model contains private information, and there is a risk of it being leaked during the data transmission between device and server.

On the other hand, federated learning consists in having a centralized server that only receives models from the edge devices, not the input data. The edge devices have models that are trained locally, and any modification of the local model is sent to the

server. The server averages and aggregates the models it receives from all the devices and generates a new model that is sent back to all the devices [22].

There are two main roles in federated learning: the clients and the server.

- Clients: edge devices with a local model that is trained using local data, which can be private. A client sends its local model to the server when it's trained, but it doesn't send and of the data used to train it, so it remains private. If the client receives a model from the server it updates the local model.
- Server: centralized database that will receive models from the clients, but not the data used to train them. The server uses the models it has received to generate a new model, and then sends it to the clients.

Federated learning is based on the following steps:

1. The server initializes a machine learning model, which can be empty or pre-trained in the server.
2. The server sends the initial model to all the devices.
3. The devices train the model individually, using their own local data to train. Since the training is done locally private data can be used without any problems.
4. After the model is trained locally, each device sends it back to the server. The server doesn't receive any of the local data, only the parameters of the model, like the weights of a neural network.
5. The server averages all of the received models to generate a single shared model, improving its accuracy.
6. The server sends the new model to all the devices.

Steps 3 to 6 can be repeated iteratively to further improve the accuracy of the models.

A representation of these steps is in the following diagram:

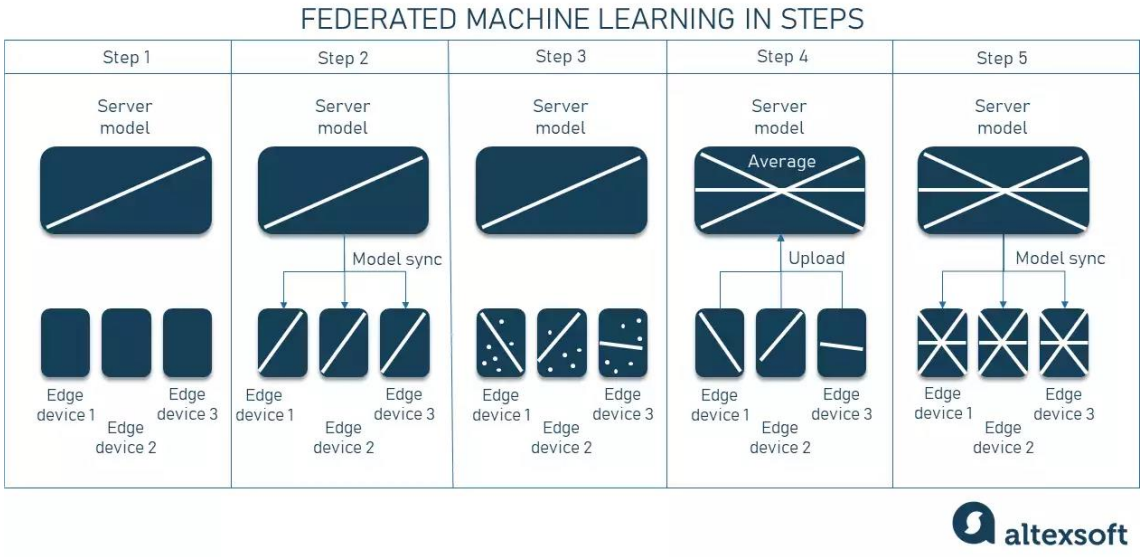

Figure 12: Federated learning steps. Source: [22]

Federated learning solves some of the issues of centralized machine learning:

- Connectivity: it still needs a stable connection to be able to send models correctly between edge devices and the server. Although if after the first iteration a device disconnects from the server it could still obtain results from the local model, and the federated learning process can continue with all the other devices.
- Network latency: the device obtains a model from the federated learning server, so it can obtain results locally and it isn't affected by the latency.
- Data privacy: since the data isn't sent to the server, only the model, private information can be used with a very low risk of leaking.

Although it also has some problems of its own:

- Efficiency of communication: in each communication we have to send a machine learning model. Depending on the size of the model this can be slow. Sometimes the number of federated learning iterations has to be reduced to improve the total time it takes.
- Heterogeneity of systems and data: the devices that are connected to the federated learning server could be different from each other, they might have different storage size, CPU, connection strength, etc. And since the devices could be in different locations the data they obtain could also be very different between the devices. This can make it so that aggregating the local models generates a worse model.

One example of practical use of federated learning is in the healthcare industry to be able to generate machine learning models using the private data of hospital patients. For example, federated learning was used to try to predict the clinical outcome of patients with COVID, using data from 20 sources around the world while maintaining anonymity [23].

Another example is in the automotive industry, since federated learning can produce results in real-time it's being used in the development of autonomous cars, using real-time updates of the conditions of the road around the car.

6. Application development

6.1. Planning update

Due to an unexpected technical error the progress of the project was delayed by a week, but by spending more hours each day the project is back on the defined schedule and is currently in the third sprint with the third version of the project, as expected in the Gantt diagram.

6.2. Methodology and rigor update

I am still using the agile methodology, doing a sprint for each version of the program.

The communication with the director is still done using Gmail and Jitsi.

The code of this project is uploaded in a Github repository [24].

6.3. Analysis of existing solution

An implementation of Federated learning with Arduino boards was developed by Nil Llisterri [25].

He implemented a federated learning system where the server is in a main computer and is executed using a python script and the clients are Arduino boards (in his case Arduino Nano 33 BLE Sense) that communicate with the server using their serial port.

The code for the federated learning server is in the file fl_server.py, and the code of the clients is in the src folder, in main.ino, neural_network.cpp and neural_network.h.

The code in the neural_network files define a feed-forward backpropagation neural network inside the Arduino board [20], using the following parameters that are defined at the start of the code, as seen in Figure 13:

```
 9   static const int PatternCount = 3;
10   static const int InputNodes = 650;
11   static const int HiddenNodes = 25;
12   static const int OutputNodes = 4;
13   static const float InitialWeightMax = 0.5;
14
15   typedef unsigned int uint;
16   static const uint hiddenWeightsAmt = (InputNodes + 1) * HiddenNodes;
17   static const uint outputWeightsAmt = (HiddenNodes + 1) * OutputNodes;
18
19   class NeuralNetwork {
20       public:
21
22           void initialize(float LearningRate, float Momentum);
23           // ~NeuralNetwork();
```

Figure 13: Parameters of the neural network. Source: neural_network.h in [25]

- PatternCount: The number of training items.
- InputNodes: The number of neurons in the input layer.
- HiddenNodes: The number of neurons in the hidden layer.
- OutputNodes: The number of neurons in the output layer.
- InitialWeightMax: Sets the maximum starting value for weights.
- LearningRate: Adjusts how much of the error is actually propagated.
- Momentum: Adjusts how much the results of the previous iteration affect the current iteration.

The amount of weights is calculated considering that the neurons in the input layer are connected to all the neurons in the hidden layer, and the neurons in the hidden layer are connected to all the neurons in the output layer. The bias of each neuron is also added.

The model takes a spoken word as an input and returns what kind of word it is as output. In this case the model takes the audio of 650 words as inputs and returns if it is "montserrat", "pedraforca", "vermell" or "blau".

The code to compute neuron activations, calculate errors and backpropagate is in neural_network.cpp.

In the main.ino file the board declares a neural network and communicates with the federated learning server. It always has to receive a command from the server to do any function, to initialize the model it has to receive an “s”, to train the model it has to receive a “t” and to do federated learning it has to receive a “>”.

The model is initialized in the setup using the values it receives from the server, and in the loop the board trains the model or starts the federated learning process. The options available in the loop can be seen in Figure 14.

```
162  void loop() {
163      digitalWrite(LEDR, HIGH);
164      digitalWrite(LEDG, HIGH);
165      digitalWrite(LEDB, HIGH);
166
167      if (Serial.available()) {
168          char read = Serial.read();
169          if (read == '>') {
170              startFL();
171          } else if (read == 't') {
172              receiveSampleAndTrain();
173          } else { // Error
174              Serial.println("Unknown command " + read);
175              while(true){
176                  digitalWrite(LEDR, LOW);
177                  delay(100);
178                  digitalWrite(LEDR, HIGH);
179              }
180          }
181      }
```

Figure 14: Main loop of the client device. Source: main.ino in [25]

Normally to train the model the user would have to press a button and speak the word each time, but to facilitate testing Nil recorded samples of each word and stored them in the datasets folder. The server sends the data of these files to the client via the serial

port. Either way after receiving data it updates the neural network and sends some values like the error to the server to build a graph later.

If it receives a command to start federated learning it sends the weights of the neural network to the server, first the hidden layer and then the output layer. After that it waits until it receives any data from the server, and when it does it replaces the weights of its neural networks with the ones it has received, since it should be the weights of the model that the federated learning server has generated.

The code for the federated learning server is contained in fl_server.py. At the start of the code it defines the variables of the neural network which have the same values as the ones in the neural_network files, as seen in Figure 15.

```
28  keywords_buttons = {
29      "montserrat": 1,
30      "pedraforca": 2,
31      "vermell": 3,
32      "blau": 4,
33      # "verd": 5
34  }
35
36  output_nodes = len(keywords_buttons)
37  test_samples_amount = 60
38  size_hidden_nodes = 25
39  size_hidden_layer = (650+1)*size_hidden_nodes
40  size_output_layer = (size_hidden_nodes+1)*output_nodes
41  momentum = 0.9
42  learningRate= 0.6
43  pauseListen = False # So there are no threads reading
```

Figure 15: Neural network variables. Source: fl_server.py in [25]

When the script is executed first it registers all connected boards, then it sends an initial model to all registered boards, by sending the weights of the neural network, first the hidden layer and then the output layer. It uses threads to be sure that all devices have received the initial model before it continues to the rest of the code.

Then it begins training the devices. It distributes the sample files in batches using batch_size. The process to distribute the samples and wait until they are used before starting federated learning can be seen in Figure 16.

```
516     threads = []
517     # Train the device
518     for batch in range(int(samples_per_device/batch_size)):
519         batch_ini_time = time.time()
520         for deviceIndex, device in enumerate(devices):
521             if experiment == 'iid' or experiment == 'train-test': method = sendSamplesIID
522             elif experiment == 'no-iid': method = sendSamplesNonIID
523 
524             thread = threading.Thread(target=method, args=(device, deviceIndex, batch_size, batch))
525             thread.daemon = True
526             thread.start()
527             threads.append(thread)
528 
529         for thread in threads: thread.join() # Wait for all the threads to end
530         print(f'Batch time: {time.time() - batch_ini_time} seconds)')
531         fl_ini_time = time.time()
532         startFL()
533         print(f'FL time: {time.time() - fl_ini_time} seconds)')
534         time.sleep(1)
535 
536     train_time = time.time()-train_ini_time
537     if debug: print(f'Trained in ({train_time} seconds)')
```

Figure 16: Send train command and do FL. Source: fl_server.py in [25]

For each sample in a batch it sends a train command “t” to the boards, waits for a confirmation from the boards, then sends all the data in the sample file and waits for a confirmation from the boards to be sure that it was received correctly by the board. Then it receives the error, num_epochs… for the graph from the boards.

After all samples from a batch have been sent the server begins the federated learning process. It receives the model of all connected boards. The weights can be received as integers or floats using the variable mixedPrecision, as seen in Figure 17:

```
352     for i in range(size_hidden_layer): # hidden layer
353         if mixedPrecision:
354             scaledWeight = readInt(d, scaledWeightsSize)
355             #scaled_in_float = readFloat(d)
356             # float_weight = readFloat(d)
357             weight = deScaleWeight(min_w, max_w, scaledWeight)
358             # if i < 5 and d.port == 'com6': print(f"[{d.port}] Recevied Weight {i}: {float_weight}")
359             # if abs(float_weight - weight) > 0.3: print(f"[{d.port}] Scaled weight: {scaledWeight} (float: {s
360         else: weight = readFloat(d)
361 
362         if debug and i % 1000 == 0 and d.port == 'com6': print(f"[{d.port}] Received Weight {i}: {weight}")
363         devices_hidden_layer[device_index][i] = weight
```

Figure 17: Receiving the hidden layer part of a model. Source: fl_server.py in [25]

When it has received all the models it generates the shared federated learning model, doing an average of the weights of all models and using the number of epochs to give more importance to some models.

```
463       # Processing models
464
465       # if sum == 0, any device made any epoch
466       if debug: print(f"Devices num epochs: {devices_num_epochs}")
467       if sum(devices_num_epochs) > 0:
468           # We can use weights to change the importance of each device
469           # example weights = [1, 0.5] -> giving more importance to the first device...
470           # is like percentage of importance :  sum(a * weights) / sum(weights)
471           ini_time = time.time() * 1000
472           hidden_layer = np.average(devices_hidden_layer, axis=0, weights=devices_num_epochs)
473           output_layer = np.average(devices_output_layer, axis=0, weights=devices_num_epochs)
474           if debug: print(f'[{d.port}] Average millis: {(time.time()*1000)-ini_time} milliseconds)')
```

Figure 18: Generation of the federated learning model. Source: fl_server.py in [25]

The code used to generate the federated learning model can be seen in Figure 18.

Then it sends the new model to all the devices. When all devices have received the new model, if there are sample batches left it stays in the loop and starts to send the samples to the boards to train again. If there aren't any samples left to send the script prints some results of the training and generates a graph before it ends the execution.

Basically, the whole project follows these steps:

1. The federated learning server sends the initial model to all the devices.
2. The federated learning server sends a command to make the devices train and send them a batch of samples.
3. After all devices have trained with all the samples of a batch, do federated learning: the server sends a command to all clients to make them send their models back to the server.
4. When it has received a model from all the clients it does an average using weights to generate the federated learning shared model.
5. The server sends the new model to all the clients.
6. Repeat steps 2 to 5 until there are no more sample batches
7. Print the training loss, the training accuracy and generate a graph.

In the previous solution all boards have to be connected to the federated learning server with a cable, to be able to use the serial port to communicate. This means that all the edge devices have to be physically close to each other to be able to do the federated learning. Although it works, realistically when we want to do federated learning we want to use a lot of devices with varied data, and most of the time they won't be in the same room, so we would need a different way to communicate the Arduino boards to the federated learning server.

For this project we have chosen to develop a solution that uses Wi-Fi to do the communications, since it allows having the edge devices far away from the server. As long as they have an internet connection, they will be able to connect to the federated learning server.

6.4. Project versions

6.4.1. Version 1: Basic version

As mentioned earlier, the objective of this project is to develop a federated learning application where the role of the clients is executed by microcontrollers, in this case the Arduino Portenta H7 boards, and where the communications between the clients and the federated learning server are done using Wi-Fi connection.

Therefore, two roles need to be implemented: the client and the federated learning server.

In order to do federated learning the models of the clients have to be trained with input data before they are sent to the server. We could introduce inputs manually using the built-in buttons and microphone of the Portenta, but it would make testing difficult since it needs large amount of inputs to work well and they wouldn't be exactly the same between executions. Therefore, we use recorded samples like in the existing solution.

However, those samples have to be sent from a computer. The federated learning server could also send those samples, but since it only communicates using Wi-Fi the samples would take longer to be sent and would go against the idea of keeping the privacy of local data.

Therefore a third role is added, experiment control, which will be in charge of sending to the clients the data needed to test the application, like the samples and the initial machine learning model. Additionally, it will control what kind tests are executed. Since it manages a large amount of data that isn't relevant to the federated learning server, we implement the communication between experiment control and the client using the Serial port instead, to make testing faster.

Then the three parts of this project are:

- Client: the device that obtains local data, trains a local model, sends it to the federated learning server, and then receives a new model generated by the server. It will be implemented using a Portenta board. Its code is in the src folder: neural_network.cpp and neural_network.h contain the code to implement a neural network inside an Arduino board, while main.ino contains the rest of the code that will be executed by the Portenta board.
- Federated learning server: the server that receives models from the clients, generates a new model and then sends it to all the clients. It will be implemented with a python script, using Flask, which is a python framework to develop web applications [26]. Its code is in FL_server_part.py.
- Experiment control: some functions to control how the testing is done. It will be implemented with a python script. Its code is in experiment_control.py.

The basic steps of the first version of the project are:

1. Experiment control sends an initial model to the clients using the Serial port

2. Experiment control sends a sample to a client using the Serial port
3. Client uses sample to train their local model
4. Client sends trained model to the federated learning server using Wi-Fi
5. If the federated learning server has received a trained model from all devices, starts the federated learning process.
6. Federated learning server generates a new model from the ones sent by the clients.
7. Federated learning server sends the generated model to all clients using Wi-Fi
8. Repeat steps 2 to 7 for all samples that will be used

A diagram representing those steps can be seen in Figure 19.

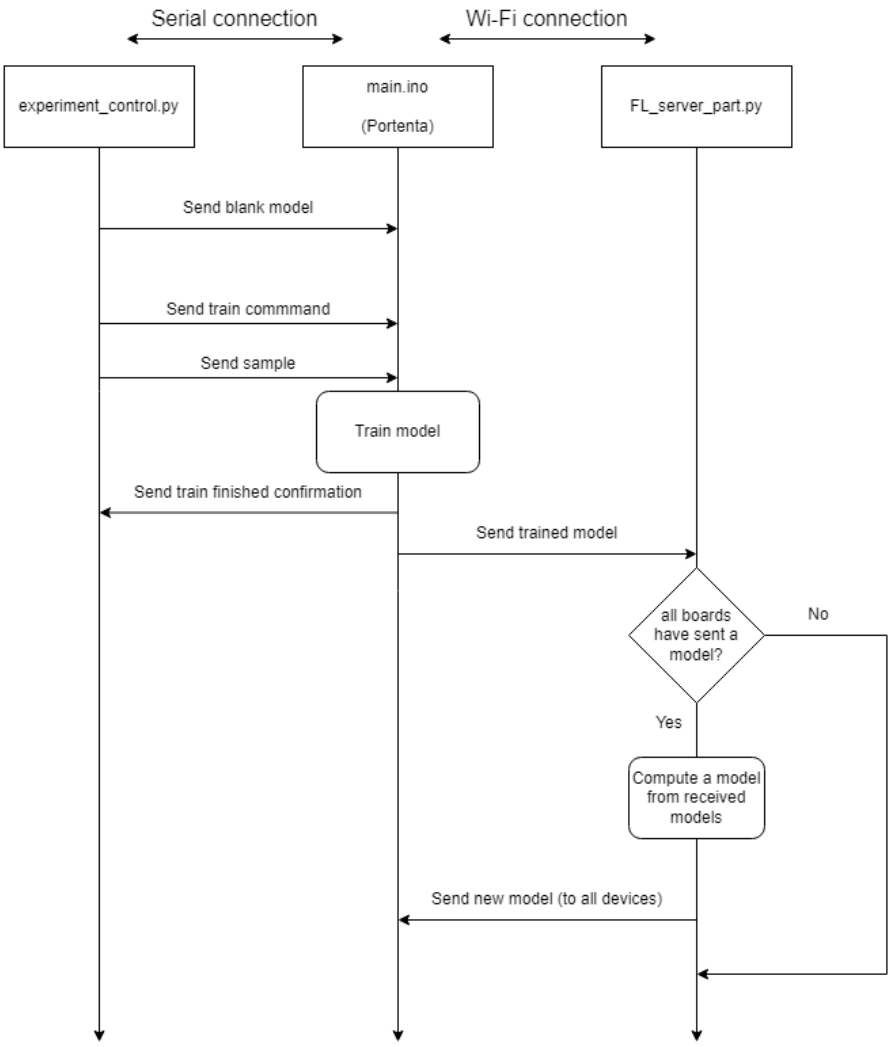

Figure 19: Basic diagram of the three parts of the project. Own elaboration

Now we will explain each part of the project in more detail.

6.4.1.1. Client

The code of the client is located in the src folder of this project.

As explained previously, the client receives an initial model and the samples from experiment control using the Serial port, and it sends and receives model data to the federated learning server using Wi-Fi.

Then, the first part of the execution of the client only involves the Serial port, since it reads the contents of the initial model sent by experiment control. Since this only needs to be done once at the beginning of the execution it's done in the setup, inside the function init_network_model(), where it reads all the values of the initial model that it receives from experiment control in the Serial port.

In theory the client would only receive data and not send anything to experiment control, however, in order to manage the execution and verify that both parts are on the correct part of their codes we added a short verification message at the beginning and the end of each process that involves Serial communication. For example, before the client starts reading data of the initial model, it waits until it receives an "s", then it begins saving the data it receives. And when it has read all the data necessary to make the model, it sends "Received new model." to experiment control, so both parts know that they can advance.

After receiving the initial model the client can already train, so it waits until it receives a "t" from experiment control, which means that it's ready to send a sample. Then it will read the data on the Serial port to obtain the contents of a file that experiment control has selected, and when it has all the data of the sample it trains the model using functions in neural_network.cpp.

After the client has trained the model it will begin the federated learning process, and won't communicate with experiment control until it's done, it will only communicate with the federated learning server using Wi-Fi.

To be able to connect the Portenta board to internet it has to be configured in the code, using the WiFi Arduino library [27].

Since it will only be connected to one network and it will be used repeatedly in each iteration of federated learning, we initialize the Wi-Fi connection in the setup, to avoid wasting time connecting to the network each time we send a message.

```
261  void connect_to_wifi() {
262    // check for the presence of the shield:
263    if (WiFi.status() == WL_NO_SHIELD) {
264      Serial.println("WiFi shield not present");
265      // don't continue:
266      while (true);
267    }
268    String fv = WiFi.firmwareVersion();
269    if (fv != "1.1.0") {
270      Serial.println("Please upgrade the firmware");
271    }
272
273    // attempt to connect to Wifi network:
274    while (status != WL_CONNECTED) {
275      Serial.print("Attempting to connect to SSID: ");
276      Serial.println(ssid);
277
278      // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
279      status = WiFi.begin(ssid, pass);
280
281      // wait 10 seconds for connection:
282      delay(10000);
283    }
284
285    // you're connected now, so print out the status:
286    printWifiStatus();
287  }
```

Figure 20: Connecting Portenta to Wi-Fi. Source: main.ino

In Figure 20 is shown the code to connect the Portenta board to a Wi-Fi network. It needs the name of the network and its password, which are stored in the “ssid” and “pass” global variables. The values of these two global variables have to be manually written at the start of the main.ino file to match the network that will be used in the execution.

As an added benefit Portenta doesn’t need any accessories to connect to Wi-Fi, although by default it needs to be close to the source of internet to work, but in the same package as the Portenta board there is an antenna that greatly increases the range of connection of the board when it’s attached.

With the Wi-Fi connection configured the Portenta board can connect to any public website, using their URL in a String format or their IP address in an IPAddress class from the WiFi Arduino library, as seen in this code fragment:

```
// server address: IP address of the flask server
IPAddress server(192,168,0,15);
String stringHost = "192.168.0.15";
```

In order to open a connection the URL or IP to connect to is needed, but the port is also necessary. Since we developed the federated learning server these values can be easily obtained and introduced in variables to connect the Portenta board to the federated learning server.

Content is sent into the receiving address using client.print().

In order to send data between the Portenta board and the federated learning server HTTP POST messages are used, and since sending models involves sending all the weights of the model, large amounts of values will be sent, which makes JSON a suitable format to store all these values and send them in the body of HTTP POST messages.

We explored some libraries to implement JSON in Arduino boards:

- Arduino_JSON [28]: the library to implement JSON offered by the official Arduino website. Although it works correctly, it's not very well documented and is a beta version.
- ArduinoJson [29]: a very well documented library that has its own website outside of Arduino. It has more functions and is very well documented, so it's the one that will be used in the project.

In order to write data in JSON format using the ArduinoJson library a JSONDocument has to be declared. There are two types: StaticJsonDocument, which uses the stack, and DynamicJsonDocument, which uses the heap. Since we will be sending all the weights of a neural network the JSONDocument will probably be very large, so we use DynamicJsonDocument.

In order to generate a DynamicJsonDocument its size has to be specified, which can be calculated using some functions that the library offers, like JSON_OBJECT_SIZE(number of elements) or JSON_ARRAY_SIZE(number of elements).

After the DynamicJsonDocument is declared its elements can be inserted. Singular values like integers, floats or strings can be added assigning them a name, like accessing a position in an array, as seen in Figure 21:

```
356   // if there's a successful connection:
357   if (client.connect(server, 5000)) {
358     Serial.println(F("connecting..."));
359
360     //send HTTP POST request for everything except the hiddenodes (because they don't fit in the document)
361     client.println(F("POST /FL/sendData HTTP/1.1"));
362     client.println("Host: " + stringHost);
363     client.println(F("User-Agent: ArduinoWiFi/1.1"));
364     client.println(F("Connection: close"));
365     client.println(F("Content-Type: application/json"));
366
367     // Use arduinojson.org/v6/assistant to compute the capacity.
368     const size_t capacity = JSON_OBJECT_SIZE(4) + JSON_ARRAY_SIZE(hiddenWeightsAmt) + JSON_ARRAY_SIZE(outputWeightsAmt) + 60;
369     DynamicJsonDocument doc(capacity);
370     // Serial.println(F("JSONdoc capacity:..."));
371     // Serial.println(capacity);
372
373     doc["message"] = ">";
374     doc["num_epochs"] = num_epochs;
```

Figure 21: Start of HTTP message and declaration of JSONDocument and adding single values. Source: main.ino

When adding arrays it's slightly different. First you a JsonArray has to be declared using createNestedArray("name"); and then the elements have to be added to it one by one, as seen in Figure 22.

```
397     //Serial.println("computing HiddenWeights...");
398     // Serial.print("the number of iterations is: ");
399     // Serial.println(hiddenWeightsAmt);
400     JsonArray deviceHiddenWeights = doc.createNestedArray("HiddenWeights");
401         // Sending hidden layer
402         char* hidden_weights = (char*) myNetwork.get_HiddenWeights();
403         for (uint16_t i = 0; i < hiddenWeightsAmt; ++i) {
404             // Serial.print("iteration: ");
405             // Serial.println(i);
406             if (mixed_precision) {
407                 scaledType weight = scaleWeight(min_weight, max_weight, float_hidden_weights[i]);
408                 scaledType casted = weight;
409                 //Serial.write((byte*) &casted, sizeof(scaledType));
410                 deviceHiddenWeights.add(casted);
411             } else {
412                 //Serial.write((byte*) &float_hidden_weights[i], sizeof(float)); // debug
413                 deviceHiddenWeights.add(float_hidden_weights[i]);
414             }
415         }
416
```

Figure 22: Creation of JsonArray and introducing its data. Source: main.ino

When all data is in the JsonDocument it has to be serialized to be able to be sent to the server. There are two functions to serialize a JsonDocument: serializeJson() and serializeJsonPretty().serializeJson() generates a document without spaces or line break between values, while serializeJsonPretty() generates a document with spaces and line breaks. In order to be sent in a HTTP message to the federated learning server it has to be in serializeJsonPretty() format, because then the content of the JSON can be easily accessed with request.get_json().

We generate a HTTP POST message that has the JSON with the data of the model in its content manually, printing the headers into the server connection as seen partially in Figure 21.

In order to fill the Content-Length field of the HTTP message we use measureJsonPretty(), to obtain the size of the JSON after it's in the correct format, as seen in Figure 23.

```
432
433     int JSONsize = measureJsonPretty(doc);
434     client.print(F("Content-Length: "));
435     client.println(JSONsize);
436     // Terminate headers
437     client.println(); //hay que dejar una linea vacía
438     // Send body
439     Serial.println(F("sendDataFL Sending message"));
440     unsigned long StartTime = millis();
441
442     // //regular version
443     serializeJsonPretty(doc, client);
444     // int stringSize = JSONdocString.length();
445     // Serial.print("Message length: ");
446     // Serial.println(stringSize);
447
448     // Serial.println();
449     unsigned long CurrentTime = millis();
450     unsigned long ElapsedTime = CurrentTime - StartTime;
451     Serial.print(F("sendDataFL Data were sent successfully in: "));
452     Serial.print(ElapsedTime);
```

Figure 23: Sending an Http message with JSON content. Source: main.ino

After the client has sent an HTTP POST message to the federated learning server with its local model in JSON format, it will wait until the federated learning server generates a new model from all the local models it has received.

Then the client will receive an HTTP POST message with the new model in JSON format, and for each value of the JSON that it reads it will overwrite the corresponding value in its local model. After all values have been read the local model will have turned into the model sent by the federated learning server.

With the new model fully obtained the client opens the Serial communications with experiment control to print the time to do federated learning and to be able to receive new samples and do another loop.

In order to give the federated learning server the IP addresses of the clients that will participate in the federated learning process and be able to see how many clients will participate, we tried to implement a function that would register the client sending a short message to the server at the beginning of the execution. However due to limitations of the Arduino devices messages can't be sent using Wi-Fi during the setup, they must be sent in the loop. Since it would generate redundant messages after the first iteration we decided to discard the register function and manage the addresses in the server side, using the already existing messages to store them.

6.4.1.2. Federated learning server

The code of the federated learning server is located in the FL_server_part.py file.

As mentioned earlier, the federated learning server is implemented in a python script using Flask.

The role of the server is to receive trained models from the clients, store them in a local database, and when it has received one trained model from each client it will generate a federated learning model and send it back to all the clients.

The server is purely reactive, it will only operate when it receives a message from a client. The only time it sends a new message is when it sends the generated model to the clients, and that only happens as a response to having received the model of the last client.

In the next figure we can see the initialization of the Flask server:

```python

if __name__ == '__main__':
    from argparse import ArgumentParser

    parser = ArgumentParser()
    parser.add_argument('-p', '--port', default=5000, type=int, help='port to listen on')
    args = parser.parse_args()
    port = args.port

    app.run(host='0.0.0.0', port=port)
```

Figure 24: Code to initialize the flask server. Source: FL_server_part.py

By default a Flask server is only accessible from the same computer where it's being executed, so external devices like the Portenta boards can't access it, even if they have its IP address and its port. To make the Flask server open to external devices the "host" parameter of the app.run() function has to be set to '0.0.0.0', which makes the server publicly available.

The port number of the federated learning server is the default value for Flask servers, 5000.

To differentiate between different kinds of messages, an URL can be assigned to a function to only be executed when the message is sent to that specific URL.

```python
#get the model of a board
@app.route('/FL/sendData', methods=['POST'])
def FlGetModel():
    #check if the board has contacted before, and if not add its address to the list
    adress = request.remote_addr
    if not (adress in federated_learning_server.client_adresses):
        #the adress isn't in the list, so we add it
        federated_learning_server.client_adresses.append(adress)

    #get device index from the list of adresses
    device_index = federated_learning_server.client_adresses.index(request.remote_addr)
    values = request.get_json()

    num_epochs = values.get('num_epochs') #int
    print("num_epochs: ")
    print(num_epochs)
    federated_learning_server.devices_num_epochs.append(num_epochs)

    min_w = values.get('min_weight') #float
    max_w = values.get('max_weight') #float
```

Figure 25: Flask server receiving a model. Source: FL_server_part.py

In Figure 25 there is an example of a function of the federated learning server, where it can be seen that it has a specific URL route assigned and it only accepts HTTP messages with the POST method. To send a message that activates this function the client has to use:

```
IPAddress server(192,168,0,15);
client.connect(server, 5000);
client.println(F("POST /FL/sendData HTTP/1.1"));

...
```

The solution to the problem of registering clients can also be seen: the server has an internal list of client addresses that is initialized empty, and for each message that it receives it checks if the address of the sender is in the list of clients. If it's not, it gets added to the list of addresses and starts working with the contents of the message.

The "request" variable contains the HTTP message that was sent to this address, and it contains all the values of the message, like the IP address of the sender, using request.remote_addr, or the body of the message, using request.get_json() since the data was sent in JSON format.

After obtaining the JSON of the message its contents can be accessed using the name of each field, with get(). All the data is stored in a local database, using the index of the address of the sender in the list of addresses to assign its position, so the data of each client can be accessed whenever it's necessary.

The message that is sent to /FL/sendData contains all the data of the trained model of a client. After storing all of it in its according position, the server checks how many clients have sent a model, and if it's equal to the total number of clients it begins to do federated learning. The number of clients that will participate is specified in the "numdevices" variable, at the beginning of the code.

The federated learning model is generated doing a weighted average of all the clients' models that are stored in the server, using the number of epochs of each client as the weights for the average.

Once the federated learning model is generated it's sent to all the clients in parallel using python threads. To send the model to the clients its data is stored in a JSON and it is sent to all the addresses in the list of clients of the server using a HTTP POST request.

The server sends the messages using requests.post, which returns a Boolean that is used to see if the message was sent correctly or not.

After it has sent the model to all the clients it won't do anything else until it receives a new message from a client.

When the script that manages the server is executed it will print on the terminal which IP addresses will be used, combined with the port in URL format. An example can be seen in the following figure:

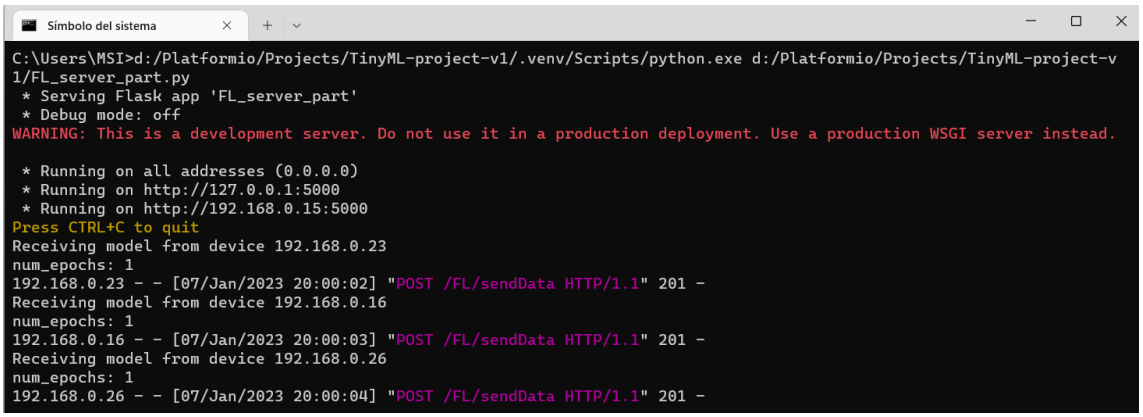

Figure 26: Example of the flask server receiving messages. Own elaboration

The first address is the local address which can only be accessed from the same machine as the execution of the script. By default its value is http://127.0.0.1:5000.

The second address is the public address, which can be accessed by any device, including the Portenta boards. In this case it's http://192.168.0.15:5000.

Whenever the server receives a message, after doing all the corresponding code including any prints, it shows a line with some information about the message:

- IP address of the sender
- Date and time when the message was received
- The header of the received message: it's colored purple if the message was received correctly without any error during its corresponding code, or colored red if there were any errors.
- A HTTP response status code: a 201 means that a POST request was successful.

In the next two figures there is an example of the server receiving small models and sending the generated model to the clients after doing federated learning, where all the data is printed for testing:

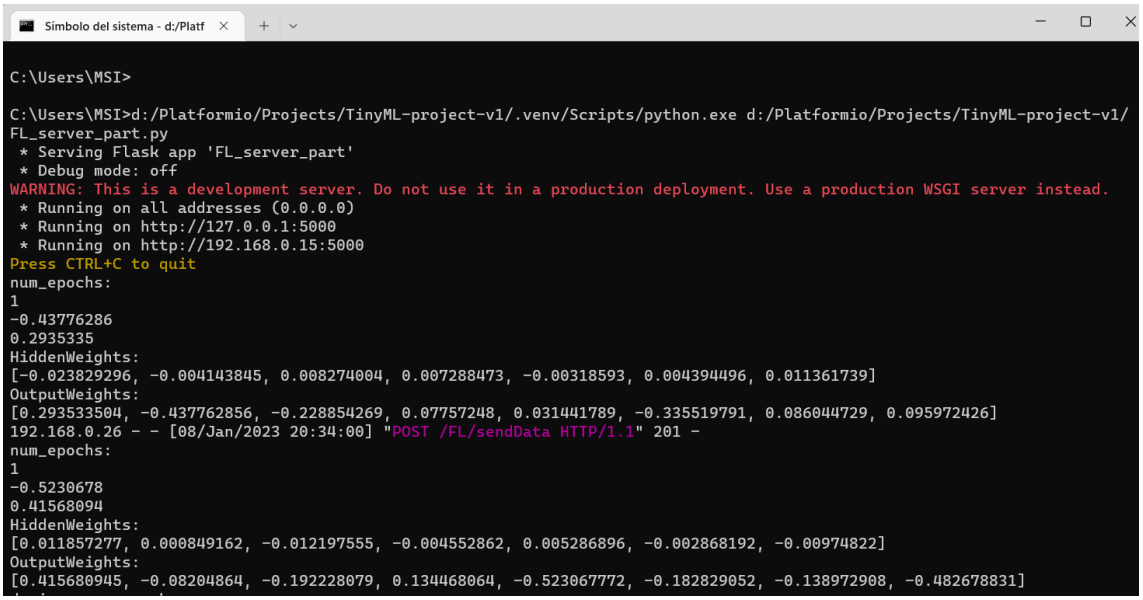

Figure 27: Example of server receiving a model in execution. Own elaboration

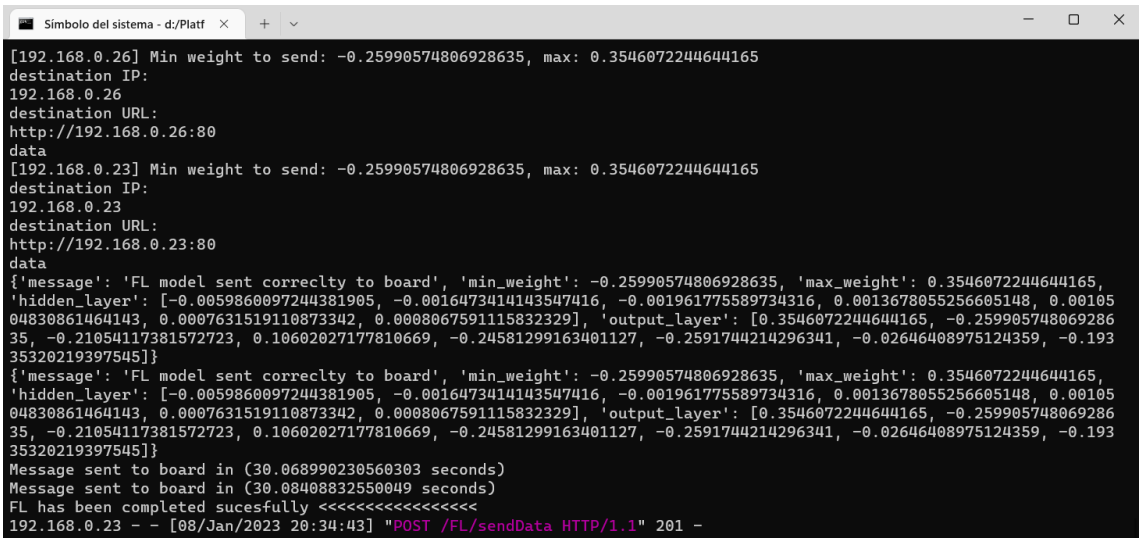

Figure 28: Example of server sending a model in execution. Own elaboration

In this example there are two clients, one with IP 192.168.0.23 and one with IP 192.168.0.26. It can be seen that the federated learning process is done as a response to receiving a /FL/sendData message, specifically when that message is from the last client that hadn't sent a model yet. In this case the device 192.168.0.26 sends its model first and when 192.168.0.23 sends its model a federated learning round begins and the server sends the generated model at the end.

6.4.1.3. Experiment control

The code of experiment control is located in the experiment_control.py file.

This script controls the flow of the experiments and manages the data that doesn't have to be in the federated learning server, like the inputs.

It communicates with the client using the Serial port.

The main tasks of experiment control are:

- Getting how many clients there will be and their ports.
- Sending an initial model to all clients
- Sending a train command to all clients and the sample to train
- Obtaining data about the performance

When it's executed, it asks the user to introduce the number of clients that will participate in the federated learning process. Afterwards it prints a list of currently available devices and their ports, and the user has to write the port of the devices that will be used.

With the ports of the participating clients it can begin sending data. First it sends the initial model, which is randomly generated, to all the clients in parallel using threads:

```python
# Send the blank model to all the devices
def sendInitWeights():
    threads = []
    for i, d in enumerate(devices):
        hidden_layer = np.random.uniform(-0.5,0.5, size_hidden_layer).astype('float32')
        output_layer = np.random.uniform(-0.5, 0.5, size_output_layer).astype('float32')

        thread = threading.Thread(target=init_network, args=(hidden_layer, output_layer, d))
        thread.daemon = True
        thread.start()
        threads.append(thread)
    for thread in threads: thread.join() # Wait for all the threads to end

# Start the execution of the script
getDevices()

sendInitWeights()

```

Figure 29: Generating the initial mode. Source: experiment_control.py

After it has sent the initial model to all the devices, the training process begins. The script selects a sample file from the datasets folders to send its contents to all clients. The sample is also sent in parallel using threads.

In order to train the model of a client, experiment control sends:

- A train command: to tell the client that the training process started
- A button number: the button that would be pressed if the sample was introduced using the microphone. It's used to assign a keyword to the sample.
- If the training is done with only forward propagation or it also does backward propagation.
- The content of the sample file.

```python
def sendSample(device, samplePath, num_button, deviceIndex, only_forward = False):
    with open(samplePath) as f:
        ini_time = time.time() * 1000
        data = json.load(f)
        if debug: print(f'[{device.port}] Sending train command')
        device.write(b't')
        startConfirmation = device.readline().decode()
        while (startConfirmation[:-2] != "ok"):
            if debug: print(f"[{device.port}] Incorrect startConfirmation:", startConfirmation)
            startConfirmation = device.readline().decode()
        if debug: print(f"[{device.port}] Train start confirmation:", startConfirmation)

        device.write(struct.pack('B', num_button))
        button_confirmation = device.readline().decode() # Button confirmation
        while (button_confirmation[:-3] != "Button "):
            if debug: print(f"[{device.port}] Incorrect button_confirmation:", button_confirmation)
            button_confirmation = device.readline().decode()
        if debug: print(f"[{device.port}] Button confirmation: {button_confirmation}")

```

Figure 30: Beginning of sending a sample. Source: experiment_control.py

In Figure 30 we can see the beginning of sending the data that is needed to train the model of a client. In order to control that the client and experiment control are coordinated during the execution, after each data that is sent to the client, experiment

control waits for a confirmation message. The script won't advance until it receives the confirmation, so it will only send data when the client is ready to receive it.

After all the required data is sent, experiment control waits until the client sends "graph", which means that it will begin to send data about the performance of the training. Experiment control stores these values to generate a graph at the end of the execution of the script.

When all the selected sample files are used, it prints the training loss and accuracy of each device and generates a graph with the performance values.

6.4.1.4. Summary

This version of the project implements Federated learning with Wi-Fi sending and receiving entire models using a single HTTP message. When testing very small neural networks this solution works properly, although it is quite slow considering how small they are.

However, this implementation doesn't work when using bigger networks. When building the JSON with the model data for the HTTP message, if the capacity assigned to the JsonDocument is too large to fit in the memory of the device the JsonDocument automatically empties itself without warning, so all the messages would be empty with this method.

Since this project should use decently sized networks, we designed a solution in the next version.

6.4.2. Version 2: Segmented models

The first idea for this version was sending the weights of the hidden layer in one message, then the weights of the output layer in another, and then any remaining data in another message. However, with just 25 neurons in the hidden layer the amount of weights was already too large for the JsonDocument.

Therefore, in order to use bigger networks the hidden node weights will have to be sent using more than one HTTP message.

In the following figure there is a diagram that represents the workflow of this version:

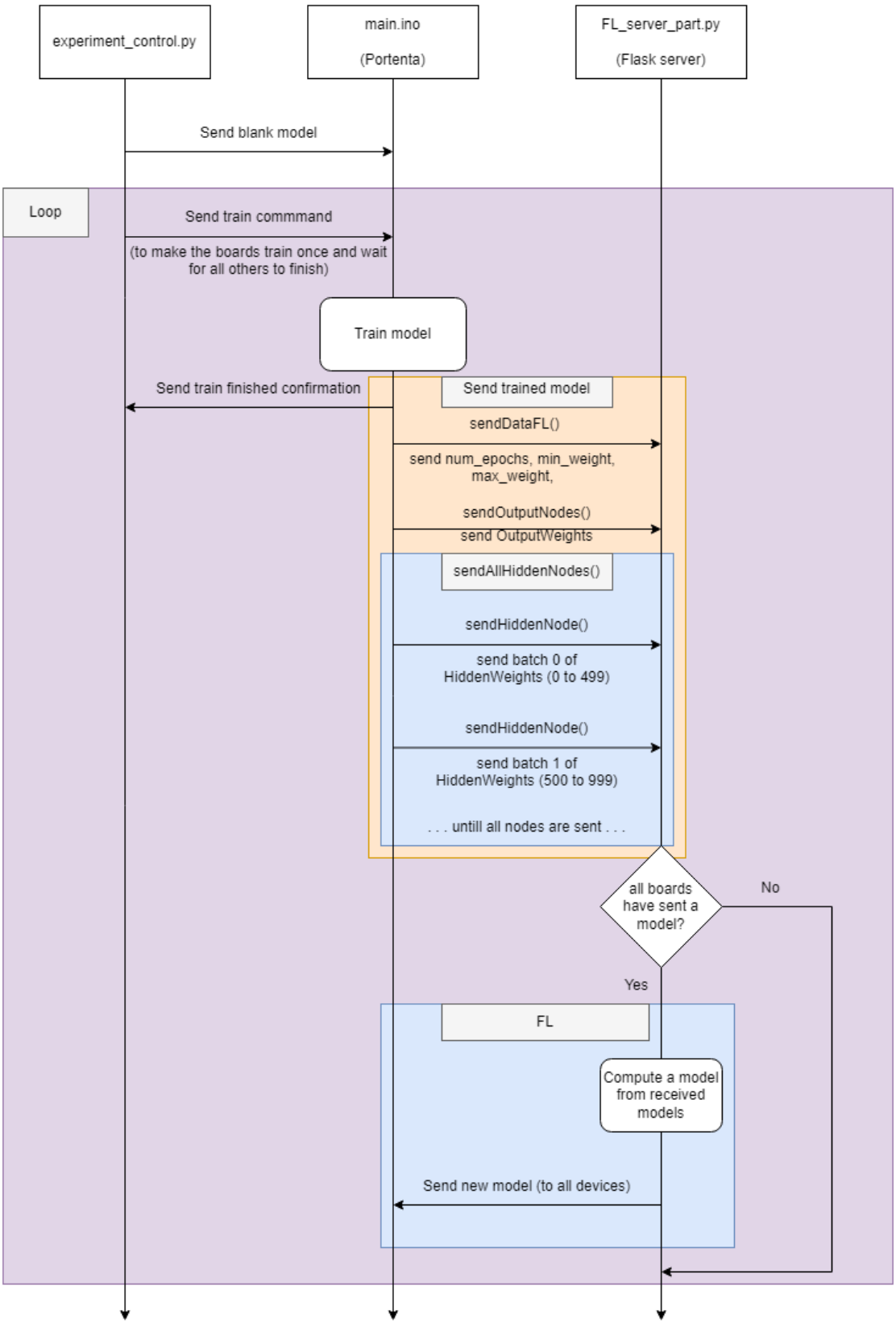

Figure 31: Diagram of version 2. Own elaboration

Now we will explain the changes to each part of the project.

6.4.2.1. Client

In the main loop, after training instead of just sending a single message to the server with sendDataFL(), now there is also sendOutputNodes() and sendAllHiddenNodes().

sendDataFL() sends the single values: num_epochs, min_weight, max_weight.

sendOutputNodes() sends the output layer weights. It's the same code that the one that added into the JSON in the previous version, but in a separate function.

sendAllHiddenNodes() will send all the hidden layer weights, but since there are so many they will be sent in batches. The number of weights per batch is defined in the variable batchSize, which can be changed to try to find an equilibrium between message size and total transmission speed.

In order to keep the batches ordered each one is assigned a batch number.

When a whole batch can't be created from the remaining weights it means that the current batch is the last one, which theoretically means that all the model was sent.

sendHiddenNode() is the function that sends a single batch of weights. It takes as parameters the number of the batch, the size of the current batch, the size of all the other batches, and the positions of the first and last weights of the batch in the complete list of hidden weights of the neural network.

The positions are used to obtain the nodes from the list to send them in the message.

In order to have more stable connections we began using the Portenta_H7_AsyncWebServer [30] library in main.ino to receive the messages from the server. This library allows to make more than one connection at the same time, and also allows to receive messages during the execution of other code, which is useful to prevent data losses. Additionally, it's faster than a regular Wi-Fi connection.

6.4.2.2. Federated learning server

The federated learning server now has three URL to receive the three different messages that the client sends.

The DataFL and OutputNodes use the same code that was used in the previous version to access its values. However, for the HiddenNodes it will receive a batch of weights instead of all of them, so it has to store the weights of each batch in a different position.

```
#  Receiving model...
# print("HiddenWeights: ")
received_hidden_layer = values.get('HiddenWeights')
# print(received_hidden_layer)
for i in range(lastInBatch): # hidden layer
    if federated_learning_server.mixedPrecision:
        scaledWeight = received_hidden_layer[i] #int
        #scaled_in_float = readFloat(d)
        # float_weight = readFloat(d)
        weight = deScaleWeight(min_w, max_w, scaledWeight)
        # if i < 5 and d.port == 'com6': print(f"[{d.port}] Recevied Weight {i}: {float_weight}")
        # if abs(float_weight - weight) > 0.3: print(f"[{d.port}] Scaled weight: {scaledWeight} (float: {scaled_in_float}),
    else: weight = received_hidden_layer[i] #float

    # Received Weight {i}: {weight}")
    federated_learning_server.devices_hidden_layer[device_index][i+batchNumber*batchSize] = weight

#when the server receives a model calculate the medium of all previously received models
# print("lastInBatch: ")
# print(lastInBatch)
# print("batchSize: ")
# print(batchSize)
print(lastInBatch != batchSize)
if (lastInBatch != batchSize):
    # the whole model is received
    federated_learning_server.num_devices_sent = federated_learning_server.num_devices_sent+1
    # if all devices have sent a model start FL
    if (federated_learning_server.num_devices_sent == federated_learning_server.numdevices):
        federated_learning_server.num_devices_sent = 0
        doFL()
```

Figure 32: Receiving a batch of weights in the server. Source: FL_server_part.py

In the previous figure we can see how the additional parameters are used to put the weights in their correct position: with i+batchNumber*batchSize. Additionally lastInBatch is used to see if this batch has less elements than the rest, to see if it's the last one, because if it is the whole model is in the server and it can begin to do federated learning.

When the federated learning sends the generated model it also uses multiple messages, using similar code to the client.

6.4.2.3. Experiment control

Experiment control doesn't have any important changes, since the modifications were done in the HTTP messages, which don't influence this part of the project.

6.4.2.4. Summary

This version of the project implements Federated learning with Wi-Fi sending and receiving models using a multiple HTTP messages. To do so, we send various data I one message, the output layer weights in another message, and the hidden layer weights divided in multiple messages.

This version works with any amount of neurons. However, the Wi-Fi messages are still very slow, which is a problem when large networks need to send a lot of messages to send their model.

We will explore why it is so slow and how to increase the speed in the next version.

6.4.3. Version 3: Faster connections and security

During the testing of the project we could see that AsyncWebServer could receive Wi-Fi messages pretty fast, which meant that the slowness of the other communications wasn't a hardware issue.

In order to see what could be the cause of the problem we used wireshark to get additional information about the HTTP messages that the Portenta sends.

Source	Destination	Protocol	Length	Info
192.168.0.11	77.234.43.225	HTTP	360	GET /R/A3sKIDAwMGI5YTY4OTU5ODQ1OGU4NTg1MmYwOTVlZWE4NmU0EgQEKBEiGI4CIgH-KggIBBCB7ZakASoICAM
77.234.43.225	192.168.0.11	TCP	60	80 → 49709 [ACK] Seq=793 Ack=307 Win=7 Len=0
192.168.0.16	192.168.0.11	TCP	60	62112 → 80 [SYN] Seq=0 Win=2144 Len=0 MSS=536
192.168.0.11	192.168.0.16	TCP	58	80 → 62112 [SYN, ACK] Seq=0 Ack=1 Win=65392 Len=0 MSS=1460
192.168.0.16	192.168.0.11	TCP	60	62112 → 80 [ACK] Seq=1 Ack=1 Win=2144 Len=0
192.168.0.16	192.168.0.11	TCP	80	62112 → 80 [PSH, ACK] Seq=1 Ack=1 Win=2144 Len=26 [TCP segment of a reassembled PDU]
192.168.0.16	192.168.0.11	TCP	124	62112 → 80 [PSH, ACK] Seq=27 Ack=1 Win=2144 Len=70 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=97 Win=65296 Len=0
192.168.0.16	192.168.0.11	TCP	84	62112 → 80 [PSH, ACK] Seq=97 Ack=1 Win=2144 Len=30 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=127 Win=65266 Len=0
192.168.0.16	192.168.0.11	TCP	60	62112 → 80 [PSH, ACK] Seq=127 Ack=1 Win=2144 Len=2 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=129 Win=65264 Len=0
192.168.0.16	192.168.0.11	TCP	70	62112 → 80 [PSH, ACK] Seq=129 Ack=1 Win=2144 Len=16 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=145 Win=65248 Len=0
192.168.0.16	192.168.0.11	TCP	60	62112 → 80 [PSH, ACK] Seq=145 Ack=1 Win=2144 Len=3 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=148 Win=65245 Len=0
192.168.0.16	192.168.0.11	TCP	60	62112 → 80 [PSH, ACK] Seq=148 Ack=1 Win=2144 Len=2 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=150 Win=65243 Len=0
192.168.0.16	192.168.0.11	TCP	60	62112 → 80 [PSH, ACK] Seq=150 Ack=1 Win=2144 Len=2 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=152 Win=65241 Len=0
192.168.0.16	192.168.0.11	TCP	60	62112 → 80 [PSH, ACK] Seq=152 Ack=1 Win=2144 Len=3 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=155 Win=65238 Len=0
192.168.0.16	192.168.0.11	TCP	60	62112 → 80 [PSH, ACK] Seq=155 Ack=1 Win=2144 Len=2 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=157 Win=65236 Len=0
192.168.0.16	192.168.0.11	TCP	60	62112 → 80 [PSH, ACK] Seq=157 Ack=1 Win=2144 Len=1 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=158 Win=65235 Len=0
192.168.0.16	192.168.0.11	TCP	60	62112 → 80 [PSH, ACK] Seq=158 Ack=1 Win=2144 Len=1 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=159 Win=65234 Len=0
192.168.0.16	192.168.0.11	TCP	60	62112 → 80 [PSH, ACK] Seq=159 Ack=1 Win=2144 Len=1 [TCP segment of a reassembled PDU]
192.168.0.11	192.168.0.16	TCP	54	80 → 62112 [ACK] Seq=1 Ack=160 Win=65233 Len=0

Figure 33: Wireshark capture of the communication. Own elaboration

On Figure 33 we can see a wireshark capture with messages between a Portenta board acting as a client and the federated learning server. In this case 192.168.0.16 is the IP address of a Portenta board and 192.168.0.11 is the IP address of the flask server.

We can see that, except for the headers at the beginning of the communication, Portenta sends fragments of one or two bytes each time.

In order to fix this problem we use the StreamUtils [31] library to force Portenta to write in chunks of bytes, massively improving the speed of the messages. This is done adding buffering to the Streams that Arduino uses, reading or writing many bytes at once.

The changes to each part of the project are:

6.4.3.1. Client

To improve the writing speed of the client we use a WriteBufferingClient from the StreamUtils library to make it write a number of bytes defined in the variable byteCapacity.

```
550
551     int JSONsize = measureJsonPretty(doc);
552     client.print(F("Content-Length: "));
553     client.println(JSONsize);
554     // Terminate headers
555     client.println(); //hay que dejar una linea vacía
556     // Send body
557     Serial.println(F("sendDataFL Sending message"));
558     unsigned long StartTime = millis();
559     //optimized for speed
560     WriteBufferingClient bufferedWifiClient{client, byteCapacity};
561     serializeJsonPretty(doc, bufferedWifiClient);
562     bufferedWifiClient.flush();
563     // //regular version
564     // serializeJsonPretty(doc, client);
565     // // int stringSize = JSONdocString.length();
566     // // Serial.print("Message length: ");
567     // // Serial.println(stringSize);
568
```

Figure 34: example of buffered writing. Source: main.ino

Declaring the WriteBufferingClient locally like in Figure 34 makes it so it works perfectly with up to 128 bytes, greatly improving the speed compared to the previous version.

However, using more than 128 bytes sometimes generates connection errors during the execution, where the connection in closed before it can end properly. Since the WriteBufferingClient is declared locally, when the function ends it's deleted, so if it's deleted before it can send all its data it will cause the error.

In order to solve this problem the WriteBufferingClient can be declared globally instead, which makes it work properly with any amount of bytes.

To be able to use larger amounts of samples, the client will now wait until it has trained with a certain amount of samples before it begins a federated learning round.

Up until now the program supposed that all connections were successful during the execution, if any message is lost its data is also lost. In reality there probably will be some connection errors during the execution, since a lot of messages are being sent.

The client won't be able to know if its message was successful until it receives a response from the server. The message will be successful if the HTTP response status code is 201.

Therefore, to be sure that all messages are successfully sent we make the client wait for the response of each message, and once it has the response it checks the HTTP response status code. If it's not the correct code it waits some time and then it tries to do the communication again from the beginning. We define a maximum amount of retries to prevent an infinite loop. When the same connection fails at least 3 times in a row, we reset the Wi-Fi connection to see if there is something wrong with it.

```
983         // wait for server response
984         while (!client.available()){
985         }
986         // delay(responseDelay);
987         if (debug_text){
988             Serial.println(F("Server response received: "));
989         }
990         ////HTTP/1.1 201 CREATED -> successful message
991         ////else connection error
992         char responseCode[13] = "HTTP/1.1 201";
993         int responseIt = 0;
994         while (client.available()){
995             char c = client.read();
996             if (responseCode[responseIt] == '2'){
997                 if (c != '2'){
998                     retry = true;
999                 }
1000                // keep the while running to empty the recieved message buffer and close the connection properly.
1001                responseIt++;
1002            }
1003            if (c == responseCode[responseIt]){
1004                responseIt++;
1005            }
1006            // Serial.print(c);
1007        }
```

Figure 35: Checking if message was sent correctly. Source: main.ino

In Figure 35 we can see the implementation to check if the message was sent correctly. Although it adds a small delay to each message, being able to know if there are errors and retry to solve them makes the program more reliable. Since connection problems happen when the data is being sent and not when it's being prepared, we move all the code to generate the JSON outside of the retry loop that tries to send the message, since the content of the JSON will always be the same.

Another problem that can be prevented is that sometimes if the Flask server receives the same kind of message from two clients at the exact same time one of them gets blocked and can't send the message. This could happen with the /FL/sendData message since it's so small. To solve this we add a small delay at the beginning of the round of federated learning, using a different value for each client. For simplicity we used the index of the device in the list of clients.

Additionally, to make sure that all the clients will be working concurrently, instead of beginning to train right after it receives the new model from the federated learning server, it will have to wait until all the other clients have also received the model, using a short message sent by the server.

6.4.3.2. Federated learning server

Although in theory all hidden layer weight batches are received in order, since they are sent using Internet there is a possibility that they don't reach the server in the same order as they are sent. If this happens with the batch containing the last weights of the hidden layer, using the method from the previous version the server would interpret it as having received all the weights from the hidden layer, and if it can it starts the federated learning process, having parts of the hidden layer empty because it hadn't received the messages with those weight batches yet.

In order to prevent that, instead of looking for the last batch to see if the layer is complete, the server will use the amount of batches received instead, which makes it so

it will have to receive exactly all the weight batches to consider the hidden layer complete. The position of each weight in the layer was already calculated in the previous version of the project, using the batch number and the batch size, so even if the messages reach the server completely unordered there would be no problems in this version and the hidden layer of the received model would be received correctly.

The total number of hidden layer weight batches will be sent by the client as another parameter in the hidden weight batch message.

As mentioned when explaining the client, while the server is sending the federated learning model to all the clients, if one of them receives the model and begins training immediately it could begin sending another model back to the server before the server has finished sending the previous model to all the other clients. If this happens some data could be lost when the server resets some variables to prepare for the next round of federated learning.

To prevent this, after the server has sent the generated model to all the clients and prepared its variables for the next round of federated learning, it sends a message to all the clients to let them know that they can begin sending their models. The message is sent using threads to try to send it to all the clients as quickly as possible, as seen in the next figure:

```
306     threads = []
307     for d in federated_learning_server.client_adresses:
308         thread = threading.Thread(target=FLSendModel, args=(d, hidden_layer, output_layer))
309         thread.daemon = True
310         thread.start()
311         threads.append(thread)
312     for thread in threads: thread.join() # Wait for all the threads to end
313 
314     print("FL has been completed sucesfully")
315 
316     # reset values for next FL
317     federated_learning_server.devices_num_epochs.clear()
318     federated_learning_server.devices_hidden_layer_batch_amount = np.zeros((federated_learning_server.numdevices))
319 
320     # Telling all registered devices that fl is finished
321     print("Telling the devices that FL is finished")
322     threads = []
323     for d in federated_learning_server.client_adresses:
324         print(d)
325         # have to add a coma in args if there is only 1 value, if not it will read it as a list of chars
326         thread = threading.Thread(target=FLSendConfirmation, args=(d,))
327         thread.daemon = True
328         thread.start()
329         threads.append(thread)
330     for thread in threads: thread.join() # Wait for all the threads to end
```

Figure 36: Sending a "FL is finished" confirmation to the clients. Source: FL_server_part.py

The message is a HTTP GET, since it doesn't need to send any values and it's faster than HTTP POST.

Additionally, the server will also try to send again any message that isn't received correctly by a client, although with a limited amount of retries to prevent infinite loops. To check if a message is received correctly it uses the return value of requests.post or requests.get, if it's false the message wasn't received correctly.

6.4.3.3. Experiment control

In order to know exactly when a round of federated learning has finished, experiment control will also receive a confirmation message when all the clients have received the model generated by the server, although this message will be sent by the client using the Serial port, right after the client receives the confirmation from the server.

6.4.3.4. Summary

Thanks to buffering the streams that Portenta uses to send Wi-Fi messages, this version of the project works at least 100 times faster when using 1024 bytes in the buffer compared to the two previous versions.

Additionally, with the checks and retries added after each message, the project can work even when the internet connection isn't very reliable, although the number of possible retries might have to be increased if the connection is expected to be very weak.

Thanks to these improvements, this version of the project should work in real scenarios.

7. Evaluation

7.1. Performance

7.1.1. Studying the influence of buffering the Wi-Fi messages

As we explained in section 6.4.3.1, a buffer can be added to the write operations of the Portenta board to make it write chunks of bytes at once instead of byte by byte. The amount of bytes that will be written is specified when declaring the buffer.

To study the improvement obtained when adding a buffer to the write operations of the Portenta board, we test executing the project using different amounts of bytes in the buffer and obtain the time that it takes to do a round of federated learning, since the buffer is used in all the write operations from the Portenta board acting as a client to the federated learning server.

We use a neural network with 650 neurons in the input layer, 25 in the hidden layer and 3 in the output layer. There will be three Portenta boards acting as clients.

In the following figures and table we can see the average times obtained from the tests, in milliseconds and in minutes.

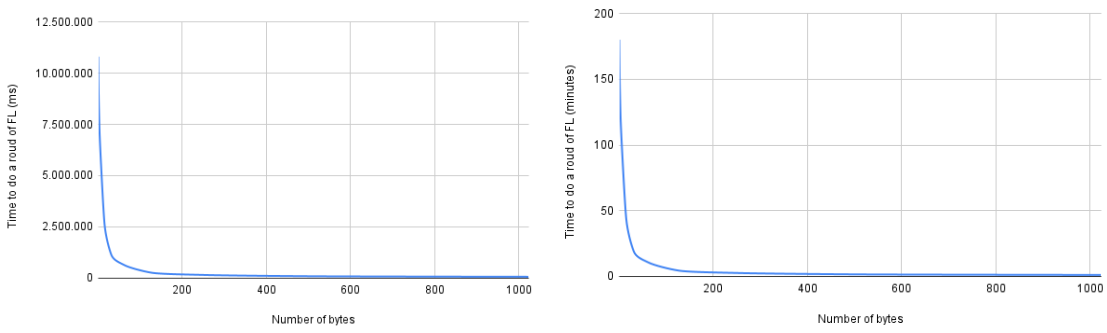

Figure 37: Time to do a round of FL using different amounts of bytes. Own elaboration

Amount of bytes used	Time to do a round of FL (ms)	Time to do a round of FL (minutes)
1 (without buffering)	10.821.534	180,36
2	9.147.852	152,46
4	7.261.421	121,02
8	5.401.741	90,03
16	2.622.096	43,70
32	1.143.412	19,06
64	629.849	10,50
128	266.942	4,45
256	157.490	2,62
512	94.992	1,58
1024	68.612	1,14

Table 9: Time to do a round of FL using different amounts of bytes

The quality of the internet connection also plays a big role in the time that will take to send the messages needed to do the round of federated learning. In order to try to have the same conditions in all the tests we did all the executions from the same physical location.

From the results we can see that adding buffering and increasing the amount of bytes that it uses generates a very large improvement in the execution time.

Although the amount of bytes to use in the buffer is duplicated in each increase, the obtained time won't be exactly half as short, mainly because the federated learning process doesn't only have the write operations, all the operations regarding the content of the messages and building a model will take the same amount of time regardless of the buffering bytes. Additionally, the strength of the internet connection also influences the time for each message.

Since using 1024 bytes in the buffer gives the best time for each round of federated learning, we will use that value for all following tests.

7.1.2. Studying the influence of the hidden weight batch size

As explained in section 6.4.2, from the second version of the project we send the weights of the hidden layer in batches to be able to use the JSON format to send them.

In order to see which would be the optimal amount of weights to send in each batch we execute the project using different amounts and check the average time to do a round of federated learning for each one.

The conditions for the tests are the same as in the previous test in section 7.2.1, using three boards with a neural network with 650 neurons in the input layer, 25 in the hidden layer and 3 in the output layer.

In the following figures and table we can see the average times obtained from the tests, in milliseconds and in minutes.

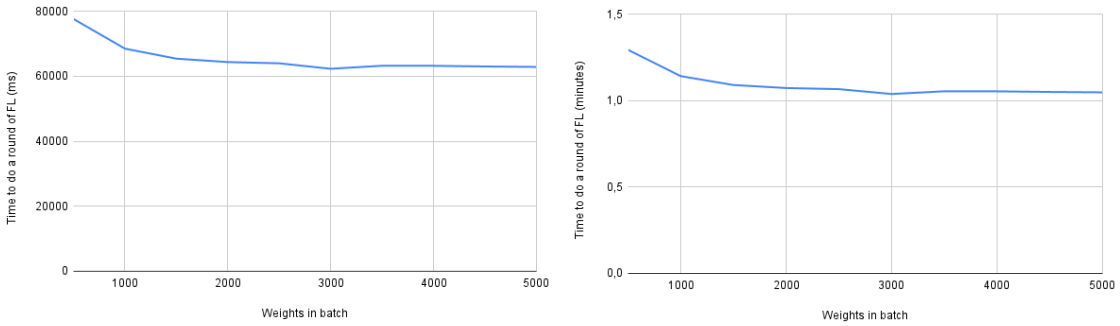

Figure 38: Time to do a round of FL using different amounts of weights in each batch. Own elaboration

Amount of weights in each batch	Time to do a round of FL (ms)	Time to do a round of FL (minutes)
500	77592	1,29
1000	68474	1,14
1500	65390	1,08
2000	64346	1,07
2500	63960	1,06
3000	62278	1,03
3500	63212	1,05
4000	63194	1,05
4500	62989	1,04
5000	62849	1,04

Table 10: Time to do a round of FL using different amounts of weights in each batch

Although at first there is a small improvement in the execution time when increasing the amount of hidden layer weights in each message, after reaching 1500 weights per batch the time to do a round of federated learning stays relatively the same. However, increasing the amount of weights in each message also increased the chance to get an error while the message was being sent, likely because with longer messages the Portenta board has to keep the connection with the server open and stable for a longer time, while with shorter messages the connection can be closed earlier.

Therefore, since the time doesn't improve too much with more than 1500 weights but the chance of error does increase, we use 1500 weights in each hidden layer weight batch.

7.1.3. Studying the performance of the model obtained with FL

To study the performance of the model we will check the Mean Squared Error and the percentage of successful inferences of the model.

We use the samples from the datasets folder, which contain sound samples of three spoken words: "montserrat", "pedraforca" and "vermell". Each kind of word is assigned

a numeric value and the inference will consist in assigning the correct numeric value to an input sound.

Instead of having a separate testing dataset, to reduce the number of data being sent to the Portenta board we use the same samples to test and train: when the board receives a sample from experiment control, it uses it to test if making an inference with its local model would give the correct result. After the testing is done and the result is obtained, the sample is used to train the model.

Since the testing is done before the training the sample will be unknown to the model at the time of testing.

The conditions for the tests are the same as in the previous test in section 7.2.1, using three boards with a neural network with 650 neurons in the input layer, 25 in the hidden layer and 3 in the output layer.

We use 120 samples to train the model of each device, doing a round of federated learning every 10 samples. The momentum of the model will be set to 0.9 and the learning rate to 0.6. It will do forward propagation and backward propagation.

In the following figure we can see the Mean Squared Error (or Loss) of the models of each Portenta board in a different color.

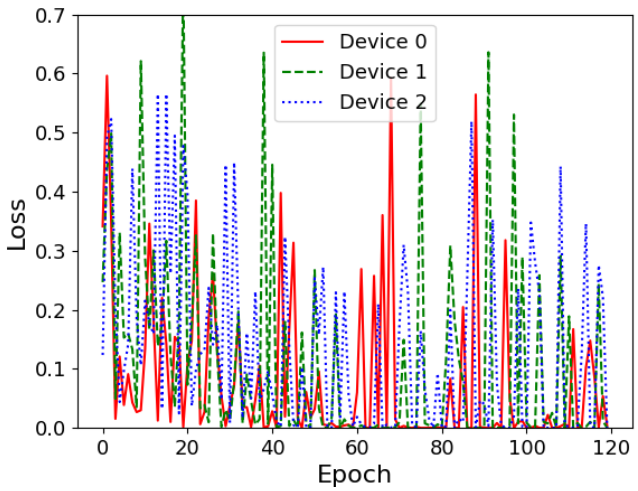

Figure 39: MSE of the models over the epochs. Own elaboration

We can see that the Mean Squared Error generally gets smaller as the number of epochs increases.

In the following figure we can see the percentage of successful inferences of each model.

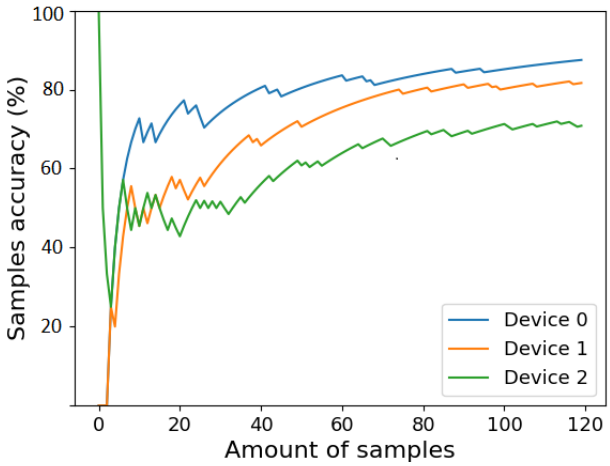

Figure 40: Accuracy of the models over the epochs. Own elaboration

The accuracy is calculated making a sum of the successful inferences divided by the total amount of inferences, which is equal to the amount of samples.

We can see that the accuracy of the model increases with the number of epochs, and that the models of the three devices have closer results with more rounds of federated learning. After 12 rounds of federated learning the models have between 70 to 90% accuracy.

7.1.4. Studying the influence of the network size in the performance

Using the same environment as in the test in section 7.2.3, we will study the effect of using neural networks of different sizes, by modifying the amount of neurons in the hidden layer.

First we try reducing the hidden layer to 5 neurons, obtaining the Mean Squared Error and accuracy represented in the following graphs:

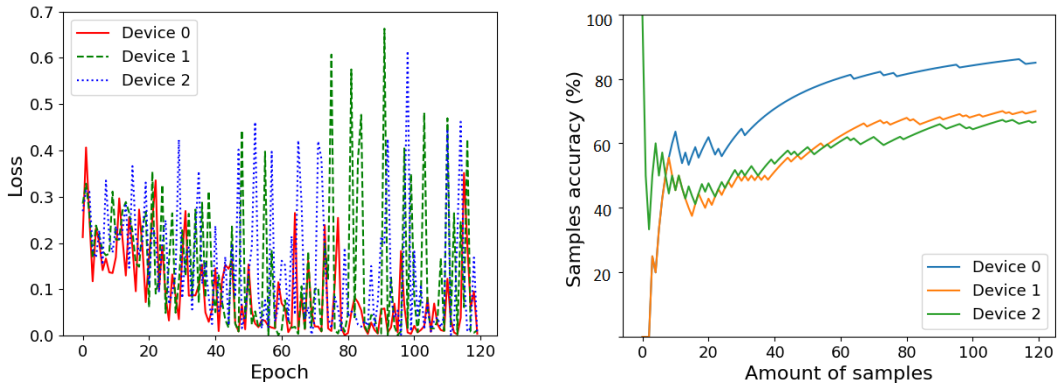

Figure 41: MSE and accuracy with 5 neurons in HL. Own elaboration

We can see that one of the models reaches 80% accuracy, but the other two only reach 60% accuracy after 12 rounds of federated learning. The final accuracies are a bit worse than using 25 neurons.

The execution of the whole process is very fast compared to using 25 neurons, since the model has less weights to be sent and received.

Duplicating the amount of neurons in the hidden layer, to have 10, we obtain the following results:

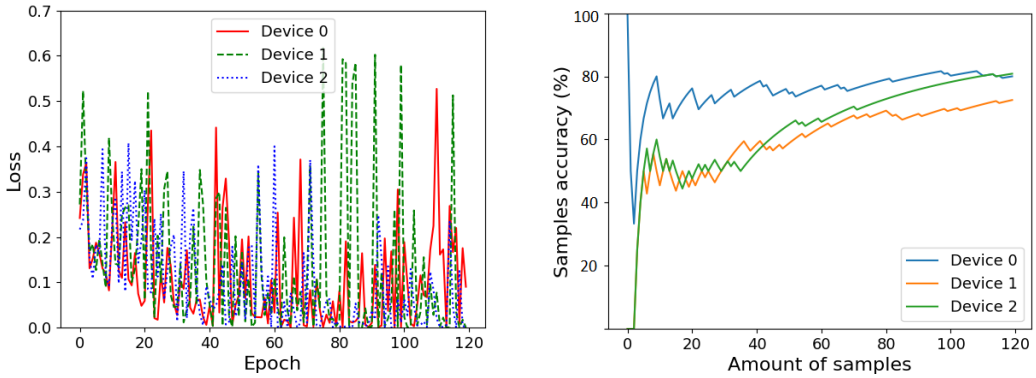

Figure 42: MSE and accuracy with 10 neurons in HL. Own elaboration

Two devices reach 80% accuracy, while the third one only reaches 70% in 12 rounds of federated learning. The final accuracies are very similar to the ones with 25 neurons, although they are slightly lower. The execution is also slightly faster compared to using 25 neurons.

We experiment with a hidden layer of 50 neurons, twice as many as the original experiment, and obtain the following results:

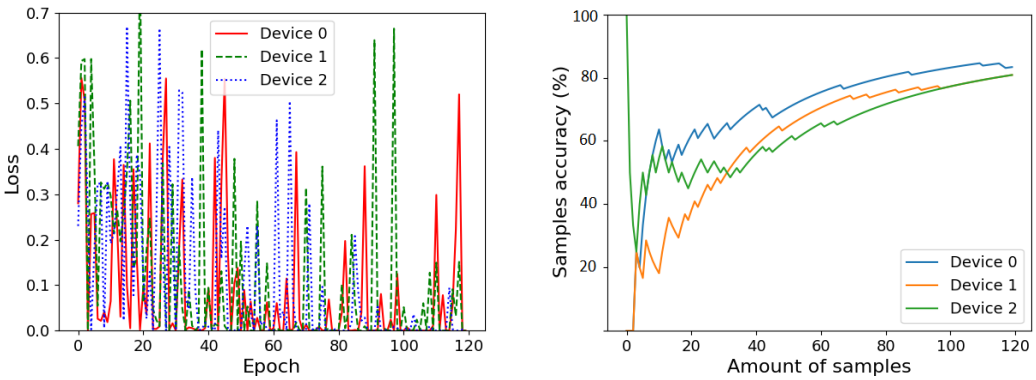

Figure 43: MSE and accuracy with 50 neurons in HL. Own elaboration

In this case the three devices reach over 80% accuracy after the 12 rounds of federated learning.

Although the final accuracies are slightly better than using 25 neurons, the execution time of the project is much slower with 50 neurons than with 25, it takes over twice as long to finish and is more prone to connection errors because of the large amount of messages sent for all the weights.

To keep a balance between accuracy and execution time we decide to use 25 neurons in the hidden layer, since it obtains good results in a decent timeframe.

8. Conclusions and future work

8.1. Technical and personal conclusions

As of the third version of the project, the federated learning process can be completely done using Wi-Fi for the communication between the edge client and the federated learning server in a good timeframe.

With this Portenta boards can be used to connect to federated learning servers in any machine, as long as an IP address connection to Internet is available.

Working with Portenta boards has been a big challenge, as suspected in the risk section. There have been a lot of problems when finding documentation, because of the limited amount of information and a lot of outdated documentation.

Knowing how to connect a Portenta board with other devices using Wi-Fi opens a lot of possibilities, since now it can be part of systems without needing a physical connection, bringing a lot of flexibility.

This project has been useful to learn how to work around the limitations of the hardware to fulfill an objective. Even outside of microcontrollers, this is a good knowledge to have, for example to be in a workplace with simple or outdated hardware.

Additionally, TinyML has a lot of potential for the future: being able to do the same job as a large machine with only microcontrollers would result in economic and ecologic improvements, since the production cost and materials used would be much lower, the power consumption would be also lower and with their small size they can be used in more situations than large computers.

8.2. Future work

The first step to continue developing this project would be trying to increase the speed of the process even further, if possible. Although at this point the application is decently fast, there could be federated learning application that require nearly immediate results.

Another option could be to leverage the two cores of the boards. This way all the available memory of the boards could be used, which would allow to have more memory available for larger neural networks.

9. References

[1] M. Bagnoli, "TinyML: Making Smart Devices Tinier than Ever," 05 January 2021. [Online]. Available: https://www.plugandplaytechcenter.com/resources/tinyml-making-smart-devices-tinier-ever/. [Accessed 22 September 2022].

[2] B. Lutkevich, "Definition: microcontroller (MCU)," [Online]. Available: https://www.techtarget.com/iotagenda/definition/microcontroller. [Accessed 23 September 2022].

[3] Arduino, "What is Arduino?," 05 February 2018. [Online]. Available: https://www.arduino.cc/en/Guide/Introduction. [Accessed 21 September 2022].

[4] Arduino, "Software," [Online]. Available: https://www.arduino.cc/en/software. [Accessed 21 September 2022].

[5] Arduino, "Community forum," [Online]. Available: https://forum.arduino.cc/. [Accessed 21 September 2022].

[6] M. Narang, "Edge Computing vs Cloud Computing: Major Differences," 04 August 2022. [Online]. Available: https://www.knowledgehut.com/blog/cloud-computing/edge-computing-vs-cloud-computing. [Accessed 25 September 2022].

[7] Arun, "An Introduction to TinyML," 10 November 2020. [Online]. Available: https://towardsdatascience.com/an-introduction-to-tinyml-4617f314aa79. [Accessed 22 September 2022].

[8] C. Dilmegani, "TinyML in 2022: Machine Learning at the Edge," 14 March 2022. [Online]. Available: https://research.aimultiple.com/tinyml/. [Accessed 23 September 2022].

[9] Arduino, "Arduino Nano 33 BLE store page," [Online]. Available: https://store.arduino.cc/products/arduino-nano-33-ble. [Accessed 24 September 2022].

[10] Arduino, "Arduino Portenta H7 store page," [Online]. Available: https://store.arduino.cc/products/portenta-h7. [Accessed 24 September 2022].

[11] indeed, "¿Cuánto se gana como uno Gestor/a de proyectos en España?," [Online]. Available: https://es.indeed.com/career/gestor-de-proyectos/salaries?from=top_sb. [Accessed 7 October 2022].

[12] indeed, "¿Cuánto se gana como uno Machine learning engineer/a en España?," indeed, [Online]. Available: https://es.indeed.com/career/machine-learning-engineer/salaries?from=top_sb. [Accessed 7 October 2022].

[13] indeed, "¿Cuánto se gana como uno Desarrollador/a de software en España?," [Online]. Available: https://es.indeed.com/career/desarrollador-de-software/salaries. [Accessed 7 October 2022].

[14] Selectra, "Precio de la luz por horas," [Online]. Available: https://tarifaluzhora.es/. [Accessed 8 October 2022].

[15] Selectra, "¿Cuáles son las mejores ofertas de Internet?," Selectra, [Online]. Available: https://selectra.es/internet-telefono/internet. [Accessed 8 October 2022].

[16] Future of Life Institute (FLI), "The Artificial Intelligence Act," [Online]. Available: https://artificialintelligenceact.eu/. [Accessed 10 December 2022].

[17] TensorFlow, "TensorFlow Lite," 26 May 2022. [Online]. Available: https://www.tensorflow.org/lite/guide. [Accessed 3 December 2022].

[18] R. Khandelwal, "A Basic Introduction to TensorFlow Lite," Towards Data Science, 15 June 2020. [Online]. Available: https://towardsdatascience.com/a-basic-introduction-to-tensorflow-lite-59e480c57292. [Accessed 3 December 2022].

[19] PlatformIO, "PlatformIO IDE," PlatformIO Labs, [Online]. Available: https://platformio.org/platformio-ide. [Accessed 17 October 2022].

[20] R. Heymsfeld, "a neural network for arduino," hobbizine, [Online]. Available: http://robotics.hobbizine.com/arduinoann.html. [Accessed 6 December 2022].

[21] J. Chen, "What Is a Neural Network?," Investopedia, 21 September 2022. [Online]. Available: https://www.investopedia.com/terms/n/neuralnetwork.asp. [Accessed 4 December 2022].

[22] AltexSoft, "Federated Learning: The Shift from Centralized to Distributed On-Device Model Training," AltexSoft, 21 February 2022. [Online]. Available: https://www.altexsoft.com/blog/federated-learning/. [Accessed 5 December 2022].

[23] H. R. A. Z. F. J. G. Q. L. M. G. F. Ittai Dayan, "Federated learning for predicting clinical outcomes in patients with COVID-19," MGH Radiology and Harvard Medical School, Boston, MA, USA, 15 September 2021. [Online]. Available: https://www.nature.com/articles/s41591-021-01506-3. [Accessed 5 December 2022].

[24] D. Ramos, "TinyML with Arduino Portenta H7," Github, [Online]. Available: https://github.com/DavidRD16/TinyML-project-v1/tree/OptimizedStable.

[25] N. Llisterri, "Federated Learning with Arduino Nano 33 BLE Sense," Github, 24 August 2022. [Online]. Available: https://github.com/NilLlisterri/TinyML-FederatedLearning/tree/mixed-precision. [Accessed 30 September 2022].

[26] Pallets, "Flask: web development, one drop at a time," Pallets, 2010. [Online]. Available: https://flask.palletsprojects.com/en/2.2.x/. [Accessed 10 October 2022].

[27] Arduino, "WiFi," Arduino, [Online]. Available: https://www.arduino.cc/reference/en/libraries/wifi/. [Accessed 3 October 2022].

[28] Arduino, "Arduino_JSON," Arduino, [Online]. Available: https://www.arduino.cc/reference/en/libraries/arduino_json/. [Accessed 13 October 2022].

[29] B. Blanchon, "arduinoJson," [Online]. Available: https://arduinojson.org/. [Accessed 16 October 2022].

[30] K. Hoang, "Portenta_H7_AsyncWebServer," Github, 2021. [Online]. Available: https://github.com/khoih-prog/Portenta_H7_AsyncWebServer. [Accessed 5 November 2022].

[31] B. Blanchon, "StreamUtils: Power-ups for Arduino Streams," Github, [Online]. Available: https://github.com/bblanchon/ArduinoStreamUtils. [Accessed 22 November 2022].