

Facultade de Informática



UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN



Desarrollo de una librería para el aprendizaje federado bajo una arquitectura peer-to-peer

Estudiante: Pedro Guijas Bravo

Dirección: Daniel Rivero Cebrián

Enrique Fernández Blanco

A Coruña, septiembre de 2022

A toda la gente que me rodea, en especial a mis padres y pareja.

Agradecimientos

Quisiera darles las gracias a quienes me han apoyado durante la realización de este proyecto, sin ellos esto no hubiera sido posible:

- A mi novia, Elena, por aguantarme todos estos años.
- A mis padres y abuelos, por estar conmigo pese a las desgracias de la vida.
- A mis tutores, por ayudarme a hacer posible un proyecto tan interesante en el que he aprendido tanto.
- A mis compañeros de piso, por crear un entorno de rendimiento académico y desconexión en los momentos oportunos.
- A mis amigos de toda la vida, por ayudarme a desconectar.
- A mi amigo Fernando, por estar siempre sonriendo.

Resumen

En la última década, la evolución del *Machine Learning* ha sido muy próspera, necesitando los modelos más fructíferos ser nutridos por grandes volúmenes de datos. A menudo, la obtención y gestión de estos datos es complicada, siendo generalmente escasos y sujetos a medidas de privacidad. El *Federated Learning* implica un cambio de paradigma en el entrenamiento de modelos de *Machine Learning*. Este nuevo enfoque permite realizar el proceso de aprendizaje sobre datos distribuidos entre una gran cantidad de clientes.

A pesar de que esta novedosa técnica trae consigo numerosas ventajas, una de sus grandes limitaciones es la necesidad de un servidor que orqueste todo el proceso de aprendizaje, suponiendo un único punto de falla. Así mismo, se necesitará disponer de una gran infraestructura para hacer escalables estos sistemas. Para tratar de solventar estas desventajas, surgirán nuevas aproximaciones denominadas *Decentralized Federated Learning*, siendo una de las soluciones más prometedoras el uso de redes *peer-to-peer*.

Ante la inexistencia de alguna librería de soporte al *Decentralized Federated Learning*, en este proyecto, se propone el desarrollo de una librería de propósito general que permita el ***Federated Learning sobre redes peer-to-peer, empleando el protocolo Gossip***. El uso del protocolo *Gossip* garantizará la tolerancia a fallos en la red *peer-to-peer*, creando un ecosistema descentralizado, escalable y robusto.

La librería busca dar soporte a toda clase de dispositivos, haciendo especial hincapié en su facilidad de uso y ampliación futura. Además de permitir el despliegue, ésta, posibilita la ejecución de simulaciones, haciendo posible la realización de pruebas en entornos controlados.

Para el desarrollo, se ha hecho uso de la metodología ágil *SCRUM*. Las iteraciones centrales se han destinado a la implementación del sistema, mientras que la inicial y final se han dedicado a la preparación del proyecto y realización de pruebas respectivamente. En las diversas pruebas realizadas, se han empleado los *datasets* de *MNIST* y *FEMNIST*, obteniendo resultados realmente similares a ejecuciones equivalentes con entrenamientos clásicos.

Abstract

In the last decade, the evolution of Machine Learning has been really thriving. The most productive models need to be fed by a large volume of data. Obtaining and managing these data

is often complicated, as they are generally scarce and subject to privacy measures. Federated Learning implies a paradigm shift in the training of Machine Learning models. This new approach allows us to carry out the learning process on data which are distributed among a large number of clients.

In spite of the fact that this new technique has numerous advantages, one of its great limitations is the need for a server that orchestrates the entire learning process, assuming a single point of failure. Moreover, a large infrastructure will be necessary to make these systems scalable. In order to solve these disadvantages, new approaches called Decentralized Federated Learning will emerge, and one of the most promising solutions will be the use of peer-to-peer networks.

Given the lack of any library to support Decentralized Federated Learning, this project proposes the development of a general purpose library that allows **Federated Learning over peer-to-peer networks, using the Gossip protocol**. The use of the Gossip protocol will guarantee fault tolerance in the network, creating a decentralized, scalable and robust ecosystem.

The library will seek to support all kinds of devices, with special emphasis on ease of use and future expansion. In addition to allowing deployment, it will enable the execution of simulations, making it possible to perform tests in controlled environments.

The agile SCRUM methodology has been used for this development. The central iterations have been used to the implementation of the system, while the initial and final iterations have been respectively dedicated to project preparation and testing. The *MNIST* and *FEMNIST* datasets have been used in the different tests carried out, obtaining results that are really similar to equivalent executions with classic training.

Palabras clave:

- Aprendizaje Máquina
- Aprendizaje Federado Descentralizado
- *peer-to-peer*
- Protocolo *Gossip*
- *Python*
- *Pytorch*

Keywords:

- Machine Learning
- Decentralized Federated Learning
- *peer-to-peer*
- *Gossip* protocol
- *Python*
- *Pytorch*

Índice general

1	Introducción	1
1.1	Objetivos	3
1.2	Estructura de la Memoria	3
2	Estudio de Mercado	5
2.1	Criterios de comparación	5
2.2	Análisis y comparación de soluciones existentes	6
2.3	Conclusión	8
3	Fundamentos	9
3.1	Fundamentos Teóricos	9
3.1.1	Redes Neuronales Artificiales	9
3.1.2	Aprendizaje Federado	11
3.1.3	<i>Peer-to-peer</i>	16
3.1.4	Protocolo <i>Gossip</i>	18
3.1.5	Criptografía	18
3.2	Fundamentos Tecnológicos	20
3.2.1	Lenguaje de Programación: Python	20

3.2.2	Librerías de ML	21
3.2.3	Librería para el manejo de conexiones: <i>TCP con sockets</i>	21
4	Metodología y Planificación	22
4.1	Análisis de viabilidad	22
4.2	Análisis de riesgos	23
4.3	Metodología	26
4.3.1	Variaciones sobre <i>SCRUM</i>	27
4.3.2	Ciclo de Vida	27
4.4	Planificación	28
4.4.1	<i>Sprints</i>	29
4.5	Resultado del seguimiento	33
5	Desarrollo de la Librería	36
5.1	<i>Sprint 0</i> : Preparación del proyecto	36
5.1.1	Análisis de Requisitos	36
5.2	<i>Sprint 1</i> : Arquitectura <i>p2p</i> completamente conectada	38
5.2.1	Análisis	38
5.2.2	Diseño	40
5.2.3	Implementación	43
5.2.4	Pruebas	44
5.3	<i>Sprint 2</i> : Aprendizaje Federado	45
5.3.1	Análisis	45
5.3.2	Diseño	47
5.3.3	Implementación	51
5.3.4	Pruebas	52

5.4	<i>Sprint 3: Conjunto test y Subconjuntos</i>	53
5.4.1	Análisis	53
5.4.2	Diseño	55
5.4.3	Implementación	56
5.4.4	Pruebas	56
5.5	<i>Sprint 4: Protocol Gossip</i>	56
5.5.1	Análisis	56
5.5.2	Diseño	57
5.5.3	Implementación	59
5.5.4	Pruebas	60
5.6	<i>Sprint 5: Cambio de topología</i>	60
5.6.1	Análisis	60
5.6.2	Diseño	61
5.6.3	Implementación	62
5.6.4	Pruebas	62
5.7	<i>Sprint 6: Cifrado en las comunicaciones</i>	63
5.7.1	Análisis	63
5.7.2	Diseño	64
5.7.3	Implementación	65
5.7.4	Pruebas	66
5.8	<i>Sprint 7: Aplicación Práctica</i>	67
5.8.1	<i>Datasets Empleados</i>	67
5.8.2	Pruebas de simulación	68
5.8.3	Validación del funcionamiento en un entorno real	70
5.8.4	Pruebas de simulación con alto número de nodos	70

5.8.5	Pruebas con datos <i>no-iid</i>	72
5.8.6	Conclusión sobre los resultados obtenidos	73
6	Conclusiones	74
7	Trabajo Futuro	76
	Lista de acrónimos	80
	Glosario	82
	Bibliografía	84
A	Planificación	89
A.1	Diagrama de <i>Gantt</i> tentativo	89
A.2	Diagrama de <i>Gantt</i> de seguimiento	91
B	Diagramas de secuencia	93
C	Arquitecturas de las <i>RR.NN.AA</i> empleadas	96

Índice de figuras

3.1	Pasos en cada ronda de Aprendizaje Federado	11
3.2	Envío de modelos locales empleando una arquitectura <i>p2p</i>	15
3.3	Tipos de topología en red Fuente: Wikimedia	17
4.1	Matriz resultante del análisis Fortalezas Oportunidades Debilidades Amenazas (FODA) del proyecto	23
4.2	Tablero <i>Kanban</i> para la visualización de la metodología <i>SCRUM</i> en el <i>sprint</i> de preparación.	30
4.3	Tablero <i>Kanban</i> para la visualización de la metodología <i>SCRUM</i> en el 1° <i>sprint</i>	30
4.4	Tablero <i>Kanban</i> para la visualización de la metodología <i>SCRUM</i> en el 2° <i>sprint</i>	31
4.5	Tablero <i>Kanban</i> para la visualización de la metodología <i>SCRUM</i> en el 3° <i>sprint</i>	31
4.6	Tablero <i>Kanban</i> para la visualización de la metodología <i>SCRUM</i> en el 4° <i>sprint</i>	32
4.7	Tablero <i>Kanban</i> para la visualización de la metodología <i>SCRUM</i> en el 5° <i>sprint</i>	32
4.8	Tablero <i>Kanban</i> para la visualización de la metodología <i>SCRUM</i> en el 6° <i>sprint</i>	33
4.9	<i>Burn Down Chart</i> del proyecto	34
5.1	Diagrama de casos de uso del 1° <i>sprint</i>	40
5.2	Diagrama de clases del 1° <i>sprint</i>	42

5.3	Diagrama de casos de uso implementados hasta el <i>sprint</i> 2. Los casos de uso en la zona verde son los nuevos casos de uso incluidos en el 2° <i>sprint</i>	47
5.4	Diagrama de clases para el 2° <i>sprint</i>	49
5.5	Diagrama de clases para el 4° <i>sprint</i>	58
5.6	Diagrama de clases para el 6° <i>sprint</i>	65
5.7	Comparativa del error y precisión sobre el conjunto test en un entrenamiento clásico y un entrenamiento federado para el experimento descrito en el apartado 5.8.2. Las métricas han sido obtenidas al finalizar cada ronda en un entrenamiento federado o cada <i>epoch</i> en un entrenamiento clásico. La cantidad de cómputo entre muestras será idéntica para ambas ejecuciones.	69
5.8	Error en el conjunto de entrenamiento del Aprendizaje Federado para el experimento descrito en el apartado 5.8.2. Recuerdese que la cantidad de cómputo es la cantidad de veces que se han empleado <i>batches</i> de 32 muestras para mejorar el modelo. Apréciase que el cómputo está paralelizado.	69
5.9	Dispositivos implicados en prueba de despliegue y sus conexiones.	70
5.10	Error y precisión en conjunto test a lo largo de un aprendizaje clásico para el experimento descrito en el apartado 5.8.4.	71
5.11	Error y precisión en conjunto test a lo largo del Aprendizaje Federado para el experimento descrito en el apartado 5.8.4.	71
5.12	Error y precisión en el conjunto test para el experimento descrito en el apartado 5.8.5.	72
5.13	Ejemplo dirección vector mejora en distribuciones de datos <i>iid</i> y <i>non-iid</i>	73
A.1	Diagrama de <i>Gantt</i> tentativo. Fragmento 1	89
A.2	Diagrama de <i>Gantt</i> tentativo. Fragmento 2	90
A.3	Diagrama de <i>Gantt</i> tentativo. Fragmento 3	90
A.4	Diagrama de <i>Gantt</i> de seguimiento. Fragmento 1	91
A.5	Diagrama de <i>Gantt</i> de seguimiento. Fragmento 2	91
A.6	Diagrama de <i>Gantt</i> de seguimiento. Fragmento 3	92

B.1	Diagrama de secuencia del proceso de conexión entre nodos.	93
B.2	Diagrama de secuencia para el inicio del aprendizaje.	94
B.3	Diagrama de secuencia para la agregación de modelos.	94
B.4	Diagrama de secuencia para la difusión de mensajes usando un protocolo <i>Gossip</i>	95
B.5	Diagrama de secuencia para la difusión de modelos usando un protocolo <i>Gossip</i>	95
C.1	Arquitectura concreta del perceptrón multicapa empleado.	97
C.2	Arquitectura concreta de la red neuronal convolucional empleada.	97

Índice de tablas

2.1	Tabla comparativa de librerías de Aprendizaje Federado	7
4.1	Especificación de la probabilidad de ocurrencia para los diferentes niveles de riesgo.	24
4.2	Especificación de la probabilidad de ocurrencia para los diferentes niveles de riesgo.	24
4.3	Especificación de riesgos subyacentes al proyecto.	26
4.4	Tabla con los costes estimados del personal en planificación inicial	28
4.5	Tabla con los costes estimados del equipo necesario	29
4.6	Tabla con los costes totales iniciales estimados del proyecto	29
4.7	Tabla con los costes finales del personal en planificación inicial	34
4.8	Tabla con los costes finales del equipo necesario	35
4.9	Tabla con los costes totales finales del proyecto	35
5.1	Tabla de casos de uso implementados para el 1º <i>sprint</i>	40
5.2	Tabla explicatoria con pruebas unitarias realizadas en el <i>sprint</i> 1.	44
5.3	Tabla de casos de uso implementados para el 2º <i>sprint</i>	47
5.4	Tabla explicatoria con pruebas unitarias realizadas en el <i>sprint</i> 2.	53
5.5	Tabla de casos de uso implementados para el 3º <i>sprint</i>	54

5.6	Tabla de casos de uso implementados para el 4° <i>sprint</i>	57
5.7	Tabla de casos de uso implementados para el 4° <i>sprint</i>	61
5.8	Tabla explicatoria con pruebas unitarias realizadas en el 5° <i>sprint</i>	63
5.9	Tabla de casos de uso implementados para el 6° <i>sprint</i>	64
5.10	Tabla explicatoria con pruebas unitarias realizadas en el 5° <i>sprint</i>	66

Introducción

EN la actualidad, la Inteligencia Artificial se ha convertido en uno de los campos más prometedores de la informática. Una de sus principales ramas, el *Machine Learning* (ML), es cada vez más popular y empleada, siendo sus impresionantes avances fruto de modelos más y más complejos, nutridos por grandes cantidades de datos.

La forma convencional de trabajar en ML consiste en la centralización de información y cómputo en un mismo lugar, implicando limitaciones. Una limitación sería relativa a la infraestructura, puesto que el mantenimiento de la información y los diferentes procesos de aprendizaje requieren de grandes recursos. Otra de las grandes limitaciones se centra en la naturaleza de la información. Los datos provienen de una gran variedad de fuentes, estando sujetos usualmente a medidas de privacidad que incluso pueden llegar a impedir su difusión. De acuerdo con estas medidas, se está generando un potencial riesgo de privacidad al usarse este enfoque centralizado para procesos de aprendizaje.

De este modo, en 2016, Google propone el *Federated Learning* (FL) [1], en español, Aprendizaje Federado. Este nuevo enfoque pretende solucionar los problemas expuestos anteriormente gracias a la creación de modelos sobre datos distribuidos. Su funcionamiento es simple, consiste en un proceso iterativo orquestado por un servidor central. En cada iteración, diferentes nodos contribuirán a un modelo global por medio de la agregación de modelos locales. Es decir, para obtener los modelos locales, los nodos realizarán paralelamente entrenamientos empleando sus propios datos y recursos.

A pesar de las evidentes ventajas que otorga no compartir explícitamente los datos, el servidor orquestador también implica una serie de limitaciones. Estas consisten en la existencia de un punto único de falla y un posible cuello de botella, limitando considerablemente la escalabilidad. Para solucionar estos problemas, se busca que los nodos consigan comunicarse

entre ellos, prescindiendo del servidor. A este conjunto de aproximaciones se las denomina Aprendizaje Federado Descentralizado [2].

El uso de redes *Peer-to-peer* (p2p) [3.1.3] y protocolos *Gossip* [3] es una de las aproximaciones más prometedoras al Aprendizaje Federado Descentralizado. La arquitectura p2p, permitirá la creación de una infraestructura donde los dispositivos o nodos se comporten de forma colaborativa entre iguales. En cuanto al protocolo *Gossip*, permitirá crear un mecanismo robusto para el intercambio de mensajes, aprovechando la redundancia de las redes p2p. De esta forma, se obtendrá un ecosistema descentralizado, que garantiza tanto la escalabilidad como la robustez. Destáquese que la tolerancia a fallos resultará crucial en entornos reales, sobre todo si se desea dar soporte a *dispositivos edge*, los cuales pese a disponer de cada vez más poder de cómputo, suelen verse afectados por problemas de disponibilidad relacionados con el uso de baterías y redes de telefonía. Lamentablemente, la mayoría de trabajo relacionado con esta aproximación está enfocado a la investigación, no existiendo librerías que permitan la fácil creación de dichos sistemas.

En consecuencia, motivado por la inexistencia de una librería de propósito general para el Aprendizaje Federado Descentralizado, en este proyecto se llevará a cabo el desarrollo una implementación de este tipo. Al ser una librería de carácter general, tratará de darse soporte a toda clase de dispositivos, buscando la ligereza del sistema centrada en el soporte a dispositivos de bajas prestaciones. Así mismo, no se establecerá una topología en específico, tratando que el usuario decida la más adecuada para su sistema. En lo tocante a las comunicaciones, se tratarán de minimizar, incluyendo métodos criptográficos para garantizar la confidencialidad y confiabilidad en las mismas.

Buscando la comodidad y flexibilidad, la librería estará orientada al usuario, siendo completamente agnóstica a los *frameworks* de ML existentes, procurando la fácil inclusión de estos. Concretamente, este proyecto se centrará en las *Redes de Neuronas Artificiales (RR.NN.AA)*, puesto que el Aprendizaje Federado apenas se ha explorado con otra clase de modelos.

Asimismo, para poder validar y probar los sistemas creados, la librería no solo estará orientada al despliegue, sino que también se contemplarán los entornos simulados. La simulación facilitará el desarrollo y uso de la librería, permitiendo una sencilla transición hacia el despliegue.

Finalmente, se llevará a cabo un ejemplo práctico real en donde se pueda validar y demostrar el funcionamiento y cualidades de la librería.

1.1 Objetivos

El objetivo de este proyecto será la **creación de una librería que dé soporte al Aprendizaje Federado sobre redes p2p, haciendo uso del protocolo Gossip**. Para poder cumplir con este objetivo, se han identificado los siguientes sub-objetivos concretos que se deben abordar para poder llevar a cabo el primero:

- **Estudio:** Estudio de Aprendizaje Federado, redes p2p y protocolos Gossip.
- **Simplicidad y Facilidad:** Se tratará de llevar estos dos principios, tanto al desarrollo de la librería, como al uso de la misma por parte del usuario final.
- **Modularidad y Abstracción:** Conceptos que junto con los anteriores facilitarán la ampliación de funcionalidades, tanto para los desarrolladores iniciales como futuros.
Un diseño basado en estos dos conceptos permitirá entre otras cosas, la fácil incorporación de nuevos *framework* de ML y algoritmos de agregación federados.
- **Robustez:** Se ha de tener en cuenta la posibilidad de fallos en los nodos, controlando su ocurrencia sin perjudicar al sistema.
- **Privacidad:** Como se comentará en el capítulo de fundamentos 3, no será suficiente con la privacidad por diseño del FL, puesto que se podría realizar ingeniería inversa. Por lo tanto, la librería debe proporcionar una capa de privacidad adicional.
- **Investigación y Despliegue:** Se pretenderá que la librería sea apta para estos dos tipos de usuarios, permitiendo una sencilla migración hacia el despliegue.
- **Pruebas:** El sistema desarrollado deberá ser cautelosamente testeado y probado, tratando de acercarse a entornos reales.

1.2 Estructura de la Memoria

La memoria ha sido estructurada en torno a los siguientes capítulos:

1. **Introducción:** Capítulo actual. Se ha tratado de introducir el proyecto.
2. **Estudio de Mercado:** Se analizarán las diferentes alternativas existentes a la librería que se quiere desarrollar.

3. **Fundamentos:** Se explicarán cuidadosamente los elementos teóricos y tecnológicos que se aplican en el proyecto. Se tratará de justificar los motivos de elección así como familiarizar al lector.
4. **Metodología y Planificación:** En este capítulo se tratará la metodología a seguir, la planificación inicial del proyecto y el seguimiento de la misma. Además, se realizará un estudio de la viabilidad y riesgos en el proyecto.
5. **Desarrollo de la librería:** Detalla el proceso de desarrollo de la librería. Dicho proceso estará dividido en *sprints* tal como marca la metodología a emplear, *SCRUM*. El primer y último *sprint* se centrarán en la planificación del proyecto y elaboración de pruebas respectivamente.
6. **Conclusiones:** En este capítulo serán tratadas las conclusiones extraídas en la realización y seguimiento del proyecto.
7. **Trabajo Futuro:** Este capítulo expondrá las diferentes líneas de trabajo futuro, no incluidas en los objetivos del proyecto.

Estudio de Mercado

EL objetivo de este capítulo es el de recopilar y analizar las diferentes librerías y *frameworks* existentes que den soporte al aprendizaje federado.

A la hora de crear un producto es necesario realizar previamente un estudio de mercado. De esta forma, se evitará desarrollar soluciones ya creadas, buscando los puntos débiles de las existentes para tratar de solventarlos y aportar mejoras.

A continuación, se definirán los criterios de comparación, se analizarán y compararán los diferentes *frameworks* y se finalizará con una conclusión.

2.1 Criterios de comparación

En este apartado, se definirán brevemente las características en las que se centrará el estudio de mercado, creando criterios que faciliten un análisis comparativo. Dichas características serán las siguientes:

- **Facilidad de Uso:** En otras palabras, la usabilidad.
- **Facilidad de Ampliación:** Es decir, la extensibilidad de la librería.
- **Dependencia de algún *framework* de ML:** El acoplamiento de la librería a algún *framework* de ML, así como la capacidad para incluir nuevos.
- **Posibilidad de simulaciones y despliegues:** La capacidad de la librería para ser ejecutada sobre entornos reales y simulaciones de dichos entornos. La simulación resulta ideal para propósitos de investigación y validación de sistemas previos al despliegue.

- **Posibilidad de soporte a modelos más allá de las [RR.NN.AA](#):** Pese a pueda no estar implementado, el diseño de la librería debe ser lo suficientemente flexible como para poder incluir nuevos tipos de modelos de [ML](#).
- **Arquitectura centralizada:** Tipo de arquitectura, centrándose en si se trata, o no, de una arquitectura centralizada.
- **Mecanismos para la distribución del código del modelo:** Es decir, si se proporcionan mecanismos para la difusión entre dispositivos del código y funcionamiento de los modelos, no únicamente de los parámetros del mismo.
- **Mecanismos para la organización de los datos:** Estos mecanismos permitirán la selección de datos, haciendo posible, entre otras cosas, la coexistencia de diferentes *datasets* en los dispositivos. Se trata de una característica que permitirá la creación de entornos federados más polivalentes.
- **Mecanismos de control de dispositivos:** Éstos, controlarán tanto el acceso a la red como a recursos dentro de la misma.
- **Mecanismos de privacidad adicionales:** Técnicas ajenas al Aprendizaje Federado que garanticen la privacidad de los datos.

2.2 Análisis y comparación de soluciones existentes

A continuación, se introducirán escuetamente las principales librerías y *frameworks*, mostrando en la tabla [2.1](#) las diferencias entre las mismas según los criterios establecidos.

- **Flower** [\[4\]](#): *Framework open-source* para la creación de sistemas de Aprendizaje Federado. Pese a no proporcionar una funcionalidad inmensamente amplia, esta librería es extremadamente amigable, ampliable y fácil de usar.
- **OpenFL** [\[5\]](#): *Framework open-source* de soporte al Aprendizaje Federado desarrollado por Intel. *OpenFL* está diseñado para ser una herramienta flexible, extensible y fácil de aprender para los científicos de datos.
- **NVIDIA Flare** [\[6\]](#): *Framework* de código abierto y extensible creado por Nvidia. Permite adaptar sistemas de [ML](#) a un entorno federado muy seguro y completo.
- **PySyft** [\[7\]](#): *Framework open-source* desarrollado por OpenMined. *PySyft* proporciona un entorno seguro y privado para *Data Science* en *Python*. *PySyft* aplica multitud de técnicas entre las que destaca el Aprendizaje Federado.

- **Tensorflow Federated** [8]: *Framework* desarrollado por el equipo de *Tensorflow*. Éste, únicamente permite simular los algoritmos de Aprendizaje Federado, no siendo posible el despliegue de dichos sistemas.
- **FATE** [9]: Proyecto *open-source* iniciado por *Webank*. Implementa múltiples protocolos de computación seguros para permitir la creación de modelos cumpliendo la normativa de protección de datos. Su ecosistema sumamente amplio y complejo, combinado con la barrera lingüística de su equipo de desarrollo, dificulta el uso de dicho *framework*.
- **PaddleFL** [10]: *Framework open-source* para el Aprendizaje Federado desarrollado por el motor de búsqueda chino *Baidu*. *PaddleFL* está basado en *PaddlePaddle* [11], siendo apto tanto para entornos de investigación como para despliegues altamente distribuidos.

	Librería a desarrollar	Flower	OpenFL	Nvidia Flare	PySyft	Tensorflow Federated	FATE	PaddleFL
Facilidad de Uso	Alta	Alta	Media	Media-Baja	Alta-Media	Media	Baja	Media-Baja
Facilidad de Ampliación	Alta	Alta	Media	Alta	Media	Baja	Baja	Media
Dependiente de algún <i>framework</i>	No	No	No	No	No	<i>TensorFlow</i>	<i>FederatedML (propio)</i>	<i>PaddlePaddle</i>
Arquitectura Centralizada	No	Si	Si	Si	Si	Si	Si	Si
Mecanismos para la distribución del código del modelo	No	No	Si	Si	Si	No	Si	Si
Mecanismos para la organización de los datos	No	No	Si	Si	Si	No	Si	Si
Mecanismos de control de dispositivos	No	No	Si	Si	Si	No	Si	No
Mecanismos de privacidad adicionales	Si	Si	Si	Si	Si	No	Si	Si
Posibilidad de simulaciones y despliegues	Si	Si	Si	Si	Si	Únicamente Simulación	Si	Si
Posibilidad de soporte a modelos más allá de las RR.NN.AA	Si	Si	No	Si	No	No	Si	No

Tabla 2.1: Tabla comparativa de librerías de Aprendizaje Federado

Como se puede apreciar en la tabla, en el instante de realización del estudio de mercado **no existe ninguna librería ni *framework* público que aborde una implementación de Aprendizaje Federado Descentralizado**, siendo el poco trabajo existente privado y completamente orientado a la investigación.

2.3 Conclusión

Las librerías de Aprendizaje Federado son, por lo general, inmaduras, escasas y con una curva de aprendizaje relativamente alta. La tabla 2.1 resalta que la mayoría de *frameworks* existentes tienen como objetivo crear sistemas complejos y completos, incluyendo mecanismos y tecnologías ajenas al Aprendizaje Federado. Obviamente, estas no serán tratadas en este proyecto, pero se tendrá en cuenta su futura inclusión.

Por el contrario, *Flower* es el único *framework* con una propuesta de valor similar a la librería a desarrollar: una librería sencilla, ampliable y orientada al usuario. De este modo, teniendo presente que no existe ninguna librería con una arquitectura descentralizada e inspirándose en el enfoque de *Flower*, se buscará crear una librería de similares características que dé soporte al Aprendizaje Federado sobre redes p2p. Así mismo, puesto que el tiempo de desarrollo es bastante limitado, se enfatizará en facilitar una futura extensibilidad por parte de terceros.

Fundamentos

ESTE capítulo tratará de explicar los fundamentos teóricos y prácticos del proyecto. Se procurará familiarizar al lector con los contenidos tratados.

3.1 Fundamentos Teóricos

Se comenzará explicando brevemente las [RR.NN.AA](#), pues su comprensión es necesaria para tratar el Aprendizaje Federado. Éste, será tratado a continuación, buscando contextualizarlo, explicarlo y tratar su situación en entornos descentralizados. Acto seguido, se abordarán los contenidos relativos al funcionamiento del protocolo *Gossip* y la arquitectura [p2p](#), sobre la cual se creará el sistema de Aprendizaje Federado.

3.1.1 Redes Neuronales Artificiales

Las [Redes de Neuronas Artificiales \(RR.NN.AA\)](#) [12] son modelos computacionales inspirados en el sistema nervioso central. Esta clase de modelos aprenden por si mismos, mediante un entrenamiento supervisado, es decir, su conocimiento no está programado de forma explícita.

Como su propio nombre indica, las redes están formadas por neuronas artificiales interconectadas masivamente, siguiendo una arquitectura determinada. Cada neurona es una unidad de procesamiento, siendo la salida de cada neurona determinada por la función [3.1](#). Las neuronas internamente realizan la suma ponderada de sus entradas por unos pesos dados (w_{ij}), w_0 es un término denominado *bias*, este siempre se sumará. Además, para evitar que las concatenaciones de neuronas den como resultado una operación de igual estructura que la de la neurona ([3.1](#)), se aplicará la función de activación φ , encargada de eliminar la linealidad entre

operaciones.

$$y_i = \varphi_i\left(\sum_{j=1}^{m_i} w_{ij}x_{ij}\right) + w_{i0} \quad (3.1)$$

donde:

y_i = valor de salida de la neurona i .

φ_i = función de activación para la neurona i .

m_i = cantidad de entradas a la neurona i .

w_{ij} = peso para la entrada j de la neurona i , w_{i0} será el *bias*.

x_{ij} = valor de entrada j de la neurona i .

El proceso de aprendizaje es el encargado de encontrar los pesos del modelo que produzcan el comportamiento deseado. De esta forma, el entrenamiento será tratado como un problema de minimización del error producido por la red. El algoritmo 1 detalla el aprendizaje, apreciándose que se trata de un proceso iterativo en el que en cada *epoch* (E) se busca mejorar los pesos del modelo (w_{ij}). Para controlar la convergencia, las mejoras ($get_upgrade(w_{ij}; train_data)$) son suavizadas por el *learning rate* (η), cuyo valor generalmente está en el intervalo $(0,1]$.

Algorithm 1 Proceso Aprendizaje en una *Red Neuronal Artificial*

```

for each epoch from 1 to  $E$  do
     $w_{ij} = w_{ij} + \eta * get\_upgrade(w_{ij}; train\_data)$ 
end for

```

Los algoritmos de descenso del *gradiente* [13] junto con la retro-propagación del error [14] son empleados como mecanismo para determinar las mejoras sobre los pesos, debido a su gran escalabilidad y flexibilidad. Éstos, emplearán un conjunto de datos representativos del problema a resolver para calcular el error del modelo, y consecuentemente, los *gradientes* de los pesos respecto a dicho error.

Debe mencionarse que cuando se dispone de muchos datos para el entrenamiento, no es usual suministrarle al algoritmo de aprendizaje todo el conjunto de datos a la vez. Esto, suele producir problemas de estancamiento del modelo. Para inducir una mayor aleatoriedad, las muestras son suministradas al algoritmo de optimización por lotes o *batches*. Así mismo, otra variación frecuente es el uso del algoritmo *Stochastic Gradient Descent* (SGD) [15], pues introduce más aleatoriedad al entrenamiento, produciendo muy buenos resultados en la práctica.

3.1.2 Aprendizaje Federado

El Aprendizaje Federado [1] es un nuevo enfoque al ML tradicional que prescinde de centralizar los datos de entrenamiento en una única máquina o centro de datos. Este novedoso paradigma permite que múltiples dispositivos entrenen un modelo de forma colaborativa, sin la necesidad de que los datos de entrenamiento salgan del dispositivo.

Como se ha mencionado en la introducción, su funcionamiento es sencillo e iterativo, orquestado por un servidor central. Cada iteración se conoce como **ronda de Aprendizaje Federado**, constando de los siguientes pasos:

1. Trasmisión del modelo global a los diferentes dispositivos.
2. Por medio de entrenamientos locales, los dispositivos tratarán de mejorar el modelo global empleando sus propios datos.
3. Envío de los modelos obtenidos en cada dispositivo al servidor orquestador.
4. Actualización del modelo global mediante el procesamiento de los diferente modelos locales, dicho proceso es denominado como **Agregación Federada**.

En la figura 3.1 pueden apreciarse de forma gráfica los diferentes pasos de cada ronda. Nuevamente, debe recalarse que solo se tratarán las RR.NN.AA, pues el Aprendizaje Federado apenas se ha tratado con otra clase de modelos de ML.

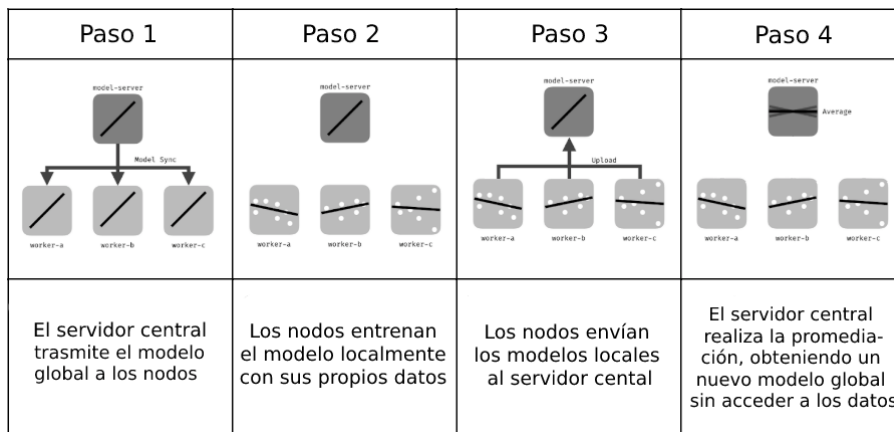


Figura 3.1: Pasos en cada ronda de Aprendizaje Federado

En consecuencia, lo realmente novedoso del Aprendizaje Federado será agregación de modelos, por lo que a continuación, se tratará el primer y principal algoritmo de agregación fede-

rada: *Federated Average* (FedAvg) [16]. Acto seguido, se abordarán las ventajas y desventajas, finalizando con el Aprendizaje Federado Descentralizado.

Federated Average (FedAvg)

La Agregación Federada es el núcleo del Aprendizaje Federado. Comprendiendo las circunstancias de su creación, así como su funcionamiento, podrá entenderse de mejor modo la forma de operar del FL.

Los modelos más fructíferos de *Deep Learning* [17] se han basado en su mayoría en SGD junto con sus extensiones o variaciones (como *Adaptive Moment Estimation* (ADAM) [18] o *Adaptive Gradient Algorithm* (AdaGrad) [19]). Así pues, la primera toma de contacto con los agregadores federados es *Federated Stochastic Gradient Descents* (FedSGD).

Con FedSGD, en cada ronda los dispositivos calcularán los **gradientes** de forma local, siendo posteriormente agregados realizando una media ponderada por el número de muestras de cada dispositivo. Al haber una inicialización común al principio de cada ronda, no se perjudica necesariamente el rendimiento del modelo promediado resultante.

En el ML tradicional, el coste de las comunicaciones es prácticamente inexistente, siendo el mayor coste el computacional. En un entorno federado sucede lo contrario, de modo que para minimizar el coste, se tratará de aumentar la computación en el dispositivo a favor de disminuir las comunicaciones. Buscando disminuir la comunicación aumentando la computación surge FedAvg, como una generalización de FedSGD, logrando entre 10-100 veces menos comunicación en comparación. En FedAvg, serán compartidos pesos del modelo y no **gradientes**, estando la cantidad de cómputo y comunicaciones controlada por 3 parámetros:

- **C:** Fracción de Clientes. Para no saturar los canales de comunicación, únicamente una fracción de dispositivos estará mejorando el modelo en cada ronda.
- **E:** Número de *epochs* o iteraciones del algoritmo de aprendizaje por ronda en cada cliente. En una ronda se realizará podrá realizar más de un paso de descenso del **gradiente**.
- **B:** Tamaño del *batch* empleado en el aprendizaje.

Por lo tanto, para un nodo con n muestras de datos, el número de actualizaciones locales por ronda será $u = E * (n/B)$. En caso de que se empleen todas las muestras en un único *batch* y $E = 1$, corresponderá con FedSGD ($u = 1$).

El pseudocódigo del Aprendizaje Federado usando **FedAvg** puede apreciarse en el Algoritmo 2, para el lado del servidor, y en el Algoritmo 3, para la parte del cliente. Resumidamente, el servidor inicializará el modelo (w_0) e iniciará el bucle de entrenamiento durante R rondas. En cada ronda, seleccionará un subconjunto de nodos (S_t) y agregará los modelos recibidos empleando una media ponderada por el número de muestras, obteniendo un nuevo modelo, $w_t + 1$. En cuanto al dispositivo, se realizará un entrenamiento local empleando **SGD** con sus datos (P_k), una duración de E *epochs* y un tamaño de *batch* de B . Al finalizar se enviarán los pesos del modelo resultante al servidor.

Algorithm 2 Federated Averaging Server Side (K clients)

```

initialize  $w_0$ 
for each round  $t$  from 1 to  $R$  do
   $train\_set\_size \leftarrow \max(C \cdot K, 1)$ 
   $S_t \leftarrow$  (subconjunto aleatorio de  $train\_set\_size$  nodos)
  for each client  $k$  in  $S_t$  do
     $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$ 
  end for
   $w_{t+1} \leftarrow \sum_{k=1}^K (n_k/n) \cdot w_{t+1}^k$ 
end for

```

Algorithm 3 Federated Averaging Client Side

```

 $\beta \leftarrow$  (split  $P_k$  en batches de tamaño  $B$ )
for each local epoch from 1 to  $E$  do
  for each batch  $b$  in  $\beta$  do
     $w \leftarrow w - \eta \nabla l(w; b)$  # SGD
  end for
end for
return  $w$  to server

```

Destáquese que existen multitud de agregadores federados basados en **FedAvg** que tratarán de mejorar la convergencia del mismo ante determinadas situaciones. Entre todos, destacan los siguientes métodos de agregación: *Federated Learning with Matched Averaging* (FedMA) [20], *Stochastic Controlled Averaging for Federated Learning* (SCAFFOLD) [21], *q-Fair Federated Average* (q-FedAvg) [22], *FedOpt*, *FedYogi*, *FedAdam* [23], *FedProx* [24].

Ventajas y Carencias del Aprendizaje Federado

El uso de sistemas de Aprendizaje Federado tiene numerosas ventajas, entre las que destacan:

- Uso de datos privados y confidenciales sin tener que preocuparse por el manejo de los datos.

- Proceso de aprendizaje continuo.
- No es necesaria una gran infraestructura más allá del servidor orquestador. El cómputo será distribuido por los diferentes dispositivos y los datos no necesitarán ser almacenados.

Así mismo, las principales desventajas consisten en:

- En cuanto al servidor orquestador, para agregar los modelos de una manera escalable y tolerante a fallos ha de disponerse de una gran infraestructura.
- La agregación de modelos no tiene mecanismos de privacidad por diseño, por lo que por medio de ingeniería inversa podría obtenerse información de los clientes. Serán empleados mecanismos como la Agregación Segura [25] u otras técnicas de encriptación para solventar este problema.
- Pérdida en el rendimiento del modelo ante clientes maliciosos y distribuciones de datos que causen que los dispositivos no dispongan de un conjunto representativo de muestras (distribuciones no *Independent and Identically Distributed Random Variables (IID)*).
- Es una tecnología relativamente nueva y en estudio.

Aprendizaje Federado Descentralizado

Ahora que se tiene una noción básica del Aprendizaje Federado, tiene que tenerse en cuenta que existen multitud de variaciones del mismo, orientadas a resolver diferentes problemáticas [26]. Este subapartado se centrará en tratar el FL siguiendo un enfoque *serveless*, es decir, sin el servidor orquestador.

Existen diferentes alternativas para crear sistemas de Aprendizaje Federado Descentralizado. Las redes p2p con grafos [27] o el uso de *blockchain* como sustitutivo del servidor orquestador [28] son una de las muchas soluciones. Sin embargo, este trabajo únicamente se centrará en las redes p2p y el protocolo *Gossip*, principalmente por su flexibilidad, ligereza, robustez y tolerancia a fallos, permitiendo su uso con dispositivos de bajas prestaciones, así como nodos más potentes.

La literatura científica del Aprendizaje Federado Descentralizado es realmente escasa, no existiendo trabajos que aborden una solución similar a la que se desarrollará en este proyecto. No

obstante, debe destacarse un trabajo que trata superficialmente dos de las partes fundamentales de este proyecto: el uso de redes *p2p* en el Aprendizaje Federado y la agregación de modelos en entornos distribuidos [29].

Este trabajo propone un sistema realmente similar a la propuesta original del Aprendizaje Federado. La gran diferenciación reside en que cada nodo recibirá y agregará los modelos locales, de igual forma que lo hace el servidor orquestador. Como resultado, al final de cada ronda todos los nodos dispondrán de la misma versión del modelo, fruto de la agregación. En la figura 3.2 puede apreciarse gráficamente como los nodos envían al resto de la red sus modelos locales.

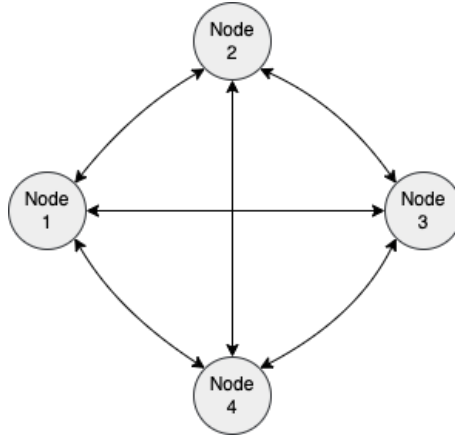


Figura 3.2: Envío de modelos locales empleando una arquitectura *p2p*.

Adicionalmente, teniendo presente el uso del protocolo *Gossip*, en el presente documento se propone una optimización para la agregación de modelos en entornos descentralizados. La mejora está basada en *FedAvg*, el algoritmo de agregación federada más común y del que parten la mayoría de alternativas.

Con *FedAvg*, la agregación es una media ponderada (fórmula 3.2), pudiendo descomponerse en un proceso iterativo (fórmula 3.3). Con el método iterativo es posible obtener resultados intermedios o agregaciones parciales que equivalgan a un conjunto de modelos ya agregados. De este modo, se buscará enviar agregaciones parciales, pues estarán condensando información, evitando enviar individualmente cada modelo. Como se puede suponer, esto supone un importante ahorro en el coste de las comunicaciones, sobre todo teniendo en cuenta que el envío de modelos es la comunicación más costosa.

$$AVG = \sum_{i=1}^n \frac{model_i * samples_i}{\sum_{i=1}^n samples_i} \quad (3.2)$$

$$AVG = \sum_{i=j}^n \frac{model_i * samples_i}{\sum_{i=1}^n samples_i} + \sum_{i=1}^j \frac{model_i * samples_i}{\sum_{i=1}^j samples_i} * \frac{\sum_{i=1}^j samples_i}{\sum_{i=1}^n samples_i} \quad (3.3)$$

donde:

n = cantidad de modelos a agregar.

j = cantidad de modelos a agregar parcialmente. ($n > j$)

Para promediar los modelos de forma correcta, los conjuntos de nodos involucrados en las agregaciones parciales deben ser conjuntos disjuntos entre si, siendo necesario llevar la cuenta de los modelos restantes por nodo.

3.1.3 *Peer-to-peer*

Peer-to-peer (p2p) es una arquitectura de *software* distribuida. Está compuesta por diferentes nodos o pares que se comportan como iguales entre sí, formando la denominada red de nodos.

En esta arquitectura no será necesario un elemento central que administre la red, los nodos se comunicarán entre sí auto-organizando la misma. De este modo, los pares son tanto proveedores como consumidores de recursos, contrastando con el modelo de arquitectura tradicional cliente-servidor.

La alta escalabilidad y la tolerancia a fallos son las dos grandes ventajas de esta arquitectura. El principal inconveniente reside en la complejidad de la administración de la red. La descentralización implica la generación de bastante tráfico y la existencia de mecanismos complejos para la gestión de la red.

A pesar de que la arquitectura p2p busca la descentralización, existen variantes que renuncian parcialmente a esta para mitigar las desventajas asociadas a la complicada gestión de los nodos. Particularmente, en este trabajo solo serán tratadas las redes p2p puras, pues garantizan una alta descentralización y escalabilidad.

Dentro de los sistemas p2p puros pueden diferenciarse dos grandes grupos, las **redes p2p no estructuradas** y las **redes p2p estructuradas**.

Redes *p2p* estructuradas

Estos sistemas *p2p* se organizan siguiendo alguna topología en específico, garantizando que cualquier nodo se pueda buscar eficientemente. Para enrutar el tráfico de manera eficiente, los nodos deben mantener listas de vecinos (frecuentemente tablas *hash* distribuidas), haciendo estas redes menos robustas ante una alta tasa de abandono, es decir, cuando la frecuencia de unión y abandono de nodos es alta.

En la figura 3.3, pueden observarse algunas de las principales topologías en red. Cada arquitectura tiene características y carencias propias, haciéndolas más o menos adecuadas dependiendo de la casuística del problema a resolver.

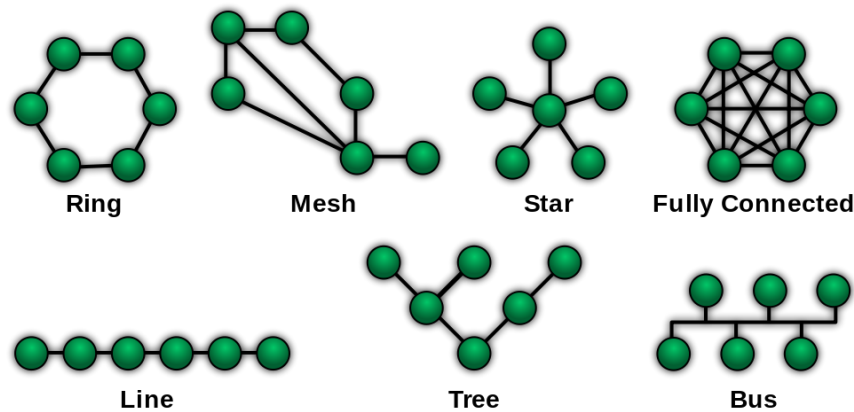


Figura 3.3: Tipos de topología en red

Fuente: [Wikimedia](#)

La rigidez que implica seguir una topología concreta, el hecho de que estas redes sean menos robustas a tasas de abandonos altas y no tener la necesidad de buscar eficientemente un dispositivo concreto en la red, han ocasionado que no se haga uso de este tipo de redes en el proyecto.

Redes *p2p* no estructuradas

Las redes *p2p* no estructuradas no imponen una estructura particular en la red. De modo que para poder localizar diferentes nodos o información, debe hacerse uso de técnicas de inundación (*flooding*) o protocolos basados en esta técnica como *Gossip*. Dado que *flooding* tiene un enfoque bruto y más ineficiente, no se tratará en esta memoria.

3.1.4 Protocolo *Gossip*

El protocolo *Gossip* es un proceso de comunicación en redes **p2p** basado en la expansión epidémica. Es usado para garantizar que un mensaje se difundirá a todos los miembros de la red, siendo usado principalmente como protocolo de difusión y protocolo de agregación.

Su funcionamiento es iterativo y sencillo, pudiendo comprenderse como una analogía a la difusión de rumores. Un elemento difundirá un rumor a una cantidad determinada de individuos aleatorios, de forma seguida, el individuo original y los conocedores del rumor lo continuarán difundiendo hasta que todos lo conozcan, es decir, hasta que converja. Como se puede intuir, su funcionamiento es adecuado en redes de gran tamaño con altos niveles de descentralización.

Entre las principales ventajas del protocolo destacan:

- **Escalabilidad:** Al realizar una difusión exponencial, el aumento del número de dispositivos no implicará problemas de rendimiento.
- **Adaptabilidad:** Pensando en dispositivos de bajas prestaciones o redes congestionadas, se podrá adaptar la tasa y tamaño de la difusión para cada nodo.
- **Robusto y tolerante a fallos:** Caídas o posibles errores no afectarán al proceso de difusión. Así mismo, el alto grado de redundancia en las comunicaciones podrá permitirnos recuperarnos de ciertos errores.
- **Extremadamente orientado a sistemas distribuidos:** Como se ha mencionado, este protocolo es especialmente útil en entornos distribuidos.

Sus desventajas son relativas a la ineficiencia comunicacional relativa a las redes **p2p** no estructuradas. Así mismo, pudiera verse como una desventaja el tiempo de convergencia (debido a que es un protocolo iterativo), pero para este trabajo se verá como una ventaja por espaciar temporalmente las comunicaciones y no sobre-saturar el sistema.

3.1.5 Criptografía

La criptografía es usada con fines de otorgar privacidad en las comunicaciones, cifrando los mensajes transmitidos. En este proyecto se usarán dos tipos de encriptación: la encriptación asimétrica y la simétrica.

Cifrado Asimétrico

El cifrado asimétrico o cifrado de clave pública se caracteriza por utilizar dos claves relacionadas, una clave pública responsable del cifrado y otra privada para el descifrado. La llave pública debe ser compartida para cifrar mensajes que únicamente el propietario de la clave privada asociada puede descifrar.

Como se puede apreciar, la principal ventaja es clara: una sencilla y segura distribución de claves, limitándose a únicamente compartir la pública. No obstante, presenta claras desventajas:

- El mensaje cifrado ocupa más espacio que el original.
- Comparado con un cifrado simétrico es computacionalmente más costoso y más lento.
- Algunos algoritmos de cifrado tienen una cantidad máxima de bytes a codificar, dependiendo estrictamente del tamaño de la clave.

Existen multitud de algoritmos de cifrado asimétrico, siendo *Rivest, Shamir y Adleman (RSA)* [30] el usado. Dicha decisión está principalmente motivada por ser el primer y más utilizado algoritmo de encriptación asimétrica.

Cifrado Simétrico

El cifrado simétrico es un método criptográfico que se basa en el uso de una misma clave para el cifrado y descifrado de mensajes. Nuevamente, existen multitud de algoritmos que permiten un cifrado simétrico, siendo el algoritmo seleccionado para esta tarea *Advanced Encryption Standard (AES)* [31]. *AES* ha sido adoptado como un estándar de cifrado por el gobierno de los Estados Unidos [32], así mismo, permite un cifrado en bloques, siendo posible alinear el cifrado con la transmisión por bloques que se usará en el sistema.

Para poder generar una clave de sesión de forma segura se hará uso del cifrado asimétrico, pues permite crear un canal seguro en el que se pueda compartir la clave. Posteriormente, el sistema usará únicamente este cifrado, pues no presenta las claras limitaciones que presenta el cifrado asimétrico.

3.2 Fundamentos Tecnológicos

En este apartado se abordarán las tecnologías sobre las que se implementará la librería. Se explicará el motivo de su elección, así como particularidades de cada tecnología.

3.2.1 Lenguaje de Programación: Python

Python [33] es un lenguaje interpretado, multiparadigma y multiplataforma. Es indudablemente el lenguaje más usado para [ML](#), contando con una inmensa comunidad y desarrollos relacionados con el sector. De este modo, teniendo presente que no existe una solución similar a la que se va a implementar, se usará este lenguaje con el objetivo de alcanzar un mayor público potencial.

Además, la sintaxis y la multitud de librerías con las que cuenta ayudarán a abstraerse de detalles irrelevantes para este proyecto, proporcionando un desarrollo rápido y ágil en comparación con otros lenguajes. Las librerías más destacables que se han empleado son las siguientes:

- **Threading**: Librería empleada para la concurrencia del sistema.
- **Pytest**: Permite la creación de pruebas unitarias que validen el funcionamiento del sistema.
- **Sphinx**: *Software* que permite la generación de documentación desde el código fuente.
- **Pycryptodome**: Librería que proporciona la implementación de los principales algoritmos criptográficos.
- **Pytorch**: *Framework* de [ML](#) empleado, en el próximo apartado 3.2.2 se tratarán en profundidad sus características y motivo de elección.
- **Sockets**: Librería que permite la comunicación entre nodos. De forma análoga, se tratarán en el apartado 3.2.3 los motivos de su elección así como su funcionamiento.

Finalmente debe mencionarse que se han planteado otros lenguajes como **Julia**, **Elixir**, **C++** o **R**. Como se ha mencionado, descartados a favor de *Python* esencialmente por no contar con una comunidad tan sólida de desarrolladores de [ML](#).

3.2.2 Librerías de ML

Hablando de *frameworks* para la creación de [RR.NN.AA](#), *Tensorflow* [34] y *PyTorch* [35] son los dos grandes candidatos. Ambos *frameworks* son de código abierto, permitiendo una sencilla programación de redes bajo un enfoque de alto nivel y la automatización del proceso de [diferenciación automática](#). *TensorFlow* está algo más centrado en la industria, mientras que *PyTorch* tiene la reputación de ser un *framework* enfocado en la investigación.

En este caso, ***PyTorch*** se adapta mejor a las necesidades del proyecto, pues permite una programación sencilla gracias a los grafos de computación dinámicos [36], muy similar a como se programa en *python*. Esto supone una gran facilidad a la hora de probar ágilmente distintos modelos o variaciones del mismo, especialmente gracias a los altos niveles de abstracción que proporciona (en particular *PyTorch Lightning*). Así mismo, la comunidad y la adopción de *PyTorch* estos últimos años es considerablemente superior a la de *Tensorflow*.

3.2.3 Librería para el manejo de conexiones: TCP con sockets

A la hora de determinar que tecnología emplear para las comunicaciones, se han planteado soluciones de alto nivel como *FastApi* [37]. No obstante, se ha optado por *sockets* para el manejo de conexiones [Transmission Control Protocol \(TCP\)](#) [38].

[TCP](#) es el principal protocolo sobre el que se basan los intercambios de información en la red. Se ha optado por usar un protocolo de bajo nivel por la flexibilidad y rapidez que proporciona. Además, la creación de conexiones *full-duplex* bidireccionales basadas en flujos de datos se adaptará fácilmente al paradigma de paso de mensajes que se desea implementar. Sin embargo, soluciones como *FastApi* basadas en [TCP](#) como protocolo base, introducen mecanismos innecesarios para el proyecto. Al mismo tiempo, esta librería no está orientada al paso de mensajes, sino a la creación de [Application Programming Interface \(API\)](#).

El diseño de [TCP](#) garantiza que el proceso de comunicación pueda recuperarse ante situaciones de corrupción, pérdida, duplicación o desorden de datos. Así mismo, este protocolo permite comunicaciones multiplataforma en redes heterogéneas, incluyendo mecanismos de control de congestión y *timeout*.

Como se ha mencionado, se hará uso de *sockets*, pues es la [API](#) para la familia de protocolos de Internet (en este caso *TCP/IP*) provista por todos los sistemas operativos modernos. Particularmente, se hará uso de los *sockets* de *python*. Este módulo, en esencia, proporciona acceso a la interfaz de *sockets* del sistema operativo bajo el paradigma de orientación a objetos en *python*.

Metodología y Planificación

EN los siguientes apartados, primeramente se detallará el análisis de viabilidad y riesgos. Acto seguido se definirá la metodología a seguir así como la planificación inicial, finalizando con el seguimiento del proyecto.

4.1 Análisis de viabilidad

Como en cualquier proyecto, será necesario realizar un análisis que ayude a determinar la decisión de si llevarlo a cabo, o no. El uso del análisis **Fortalezas Oportunidades Debilidades Amenazas (FODA)** [39], permitirá buscar y analizar, de forma proactiva y sistemática todas las variables que intervienen en el proyecto, con el fin de determinar la viabilidad del mismo.

De este modo, la figura 4.1 muestra la matriz **FODA** analizando la situación del proyecto. A grosso modo, las fortalezas y oportunidades estarán asociadas a lo novedoso y útil que puede resultar el sistema. La utilidad del sistema se ve potenciada teniendo en cuenta el auge de la Inteligencia Artificial en todos los sectores de la sociedad. En cambio, las debilidades y amenazas están sujetas nuevamente al riesgo que implica la realización de un producto novedoso, como pudiera ser la falta de información y la incertidumbre ante la acogida de los posibles clientes.

Como se ha tratado en el apartado de fundamentos 3.2, la librería es perfectamente abordable con las tecnologías mencionadas. Así mismo, como se verá en próximos apartados, la estimación en tiempo y costes es completamente asumible.

Finalmente, se concluye en que el proyecto es apto para ser realizado.

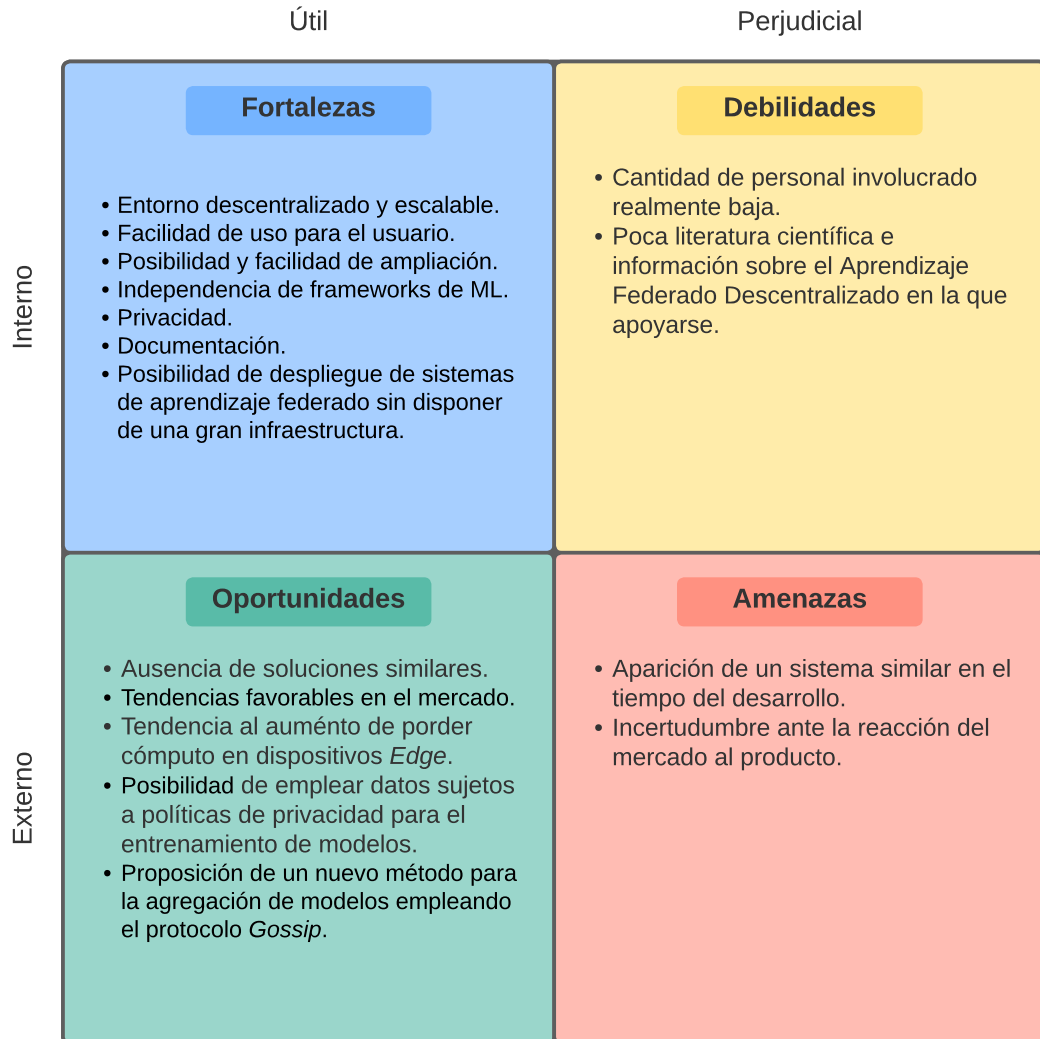


Figura 4.1: Matriz resultante del análisis FODA del proyecto

4.2 Análisis de riesgos

En este apartado se elaborará un análisis sobre los riesgos a los que estará sometido el desarrollo del proyecto. Para cada riesgo detectado, se le asociará probabilidad de ocurrencia y un nivel de criticidad. Dichos valores serán discretos, siendo definidos los intervalos que representa cada unidad en las tablas 4.1 y 4.2 respectivamente. Además, serán tratadas las consecuencias que ocasionarán los diferentes riesgos así como las medidas paliativas o preventivas a aplicar.

Nivel de aparición	Probabilidad de ocurrencia
Muy bajo	Menos de un 10%
Bajo	Entre un % y un 25%
Medio	Entre un 25% y un 50%
Alto	Entre un 50% y un 75%
Muy Alto	Más de un 75%

Tabla 4.1: Especificación de la probabilidad de ocurrencia para los diferentes niveles de riesgo.

Nivel de criticidad	Nivel de apreciación
Escaso	Se aprecia muy poco en el proyecto
Leve	Se aprecia poco en el proyecto
Moderado	Se aprecia en el proyecto
Crítico	Se aprecia mucho en el proyecto
Extremo	Se cancelaría el proyecto

Tabla 4.2: Especificación de la probabilidad de ocurrencia para los diferentes niveles de riesgo.

De esta forma, en la tabla 4.3, se enumerarán los riesgos detectados, siendo ordenados por un nivel de criticidad descendente. Como se podrá apreciar, algunos riesgos son inevitables y para otros, se tratará de prevenir o mitigar sus efectos.

Riesgo 1: Aparición de una librería idéntica			
Probabilidad	Baja	Nivel de criticidad	Extremo
Causas	Dependiendo de lo similar y sólida que sea la librería podría llegarse a cancelar el proyecto.		
Medidas preventivas	Adopción de metodología ágil que permita cambiar los requisitos buscando la diferenciación con la librería competente.		
Riesgo 2: Baja del estudiante			
Probabilidad	Media	Nivel de criticidad	Crítica

Continuación de la tabla 4.3.

Causas	Dependiendo de la gravedad, podría arrastrar grandes retrasos en el proyecto.		
Medidas preventivas	Asunción del riesgo.		
Riesgo 3: Rendimiento de modelos considerablemente inferiores a entrenamientos clásicos			
Probabilidad	Baja	Nivel de criticidad	Crítico
Causas	Supondrá analizar cautelosamente el sistema, tratando de detectar y solucionar los errores que lo induzcan.		
Medidas	Realización continua de pruebas empleando <i>toy problems</i> .		
Riesgo 4: Pérdida del código fuente			
Probabilidad	Media	Nivel de criticidad	Crítico
Causas	Supondrá volver a realizar todo el trabajo.		
Medidas	Uso de <i>software</i> de control de versiones y <i>backups</i> .		
Riesgo 5: Sistema demasiado complejo			
Probabilidad	Media	Nivel de criticidad	Moderada
Causas	Supondrá aplicar re-ingeniería, retrasando considerablemente el proyecto.		
Medidas	Se aplicarán multitud de patrones y principios de diseño.		
Riesgo 6: Pérdida o destrucción del equipo			
Probabilidad	Baja	Nivel de criticidad	Moderada
Causas	Implicaría un leve retraso y el coste económico de volver a adquirir el equipo.		
Medidas	Asunción del riesgo.		
Riesgo 7: Baja de algún director del proyecto			
Probabilidad	Media	Nivel de criticidad	Moderada
Causas	Podría provocar algún leve retraso.		
Medidas	Asunción del riesgo.		
Riesgo 8: Rechazo solicitud para uso del CESGA			

Continuación de la tabla 4.3.

Probabilidad	Muy Baja	Nivel de criticidad	Moderada
Causas	Supondrá prescindir de los resultados de simulaciones con alto número de nodos.		
Medidas	Asunción del riesgo.		
Riesgo 9: Malas estimaciones en el tiempo de desarrollo			
Probabilidad	Alta	Nivel de criticidad	Leve
Causas	Dependiendo de la gravedad, inducirá retrasos en en proyecto.		
Medidas	Estimación temporal con holgura, sin estimar tiempos demasiado acortados.		

Tabla 4.3: Especificación de riesgos subyacentes al proyecto.

En conclusión, los riesgos que afectan a este trabajo, así como sus medidas paliativas o preventivas son aceptables para poder llevar a cabo el proyecto.

4.3 Metodología

Para la realización del proyecto, se ha hecho uso de una **metodología ágil**. Concretamente, se ha empleado una **adaptación de SCRUM** [40] para proyectos individuales. Este marco de trabajo garantizará una rápida obtención de resultados intermedios mediante un desarrollo iterativo dividido en *sprints*. Así mismo, *SCRUM* permite no concretar inicialmente los requisitos, y al tratarse de una metodología ágil, permitirá una rápida respuesta y ajuste del proyecto ante la posible aparición de nuevos *papers* o librerías. De esta forma, *SCRUM* es altamente flexible y rápido, cuadrando a la perfección con los posibles cambios de requisitos y naturaleza del proyecto.

Como se ha mencionado, *SCRUM* es un marco de trabajo y no una metodología, permitiendo adaptarlo sencillamente a cada proyecto. En este caso, se modificará levemente para que encaje con un proyecto individual tutorizado.

4.3.1 Variaciones sobre SCRUM

En el proyecto, únicamente estarán involucrados los directores y el alumno. El alumno será responsable de la realización del trabajo, mientras que los directores guiarán, evaluarán y validarán el mismo. Por este motivo, los roles de SCRUM tendrán que ser adaptados. El resultado es el siguiente:

- **Equipo de desarrollo (*Development Team*):** Rol equivalente al del SCRUM original, con la salvedad de que únicamente habrá un desarrollador, el alumno.
- **Responsable del funcionamiento de SCRUM (*Scrum Master*):** Rol llevado a cabo por el alumno, idéntico al rol de SCRUM original.
- **Propietario del producto (*Product Owner*):** Rol desempeñado por los tutores. El *Product Owner* gestionará el *backlog* y asegurará que el trabajo realizado sea adecuado desde la perspectiva del Trabajo de Fin de Grado (TFG). Las historias de usuario serán substituidas por **casos de uso**, pues permiten una especificación más detallada de lo que el sistema debe hacer ante las iteraciones de diferentes actores.

Tras finalizar cada *sprint*, el *Product Owner* intervendrá evaluando y opinando sobre los resultados obtenidos. Asimismo, en caso de haber algún contratiempo, se realizarán reuniones esporádicas y consultorías. En definitiva, deben de estar siempre en contacto para cubrir el vacío que deja la ausencia de más opiniones.

Debido a la disponibilidad de los tutores, así como el hecho de que el equipo de desarrollo esté formado por un único miembro, se eliminarán las reuniones diarias (*daily meeting*). No obstante, por parte del alumno debe existir una continua auto-reflexión, destacando el análisis, crítica y adaptación.

Finalmente, debe mencionarse el uso de un tablero *Kanban* para la rápida visualización del estado del proyecto. En las figuras de la planificación, por ejemplo en la figura 4.3, puede observarse dicho tablero.

4.3.2 Ciclo de Vida

El ciclo de vida propuesto para la elaboración del proyecto consta de las siguientes fases:

1. **Agregación de tareas al *backlog*:** Primeramente, se determinarán los requisitos deseables desde un punto de vista de alto nivel. Acto seguido, se segmentarán los requisitos en tareas muy concretas que serán añadidas al *backlog*.

2. **Organización *sprints*:** Las tareas del *backlog* serán distribuidas a diferentes *sprints*.
3. **Ejecución *sprint*:** Desarrollo de cada *sprint*. Las fases que conformarán cada *sprint* serán: **análisis, diseño, implementación y pruebas**. Su duración será de entre 2 y 4 semanas aproximadamente.
4. **Inspección y adaptación:** Análisis y validación de tanto los resultados obtenidos como el proceso (orientado a mejorar debilidades en próximos *sprints*).
5. **Análisis por parte tutores:** Con el *sprint* finalizado, se mostrarán los resultados a los tutores. Es una de las etapas más importantes, pues es donde se obtendrá un *feedback* de un punto de vista diferente al del alumno.
6. Vuelta al paso 1. hasta finalizar las tareas del *backlog* y hallarse satisfecho con la calidad obtenida.

4.4 Planificación

A pesar del dinamismo de la metodología usada, para la estimación y organización inicial del proyecto, se ha creado un **diagrama de Gantt tentativo**, éste se encuentra detallado en las tablas A.1, A.2 y A.3. Los *sprints* planteados en esta planificación temporal serán tratados en el próximo subapartado. Como puede apreciarse, el inicio del proyecto está datado el **21 de febrero de 2022**, siendo el **27 de julio de 2022** la fecha estimada de finalización del mismo.

En cuanto a los costes, en las tablas 4.4, 4.5 y 4.6 se detallan los costes humanos, materiales y totales respectivamente. Destáquese que los salarios de cada perfil han sido obtenidos haciendo la media de la oferta salarial española en [talent.com](https://www.talent.com).

Equipo	Tiempo dedicado (h)	Precio (€/h)	Coste Total (€)
Analista	224	15,36	3360
Programador	920	14,42	13432
Director	108	25,73	2778,84
		Total	19570,84

Tabla 4.4: Tabla con los costes estimados del personal en planificación inicial

Equipo	Tiempo utilizado (h)	Precio (€/h)	Coste Total (€)
MacBook Pro 13"	1252	0,169	211,588
Ordenador Sobremesa	40	0,219	8,76
Raspberry Pi 4	40	0,007	0,28
Nodo Cesga	3	1,92	5,76
		Total	226,388

Tabla 4.5: Tabla con los costes estimados del equipo necesario

Coste estimados asociados al personal	19570,84
Coste estimados asociados al equipo	226,388
Total	19797,228

Tabla 4.6: Tabla con los costes totales iniciales estimados del proyecto

4.4.1 Sprints

Como se ha comentado en el apartado previo 4.3, el desarrollo de la librería se dividirá en breves períodos temporales, resultando un desarrollo iterativo incremental. A continuación, se detallarán los intervalos temporales o *sprint* planteados inicialmente para el proyecto. Destáquese que durante el desarrollo, esta planificación podrá verse modificada.

Sprint 0: Preparación del proyecto

Sprint inicial del proyecto y uno de los más densos. En éste se estudiarán las bases teóricas y tecnologías a aplicar. Estas tareas serán indispensables para analizar la viabilidad y riesgos del proyecto así como realizar un estudio de mercado. Finalmente se ha realizado la planificación inicial tentativa del proyecto. En la figura 4.3, podrán observarse las tareas que formarán el *sprint*.

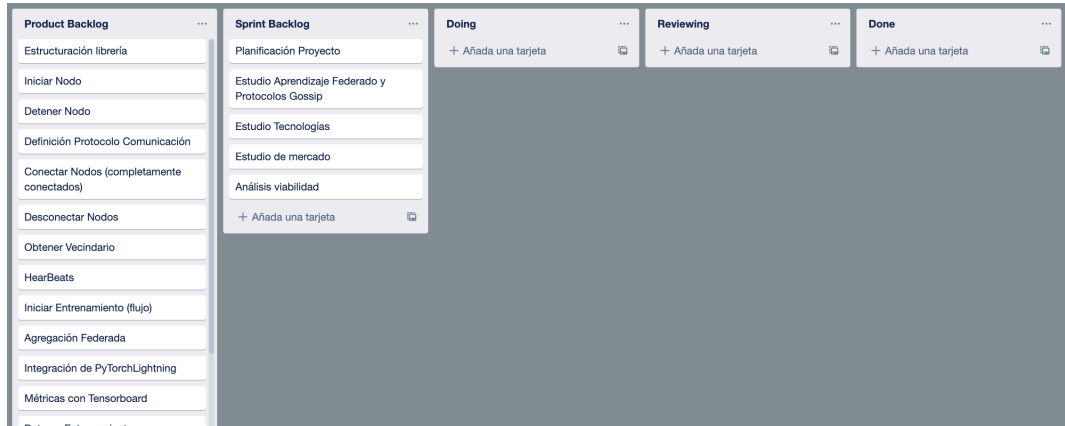


Figura 4.2: Tablero *Kanban* para la visualización de la metodología *SCRUM* en el *sprint* de preparación.

Sprint 1: Arquitectura p2p completamente conectada

Primer *sprint* dedicado al desarrollo. En éste, se buscará estructurar la librería y crear un sistema *p2p* base sobre el implementar el Aprendizaje Federado en próximas iteraciones. Nuevamente, en la figura 4.4 se observarán las diferentes tareas que conforman el *sprint*.

Destáquese que con el fin de facilitar el desarrollo, los nodos se organizarán en una arquitectura completamente conectada temporal.

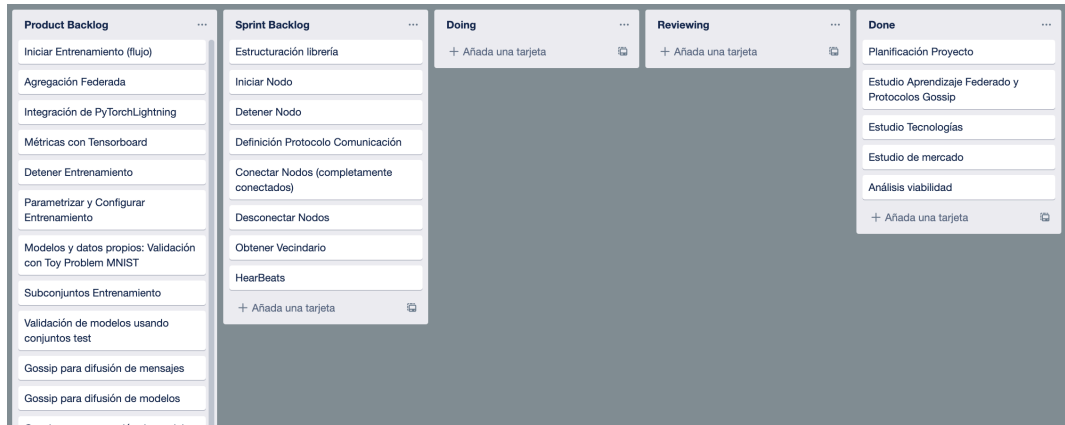


Figura 4.3: Tablero *Kanban* para la visualización de la metodología *SCRUM* en el 1º *sprint*.

Sprint 2: Aprendizaje Federado

Sprint destinado a la inclusión del Aprendizaje Federado en la red **p2p**. Obsérvese en la figura 4.5 las diferentes tareas que conforman el *sprint*.

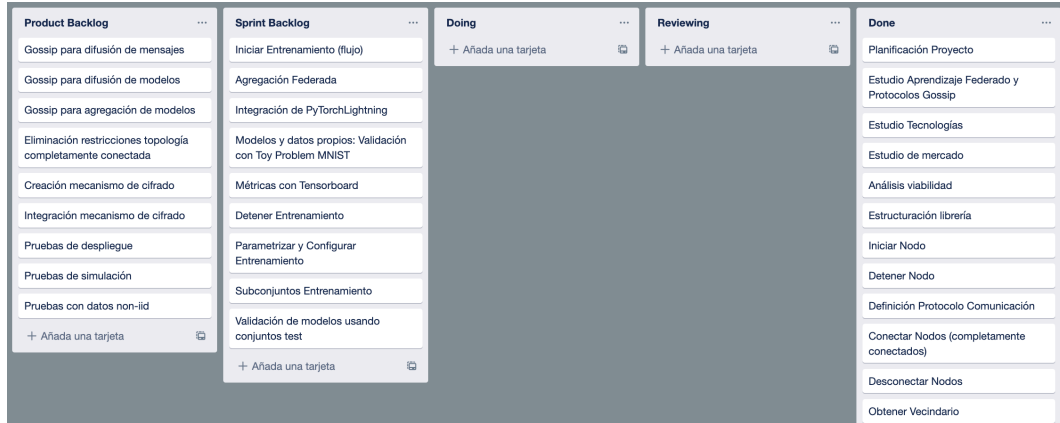


Figura 4.4: Tablero *Kanban* para la visualización de la metodología *SCRUM* en el 2º *sprint*.

Sprint 3: Protocolo Gossip

El tercer *sprint* de desarrollo del proyecto tendrá como objetivo implementar e integrar el protocolo *Gossip* en las comunicaciones de la red **p2p**. De forma análoga a los *sprints* previos, en la figura 4.5 se detallan las tareas de esta iteración.

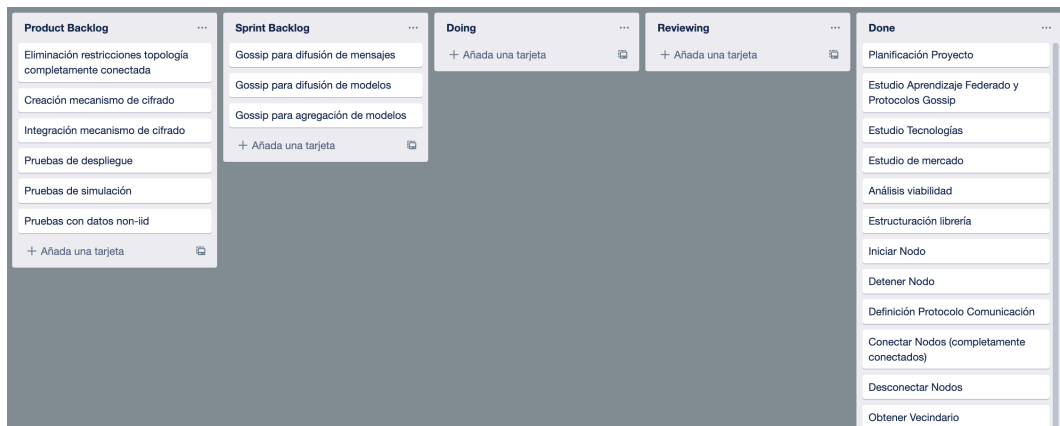


Figura 4.5: Tablero *Kanban* para la visualización de la metodología *SCRUM* en el 3º *sprint*.

Sprint 4: Cambio de topología

El objetivo de este 4º *sprint* será el de eliminar la topología completamente conectada implementada temporalmente en el 1º *sprint*. Gracias a ya tener implementado el protocolo *Gossip*, únicamente se tendrá que adaptar el sistema, no crear funcionalidades como tal. De nuevo, en la figura 4.6, podrá observarse el tablero *Kanban* con las tareas para este *sprint*.

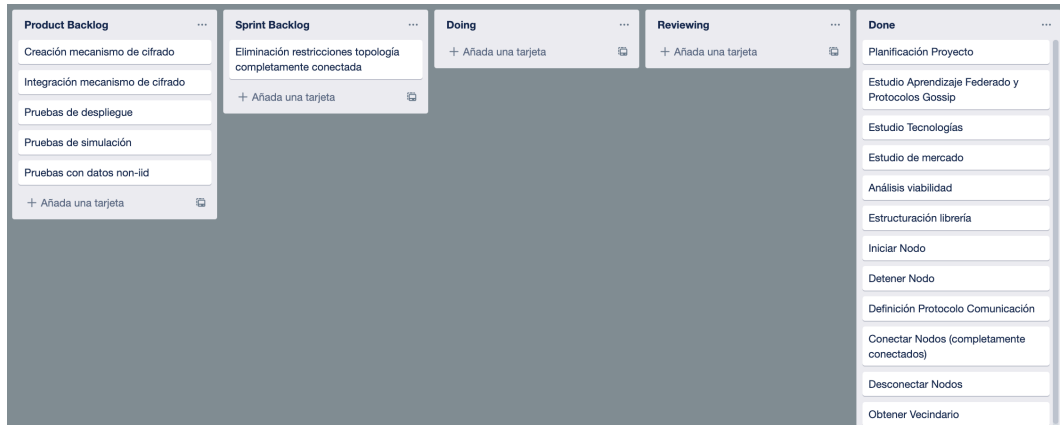


Figura 4.6: Tablero *Kanban* para la visualización de la metodología *SCRUM* en el 4º *sprint*.

Sprint 5: Cifrado en las comunicaciones

Último *sprint* de desarrollo. En éste, se buscará cifrar las comunicaciones entre nodos del sistema *p2p*, buscando dotar al sistema de confidencialidad y privacidad en la comunicación. Véanse en la figura 4.7 las tareas a realizar.

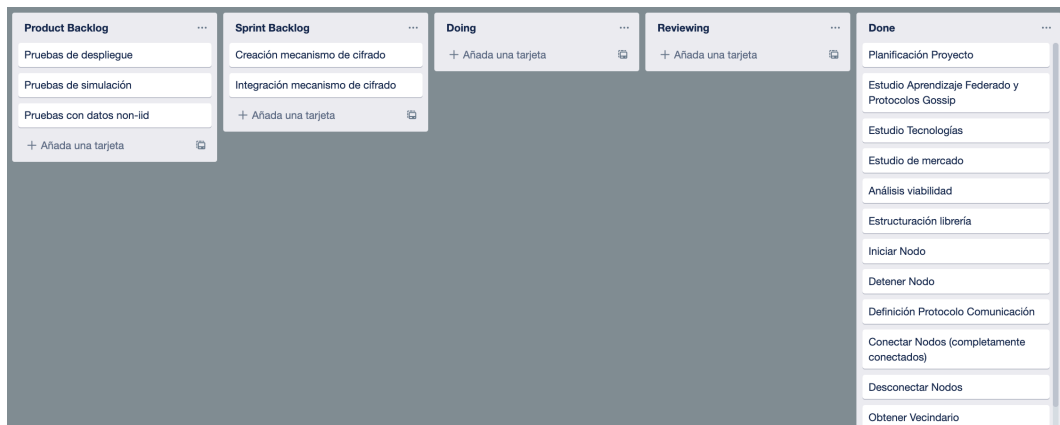


Figura 4.7: Tablero *Kanban* para la visualización de la metodología *SCRUM* en el 5º *sprint*.

***Sprint* 6: Aplicación Práctica**

En éste último *sprint* del proyecto, se ejecutarán diferentes pruebas, validando y estudiando el comportamiento de la librería desarrollada en diferentes entornos. Nuevamente, en la figura 4.5 puede apreciarse el tablero *Kanban* con las diferentes tareas que conforman el *sprint*.

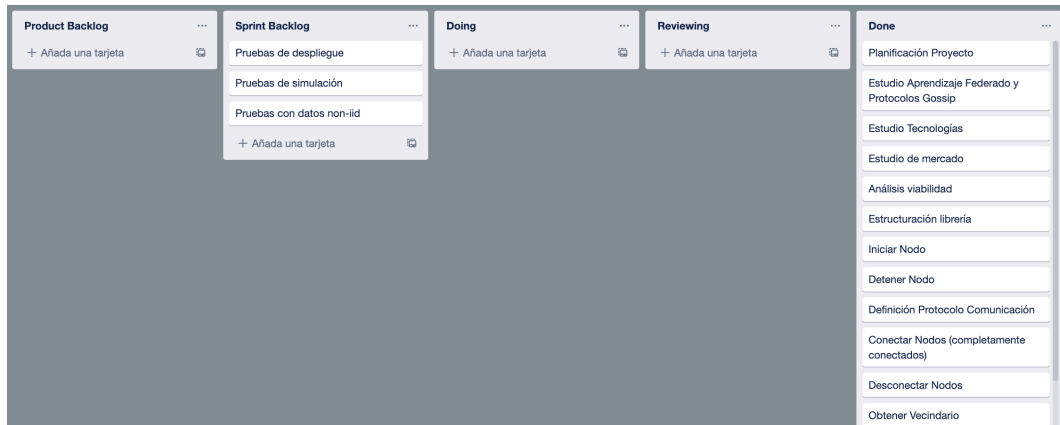
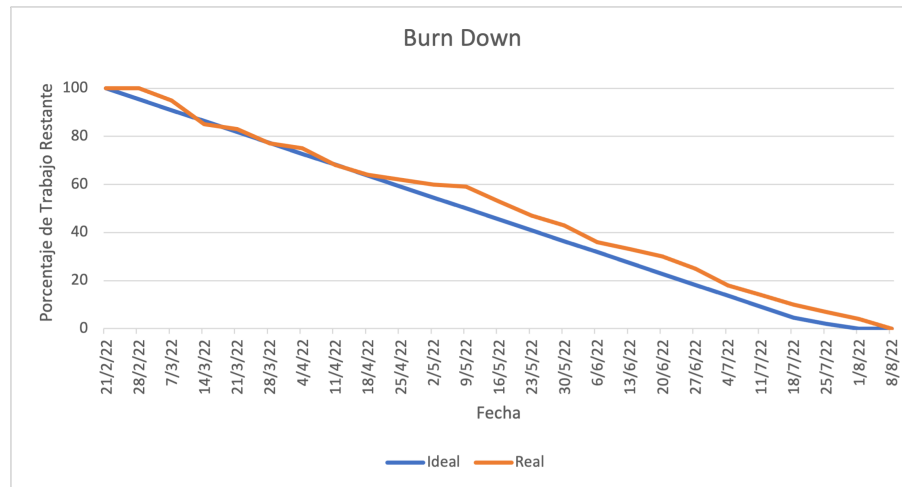


Figura 4.8: Tablero *Kanban* para la visualización de la metodología *SCRUM* en el 6° *sprint*.

4.5 Resultado del seguimiento

Gracias a que no han existido modificaciones en los requisitos, la planificación tentativa ha sido, por lo general, bastante acertada. No obstante, se ha producido un retraso de una semana en el 2° *sprint*. Tratando de no sobrecargar dicho *sprint*, las tareas de validación con conjuntos test y creación de conjuntos de entrenamiento serán trasladadas a un nuevo *sprint* que precederá al tercero.

Para visualizar el progreso del proyecto se empleará un *Burn Down Chart*. Éste es usado en metodologías ágiles como *SCRUM* para visualizar el trabajo pendiente. Las tareas del *backlog* serán cuantizadas proporcionalmente a su cantidad de trabajo asociada, restando ese valor del trabajo total cuando se finalicen. De este modo, en la figura 4.9 se aprecia el progreso del proyecto frente a su ejecución ideal. Apréciase el retraso que se produce con el inicio del 2° *sprint*, del cual el proyecto no se volverá a recuperar.

Figura 4.9: *Burn Down Chart* del proyecto

De forma similar, en los diagramas de *Gantt* de seguimiento A.4, A.5 y A.6 puede apreciarse la variación con las tareas planificadas de forma tentativa. Nuevamente, debe apreciarse el retraso inducido por la inclusión del *sprint* mencionado previamente, siendo las demás estimaciones correctas.

Los costes que ha implicado el retraso han sido ínfimos. Los costes humanos, materiales y totales serán detallados en las tablas 4.7, 4.8 y 4.9 respectivamente.

Equipo	Tiempo dedicado (h)	Precio (€/h)	Coste Total (€)
Analista	232	15,36	3563,52
Programador	987,2	14,42	14235,424
Director	116	25,73	2984,68
Total			20783,624

Tabla 4.7: Tabla con los costes finales del personal en planificación inicial

Equipo	Tiempo utilizado (h)	Precio (€/h)	Coste Total (€)
MacBook Pro 13"	1335,2	0,169	225,6488
Ordenador Sobremesa	40	0,219	8,76
Raspberry Pi 4	40	0,007	0,28
Nodo Cesga	3	1,92	5,76
		Total	240,449

Tabla 4.8: Tabla con los costes finales del equipo necesario

Coste finales asociados al personal	20783,624
Coste finales asociados al equipo	240,449
Total	21024.0728

Tabla 4.9: Tabla con los costes totales finales del proyecto

Desarrollo de la Librería

ESTE capítulo detallará los pasos necesarios para la elaboración y validación de la librería de soporte al Aprendizaje Federado sobre redes *p2p*. Siendo coherentes con la metodología, habrá una sección por cada *sprint*. Los *sprints* centrales abordarán el desarrollo del sistema, dedicando el primero a la preparación del proyecto y el último a la elaboración de pruebas sobre el sistema desarrollado.

5.1 *Sprint* 0: Preparación del proyecto

Sprint inicial de preparación del proyecto. En este intervalo temporal, se han realizado diversas actividades previas al desarrollo de la librería. Por lo tanto, se han realizado tareas relativas a la planificación, estudio de mercado, análisis de viabilidad y análisis de riesgos del proyecto, así como tareas de estudio sobre los conceptos y tecnologías a aplicar (véanse en los capítulos 2, 3 y 4 respectivamente).

En el próximo apartado únicamente se tratarán los requisitos de la librería, pues la mayoría del trabajo realizado en este *sprint* ya ha sido tratado en la memoria.

5.1.1 Análisis de Requisitos

En esta sección se especificarán los requisitos de la librería. Dichos requisitos están ampliamente relacionados con los objetivos del proyecto.

Requisitos Funcionales

- RF1* El usuario podrá gestionar un nodo, entiendo como tal el poder iniciar y detener su operativa.
- RF2* El usuario podrá gestionar el vecindario de un nodo. Es decir, podrá conectarse y desconectarse de otros nodos así como obtener sus vecinos actuales.
- RF3* El usuario podrá gestionar el Aprendizaje Federado en la red por medio de un nodo, iniciándolo o deteniéndolo. Al inicio, el usuario deberá establecer el número de rondas y *epochs*.
- RF4* El usuario podrá visualizar información del proceso y estado del aprendizaje. También, podrá visualizar gráficas y métricas del entrenamiento.
- RF5* Los nodos se auto-organizarán en el proceso de aprendizaje. La elección de candidatos para la mejora del modelo y validación en cada ronda también será auto-gestionada.
- RF6* El usuario podrá agregar o modificar el modelo y los datos, permitiendo una fácil integración con cualquier framework de aprendizaje máquina.
- RF7* El usuario tendrá la posibilidad de crear y usar de diferentes agregadores federados de forma sencilla.
- RF8* El usuario podrá establecer fácilmente valores a los parámetros del sistema. Estos parámetros serán relativos a *gossip*, *timeouts* y valores de configuración del Aprendizaje Federado, entre otros.
- RF9* A la hora de iniciar un nodo, el usuario podrá indicar si su futura actividad formará parte de un entorno simulado o desplegado. La principal diferencia residirá en que para los entornos simulados se prescindirá de elementos que produzcan un *overhead* innecesario para una ejecución simulada. Por ejemplo, en entornos simulados debe prescindirse del cifrado.

Requisitos No Funcionales

- RNF1* **Disponibilidad:** Requisito no funcional implícito en *p2p*.
- RNF2* **Robustez:** El sistema debe ser tolerante a fallos, continuando con su operativa pese a la aparición errores.

- RNF3 Escalabilidad:** No deben existir problemas de rendimiento con el aumento del número de nodos.
- RNF4 Privacidad:** Puesto que la privacidad es una de las características más deseables del Aprendizaje Federado, se incluirán mecanismos adicionales para preservar la privacidad en las comunicaciones.
- RNF5 Extensibilidad:** La librería debe estar diseñada orientada a una sencilla ampliación de sus características. Destacarán por lo tanto la posible inclusión de nuevas librerías de **ML** así como la ampliación de algoritmos de agregación federada.
- RNF6 Usabilidad:** El uso de la librería debe estar orientado al usuario, permitiendo un uso intuitivo de la misma. La existencia de **documentación** facilitará la curva de aprendizaje inicial.
- RNF7** Los nodos deben poder ser ejecutados por dispositivos de bajas prestaciones.
- RNF8** El comportamiento por parte del nodo será completamente asíncrono y no bloqueante, siendo ejecutado en *background*.

5.2 *Sprint* 1: Arquitectura *p2p* completamente conectada

Este *sprint*, se centrará en la creación de la estructura inicial de la librería, así como un sistema *p2p* sencillo (véase el apartado 3.1.3). A pesar de que aún no se incorporará ninguna funcionalidad de aprendizaje, el diseño de la red y los protocolos de comunicación deben tener presente la inminente inclusión de los mismos.

5.2.1 Análisis

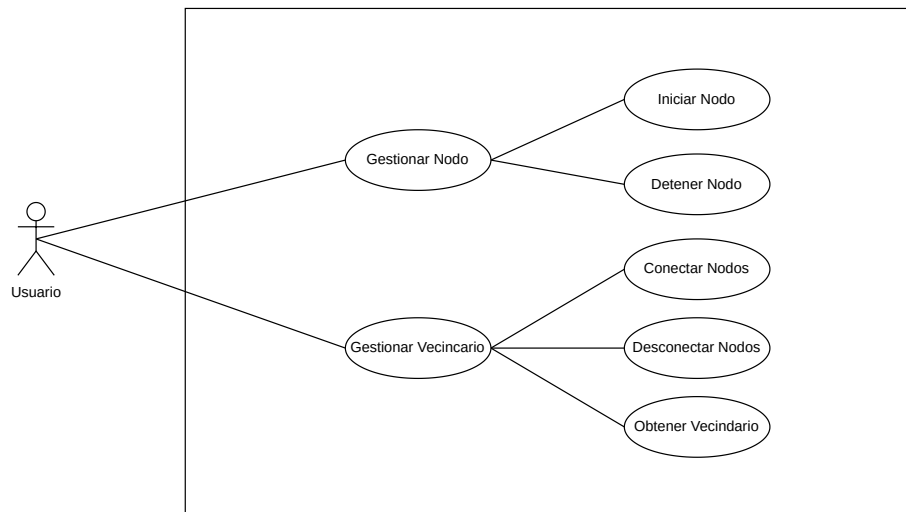
Los casos de uso a implementar en este *sprint* se encuentran en el diagrama de casos de uso de la figura 5.1, siendo detallados en la tabla 5.1. Debe apreciarse que únicamente son abordados casos de uso básicos para la gestión de la red *p2p*, existiendo un único actor: el usuario de la librería. Esto es debido a que el usuario será el único capaz de interactuar con la red *p2p* por medio de los nodos que posea.

Con el objetivo de facilitar el desarrollo inicial, se hará uso de una topología completamente conectada. Esta decisión será de carácter temporal, debido a que una red completamente conectada tiene una escalabilidad muy limitada.

CU-1: Iniciar Nodo	
Descripción	El usuario inicia el nodo. Involucra su creación.
Requisitos abordados	RF1, RF9.
Precondiciones	El nodo no debe estar iniciado.
Postcondiciones	El nodo estará a la escucha de conexiones entrantes.
Flujo Alternativo	El nodo no será iniciado pues ya se encuentra iniciado.
CU-2: Detener Nodo	
Descripción	El usuario detiene la operativa de un nodo en concreto.
Requisitos abordados	RF1.
Precondiciones	El nodo debe estar iniciado.
Postcondiciones	Todos los procesos que pudiese estar ejecutando el nodo serán detenidos.
Flujo Alternativo	El nodo no se detendrá pues ya se encuentra detenido.
CU-3: Conectar Nodos	
Descripción	El usuario conecta un nodo con otro. En este caso, para garantizar una topología completamente conectada también se conectará a todos los nodos vecinos del nodo contra el que se desea realizar una conexión.
Requisitos abordados	RF2.
Precondiciones	Los nodos a conectar deben estar iniciados y no debe existir una conexión entre ellos.
Postcondiciones	Los nodos deben resultar conectados.
Flujo Alternativo	La conexión no se efectuará.
CU-4: Desconectar Nodos	
Descripción	El usuario desconecta un nodo de otro.
Requisitos abordados	RF2.
Precondiciones	Los nodos a desconectar deben estar iniciados y conectados.
Postcondiciones	Los nodos deben resultar desconectados.
Flujo Alternativo	Los nodos no se desconectarán al no hallarse conectados.

Continuación de la tabla 5.1.

CU-5: Obtener Vecindario	
Descripción	El usuario puede solicitar la lista de vecinos del nodo.
Requisitos abordados	RF2.
Precondiciones	El nodo debe estar iniciado.
Postcondiciones	Ninguna.
Flujo Alternativo	Se devolverá una lista vacía pues un nodo no iniciado no tiene vecinos.

Tabla 5.1: Tabla de casos de uso implementados para el 1° *sprint*.Figura 5.1: Diagrama de casos de uso del 1° *sprint*.

5.2.2 Diseño

Se ha tratado de crear una base robusta y ampliable, diseñada de forma flexible y sencilla, buscando facilitar el uso, la comprensión y la modificación por parte del usuario. De este modo, en la figura 5.2 puede observarse el diseño planteado, siendo sus principales clases las siguientes:

- **BaseNode:** Nodo base. Implementa funcionalidades básicas de un nodo p2p.
- **NodeConnection:** Clase encargada de gestionar las conexiones entre nodos.

- **Heartbeater**: Implementa un mecanismo básico de *heartbeat*.
- **CommunicationProtocol**: Clase encargada de gestionar la comunicación entre los nodos. Definirá un protocolo de mensajes de modo que, cuando un mensaje sea válido, ejecutará el código correspondiente.

Puesto que la comunicación entre nodos es una de las partes que más se ampliará, se ha diseñado haciendo uso del **patrón comando**, buscando fáciles ampliaciones y modificaciones en el repertorio de comandos.

- **Events, Observable y Observer**: Con el objetivo de minimizar el acoplamiento, se ha aplicado el **patrón observador**. Este será empleado para controlar los diferentes eventos que puedan suceder en los componentes de *BaseNode*. Nuevamente, el patrón garantiza un proceso de ampliación trivial.

Para comprender de forma más detallada el funcionamiento de los nodos, en el diagrama de secuencia de la figura B.1 se detalla el proceso de conexión. Debe prestarse especial atención a como el nodo recibe un mensaje para la conexión, empleando *CommunicationProtocol* para su comprensión, y al propósito de los bucles: el de *BaseNode* para crear nuevas conexiones y el de *NodeConnection* para recibir mensajes de una conexión.

A continuación, se comentarán algunos aspectos de diseño relativos a dos de los puntos más importantes de la librería como son: el protocolo de comunicación y la robustez ante posibles caídas o fallos de este.

Protocolo de comunicación

Como se ha mencionado con anterioridad, se ha decidido usar *TCP* como protocolo base para el manejo de comunicaciones (véase en el apartado 3.2.3 la justificación de la decisión). Este protocolo será, por tanto, el encargado de gestionar las transmisiones entre los diferentes nodos. Las transmisiones serán codificadas en *8-bit Unicode Transformation Format (UTF-8)* debido a que permite representar cualquier carácter *unicode*. Los mensajes planteados para el protocolo de comunicaciones serán tratados a continuación:

- **BEAT**: Mensaje utilizado para el mecanismo *heartbeat*. Cuando se lleve una cantidad de tiempo determinada sin recibir ninguno, se asumirá algún tipo de contratiempo en la ejecución.
- **CONNECT <ip> <port> <full>**: Mensaje usado para establecer la conexión entre dos nodos. Se indica la dirección y el puerto del nodo que origina el mensaje para facili-

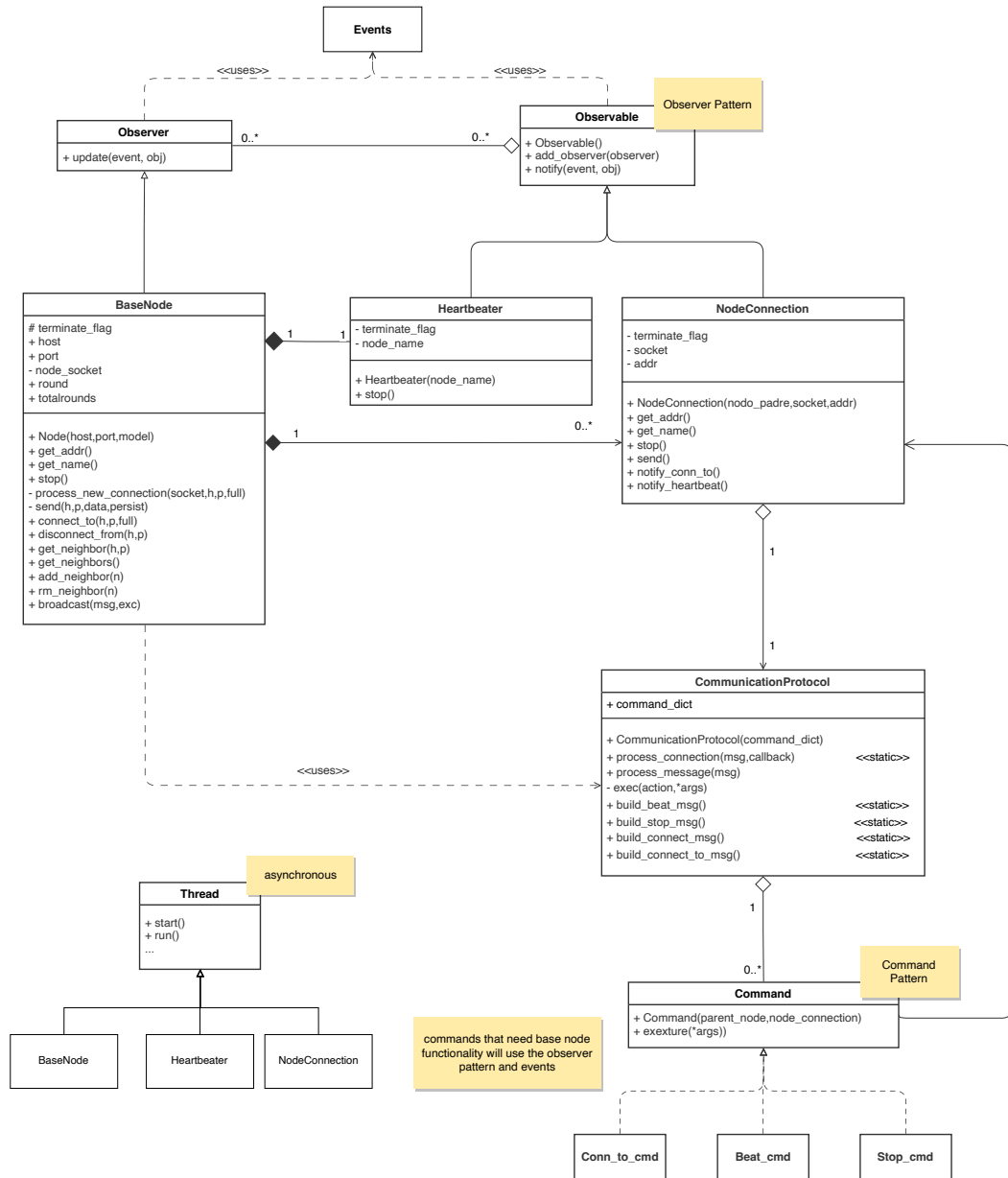


Figura 5.2: Diagrama de clases del 1º sprint

tar el manejo de conexiones. Si la opción *full* está activa, se emitirán a todos los vecinos mensajes *CONNECT_TO*. Haciendo esto con todas las conexiones originará una red completamente conectada.

- **CONNECT_TO** *<ip>* *<port>* Mensaje usado para indicarle a un nodo que inicie una conexión con otro.
- **STOP**: Este mensaje finaliza la conexión entre dos nodos.

Debe recalcar que la clase *CommunicationProtocol* opera como el componente *invoker* dentro del **patrón comando**, recibiendo mensajes procedentes de conexiones *TCP* y computando los comandos correspondientes. En la figura B.2 del próximo *sprint* podrá apreciarse el funcionamiento de este componente en un diagrama de secuencia.

Robustez

A estas alturas, como aún no se está ejecutando un proceso de Aprendizaje Federado, las caídas abruptas no implicarán grandes consecuencias. No obstante, se ha asegurado de que las caídas no perjudiquen el sistema.

Con el uso de *heartbeats* y el control en las conexiones *TCP*, se tendrá noción de si un determinado nodo está funcionando correctamente, para en caso contrario, eliminarlo del vecindario.

5.2.3 Implementación

En primer lugar, el código se ha estructurado y organizado siguiendo los estándares de *python* [41]. Se ha hecho uso de *wheel*, una de las herramientas que proporciona el ecosistema de *python* para distribuir librerías empaquetadas (*wheel packaging standard* [42]). Así, también, destaca la configuración de *sphinx* para la documentación de la librería.

Se han usado las librerías *threading* para el asincronismo y *socket* para manejar las conexiones *TCP*, siendo una implementación trivial, pues el principal objetivo ha sido el diseño. No obstante, cabe destacar un par de elementos dentro de la implementación como son:

- Por cada vecino que tenga un nodo, se dejará una conexión abierta.
- Los *heartbeats* no son procesados, se usan para resetear un *timeout* ya implementado por la clase *socket* (al recibir un mensaje se resetea).
- Si la conexión de un nodo con otro es interrumpida, esta no se levantará, rompiendo la estructura completamente conectada. Se ha decidido dejarlo así puesto que la topología completamente conectada es únicamente temporal.
- La recepción de mensajes usando *sockets* trabaja con *buffers*, es decir, cuando se recibe un mensaje, realmente se leen una cantidad determinada de bytes en el *buffer*.

Este funcionamiento combinado con el uso de *threads* puede propiciar que se estén mandando varios mensajes a la vez, acumulándolos en el *buffer* de recepción. La lectura y procesamiento multi-mensaje ha mitigado este problema.

5.2.4 Pruebas

Se han realizado múltiples pruebas de unidad, las cuales han buscado validar la funcionalidad y comprobar la tolerancia a fallos del sistema `p2p` básico. En la tabla 5.2 puede apreciarse una descripción de las pruebas realizadas, validando aparte de la funcionalidad, la tolerancia a fallos. Las pruebas han demostrado un buen funcionamiento del sistema, proporcionando una cobertura del 91% del código.

Prueba 1.1: Conexión de nodos	
Descripción	Se iniciarán y conectarán dos nodos, validando que se agreguen al vecindario.
Resultados Obtenidos	Dos nodos perfectamente conectados.
Prueba 1.2: Conexión a nodos inválidos	
Descripción	Tratará de conectarse a un nodo con una dirección inválida, asegurándose que no se agregue ningún nuevo nodo a la lista de vecinos.
Resultados Obtenidos	Un nodo sin nuevos vecinos.
Prueba 1.3: Conexión nodos completamente conectados	
Descripción	Se conectarán y desconectarán cuatro nodos, validando que se mantenga una topología completamente conectada.
Resultados Obtenidos	Nodos completamente conectados, respetando la topología.
Prueba 1.4: Validación multimensaje	
Descripción	Se enviarán a un nodo múltiples mensajes en un solo envío. El nodo destinatario debe comprender su significado sin producir errores.
Resultados Obtenidos	Comprensión de múltiples mensajes sin errores.
Prueba 1.5: Validación mensajes erróneos	
Descripción	Dados dos nodos conectados, se creará un mensaje erróneo, validando que los nodos se desconecten ante la ocurrencia del mismo.
Resultados Obtenidos	Dos nodos desconectados.
Prueba 1.6: Validación de operabilidad sin errores ante la ocurrencia de los mismos	
Descripción	Dados cuatro nodos completamente conectados, se irán introduciendo errores en los mismos, buscando la desconexión de nodos para no perjudicar al sistema. Los errores introducidos son: cierre abrupto del <code>socket</code> y detención del <code>heartbeater</code> .
Resultados Obtenidos	Control de los errores por parte de los nodos.

Tabla 5.2: Tabla explicatoria con pruebas unitarias realizadas en el *sprint* 1.

5.3 *Sprint* 2: Aprendizaje Federado

Este *sprint* tendrá como objetivo implementar Aprendizaje Federado sobre la red p2p ya creada.

5.3.1 Análisis

Los nuevos casos de uso a implementar permitirán, en esencia, el inicio y detenimiento del aprendizaje, permitiendo la modificación de elementos involucrados en el mismo y la visualización de resultados. La figura 5.3 muestra el diagrama de casos de uso actualizado, siendo detallados los nuevos casos de uso en la tabla 5.3.

CU-6: Iniciar Entrenamiento	
Descripción	El usuario iniciará el proceso de Aprendizaje Federado en los nodos conectados a la red. Debe especificarse el número de rondas y <i>epochs</i> a realizar. El modelo a inicializar en la red será el modelo del nodo que inicie el proceso.
Requisitos abordados	RF3.
Precondiciones	El nodo debe estar iniciado y sin ningún proceso de aprendizaje en ejecución.
Postcondiciones	El proceso de aprendizaje será iniciado en toda la red de nodos.
Flujo Alternativo	No comenzará el aprendizaje.
CU-7: Detener Entrenamiento	
Descripción	El usuario detendrá el proceso de Aprendizaje Federado en los nodos conectados a la red.
Requisitos abordados	RF3.
Precondiciones	El proceso de aprendizaje debe estar siendo ejecutado.
Postcondiciones	Todos los procesos relativos al aprendizaje han de finalizar.
Flujo Alternativo	No se detendrá el aprendizaje
CU-8: Visualizar Métricas e información relativa al entrenamiento	

Continuación de la tabla 5.3.

Descripción	El usuario podrá consultar y visualizar información relativa al proceso de aprendizaje así como información y métricas relativas al rendimiento de los modelos. Aún no se abordarán las métricas relativas al conjunto test para la validación del modelo.
Requisitos abordados	RF4.
Precondiciones	Debe haberse iniciado al menos un entrenamiento.
Postcondiciones	Ninguna.
Flujo Alternativo	No existirá información a ser visualizada.
CU-9: Establecer Modelo y Datos	
Descripción	El usuario podrá establecer el conjunto de datos así como un modelo en específico.
Requisitos abordados	RF6.
Precondiciones	Disponer de un nodo.
Postcondiciones	El modelo y datos del nodo serán los establecidos.
Flujo Alternativo	No se podrá ejecutar el caso de uso.
CU-10: Establecer Agregador Federado	
Descripción	El usuario podrá establecer un algoritmo de agregación federada arbitrario.
Requisitos abordados	RF7.
Precondiciones	Disponer de un nodo.
Postcondiciones	El agregador del nodo será el establecido.
Flujo Alternativo	No se podrá ejecutar el caso de uso.
CU-11: Configurar Experimento	
Descripción	El usuario podrá establecer parámetros relativos al funcionamiento del sistema.
Requisitos abordados	RF8
Precondiciones	Disponer de al menos, un nodo.
Postcondiciones	Los parámetros de funcionamiento serán los establecidos.

Continuación de la tabla 5.3.

Flujo Alternativo	No se podrá ejecutar el caso de uso.
-------------------	--------------------------------------

Tabla 5.3: Tabla de casos de uso implementados para el 2º *sprint*.

Figura 5.3: Diagrama de casos de uso implementados hasta el *sprint* 2. Los casos de uso en la zona verde son los nuevos casos de uso incluidos en el 2º *sprint*.

5.3.2 Diseño

Continuando con las líneas de diseño del *sprint* previo, se ha tratado de integrar el Aprendizaje Federado de una forma consistente y ampliable, primando un buen diseño. En la figura 5.4, se detalla el diagrama de clases correspondiente, siendo las principales clases agregadas las siguientes:

- **Node**: Amplia *BaseNode* permitiendo la operabilidad de un sistema de Aprendizaje Federado sobre una red p2p.
- **NodeLearner**: Interfaz que proporciona el manejo, codificación y decodificación de modelos de ML. Esta clase es lo que se conoce como plantilla en el **patrón homónimo**.
- **LightningLearner**: Como se ha comentado en el apartado 3.2, se ha decidido hacer uso de *PyTorch* como *framework* de ML. Por lo tanto, esta clase es una implementación de *NodeLearner* empleando *PyTorch Lightning*.
- **FederatedTensorboardLogger**: Como bien indica su nombre, permite hacer *logging* de toda la información relativa a ejecuciones federadas. Se ha adaptado un *logger* básico de *PyTorch* a un entorno federado.
- **Datamodules y Modelos**: Los modelos y datos empleados serán explicados en su correspondiente apartado en el *sprint* de pruebas 5.8.1. Destáquese el uso del **patrón singleton** en la carga de los datos con *MnistFederatedDM* para evitar operaciones de lecturas innecesarias en entornos simulados.
- **Aggregator**: Clase encargada de realizar la agregación. Su diseño trata de garantizar una sencilla extensibilidad gracias al **patrón plantilla**.
- **FedAvg**: Extiende *Aggregator*, implementando el algoritmo de agregación federada *FedAvg* tratado en el apartado 3.1.2.
- **Settings**: Debido a que se está trabajando con multitud de parámetros variable a configurar por el usuario, se ha decidido condensarlos todos en una misma clase.

Para comprender de forma más detallada el funcionamiento completo del sistema, las figuras B.2 y B.3 detallan diagramas de secuencia para dos de las partes más importantes de este *sprint*: la secuencia de inicio del aprendizaje y la agregación de modelos.

El primer diagrama detalla los pasos del aprendizaje y la comunicación entre un par de nodos, donde el primer nodo inicia el proceso de aprendizaje. Nuevamente, debe prestarse especial atención en como los mensajes recibidos en *NodeConnection* son comprendidos empleando *CommunicationProtocol* y ejecutados por medio del comando correspondiente. En cuanto al segundo diagrama, muestra el proceso de recepción de un modelo. Pueden apreciarse dos circunstancias, la inicialización del modelo o la agregación del modelo empleando *FedAvg*. Además, se detalla como se inicia el *timeout* de la agregación.

De igual modo que en el *sprint* previo, se comentarán los aspectos de diseño más importantes relacionados con protocolo de comunicación y la robustez.

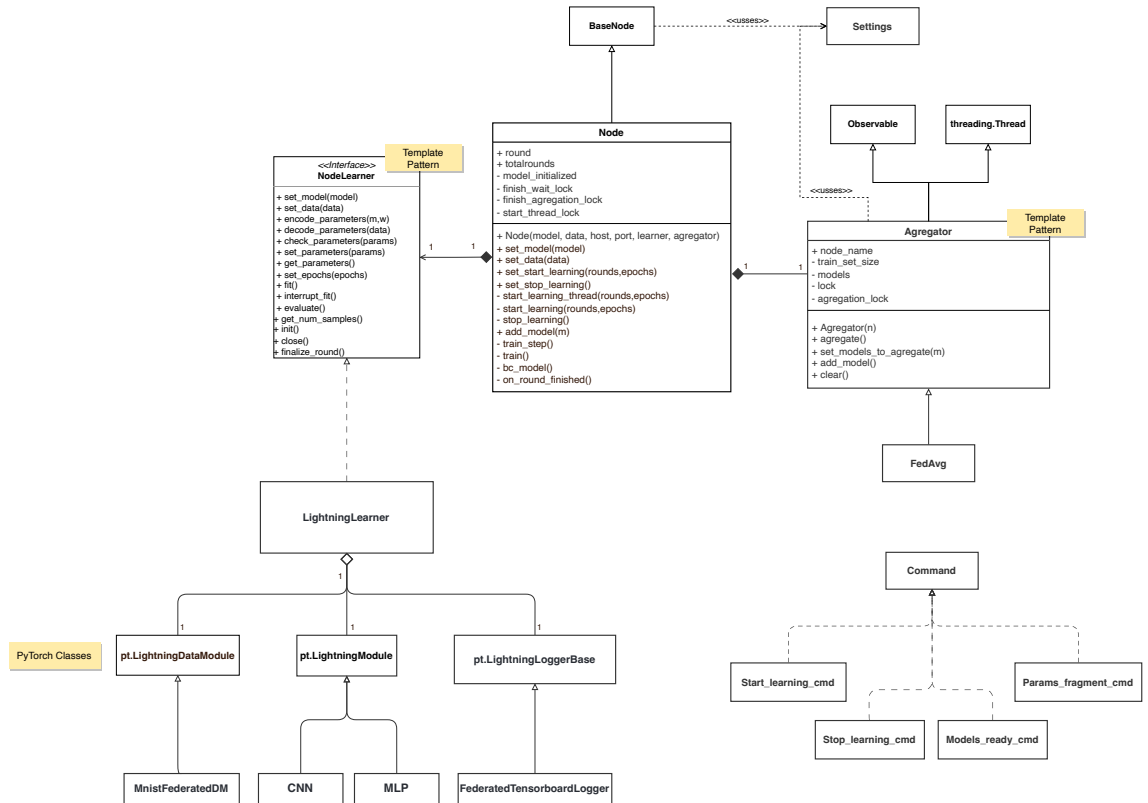


Figura 5.4: Diagrama de clases para el 2º sprint.

Protocolo de comunicación

Debido a que se necesita transmitir los pesos del modelo, el protocolo de comunicación soportará un nuevo tipo de mensaje. Los nuevos mensajes agregados al protocolo pueden clasificarse en:

- **Texto codificado en UTF-8:** Idénticos a los mensajes ya creados. Los nuevos mensajes incorporados son:
 - **START_LEARNING <rounds> <epochs>:** Indica el inicio del proceso de aprendizaje, así como el número de rondas y los *epochs* por ronda.
 - **STOP_LEARNING:** Comunica la finalización abrupta del proceso de aprendizaje.
 - **MODELS_READY <round>:** El nodo que manda el mensaje indica que ya dispone del modelo agregado para la ronda.
- **Modelos Serializados:** Los modelos y el número de muestras usadas serán serializados, fragmentados y enviados en diferentes mensajes. Su estructura será la siguiente:

- **PARAMS <data>**: Para todos los fragmentos, salvo el último.
- **PARAMS <data> \PARAMS**: Para el último mensaje, indica el final del envío.

Robustez

Partiendo de la robustez en las conexiones creada previamente, se tratará este mismo problema pero durante el entrenamiento, contemplando nuevas situaciones. De este modo, los errores que pueden darse son los siguientes:

- **Caídas de nodos durante el entrenamiento**: No se puede hacer nada. El nodo será eliminado del entrenamiento.
- **Caídas de conexiones durante el entrenamiento**: Realmente, un nodo no puede diferenciar si otro ha caído completamente o ha sido únicamente la conexión entre los mismos.

Se han planteado mecanismos que controlen los modelos agregados por nodo, de forma que al finalizar todos los nodos, se validen la cantidad de modelos agregados, igualándolos si fuese necesario. Finalmente, siguiendo el principio *Keep It Simple, Stupid! (KISS)* y teniendo en cuenta la inminente implementación de protocolos *Gossip* mitigarán estos errores por diseño, se ha decidido no implementar dicho mecanismo.

Debe tenerse en cuenta que las caídas de conexiones no tendrán grandes consecuencias. El hecho de que se parta de una inicialización común minimizará las futuras variaciones entre modelos y las consecuencias se resumen en una posible peor *performance* del modelo.

- **Modelos Incompatibles**: Error detectado en la inicialización del modelo. El modelo que no haya iniciado el entrenamiento será detenido por motivos de incompatibilidad.
- **Binarios Erróneos**: *TCP* proporciona una comunicación fiable, por lo que no deben de existir problemas en la transmisión de los mensajes.

Para lidiar con errores no contemplados o problemas de rendimiento por parte de los nodos o componentes de red intermedios, se han incluido mecanismos de *timeout*. En este *sprint* únicamente ha sido integrado en el proceso de agregación, comenzando la cuenta atrás desde que se agregue el primer modelo, incluido el local.

5.3.3 Implementación

En primer lugar, se han implementado las comunicaciones y el flujo del aprendizaje, realizando una simple promediación. En esta primera toma de contacto, debe destacarse de nuevo la librería *threading* de *python*, pues los procesos de agregación y entrenamiento son procesados en *background*. Así mismo, se ha hecho uso de *Locks* para hacer *thread safe* secciones críticas y crear mecanismos de sincronización.

Tras la implementación del flujo de Aprendizaje Federado, se ha procedido a crear los diferentes módulos de *Pytorch Lightning*. Para la transmisión de modelos, se han probado diferentes métodos, siendo la serialización el método con mejores resultados. Se ha usado la librería *pickle*, pues incluye métodos para la serialización y des-serIALIZACIÓN de estructuras de objetos de *python*.

La transmisión de los modelos ha generado bastantes problemas. Los parámetros serializados han sido fragmentados de modo que ocupen el máximo tamaño de mensaje, produciendo problemas en la recepción si coinciden con mensajes de inferior tamaño en el *buffer*. Esto des-alineará completamente la lectura de fragmentos induciendo errores en la des-serIALIZACIÓN. Para solventar este fallo, se detectarán las colisiones de fragmentos y mensajes. Ante la detección de colisiones, se procesarán los mensajes de texto y, posteriormente, se leerá del *buffer* únicamente los bytes restantes del fragmento del modelo a procesar.

También ha surgido otro problema en la recepción de fragmentos de modelos. Este es debido a que la recepción de mensajes en *TCP* con *sockets* no garantiza que se lea un fragmento de igual tamaño al enviado. De este modo, ha sido necesaria la inclusión de mecanismos que garanticen el tamaño del bloque.

En cuanto a la tolerancia a fallos, contemplando las situaciones planteadas en el apartado previo y el uso de *timeouts*, se han controlado satisfactoriamente los fallos que pudieran suceder.

Con el proceso de aprendizaje acabado, se procedió a crear un *logger* usando *TensorBoard* que le permitiese al usuario visualizar métricas e información del entrenamiento de forma sencilla y visual. El *logger* ha sido implementado partiendo de la clase base *LightningLoggerBase*, residiendo su peculiaridad en una adaptación para ser persistente y consistente ante diferentes entrenamientos en cada ronda de aprendizaje.

Finalmente, debe mencionarse que el número de ficheros abiertos aumentará con las dimensiones del experimento, pudiendo exceder las limitaciones del sistema operativo. Un simple comando, "*ulimit -n {MAX_OPEN_FILES}*", bastaría para solucionarlo, ténganse cuidado al eliminar restricciones de seguridad impuestas por el sistema operativo.

5.3.4 Pruebas

En primer lugar, debe dejarse claro que un sistema altamente tolerante a fallos hace que los test pierdan potencial, fallando únicamente en situaciones catastróficas.

De forma similar al *sprint* previo, se han creado diferentes pruebas de unidad, pudiendo observarse en la tabla 5.4. Las pruebas tendrán como objetivo validar el funcionamiento sin errores del sistema, tanto con una operativa normal, como para ejecuciones con errores inducidos. El análisis del rendimiento y resultados será tratado en el *sprint* dedicado a las pruebas 5.8.

Prueba 2.1: Codificar y decodificar parámetros.	
Descripción	Se codificarán y decodificarán parámetros validando que los parámetros decodificados sean idénticos a los originales.
Resultados Obtenidos	Parámetros decodificados idénticos al original.
Prueba 2.2: Promediación sencilla usando el agregador.	
Descripción	Se promediarán diferentes valores usando el agregador validando que los resultados sean iguales a los esperados.
Resultados Obtenidos	Valores obtenidos idénticos al los esperados.
Prueba 2.3: Promediación de modelos.	
Descripción	Se promediarán diferentes valores usando el agregador validando que los resultados sean iguales a los esperados.
Resultados Obtenidos	Modelos obtenidos idénticos al los esperados.
Prueba 2.4: Validación convergencia.	
Descripción	Prueba parametrizada ($N=1/2$ $X=2$). Dada una red con N nodos, se iniciará un entrenamiento de X rondas. Se revisará que los modelos resultantes al final del entrenamiento sean los mismos para todos los nodos.
Resultados Obtenidos	Entrenamiento finalizado con modelos idénticos para todos los nodos.
Prueba 2.5: Interrupción del entrenamiento.	
Descripción	Se iniciará un entrenamiento con un alto número de rondas. Posteriormente, el entrenamiento será detenido.
Resultados Obtenidos	Entrenamiento detenido.
Prueba 2.6: Caída de un nodo durante el entrenamiento.	
Descripción	Dados cuatro nodos conectados se iniciará el aprendizaje. Acto seguido, uno de estos nodos se desconectará repentinamente.

Continuación de la tabla 5.4.

Resultados Obtenidos	Continuación del proceso de aprendizaje pese a la caída de uno de los nodos participantes.
Prueba 2.7: Error en decodificación de modelos.	
Descripción	Se agregarán bytes en el paso del modelo que provoquen un error en la decodificación del mismo. Se validará que se detenga el nodo afectado, desconectándose del resto.
Resultados Obtenidos	Desconexión del nodo afectado por el error en la decodificación del modelo.
Prueba 2.8: Aprendizaje con modelos diferentes.	
Descripción	Dados dos nodos, cada uno con un modelo diferente, se iniciará el aprendizaje. Se validará que el nodo que no inicia el aprendizaje se detenga, desconectándose del resto.
Resultados Obtenidos	Desconexión del nodo que no inicia el proceso de aprendizaje.

Tabla 5.4: Tabla explicatoria con pruebas unitarias realizadas en el *sprint* 2.

5.4 *Sprint* 3: Conjunto test y Subconjuntos

El objetivo de este *sprint* es el de agregar funcionalidades de Aprendizaje Federado no abordadas por la iteración previa.

5.4.1 Análisis

En esencia, se modificarán los pasos del aprendizaje para permitir una evaluación correcta de los modelos resultantes usando conjuntos de test. También se incluirán los subconjuntos en el entrenamiento, permitiendo la parametrización de los nodos participantes en cada ronda.

Como consecuencia directa, este *sprint* no implementará casos de uso nuevos, pero modificará levemente los existentes. Los casos de uso modificados son especificados en la tabla 5.5.

CU-3: Conectar Nodos	
Descripción	El usuario conecta un nodo con otro. En este caso, para garantizar una topología completamente conectada también se conectará a todos los nodos vecinos del nodo contra el que se desea realizar una conexión.

Continuación de la tabla 5.5.

Requisitos abordados	RF2.
Precondiciones	Los nodos a conectar deben estar iniciados, no debe existir una conexión entre ellos y ninguno debe estar ejecutando un proceso de aprendizaje.
Postcondiciones	Los nodos deben resultar conectados.
Flujo Alternativo	Si las precondiciones no se cumplen, no se ejecutará la conexión.
CU-6: Caso de uso Iniciar Entrenamiento	
Descripción	El usuario iniciará el proceso de Aprendizaje Federado en los nodos conectados a la red. Debe especificarse el número de rondas y <i>epochs</i> a realizar, siendo otros parámetros como el tamaño del conjunto de entrenamiento leídos de los ajustes. El modelo a inicializar en la red será el modelo del nodo que inicie el proceso.
Requisitos abordados	RF3, RF5.
Precondiciones	El nodo debe estar iniciado y sin ningún proceso de aprendizaje en ejecución.
Postcondiciones	Ninguna.
Flujo Alternativo	No comenzará el aprendizaje.
CU-8: Visualizar Métricas e información relativa al entrenamiento	
Descripción	El usuario podrá consultar y visualizar información relativa al proceso de aprendizaje así como información y métricas relativas al rendimiento de los modelos.
Requisitos abordados	RF4.
Precondiciones	Entrenamiento debe haberse iniciado al menos un entrenamiento.
Postcondiciones	Ninguna.
Flujo Alternativo	No existirá información a ser visualizada.

Tabla 5.5: Tabla de casos de uso implementados para el 3° *sprint*.

5.4.2 Diseño

Al ser una ampliación del *sprint* previo, la parte de diseño para el Aprendizaje Federado ya ha sido abordada. No obstante, deben mencionarse la inclusión de nuevos mensajes en el protocolo de comunicación y sus respectivos comandos:

- ***METRICS*** *<round>* *<loss>* *<metric>*: Mensaje usado para transmitir los resultados de la evaluación del modelo con el conjunto test de cada nodo. Su comando asociado será ***Metrics_cmd***.

Destáquese que las métricas únicamente serán enviadas si al nodo se le indicó que no se está ejecutando el aprendizaje en un entorno federado.

- ***VOTE_TRAIN_SET***(*<node>* *<punct>*)* ***VOTE_TRAIN_SET_CLOSE***: Mensaje usado para transmitir los votos de cada nodo. El número de votos será variable, dependiendo del tamaño de conjunto de entrenamiento especificado. El comando encargado de ejecutar la lógica cuando se reciba esta clase de mensajes será ***Vote_train_set_cmd***.

En cuanto al diseño de las nuevas funcionalidades a implementar, a continuación se especificarán las decisiones de diseño planteadas:

- El método usado para los mecanismos de votación consiste en un consenso basado en el azar. Cada nodo seleccionará candidatos, asignándole a cada uno un valor. Posteriormente, enviará las votaciones y esperará a recibir las de los otros nodos (proceso similar a la espera de modelos para la agregación). Tras la recepción de todos los votos, se sumarán las puntuaciones y se seleccionarán los nodos con más puntuación para formar el conjunto de entrenamiento. Los empates serán resueltos por el orden alfabético del nombre de cada nodo.
- Debido a que las conexiones de nuevos nodo en el instante de votación podrán causar diferencias en el conjunto de entrenamiento, se han decidido bloquear las conexiones durante el entrenamiento. Este suceso se produce cuando en la red existen nodos que ya han dado por finalizada la votación, pero un nuevo nodo se conecta a alguno que aún no la ha dado por finalizada.
- Para evitar realizar dos votaciones, se ha decidido que los nodos del conjunto de entrenamiento validen el modelo de forma previa al entrenamiento. Con la finalización de la última ronda, todos los nodos realizarán una última validación del modelo.

5.4.3 Implementación

Para la incorporación de estas dos funcionalidades se ha modificado la secuencia de pasos en el entrenamiento, siendo su inclusión trivial.

5.4.4 Pruebas

Para este *sprint* no se han realizado nuevas pruebas unitarias, siendo suficientes para validar el correcto funcionamiento del sistema las detalladas en la tabla 5.4. Éstas, han sido nuevamente ejecutadas obteniendo idénticos resultados. Adicionalmente, se han visualizando las métricas, comprobando que sean correctas, así como que participe únicamente el subconjunto de entrenamiento en cada ronda.

Debe recordarse que en el *sprint* destinado a las pruebas 5.8, se analizarán detalladamente los resultados obtenidos en el aprendizaje.

5.5 *Sprint 4: Protocol Gossip*

Este *sprint* estará destinado a la inclusión del protocolo *Gossip*. Dicho protocolo será el encargado de difundir mensajes y modelos, agregando una gran tolerancia a fallos e implementando un mecanismo para la difusión de mensajes entre pares no directamente conectados.

5.5.1 Análisis

De igual forma que en el *sprint* previo, no se implementarán nuevos casos de uso, sino que se refinarán los ya creados. Los casos de uso afectados se detallan en la tabla 5.7.

CU-6: Iniciar Entrenamiento	
Descripción	El usuario iniciará el proceso de Aprendizaje Federado en los nodos conectados a la red. Debe especificarse el número de rondas y <i>epochs</i> a realizar, siendo otros parámetros como el tamaño del conjunto de entrenamiento leídos de los ajustes. El modelo a inicializar en la red será el modelo del nodo que inicie el proceso. Lo relativo a la comunicación entre nodos será llevada a cabo mediante el protocolo <i>Gossip</i> .

Continuación de la tabla 5.6.

Requisitos abordados	RF3, RF5.
Precondiciones	El nodo debe estar iniciado y sin ningún proceso de aprendizaje en ejecución.
Postcondiciones	El proceso de aprendizaje será iniciado en toda la red de nodos.
Flujo Alternativo	No comenzará el aprendizaje.
CU-7: Detener Entrenamiento	
Descripción	El usuario detendrá el proceso de Aprendizaje Federado en los nodos conectados a la red. La difusión de este mensaje será usando el protocolo <i>Gossip</i>
Requisitos abordados	RF3.
Precondiciones	El proceso de aprendizaje debe estar siendo ejecutado.
Postcondiciones	Todos los procesos relativos al aprendizaje han de finalizar.
Flujo Alternativo	No se detendrá el aprendizaje

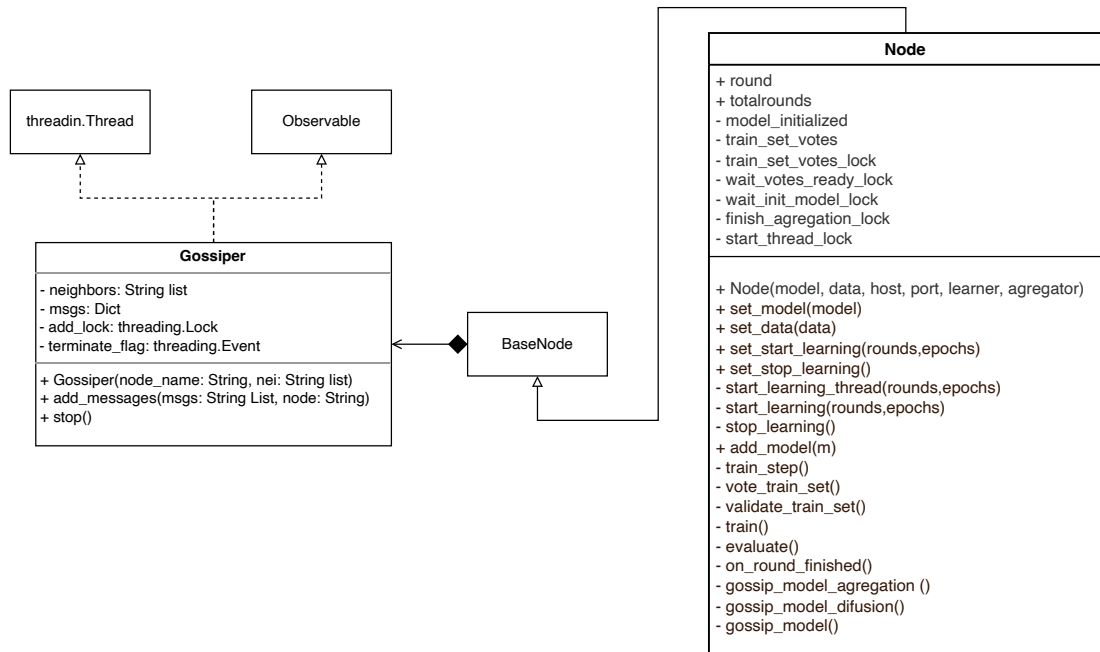
Tabla 5.6: Tabla de casos de uso implementados para el 4º *sprint*.

5.5.2 Diseño

Para detallar el diseño en la inclusión del protocolo *Gossip*, primeramente será tratada la difusión de mensajes, seguida de la difusión y agregación de modelos. En la figura 5.5, puede observarse el diagrama de clases que detalla los elementos incluidos y modificados en este *sprint*.

El protocolo *Gossip* para la difusión de mensajes es abordado con la creación de un nuevo componente (***Gossiper***). Este, reenviará los mensajes que *BaseNode* vaya recibiendo, siguiendo una cola *First In, First Out* (FIFO). Destáquese que los componentes de *BaseNode* se comunican con él por medio de eventos y el patrón observador.

La principal problemática de este protocolo es como tratar los ciclos infinitos en los mensajes. En primera instancia, se plantearon 2 opciones: difusión de mensajes de confirmación de recepción o *hasing* de mensajes. Como los mensajes de confirmación tendrían un tamaño similar a los mensajes en sí, se ha optado por calcular el *hash* los mensajes, manteniendo una lista de los últimos procesados. En la figura B.4, puede observarse el diagrama de secuencia para la

Figura 5.5: Diagrama de clases para el 4º *sprint*.

difusión de mensajes, apreciándose la dinámica de selección y envío de mensajes pendientes siguiendo una frecuencia determinada.

Los mensajes que deben ser difundidos por la red serán modificados, agregándole un **identificador único o hash** al final del mismo. Dicho *hash*, estará formado por el mensaje, el instante de tiempo de creación y un número aleatorio. Los mensajes modificados serán: *START_LEARNING*, *STOP_LEARNING*, *VOTE_TRAIN_SET* y *METRICS*. Para estos dos últimos, ha tenido que incluirse el creador del mismo, pues hasta ahora, se suponía que quien enviaba el mensaje era también su creador.

En lo tocante a la difusión de modelos, se crearon diferentes métodos en *Node*. Se usarán mensajes de confirmación, pues en este caso, reenviar modelos innecesariamente sería altamente costoso. Así mismo, para la agregación de modelos, todos los nodos del conjunto de entrenamiento deben ser conocedores de los modelos que posee cada nodo.

En la figura B.5 puede apreciarse el diagrama de secuencia para la difusión de modelos. Su flujo será exactamente igual a la difusión de mensajes, con la salvedad de que se enviarán modelos, empleando métodos de *Node* en lugar del componente *Gossiper*. Téngase en cuenta, que para la difusión de modelos podrán existir dos situaciones, siendo la principal diferencia el modelo que se envía:

- **Difusión de modelos:** Proceso realmente similar a la difusión de mensajes. Únicamente busca que todos los nodos dispongan del mismo modelo. La difusión de modelos será necesaria en dos situaciones: la inicialización de los modelos y la difusión de modelos posterior a la agregación. El nuevo mensaje en el protocolo de comunicación será *MODEL_INITIALIZED* con su respectivo comando *Model_initialized_cmd*, pues *MODELS_READY* se seguirá utilizando de igual modo que antes.
- **Agregación de modelos:** El protocolo *Gossip* será usado para la agregación de modelos. Su funcionamiento es idéntico a la difusión, con la peculiaridad de que la información que difunda no será idéntica para todos los nodos.

Se aplicará la optimización a la agregación propuesta en el apartado 3.1.2. De este modo, cada nodo enviará las agregaciones parciales que le resulten convenientes a sus diferentes vecinos.

Recuérdese que para poder aplicar dicha optimización, el conjunto de modelos que conforman la agregación parcial debe ser disjunto al conjunto de modelos ya agregados por un nodo. De esta forma, inmediatamente después de que se agregue un modelo, se difundirá por la red los modelos que se han agregado. Para esto, se ha creado un nuevo mensaje *MODELS_AGGREGATED* con su respectivo comando *Models_aggregated_cmd*. En cuanto a la agregación en sí, será idéntica a la propuesta en el [sprint 2](#) (véase el diagrama de secuencia en la figura B.3).

Finalmente, debe mencionarse que si se desea usar un algoritmo de agregación federada no basado en [FedAvg](#), podrá afrontarse la agregación de 2 formas: no emplear las agregaciones parciales, es decir, difundiendo únicamente un modelo por iteración del bucle *Gossip* o redefinir la forma de operar de las agregaciones parciales en *Aggregator*.

5.5.3 Implementación

El cambio más sustancial que implica *Gossip* en la recepción de modelos es el cambio de una espera pasiva a una activa. Es decir, en [sprints](#) previos, se hacía uso de *locks* que esperaban la recepción de modelos, validando tras cada recepción si todos los nodos habían finalizado. Ahora, gracias al uso de *Gossip*, las esperas serán activas, mandando, si es posible, los modelos que precise su vecindario. De igual forma que en versiones previas, existirán mecanismos de *timeout*, siendo accionados en los bucles *Gossip* cuando los nodos vecinos se queden estancados. Por ejemplo, un nodo lleva más de cierto umbral de rondas sin finalizar ni notificar un cambio de estado.

En cuanto al protocolo *Gossip* para la difusión de mensajes, debe destacarse que se ha tenido

un excesivo cuidado con el hecho de que mensajes duplicados no puedan dañar el sistema. Los mensajes duplicados pueden suceder cuando dos *NodeConnections* reciben al unísono un mismo mensjae.

Finalmente, respecto a la forma de especificar los nodos involucrados en las agregaciones parciales, en vez de transmitir únicamente el modelo, se ha decidido transmitir una tripla que contenga dicha información: modelo, nodos implicados, número de muestras totales.

5.5.4 Pruebas

Para este *sprint* no se han realizado pruebas específicas, siendo las existentes suficientes para garantizar el funcionamiento sin errores. De este modo, se han ejecutado nuevamente las pruebas detalladas en las tablas 5.2 y 5.4, obteniendo los mismos resultados.

Sin embargo, en el próximo *sprint* se generalizará la arquitectura, resultando imposible su funcionamiento sin el protocolo *Gossip*. De esta forma, las pruebas unitarias del próximo *sprint* validarán adicional e indirectamente el actual.

5.6 *Sprint* 5: Cambio de topología

El sistema *p2p* ha sido desarrollado empleando una topología completamente conectada, buscando únicamente la facilidad en el desarrollo. Desde el principio, se ha buscado flexibilidad y escalabilidad a la hora de creación de la arquitectura, así como la ligereza en el nodo, orientada al soporte de dispositivos de bajas prestaciones.

5.6.1 Análisis

Se adaptará el sistema para que funcione con cualquier tipo de topología, formando redes no estructuradas. El único caso de uso afectado es tratado en la tabla 5.7.

CU-3: Conectar Nodos	
Descripción	El usuario conecta un nodo con otro. Dado que el protocolo <i>Gossip</i> es usado, no es necesario que dos nodos estén directamente conectados para poder intercambiar mensajes.
Requisitos abordados	RF2.
Precondiciones	Los nodos a conectar deben estar iniciados, no debe existir una conexión entre ellos y ninguno debe estar ejecutando un proceso de aprendizaje.
Postcondiciones	Los nodos deben resultar conectados.
Flujo Alternativo	Si las precondiciones no se cumplen, no se ejecutará la conexión.

Tabla 5.7: Tabla de casos de uso implementados para el 4º *sprint*.

5.6.2 Diseño

El hecho de ya tener diseñados e implementados los mecanismos de *Gossip* pudiera parecer suficiente para migrar a cualquier otra topología. No obstante, será necesario disponer de un directorio de los nodos conectados a la red, pues tal como está diseñado el sistema no hay forma de saberlo. De este modo, se han planteado dos opciones para mantener una lista de vecinos activos:

1. **Lista mantenida por eventos:** La idea consiste en mantener una lista gestionada por eventos. Se precisarán mensajes de agregación y eliminación de nodos, necesitando mantener la información de que nodos están conectados a otros.
2. **Lista mantenida por *timeouts*:** Aprovechando los mensajes *heartbeat*, serán modificados indicando el nodo emisor y siendo difundidos empleando *Gossip*. El objetivo será mantener una lista de nodos activos en la red. Para mantener actualizada la lista, se borrarán los nodos de los que no se haya recibido un *heartbeat* en cierto intervalo de tiempo, permitiendo parametrizar su comportamiento.

Todas las situaciones en las que la caída de un nodo pueda causar inanición, como la agregación, votación o difusión de modelos, disponen de *timeouts* propios. Por lo tanto, a pesar de que se pueda tardar algo más en determinar la caída de un nodo, la simplicidad de este mecanismo hará que sea la opción a implementar.

En cuanto al diseño general del sistema no ha habido cambios relevantes, siendo los diagramas

de las figuras 5.2, 5.4 y 5.5 fieles al diseño del sistema.

5.6.3 Implementación

El proceso de implementación no ha dado pie a ninguna particularidad excesivamente importante, sin embargo, se comentarán las decisiones de implementación importantes a tener en cuenta:

- El conjunto de entrenamiento estará completamente conectado. Trás la eleccion de candidatos, estos se conectarán entre si. Dicha decisión está motivada para acelerar el proceso de agregación, así como hacerlo más tolerante a fallos (a más interconexiones, más redundancia). Fallos o imposibilidades en las conexiones no implicará ningún inconveniente gracias al protocolo *Gossip*.
- *Gossip* está diseñado de forma que un nodo, únicamente atenderá a sus vecinos directamente conectados. En consecuencia, para las redes no completamente conectadas, el único mecanismo de espera global será la votación al inicio de cada ronda.
- Nuevamente, como sucedía en los *sprints* previos, el punto más crítico del sistema es la votación de la primera ronda. En este caso, un mensaje *heartbeat* no difundido por toda la red podría originar inconsistencias en el *trainset*. De este modo, se agregará un tiempo de espera parametrizable. Esta espera buscará garantizar que la convergencia de mensajes *heartbeat* se ha dado antes de bloquear los nodos participantes en el entrenamiento.

5.6.4 Pruebas

En relación con las pruebas unitarias, se han creado nuevas, así como modificado levemente algunas debido al cambio de topología. En la figura 5.8 podrán observarse todas las pruebas que validan los cambios realizados en este *sprint*, siendo las tres primeras pruebas modificadas.

Asimismo, se ha validado que el rendimiento de los modelos resultantes sea el mismo que usando topologías completamente conectadas.

Prueba 1.1: Conexión de nodos	
Descripción	Se iniciarán y conectarán dos nodos, validando que se agreguen al vecindario y a la lista de nodos en la red.

Continuación de la tabla 5.8.

Resultados Obtenidos	Dos nodos perfectamente conectados.
Prueba 1.3: Conexión nodos completamente conectados	
Descripción	Se conectarán y desconectarán cuatro nodos, validando que se mantenga la topología y se actualice la lista de nodos conectados a la red.
Resultados Obtenidos	Nodos completamente conectados, respetando la topología.
Prueba 1.6: Validación de operabilidad sin errores ante la ocurrencia de los mismos	
Descripción	Dados cuatro nodos completamente conectados, se irán introduciendo errores en los mismos, buscando la desconexión de nodos para no perjudicar la red. Los errores introducidos son: cierre abrupto del <i>socket</i> y detención del <i>heartbeater</i> . Los nodos afectados deberán ser eliminarlos del vecindario y de la red.
Resultados Obtenidos	Eliminación de nodos con errores de la red.
Prueba 5.1: Conexión de nodos no completamente conectados	
Descripción	Cuatro nodos serán conectados siguiendo una topología en anillo. Se revisarán los vecindarios, asegurando que se cumple la topología, también se validará que todos los nodos estén disponibles en la red.
Resultados Obtenidos	Nodos conectados y visibles cumpliendo la topología.
Prueba 5.2: Ejecución del proceso de aprendizaje en redes con diferentes topologías: Anillo, Estrella, Línea	
Descripción	Cuatro nodos serán conectados siguiendo en diferentes topologías. Se validará que el entrenamiento sea exitoso para todos los nodos.
Resultados Obtenidos	Entrenamientos exitosos.

Tabla 5.8: Tabla explicatoria con pruebas unitarias realizadas en el 5º *sprint*.

5.7 *Sprint* 6: Cifrado en las comunicaciones

Este *sprint* se centrará en la inclusión de mecanismos que garanticen la privacidad en las comunicaciones. El principal objetivo del cifrado de las comunicaciones será el de evitar ataques de ingeniería inversa que comprometan los datos.

5.7.1 Análisis

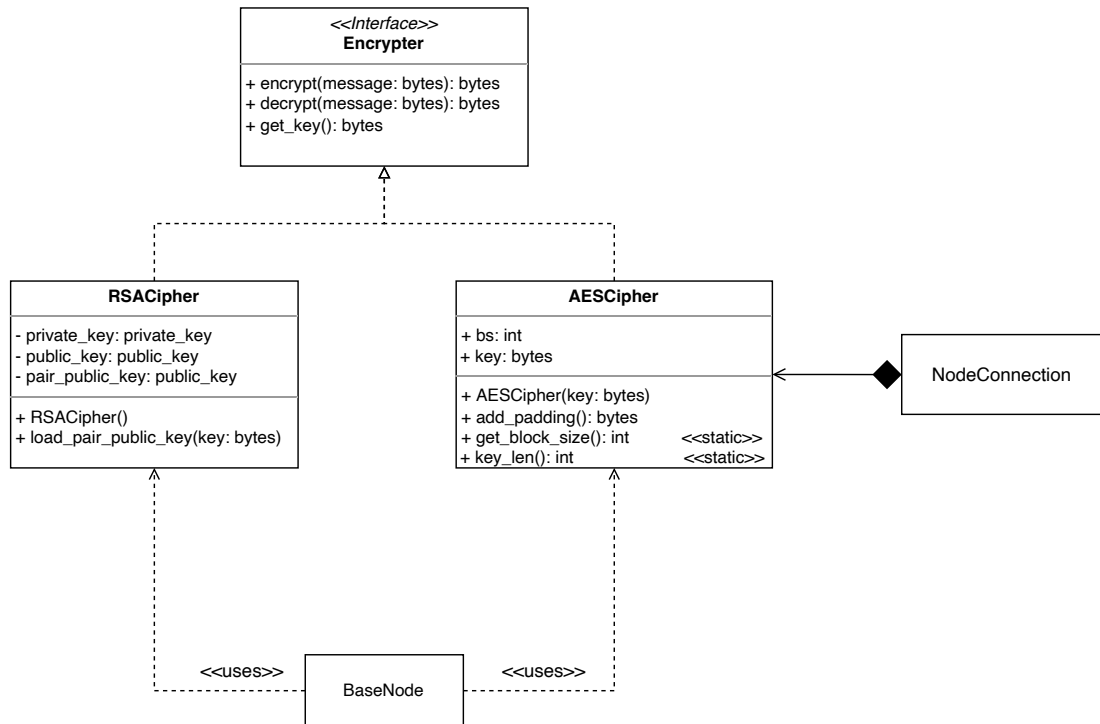
Ante la inclusión de comunicaciones cifradas, el caso de uso afectado se detalla en la tabla 5.9.

CU-6: Iniciar Entrenamiento	
Descripción	El usuario iniciará el proceso de Aprendizaje Federado en los nodos conectados a la red. Debe especificarse el número de rondas y <i>epochs</i> a realizar, siendo otros parámetros como el tamaño del conjunto de entrenamiento leídos de los ajustes. El modelo a inicializar en la red será el modelo del nodo que inicie el proceso. Lo relativo a la comunicación entre nodos será llevada a cabo mediante el protocolo <i>Gossip</i> , siendo cifrada si el usuario lo indica.
Requisitos abordados	RF3, RF5.
Precondiciones	El nodo debe estar iniciado y sin ningún proceso de aprendizaje en ejecución.
Postcondiciones	El proceso de aprendizaje será iniciado en toda la red de nodos.
Flujo Alternativo	No comenzará el aprendizaje.

Tabla 5.9: Tabla de casos de uso implementados para el 6° *sprint*.

5.7.2 Diseño

Tal como se comenta en los fundamentos de la encriptación 3.1.5, será necesario establecer un cifrado asimétrico para poder intercambiar la clave compartida del cifrado simétrico. Recuerdese que será usada la librería *pycryptodome*, pues implementa los algoritmos criptográficos a usar. En la figura 5.6, se detalla el diagrama de clases correspondiente a la inclusión de los algoritmos de cifrado.

Figura 5.6: Diagrama de clases para el 6° *sprint*.

5.7.3 Implementación

Al ser **AES** un algoritmo de cifrado por bloques, debe tenerse en cuenta que el tamaño de bloque usado por los nodos debe ser múltiplo del tamaño de bloque empleado por **AES**. Para facilitar el uso y reducir los errores que pueda cometer el usuario, el tamaño de bloque especificado será automáticamente redimensionado al próximo tamaño de bloque válido.

En lo referente a mensajes de texto codificados, es necesario aplicar un relleno adicional para alcanzar un tamaño múltiplo del tamaño de bloque de **AES**. Se ha decidido incluir espacios, pues es un carácter irrelevante para el significado del mensaje, ahorrando la tarea de tener que eliminar el relleno. Además, dicho carácter codificado en **UTF-8** ocupa 1 *byte*, la unidad mínima.

Finalmente, dado que la encriptación de mensajes añade un *overhead* innecesario en entornos simulados, el cifrado de mensajes será únicamente usado en entornos no simulados.

5.7.4 Pruebas

En la tabla 5.10, se detallan las nuevas pruebas unitarias agregadas. En vista de los resultados obtenidos, se puede concluir que ante la existencia de comunicaciones cifradas, la funcionalidad del sistema es idéntica.

Prueba 6.1: Cifrado simple usando RSA	
Descripción	Únicamente se creará un par clave pública/privada. Con estas claves, debe cifrarse y descifrarse un texto codificado dado.
Resultados Obtenidos	El texto descifrado coincide con el original.
Prueba 6.2: Cifrado entre pares usando RSA	
Descripción	Se crearán dos pares clave pública/privada. Empleando las claves del otro par, se cifrará y descifrará de texto codificado.
Resultados Obtenidos	El texto descifrado coincide con el original.
Prueba 6.3: Cifrado simple usando AES	
Descripción	Se creará una clave, esta será empleada para cifrar y descifrar de texto codificado.
Resultados Obtenidos	El texto descifrado coincide con el original.
Prueba 6.4: Cifrado entre pares usando AES	
Descripción	Se creará una clave que será compartida, la clave se empleará para cifrar y descifrar de texto codificado.
Resultados Obtenidos	El texto descifrado coincide con el original.
Prueba 6.5: Cifrado de modelos usando AES	
Descripción	Se creará una clave, la clave se empleará para cifrar y descifrar un modelo fragmentado.
Resultados Obtenidos	El modelo descifrado coincide con el original.
Prueba 6.6: Ejecución de entrenamiento usando comunicaciones cifradas.	
Descripción	Se ejecutará un entrenamiento simple entre dos nodos usando comunicaciones cifradas.
Resultados Obtenidos	Entrenamiento satisfactorio sin errores.

Tabla 5.10: Tabla explicatoria con pruebas unitarias realizadas en el 5° *sprint*.

5.8 *Sprint* 7: Aplicación Práctica

A diferencia de los demás *sprints*, este se centrará únicamente en la realización de pruebas. Éstas, serán bastante más exhaustivas que las realizadas en los *sprints* de desarrollo, en donde no se analizaban los resultados obtenidos ni el rendimiento.

En el próximo apartado se tratarán los *datasets* y modelos empleados para las pruebas. En el resto, se detallarán, explicarán y analizarán las diferentes pruebas realizadas, finalizando con una breve conclusión general.

5.8.1 *Datasets* Empleados

Para la ejecución de pruebas se usará el dataset clásico *Modified National Institute of Standards and Technology dataset* (MNIST) [43] y su variación para entornos federados, *Federated Extended Modified National Institute of Standards and Technology dataset* (FEMNIST). A continuación, los *datasets* serán detallados, tratando a mayores las RR.NN.AA empleadas para resolverlos.

MNIST

MNIST es una base de datos de imágenes de 28x28 píxeles sobre dígitos escritos a mano. El *dataset* consta de 60.000 imágenes para el entrenamiento y 10.000 imágenes para test.

Dado que va a ser usado en un entorno federado, se distribuirán las muestras a cada nodo. El *dataset* será fraccionado en conjuntos disjuntos, siendo asignado cada subconjunto a un nodo en concreto. Los subconjuntos serán obtenidos siguiendo una distribución IID, es decir, cada nodo dispondrá de un subconjunto representativo de muestras. Por lo tanto, si se tuviesen dos nodos, MNIST se partirá de forma balanceada y aleatoria, distribuyendo el dataset original entre estos dos nodos.

En cuanto a los modelos usados, el simple *MultiLayer Perceptron* (MLP) de dos capas ocultas detallado en la figura C.1 ha demostrado tener la capacidad suficiente para obtener buenos resultados. En la gráfica 5.7, puede apreciarse la obtención de aproximadamente el 98% de precisión en el conjunto test

FEMNIST

FEMNIST ha sido creada con el objetivo de creación de un marco de evaluación comparativa para el aprendizaje en entornos federados (LEAF [44]). Es un *dataset* derivado de *Extended Modified National Institute of Standards and Technology dataset* (EMNIST) [45], el sucesor de MNIST. EMNIST, no solo amplía notoriamente el número de muestras, sino que añade nuevas clases. Se añadirán letras mayúsculas y minúsculas escritas a mano, conformando un total de 62 tipos de muestra.

Para adaptar EMNIST a un entorno federado, FEMNIST divide los datos basándose en el escritor del dígito o carácter. Como resultado, FEMNIST consta de 80.5263 muestras repartidas entre 3,550 usuarios, con una media de 227 muestras por usuario aproximadamente.

En cuanto al modelo usado para resolver FEMNIST y EMNIST, está demostrado que es necesario el uso de algún modelo más complejo y potente como pudieran ser las *Convolutional Neural Network* (CNN) [46]. Concretamente, la red neuronal convolucional empleada se encuentra detallada en la figura C.2.

5.8.2 Pruebas de simulación

La primera prueba consistirá en el entrenamiento de un MLP para la resolución de MNIST bajo el paradigma federado y clásico, comparando el rendimiento y resultados de ambas ejecuciones. Destáquese que para la ejecución en el entorno federado, se realizará en una red de 6 nodos, usando como *trainset* la red completa.

Debido a que para comparar correctamente ambos métodos la cantidad de cómputo debe ser estrictamente la misma, la ejecución federada consistirá en 10 rondas de 1 *epoch* por nodo, es decir, en cada ronda se usarán una única vez todos los datos del conjunto entrenamiento para mejorar el modelo. Mientras tanto, para el entrenamiento clásico se ejecutarán 10 *epochs*.

En la figura 5.7, se muestran la precisión y el error obtenidos sobre el conjunto test. Estas métricas servirán para determinar la bondad del modelo, es decir, su capacidad de inferencia. Adicionalmente, en la figura 5.8 se muestra la evolución del error de los nodos en el conjunto de entrenamiento. Debe apreciarse como se reduce dicho error, aún cuando se producen agregaciones de modelos.

En general, las métricas obtenidas presentan una evolución y valores realmente similares para ambos entrenamientos. Ante la similitud de los resultados, la ganancia en tiempo debido a la paralelización del cómputo es evidente.

A pesar de la clara ganancia en paralelización, deben tenerse en cuenta los costes relativos a la comunicación, siendo el envío de modelos el coste más alto del sistema. Particularmente para este experimento, al trabajar sobre un [MLP](#), un modelo realmente simple y poco pesado, la comunicación no ha sido altamente costosa. No obstante, para la próxima prueba de simulación, se trabajará con [CNN](#), siendo el envío de modelos un factor altamente determinante en el coste final del sistema.

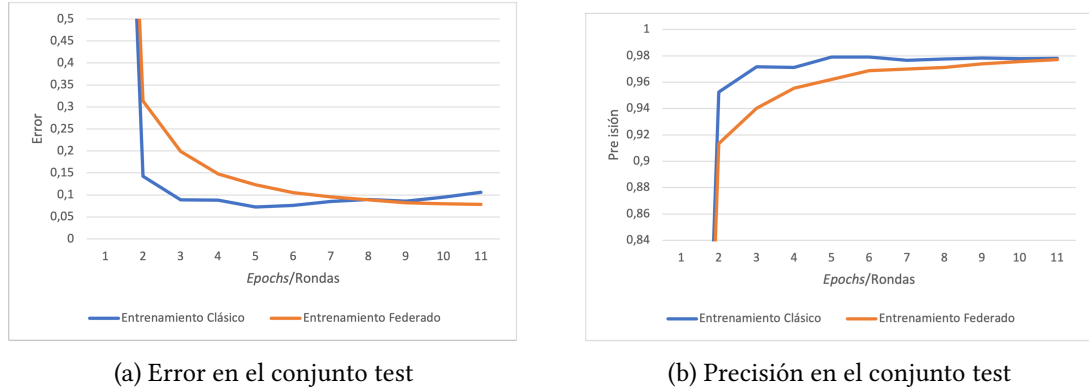


Figura 5.7: Comparativa del error y precisión sobre el conjunto test en un entrenamiento clásico y un entrenamiento federado para el experimento descrito en el apartado 5.8.2. Las métricas han sido obtenidas al finalizar cada ronda en un entrenamiento federado o cada *epoch* en un entrenamiento clásico. La cantidad de cómputo entre muestras será idéntica para ambas ejecuciones.

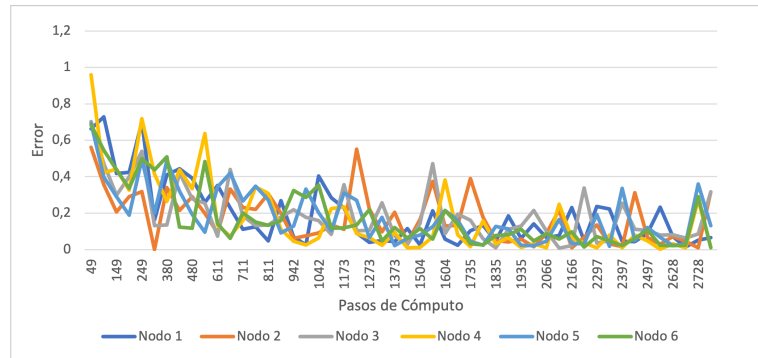


Figura 5.8: Error en el conjunto de entrenamiento del Aprendizaje Federado para el experimento descrito en el apartado 5.8.2. Recuérdese que la cantidad de cómputo es la cantidad de veces que se han empleado *batches* de 32 muestras para mejorar el modelo. Apréciase que el cómputo está paralelizado.

5.8.3 Validación del funcionamiento en un entorno real

La prueba a realizar buscará ser idéntica a la realizada en el apartado previo, pero en un entorno de despliegue. Se crearán un total de 6 nodos, estando distribuidos en 3 máquinas como se puede observar en la figura 5.9. En el *MacBook* residirán 2 nodos, en el ordenador de sobremesa 3 y finalmente en la *Raspberry* 1. Los nodos estarán organizados siguiendo una topología en línea.

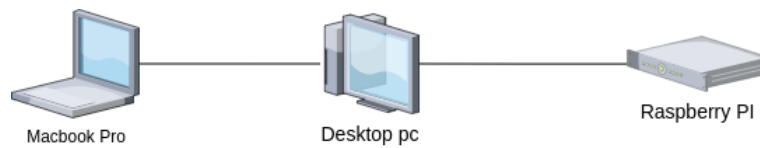


Figura 5.9: Dispositivos implicados en prueba de despliegue y sus conexiones.

Destáquese que al contar con dispositivos de prestaciones muy diferentes, será necesario ajustar los parámetros de la librería de forma consecuente. Todos los valores relativos a la espera de recepción de modelos serán establecidos permitiendo largas esperas, pues en comparación, la *Raspberry* presenta carencias en *hardware* y consecuentemente en rendimiento.

La ejecución ha sido exitosa, arrojando los mismos resultados que la prueba previa (véanse las figuras 5.7 y 5.8). De este modo, se valida el funcionamiento del sistema en un entorno real, es decir, un entorno distribuido.

5.8.4 Pruebas de simulación con alto número de nodos

Debido a la ausencia de pruebas con un alto número de nodos y pocas muestras por nodo, así como la falta de pruebas con modelos medianamente complejos, se ha decidido elaborar una simulación que se acerque más a un caso de uso real. Para esto, se hará uso de un nodo con 64 *cores* del [Centro de Supercomputación de Galicia \(CESGA\)](#).

El *dataset* empleado, [FEMNIST](#) ha sido descrito en el apartado 5.8.1. Sin embargo, no se ha utilizado el conjunto de datos al completo, ya que únicamente se crearán 40 nodos. En lo relativo a la configuración del experimento federado, el *trainset* será de 4 nodos, siendo ejecutadas 10 rondas. Los nodos del *trainset* computarán 2 *epochs* por ronda.

En cuanto al experimento equivalente empleando un entrenamiento clásico, nuevamente se ejecutará sobre los mismos datos y con la misma cantidad de cómputo. Teniendo en cuenta que para cada ronda de Aprendizaje Federado se emplearán 2 veces el 10 % de las muestras del *dataset*, un entrenamiento clásico equivalente debe tener una duración de 24 *epochs*. Recuérdese

que en cada *epoch* se iterarán una única vez todos los datos del conjunto de entrenamiento.

En las figura 5.10 y 5.11 se mostrará la evolución de los modelos, reflejada en el error y precisión sobre el conjunto test. Teniendo únicamente en cuenta los valores finales, se han obtenido resultados realmente similares. Por un lado, el entrenamiento federado ha logrado un error de 0.81 y una precisión del 78.2%, con una desviación típica de 0.34 y 13% respectivamente. En cuanto al entrenamiento clásico, se ha obtenido un error de 0.79 y una precisión del 80% aproximadamente, apreciándose algo de sobre-entrenamiento en la gráfica del error. A diferencia de otras pruebas realizadas donde los nodos disponían de más muestras, en el Aprendizaje Federado puede apreciarse un notorio retraso en el alcance de valores de convergencia.

Finalmente, en lo referente al costo de las comunicaciones, el uso de un modelo 9 veces más grande que *MLP* ha incrementado notoriamente el volumen y costo de las comunicaciones. Al coste computacional de la ejecución federada deben agregarse los costes de transmisión de 15 GB de parámetros de modelos.

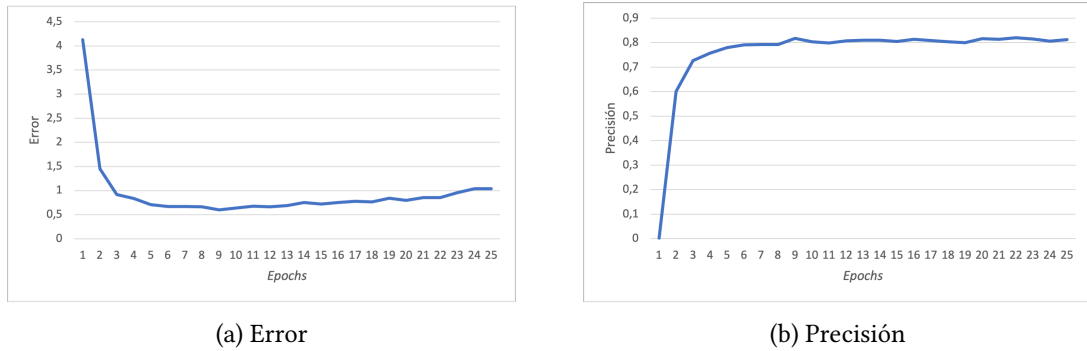


Figura 5.10: Error y precisión en conjunto test a lo largo de un aprendizaje clásico para el experimento descrito en el apartado 5.8.4.

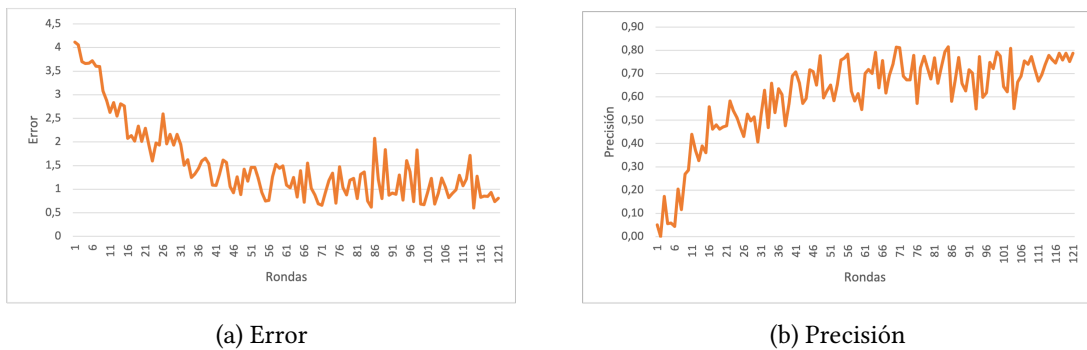


Figura 5.11: Error y precisión en conjunto test a lo largo del Aprendizaje Federado para el experimento descrito en el apartado 5.8.4.

5.8.5 Pruebas con datos *no-iid*

Debido a que las pruebas realizadas hasta el momento han sido siguiendo distribuciones de datos IID, es necesario abordar como se comporta el sistema ante distribuciones no IID. Es decir, analizar el rendimiento de los modelos resultantes cuando los nodos no dispongan de un conjunto de muestras representativas del problema a resolver. La obtención de buenos resultados con diferentes distribuciones depende esencialmente del agregador federado, como ya se verá, *FedAvg* tiene problemas con este tipo de distribuciones [47].

En la prueba a realizar, se recreará el peor escenario de distribución de datos. Para este escenario, se ha modificado *MnistFederatedDM*, ordenando el *dataset* por tipo muestras y distribuyéndolo disjunta y ordenadamente a cada nodo.

Para la prueba, se crearán 6 nodos, disponiendo cada uno de un máximo de 2 tipos de muestra. En cuanto a los parámetros del Aprendizaje Federado, se establecerá un *trainset* de 4 nodos, ejecutando 100 rondas. En cada ronda, los nodos únicamente ejecutarán 1 *epoch*.

Las métricas sobre el conjunto test resultantes de la ejecución del experimento pueden observarse en la figura 5.12. Como puede apreciarse, los resultados obtenidos son notoriamente inferiores a la ejecución del apartado 5.8.2. Apréciase que en estas 100 rondas no se ha dado ninguna convergencia entorno a ningún valor, alcanzando en la última un valor medio del 64.6%, con una desviación típica realmente alta, del 20.9%

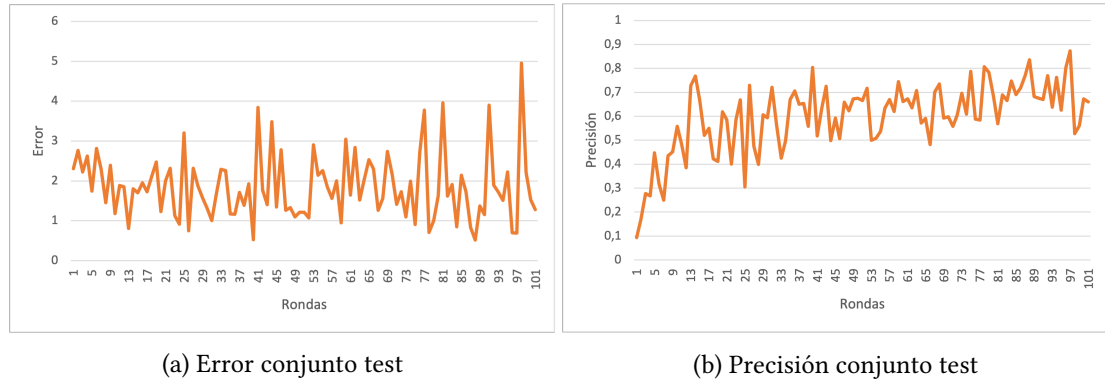


Figura 5.12: Error y precisión en el conjunto test para el experimento descrito en el apartado 5.8.5.

El principal motivo de que se produzca esta gran diferenciación en el rendimiento del modelo con esta distribución de datos, es debido a que las direcciones de mejora propuestas por cada entrenamiento local no tienen una clara direccionalidad común. En la figura 5.13, puede apreciarse gráficamente que para un conjunto de datos representativo, las direcciones de mejora

propuestas por cada nodo serán similares. No obstante, si las muestras no son representativas como sucede con la distribución no IID, las direcciones de mejora se contrarrestarán en la agregación, no resultando para nada efectiva.

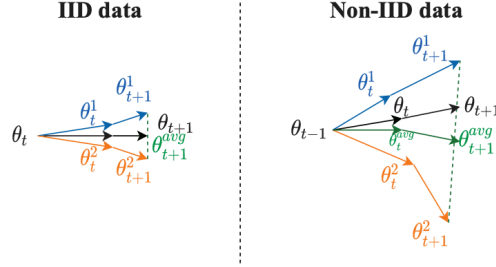


Figura 5.13: Ejemplo dirección vector mejora en distribuciones de datos *iid* y *non-iid*.

5.8.6 Conclusión sobre los resultados obtenidos

Las diferentes pruebas realizadas han demostrado un buen comportamiento del sistema. Siendo comparado con un entrenamiento clásico equivalente, se han alcanzado valores de error y precisión similares en cantidades de cómputo idénticas, únicamente se han apreciado dos carencias: el retraso en la convergencia estrictamente ligado a la cantidad de muestras del nodo y los problemas con conjuntos de datos no representativos producidos por distribuciones no IID.

La primera carencia, es completamente asumible. En la prueba del apartado 5.8.4 la convergencia se ha retrasado aproximadamente 48 rondas. Una cifra, para nada disparatada teniendo en cuenta el diminuto tamaño del *trainset* y el bajo número de muestras por nodo (227 aproximadamente) para esta ejecución.

En cuanto a los problemas con distribuciones no IID, debe tenerse en cuenta que se ha ejecutado el peor caso con un algoritmo de agregación (FedAvg) que tiene un mal comportamiento ante estas situaciones.

Relacionado con los costes en la comunicación del Aprendizaje Federado, se considera que no implicarán una desventaja al uso del sistema. No obstante, deben tenerse en cuenta los tiempos de difusión del modelo sobre la red, o las esperas por dispositivos con menos prestaciones.

Como conclusión, si un usuario que no dispone de una gran infraestructura le resultan atractivas las ventajas del Aprendizaje Federado, la librería suplirá todas sus necesidades.

Conclusiones

COMO se ha detallado en el capítulo dedicado al desarrollo 5, el objetivo de **elaborar una librería de soporte al Aprendizaje Federado sobre redes p2p** ha sido logrado con creces. En comparación con entrenamientos tradicionales, se ha conseguido un rendimiento y unos resultados realmente satisfactorios.

Relacionado con el potencial del FL, debe tenerse en cuenta que no únicamente la privacidad y el cambio de paradigma en la gestión de datos son elementos a tener en cuenta. La paralelización del cómputo en el entrenamiento será una de las grandes características del Aprendizaje Federado. Como se ha tratado en la introducción, la tendencia de los dispositivos *edge* a disponer cada vez de más capacidad de cómputo y datos, hacen que el FL disponga de un entorno ideal para poder demostrar su potencial en los próximos años.

En cuanto a los costes de comunicaciones, se considera que si bien son un nuevo tipo de coste a tener en cuenta, no implican ninguna desventaja añadida. Como se ha visto en la sección de pruebas, para una ejecución de 40 nodos y 120 rondas se han transmitido un total de 15 GB en toda la ejecución, resultando en una media de 0.375 GB en tráfico por nodo, para nada una cifra disparatada.

Pese al gran potencial y las claras ventajas que implican el Aprendizaje Federado, la principal traba de este proyecto ha estado ligada desde el principio a la escasa trayectoria del mismo en la literatura científica. Siendo uno de los puntos más críticos el planteamiento teórico de la viabilidad del proyecto, sin la existencia de una solución similar. Así mismo, debe tenerse en cuenta que para la agregación de modelos, **se ha propuesto una optimización que permite reducir considerablemente la cantidad comunicaciones entre nodos.**

En lo tocante a la librería desarrollada, la tolerancia a fallos ha sido uno de los requisitos incuestionables y esenciales, principalmente para tratar de garantizar un soporte a entornos

reales. En dichos entornos, los errores no serán únicamente inducidos por los dispositivos finales, si no que los dispositivos de red intermedios también serán un punto crítico, pudiendo inducir latencias o desconexiones entre pares. La redundancia, es uno de los mejores métodos para mitigar esta clase de errores, sin embargo, debe existir un equilibrio con el volumen de tráfico.

La topología elegida será un factor altamente determinante a la hora de emitir mensajes innecesarios. Teniendo en cuenta la implementación del protocolo *Gossip*, topologías con un alto número de vecinos por nodo como redes completamente conectadas implicarán más tráfico en comparación con topologías con bajo número de vecinos como pudiera ser una topología en bus. Naturalmente, no existe una opción mejor, debe buscarse un equilibrio. Por ejemplo, en redes con topología en bus, un fallo en un nodo causará la fractura de la red en dos subredes inconexas, imposibilitando la comunicación entre las mismas.

A pesar de que el funcionamiento de la librería es realmente satisfactorio, existen multitud de apartados que deben ser mejorados. Al ser estas futuras mejoras externas al trabajo y sus objetivos, serán tratadas en el apartado de trabajo futuro 7.

En definitiva, los objetivos planteados en el apartado 1.1, así como los requisitos propuestos, han sido abordados satisfactoriamente. La existencia de una librería base de código abierto como la desarrollada implica una base sólida para futuros desarrollos, ya bien sea orientados a investigación o creación de una plataforma usable.

Trabajo Futuro

EN este capítulo se tratarán las diferentes líneas de trabajo a realizar sobre el desarrollo realizado. Como se menciona en el capítulo de introducción, se ha puesto especial énfasis en la extensibilidad de la librería, pues el trabajo se ha centrado en la creación de una base sólida y funcional del Aprendizaje Federado sobre redes p2p.

De entre la multitud de líneas por las que se pueda continuar el desarrollo, se tratarán las que se consideran como las principales:

- **Nuevos algoritmos de agregación federada:** Actualmente, únicamente se encuentra implementado [FedAvg](#). De los agregadores, depende en gran medida el rendimiento del modelo obtenido, resultando interesante disponer de un repertorio de algoritmos de agregación para aportar versatilidad a la librería. Por ejemplo, para la obtención de mejores modelos sobre distribuciones no IID, sería interesante disponer de algoritmos como *FedProx* [24].

Destáquese que la mayoría de algoritmos parten de [FedAvg](#), siendo posible realizar agregaciones parciales, no obstante, debiera tenerse especial cuidado con algoritmos de agregación que no sean compatibles con agregaciones parciales.

- **Inclusión de nodos en caliente:** Como se ha tratado en el apartado de implementación del 3º [sprint 5.4](#), se ha decidido bloquear la inclusión de nodos en caliente. A pesar de que el diseño sea apto, ante la posibilidad de inconsistencias en las votaciones se ha preferido evitar este problema.

La inclusión de un mecanismo que mitigue las inconsistencias que se puedan ocasionar en las votaciones, permitiría la conexión de nodos en caliente sin ningún riesgo aparente.

- **Optimización bucle *gossip* para el envío de modelos:** La implementación actual es correcta para el envío sin errores, no obstante, cuando algún nodo presenta problemas, se le envían modelos innecesaria y repetidamente. Con el fin de ahorrar ancho de banda, pudiera buscarse la inclusión de un mecanismo particular para los nodos que aparentan tener problemas de rendimiento o conexión. El objetivo será el de detener o bajar la frecuencia de envío a nodos que no estén reaccionando adecuadamente ante envío de modelos.
- **Secure Agregation:** La encriptación en las comunicaciones evita que se capture tráfico a usuarios ajenos a la red, sin embargo, nodos maliciosos podrán comprometer la información privada de otro nodo [48] en la recepción de modelos locales. Para solucionar este problema, sería interesante hacer uso de protocolos *Secure Multi-Party Computation*, concretamente, *Secure Agregation* [25] permitirá realizar agregaciones sin revelar cualquier información sobre los modelos locales de cada nodo.
- **Terminal iterativa:** Actualmente, si se desea controlar un nodo en tiempo real debe hacerse uso de la terminal iterativa de *python* o plantearse la creación de algún *script*. En consecuencia, buscando la comodidad del usuario final, pudiera plantearse la creación de un intérprete de comandos sobre el nodo, obviamente iterativo.
- **Tolerancia a recepción incompleta de mensajes:** De forma similar a lo que sucede con las recepciones incompletas en los fragmentos del modelo, pudiera suceder que si se envían una gran cantidad de mensajes sobre una misma conexión *TCP* algún mensaje se lea parcialmente provocando errores. Esto puede suceder si se sobrepasa el tamaño de recepción, o si en el instante de lectura se recibe un fragmento de un mensaje que aun se encuentra enviando. Este error es realmente improbable, pero ciertamente pudiera suceder. Su solución es trivial, detectar cuando un mensaje no se encuentra completo para leer el fragmento restante.
- **Pruebas con un elevado número de nodos:** A pesar de que *sockets* y *TCP* permitan despliegues y simulaciones locales, si se desea hacer pruebas a gran escala, *TCP* introduce un *overhead* innecesario en las simulaciones. Por este motivo, se plantea la creación de un protocolo de comunicación alternativo para simulaciones locales que busquen el estudio del sistema con un alto número de nodos.
- **Trabajo orientado al despliegue:** Como se ha podido observar, la mayor parte del esfuerzo ha sido dedicado a la parte de simulación, validando únicamente un igual funcionamiento en el despliegue. No obstante, si se quisiese usar la librería en un entorno de producción, debiera estudiarse y verificarse aún más rigurosamente el funcionamiento del sistema. Además, debieran mejorarse aspectos relativos al funcionamiento como

podrían ser:

- **Comparación de los ajustes del nodo:** Para tratar de prevenir errores, deberá validarse que los nodos estén operando con los mismos ajustes.
- **Asociar votos a rondas:** Con el fin de evitar inconsistencias provocadas por nodos con mal funcionamiento, debieran asociarse los votos al momento o ronda que fueron emitidos.

En cuanto a la seguridad en la red, deberán plantearse los siguientes mejoras:

- **Autenticación de ingreso a la red:** La autenticación de usuarios será necesaria para tener un control de los usuarios en la red. Sobre todo, teniendo en cuenta que una de las herramienta de privatización de servicios más usada, las *Virtual Private Network* (VPN), son ineficaces debido al uso del protocolo *Gossip*.
- **Ataques:** Deben tenerse en cuenta la ocurrencia de ataques. A pesar de que los ataques relativos a la privacidad estarán cubiertos por el protocolo *Secure Aggregation*, ataques que busquen la desestabilización de la red, o el "envenenamiento" deben de tener algún mecanismo de defensa.

Finalmente, destáquese que el código fuente de la librería estará publicado en *GitHub* bajo una licencia **GNU General Public License v3.0** [49]. Esta licencia es ampliamente usada en el mundo del software libre y código abierto, permitiendo a terceros usar, estudiar, compartir y modificar el software creado.

Apéndice

Lista de acrónimos

- AdaGrad** *Adaptive Gradient Algorithm.* 12
- ADAM** *Adaptive Moment Estimation.* 12
- AES** *Advanced Encryption Standard.* 19, 65
- API** *Application Programming Interface.* 21
- CESGA** *Centro de Supercomputación de Galicia.* 70
- CNN** *Convolutional Neural Network.* 68, 69, 96
- EMNIST** *Extended Modified National Institute of Standards and Technology dataset.* 68, 96
- FedAvg** *Federated Average.* 12, 13, 15, 59, 72, 73, 76
- FedMA** *Federated Learning with Matched Averaging.* 13
- FedSGD** *Federated Stochastic Gradient Descents.* 12
- FEMNIST** *Federated Extended Modified National Institute of Standards and Technology dataset.* 67, 68, 70
- FIFO** *First In, First Out .* 57
- FL** *Federated Learning.* 1, 3, 12, 14, 74
- FODA** *Fortalezas Oportunidades Debilidades Amenazas.* v, 22, 23
- IID** *Independent and Identically Distributed Random Variables.* 14, 67, 72, 73, 76
- KISS** *Keep It Simple, Stupid!.* 50

ML *Machine Learning*. 1–3, 5, 6, 11, 12, 20, 38, 48

MLP *MultiLayer Perceptron*. 67–69, 71, 96

MNIST *Modified National Institute of Standards and Technology dataset*. 67, 68, 96

p2p *Peer-to-peer*. 2, 3, 8, 9, 14–18, 30–32, 36, 38, 40, 44, 45, 48, 60, 74, 76, 93

q-FedAvg *q-Fair Federated Average*. 13

RR.NN.AA *Redes de Neuronas Artificiales*. 2, 6, 9, 11, 21, 67, 96

RSA *Rivest, Shamir y Adleman*. 19

SCAFFOLD *Stochastic Controlled Averaging for Federated Learning*. 13

SGD *Stochastic Gradient Descent*. 10, 12, 13

TCP *Transmission Control Protocol*. 21, 41, 43, 50, 51, 77

TFG *Trabajo de Fin de Grado*. 27

UTF-8 *8-bit Unicode Transformation Format*. 41, 65

VPN *Virtual Private Network*. 78

Glosario

Independent and Identically Distributed Random Variables En teoría de probabilidad y estadística, para que un conjunto de variables aleatorias se consideran independientes e idénticamente distribuidas (*iid*), cada variable aleatoria debe tener la misma distribución de probabilidad, siendo todas mutuamente independientes.

Centrándose en la distribución de muestras entre los diferentes clientes de un entorno federado, una distribución de este tipo causará que los conjuntos de datos de cada cliente sean similares, disponiendo cada uno de un conjunto representativo del problema a resolver. . 14, 80

Secure Multi-Party Computation *Secure Multi-Party Computation* o cómputo de preservación de la privacidad es un subcampo de la criptografía que trata que diferentes entidades colaboren con su información preservando su privacidad y confidencialidad. Por lo tanto, el objetivo será el de computar sobre información privada de diferentes entidades, revelando únicamente el resultado y no la información aportada por cada participante.. 77

batch A la hora de ejecutar un *epoch*, la cuantía total de muestras no serán introducidas a la vez en algoritmo de aprendizaje automático. Las muestras serán introducidas iterativamente, siendo el *batch* el número de ejemplos que se pasan al algoritmo en cada iteración del aprendizaje. . vi, 10, 12, 13, 69

epoch En el contexto del aprendizaje automático, un *epoch* puede describir como un ciclo completo a través de todo el conjunto de datos de entrenamiento para mejorar el modelo. Los *epoches* indican la cantidad de iteraciones que el algoritmo de aprendizaje automático ejecutará durante ese entrenamiento.

En el contexto de este proyecto, los *epoches* serán designados para indicar cantidad de cómputo local en los nodos.. 10, 12, 13, 37, 45, 49, 54, 56, 64, 68, 70, 71, 82

heartbeat Un *heartbeat* es una señal periódica enviada por un componente *hardware* o *soft-*

ware que indica un funcionamiento normal, sin errores. El mecanismo *heartbeat* es una de las técnicas más comunes en los sistemas para proporcionar alta disponibilidad y tolerancia a fallos. [41](#), [43](#), [61](#), [62](#)

sprint Es cada uno de los ciclos o iteraciones que se van a tener dentro de un proyecto con una metodología *SCRUM*. [vi](#), [4](#), [26–34](#), [36](#), [38](#), [43](#), [45](#), [47](#), [48](#), [50](#), [52](#), [53](#), [55–60](#), [62](#), [63](#), [65](#), [67](#), [76](#), [93](#), [94](#)

unicode *Unicode* es un estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de numerosos idiomas y disciplinas técnicas, además de textos clásicos de lenguas muertas. El término Unicode proviene de los tres objetivos perseguidos: universalidad, uniformidad y unicidad. [41](#)

diferenciación automática En matemáticas y álgebra informática, la diferenciación automática, también llamada diferenciación algorítmica, es un conjunto de técnicas para evaluar la derivada de funciones numéricas especificadas en un programa informático. Centrándose en el sector de la inteligencia artificial, estas técnicas serán empleadas para realizar el descenso del gradiente. . [21](#)

dispositivos edge Los dispositivo *edge* son hardware físico en ubicaciones remotas del perímetro de la red. Estos dispositivos disponen de suficiente memoria, potencia de procesamiento y recursos informáticos para recopilar datos y procesarlos casi en tiempo real, con la ayuda limitada de otras partes de la red.
En definitiva, un dispositivo *edge* es donde se recopilan y procesan los datos. [2](#), [74](#)

gradiente En matemáticas, el gradiente es una generalización de la derivada. Mientras que una derivada se puede definir solo en funciones de una sola variable, devolviendo un escalar, para funciones de varias variables, se empleará el gradiente, devolviendo un valor vectorial.

El gradiente representa la pendiente de la recta tangente a la gráfica de una función. Es decir, el gradiente tomará valores más altos en puntos de la gráfica con mayor incremento. . [10](#), [12](#)

Bibliografía

- [1] B. McMahan and D. Ramage, “Federated learning: Collaborative machine learning without centralized training data,” 2017. [En línea]. Disponible en: <https://ai.googleblog.com/2017/04/federated-learning-collaborative.html>
- [2] R. M. Parizi, A. Dehghantanha, Q. Zhang, and K. Franke, “Decentralized federated learning: An introduction and the road ahead,” in *Annual Hawaii International Conference on System Sciences*. IEEE Computer Society, 2021.
- [3] M. Jelasity, “Gossip,” in *Self-organising software*. Springer, 2011, pp. 139–162.
- [4] “Flower: A friendly federated learning framework.” [En línea]. Disponible en: <https://flower.dev/>
- [5] G. A. Reina, A. Gruzdev, P. Foley, O. Perepelkina, M. Sharma, I. Davidyuk, I. Trushkin, M. Radionov, A. Mokrov, D. Agapov, J. Martin, B. Edwards, M. J. Sheller, S. Pati, P. N. Moorthy, S.-h. Wang, P. Shah, and S. Bakas, “Openfl: An open-source framework for federated learning,” 2021. [En línea]. Disponible en: <https://arxiv.org/abs/2105.06413>
- [6] “Nvidia flare.” [En línea]. Disponible en: <https://developer.nvidia.com/flare>
- [7] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, “A generic framework for privacy preserving deep learning,” 2018. [En línea]. Disponible en: <https://arxiv.org/abs/1811.04017>
- [8] “Tensorflow federated.” [En línea]. Disponible en: <https://www.tensorflow.org/federated>
- [9] “Fate.” [En línea]. Disponible en: <https://fate.fedai.org/>
- [10] “Paddlefl.” [En línea]. Disponible en: <https://github.com/PaddlePaddle/PaddleFL>

- [11] T. W. H. W. Yanjun Ma, Dianhai Yu, “Paddlepaddle: An open-source deep learning platform from industrial practice,” *Frontiers of Data and Computing*, vol. 1, no. 1, p. 105, 2019. [En línea]. Disponible en: http://www.jfdc.cn/cn/EN/abstract/article_2.shtml
- [12] R. F. López and J. M. F. Fernández, *Las redes neuronales artificiales*. Netbiblo, 2008.
- [13] S. Ruder, “An overview of gradient descent optimization algorithms,” 2016. [En línea]. Disponible en: <https://arxiv.org/abs/1609.04747>
- [14] D. E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin, “Backpropagation: The basic theory,” *Backpropagation: Theory, architectures and applications*, pp. 1–34, 1995.
- [15] J. Kiefer and J. Wolfowitz, “Stochastic estimation of the maximum of a regression function,” *The Annals of Mathematical Statistics*, pp. 462–466, 1952.
- [16] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, “Communication-efficient learning of deep networks from decentralized data,” 2016. [En línea]. Disponible en: <https://arxiv.org/abs/1602.05629>
- [17] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, jan 2015. [En línea]. Disponible en: <https://doi.org/10.1016%2Fj.neunet.2014.09.003>
- [18] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014. [En línea]. Disponible en: <https://arxiv.org/abs/1412.6980>
- [19] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization.” *Journal of machine learning research*, vol. 12, no. 7, 2011.
- [20] H. Wang, M. Yurochkin, Y. Sun, D. S. Papailiopoulos, and Y. Khazaeni, “Federated learning with matched averaging,” *CoRR*, vol. abs/2002.06440, 2020. [En línea]. Disponible en: <https://arxiv.org/abs/2002.06440>
- [21] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, “SCAFFOLD: stochastic controlled averaging for on-device federated learning,” *CoRR*, vol. abs/1910.06378, 2019. [En línea]. Disponible en: <http://arxiv.org/abs/1910.06378>
- [22] T. Li, M. Sanjabi, and V. Smith, “Fair resource allocation in federated learning,” *CoRR*, vol. abs/1905.10497, 2019. [En línea]. Disponible en: <http://arxiv.org/abs/1905.10497>
- [23] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, “Adaptive federated optimization,” 2020. [En línea]. Disponible en: <https://arxiv.org/abs/2003.00295>

- [24] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, “On the convergence of federated optimization in heterogeneous networks,” *CoRR*, vol. abs/1812.06127, 2018. [En línea]. Disponible en: <http://arxiv.org/abs/1812.06127>
- [25] K. A. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for federated learning on user-held data,” in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016. [En línea]. Disponible en: <https://arxiv.org/abs/1611.04482>
- [26] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, “A survey on federated learning systems: Vision, hype and reality for data privacy and protection,” *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021. [En línea]. Disponible en: <https://doi.org/10.1109%2Ftkde.2021.3124599>
- [27] A. Lalitha, O. C. Kilinc, T. Javidi, and F. Koushanfar, “Peer-to-peer federated learning on graphs,” 2019. [En línea]. Disponible en: <https://arxiv.org/abs/1901.11173>
- [28] Z. Wang and Q. Hu, “Blockchain-based federated learning: A comprehensive survey,” 2021. [En línea]. Disponible en: <https://arxiv.org/abs/2110.02182>
- [29] T. Wink and Z. Nochta, “An approach for peer-to-peer federated learning,” in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2021, pp. 150–157.
- [30] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, p. 120–126, feb 1978. [En línea]. Disponible en: <https://doi.org/10.1145/359340.359342>
- [31] F. I. Processing and A. The, “Announcing the advanced encryption standard (aes).”
- [32] E. Barker, “Guideline for using cryptographic standards in the federal government: Cryptographic mechanisms,” 2016-08-22 2016.
- [33] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.
- [34] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous

- systems,” 2015, software available from tensorflow.org. [En línea]. Disponible en: <https://www.tensorflow.org/>
- [35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems* 32. Curran Associates, Inc., 2019, pp. 8024–8035. [En línea]. Disponible en: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- [36] C. E. Perez, “Pytorch, dynamic computational graphs and modular deep learning,” Mar 2019. [En línea]. Disponible en: <https://medium.com/intuitionmachine/pytorch-dynamic-computational-graphs-and-modular-deep-learning-7e7f89f18d1>
- [37] “Fastapi.” [En línea]. Disponible en: <https://fastapi.tiangolo.com/>
- [38] L. M. Crespo Martínez and F. A. Candelas-Herías, *Introducción a TCP/IP: sistemas de transporte de datos*. Universidad de Alicante, 1998.
- [39] S. Ghazinoory, M. Abdi, and M. Azadegan-Mehr, “Swot methodology: a state-of-the-art review for the past, a framework for the future,” *Journal of business economics and management*, vol. 12, no. 1, pp. 24–48, 2011.
- [40] J. P. Alexander Menzinsky, Gertrudis López, “Scrum manager,” *Iubaris Info 4 Media SL*, 2016.
- [41] “Python enhancement proposals (peps),” Tech. Rep. [En línea]. Disponible en: <https://peps.python.org/>
- [42] D. Holth, “The wheel binary package format 1.0,” PEP 427, 2012. [En línea]. Disponible en: <https://peps.python.org/pep-0427/>
- [43] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [En línea]. Disponible en: <http://yann.lecun.com/exdb/mnist/>
- [44] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, “Leaf: A benchmark for federated settings,” 2018. [En línea]. Disponible en: <https://arxiv.org/abs/1812.01097>
- [45] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, “EMNIST: an extension of MNIST to handwritten letters,” *CoRR*, vol. abs/1702.05373, 2017. [En línea]. Disponible en: <http://arxiv.org/abs/1702.05373>

- [46] A. Botalb, M. Moinuddin, U. M. Al-Saggaf, and S. S. A. Ali, “Contrasting convolutional neural network (cnn) with multi-layer perceptron (mlp) for big data analysis,” in *2018 International Conference on Intelligent and Advanced System (ICIAS)*, 2018, pp. 1–5.
- [47] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated learning with non-iid data,” *CoRR*, vol. abs/1806.00582, 2018. [En línea]. Disponible en: <http://arxiv.org/abs/1806.00582>
- [48] R. Shokri, M. Stronati, and V. Shmatikov, “Membership inference attacks against machine learning models,” *CoRR*, vol. abs/1610.05820, 2016. [En línea]. Disponible en: <http://arxiv.org/abs/1610.05820>
- [49] “Gnu general public license,” Free Software Foundation. [En línea]. Disponible en: <http://www.gnu.org/licenses/gpl.html>

Planificación

EN los siguientes apartados se detallará la planificación inicial tentativa y el seguimiento de la misma.

A.1 Diagrama de Gantt tentativo

El diagrama de Gantt de las figuras A.1, A.2 y A.3 muestra la planificación inicial tentativa. Para más información sobre los costes estimados o la planificación en sí, véase el apartado 4.4.

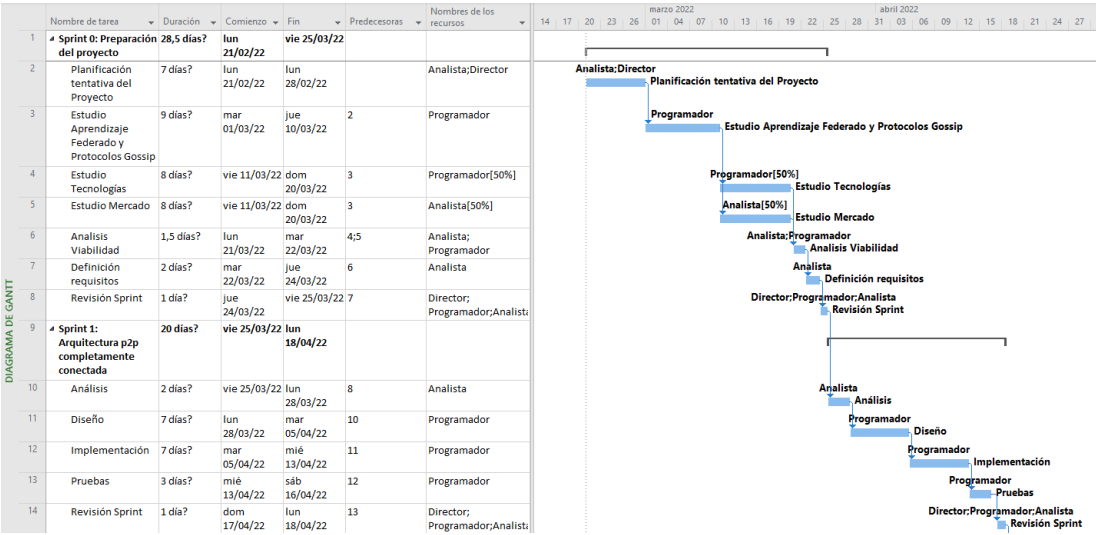


Figura A.1: Diagrama de Gantt tentativo.
Fragmento 1

APÉNDICE A. PLANIFICACIÓN

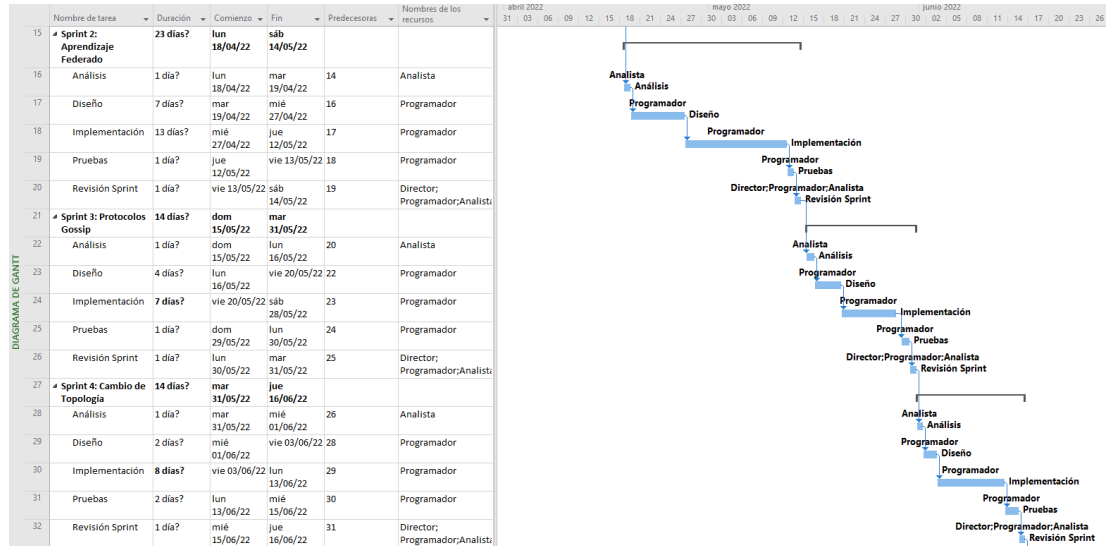


Figura A.2: Diagrama de *Gantt* tentativo.
Fragmento 2

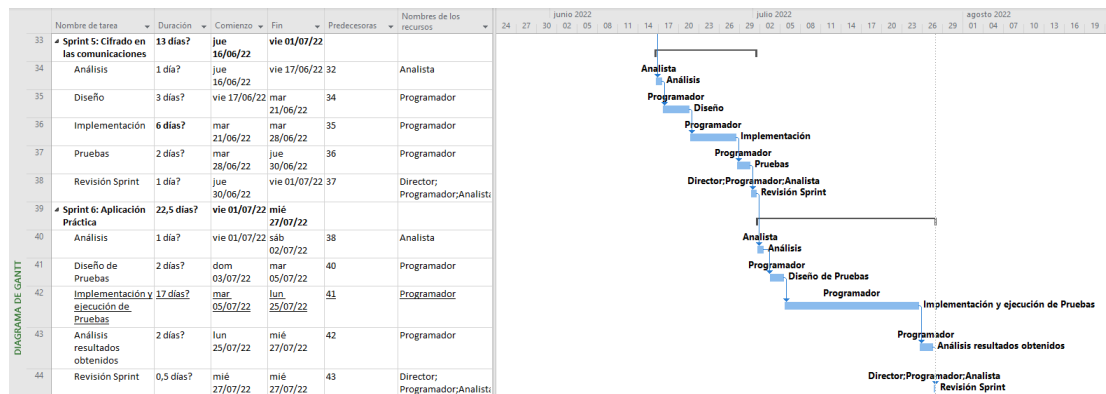


Figura A.3: Diagrama de *Gantt* tentativo.
Fragmento 3

A.2 Diagrama de *Gantt* de seguimiento

En las figuras A.4, A.5 y A.6 se muestra el diagrama de *Gantt* de seguimiento, este detalla los desvíos frente a la planificación tentativa. Nuevamente, para más información acerca del seguimiento y sus costes puede consultarse el apartado 4.5.

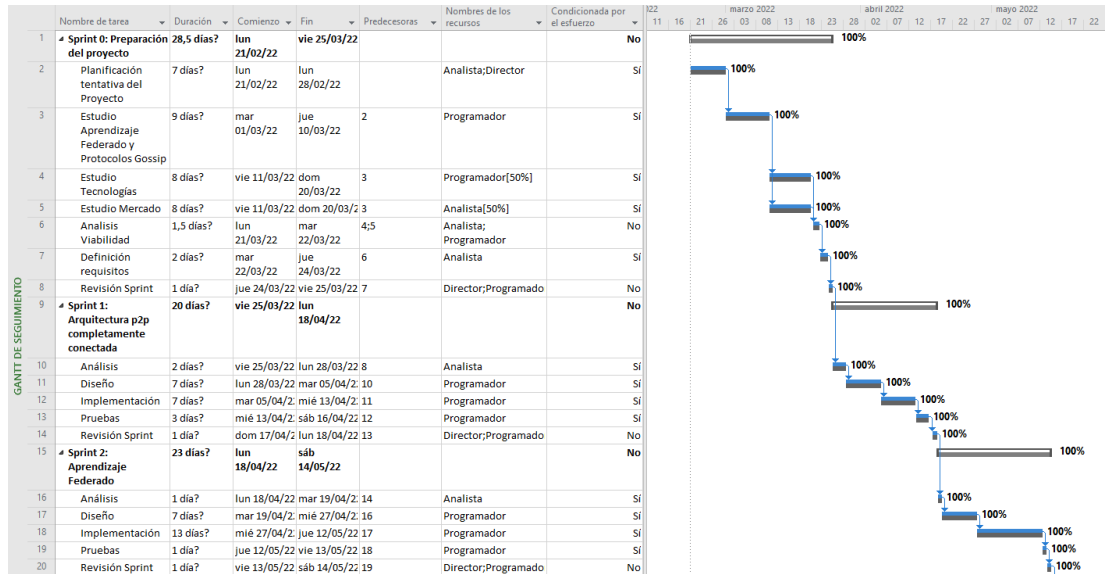


Figura A.4: Diagrama de *Gantt* de seguimiento.
Fragmento 1

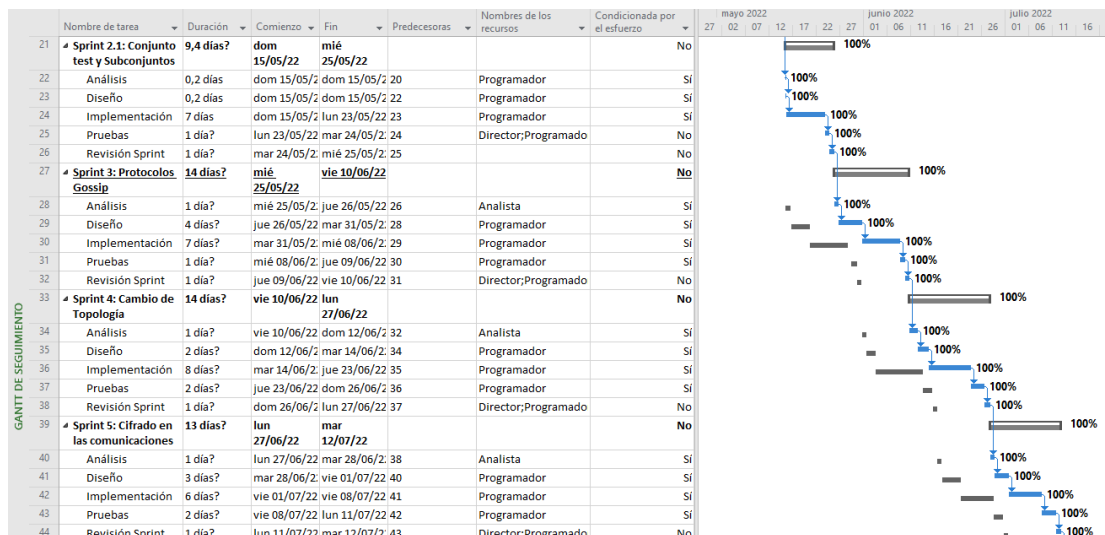


Figura A.5: Diagrama de *Gantt* de seguimiento.
Fragmento 2

APÉNDICE A. PLANIFICACIÓN

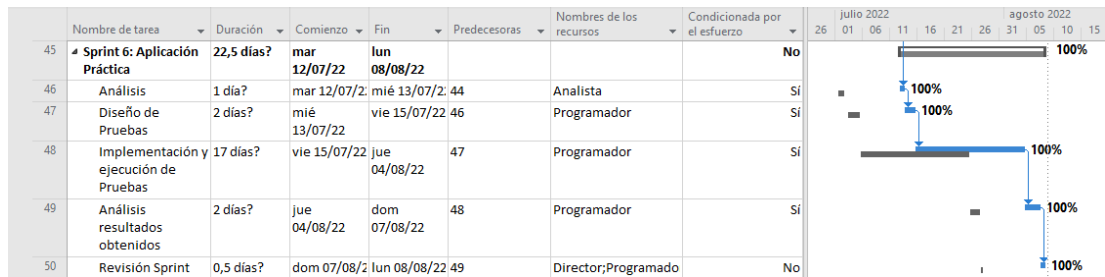


Figura A.6: Diagrama de *Gantt* de seguimiento.
Fragmento 3

Diagramas de secuencia

A continuación, se incluirán los diagramas de secuencia que detallan el funcionamiento de los componentes más importantes de la librería.

Primeramente, el diagrama de secuencia de la figura B.1 muestra el funcionamiento de una de las partes esenciales del sistema p2p, la conexión entre nodos. Esta funcionalidad ha sido implementada en el *sprint* 1, pudiendo encontrar una explicación detallada de dicho diagrama en el apartado 5.2.2.

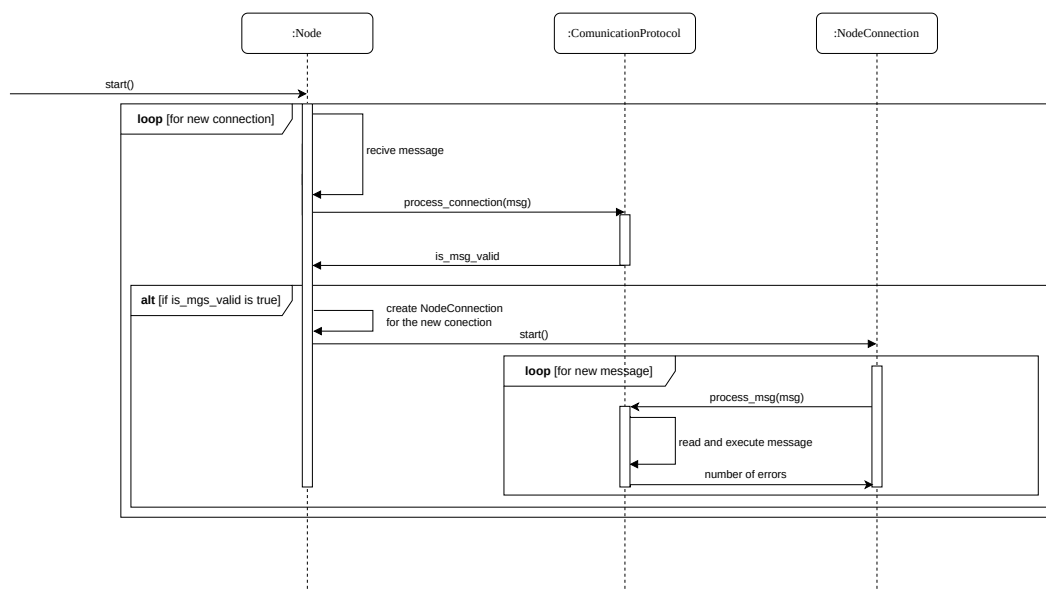


Figura B.1: Diagrama de secuencia del proceso de conexión entre nodos.

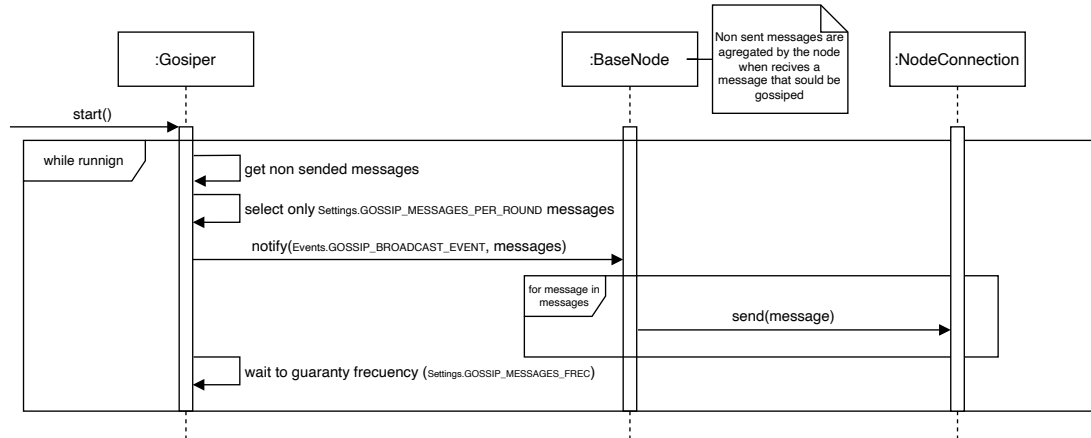


Figura B.4: Diagrama de secuencia para la difusión de mensajes usando un protocolo *Gossip*.

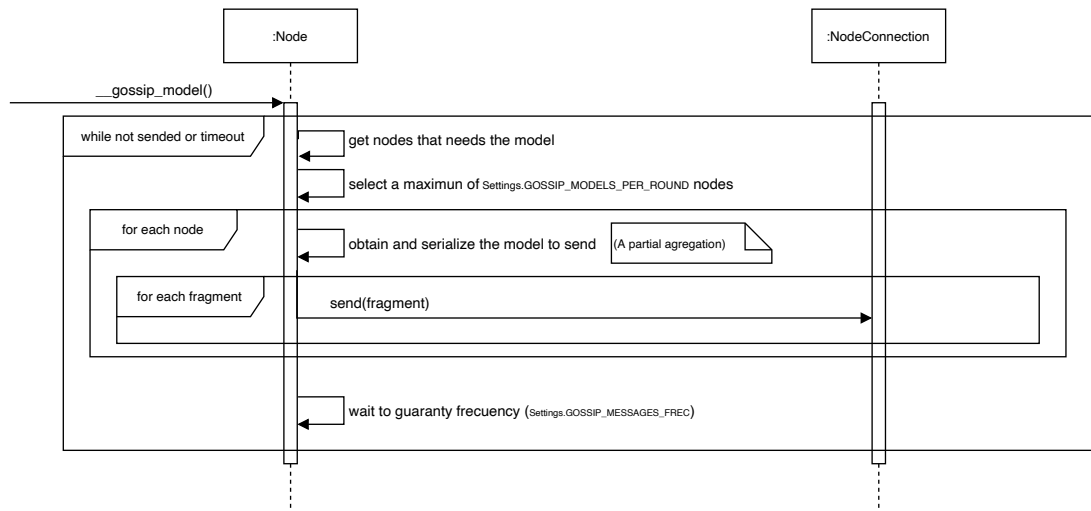


Figura B.5: Diagrama de secuencia para la difusión de modelos usando un protocolo *Gossip*.

Arquitecturas de las *RR.NN.AA* empleadas

ESTE apéndice se destinará a explicar los modelos de *RR.NN.AA* empleados en el proyecto para la resolución de *MNIST* y *EMNIST*.

El primero y el más sencillo es el que se detalla en la figura C.1. Este modelo es un *MLP*, y consta de 4 capas completamente conectadas de forma contigua. La primera, o capa de entrada, tendrá una neurona por píxel en la imagen, mientras que la última capa tendrá tantas neuronas como tipos de muestra hay en el problema a resolver, es decir, 10 neuronas que representan los números del 0 al 9. Las dos capas ocultas de 256 y 128 neuronas son empleadas para que el modelo aprenda las características que diferencian las clases existentes.

En cuanto al segundo modelo, la arquitectura concreta de la *CNN* empleada se encuentra detallada en la figura C.2. A grosso modo, las capas convolucionales extraerán características de las imágenes, mientras que las capas *max pool* reducirán el tamaño de los mapas de características. El objetivo será disponer de muchos mapas de características de baja resolución que serán introducidos a un *MLP*, similar al modelo tratado previamente. De este modo, la concatenación de convoluciones y capas *max pool* permitirán obtener las características más salientables de la imagen, facilitando notoriamente el trabajo del clasificador (un *MLP* de una sola capa oculta).

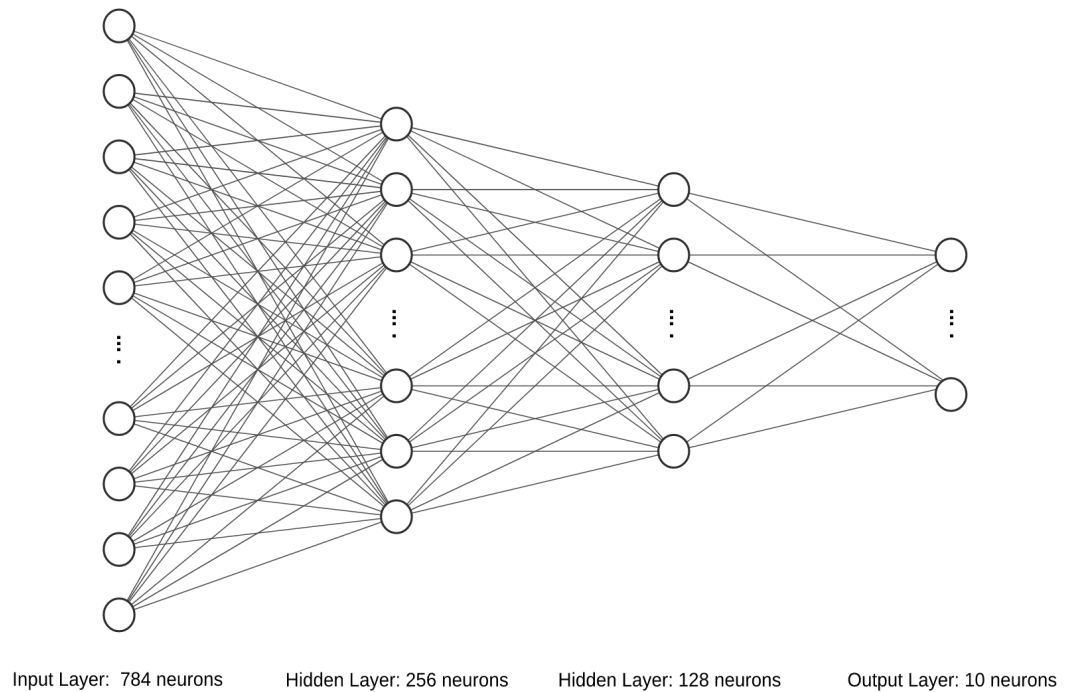


Figura C.1: Arquitectura concreta del perceptrón multicapa empleado.

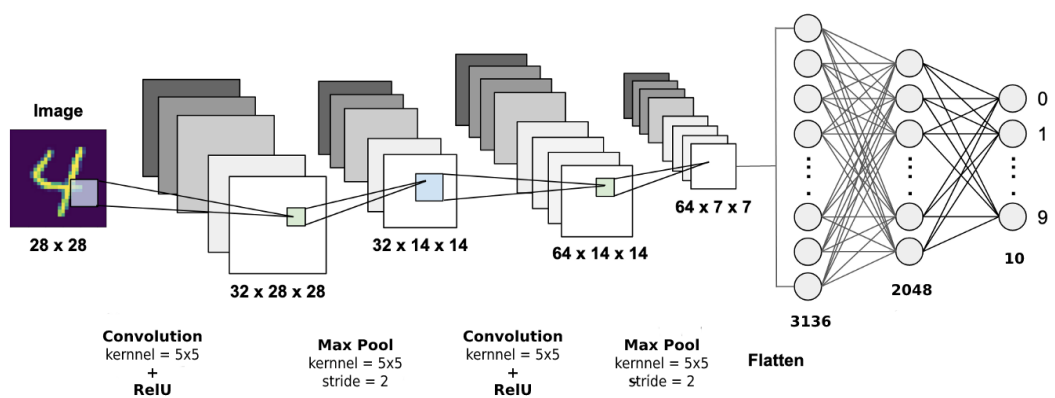


Figura C.2: Arquitectura concreta de la red neuronal convolucional empleada.