DEPARTMENT OF
COMPUTER SCIENCE

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

RAFAEL MARTINS SANTOS COSTA

BSc in Computer Science and Engineering

TOWARDS DECENTRALIZED SPLIT LEARNING

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon

Draft: *February 9, 2025*

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

TOWARDS DECENTRALIZED SPLIT LEARNING

RAFAEL MARTINS SANTOS COSTA

BSc in Computer Science and Engineering

Adviser: João Leitão
Associate Professor, NOVA University Lisbon

Co-adviser: Dimitra Tsigkari
Research Scientist, Telefónica Research (Spain)

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon

Draft: *February 9, 2025*

Abstract

In recent years, the rise of data-driven applications has driven the development of decentralized learning frameworks such as Federated Learning (FL) and Split Learning (SL), designed to improve data privacy and distribute computational load across multiple devices. However, current FL systems still rely heavily on centralized aggregators, which imposes privacy and efficiency limitations due to the centralized orchestration.

This thesis proposes a fully decentralized platform that combines FL and SL methodologies to overcome these challenges. Our approach aims at design a self-managed system for collaborative decentralized machine learning, minimizing human intervention and explicitly dynamic delegation protocols and adaptive role assignment mechanisms. Participants can collaboratively train machine learning models in a secure environment, with devices autonomously handling changes in the network, such as device failures and connectivity changes.

Our research aims therefore, at bridging a gap between current architectures for Federated and Split Learning and the modern design and operation of decentralized systems, providing to the latter the benefits of central points of coordination while providing self-management and fault-tolerance.

Keywords: decentralized platform, FL, SL, collaboration, adaptability

Resumo

Nos últimos anos, o aumento de aplicações baseadas em dados impulsionou o desenvolvimento de estruturas de aprendizagem descentralizadas, como o Federated Learning (FL) e o Split Learning (SL), concebidas para melhorar a privacidade dos dados e distribuir a carga computacional por vários dispositivos. No entanto, os sistemas FL atuais ainda dependem fortemente de agregadores centralizados, o que impõe limitações de privacidade e eficiência devido à orquestração centralizada.

Esta tese propõe uma plataforma totalmente descentralizada que combina as metodologias FL e SL para superar estes desafios. A nossa abordagem visa projetar um sistema autogerido para aprendizagem automática descentralizada e colaborativa, minimizando a intervenção humana e explicitamente protocolos de delegação dinâmicos e mecanismos de atribuição de funções adaptáveis. Os participantes podem treinar modelos de aprendizagem automática de forma colaborativa num ambiente seguro, com os dispositivos a lidarem autonomamente com alterações na rede, como falhas de dispositivos e alterações de conectividade.

A nossa investigação visa, portanto, preencher a lacuna entre as atuais arquiteturas de Federated e Split Learning e o design e operação modernos de sistemas descentralizados, proporcionando a estes últimos os benefícios de pontos centrais de coordenação, ao mesmo tempo que proporciona autogestão e tolerância a falhas.

Palavras-chave: plataforma totalmente descentralizada, FL, SL, colaboração, adaptabilidade

Contents

List of Figures

List of Tables

Acronyms

CFL Centralized Federated Learning *(pp. 11, 12, 17)*

DFL Decentralized Federated Learning *(pp. 12–18)*

FL Federated Learning *(pp. 2, 5, 11, 14, 16, 18, 22, 23)*

IoT Internet of Things *(pp. 15, 19)*

MI Model Inversion *(p. 15)*
ML Machine Learning *(p. 25)*

NN Neural Network *(p. 18)*

P2P Peer-To-Peer *(pp. 5–7, 9, 14)*
PSL Parallel Split Learning *(p. 21)*

SFL Split Federated Learning *(pp. 22–25)*
SL Split Learning *(pp. iv, 5, 18, 20–25)*

1

Introduction

In recent years, the exponential growth of data generated through distributed devices and platforms has motivated the development and evolution of collaborative learning methods such as Federated Learning (FL) [3]. Federated learning allows a network of participants to train a machine learning model without centralizing data, addressing critical issues related to data privacy and security. However, FL still faces inherent limitations [30], particularly in its dependence on a central aggregator and the computational resources required on individual devices to train complex models. Furthermore, reliance on centralized platforms raises significant privacy concerns as they can introduce vulnerabilities or potential access points to sensitive data. Alternatively, Split Learning (SL) [63, 43] has emerged as a promising approach within distributed learning paradigms, allowing computational load to be shared among participants, which can help alleviate resource constraints while data used in the training process remains local at each device.

In the absence of an effective mechanism to manage and reconfigure these processes without human intervention, particularly in terms of fault tolerance, the objective is to design, implement, and evaluate a fully self-managed and decentralized platform capable of leveraging SL techniques. This platform will allow participants to configure and reconfigure independently, even in the presence of dynamic network conditions. The exchange of information and the establishment of a sense of community among the participants will be facilitated by Babel [15], a tool that will be introduced in later sections of the next chapter.

This decentralized platform seeks to minimize human intervention and ensure robustness by allowing participants to help each other in the training process, thus addressing privacy concerns that arise from centralized architectures.

During the Federated Learning process, at the conclusion of each round, the aggregator awaits the completion of activity and the transmission of data by all clients. Following this, the data is aggregated and advanced to the next round. In the scenario of a client experiencing delays in the training process, the round's conclusion is also postponed. The integration of Split Learning enables clients with computational resource needs to share the load with other clients or more capable machines, thereby avoiding delays in the time

per training round. Our primary focus is not on optimizing neural network training by improving Federated Learning and Split Learning, but rather on optimizing the processing workflow between clients to address the current challenges of fault tolerance and work distribution.

This research will first address the current state of the art in decentralized learning systems and evaluate existing methods, followed by the design of a novel decentralized SL architecture. Through this approach, we aim to improve the efficiency of decentralized federated systems [47], ultimately contributing to a more robust and flexible infrastructure for distributed learning, that requires significantly less intervention during its operation.

1.1 Objective

The objective of this thesis is to develop a fully decentralized federated learning platform that integrates key principles of Split Learning (SL) to enhance data ownership, efficiency, and robustness. By moving away from depending on a central aggregator, this work aims at building a fault-tolerant and autonomous framework where participants can collaborate without compromising data privacy (i.e., without sending their own local data to a central location or other participants).

To achieve this, we will focus on the following key areas:

1. **Decentralized Federated Learning:**

   Design and implement a federated learning [47] system that operates in a decentralized manner, reducing reliance on central servers and enabling greater autonomy for participants. (This point will be studied and developed in line with another student's master's dissertation.)

2. **Split Learning:**

   Incorporate split learning [43] techniques to distribute unbalanced computational loads among participants of a Federated Learning (FL) framework, allowing resource-constrained devices to participate effectively in the model training without overextending their capabilities, or imposing addicional delays on the overall program of the training.

3. **Delegation Protocols:**

   We plan to develop delegation protocols that allow to assign an idle node with enough processing capabilities available as a helper to train a model. Split Learning, to be effective, needs to identify participants with different loads (or devices) that are not effectively participating on the FL training process, and coordinate - at runtime - on sharing the processing of more overloaded devices We encapsulate these activities as a new abstraction to what we call Delegation Protocol.

4. **Helper Membership Management:**

   Establish mechanisms for resource/load tracking mechanisms at membership management layers, enabling participants to support one another in the training process and foster a robust, collaborative learning environment, that can easily self-adapt to natural dynamics of the system (e.g. resource availability or churn)

1.2 Expected contributions and Results

The main contributions and results that are expected from this work are:

1. **The Design, Implementation, and Evaluation of a Novel Split Learning Scheme:**

   A novel platform based on Babel [15] that offers support for both split learning and federated learning on servers, personal computers and/or Android devices.

2. **Demonstration with realistic applications scenarios:**

   Two demonstrators will showcase the practical applicability of the developed solution and technology. These demonstrators will represent real-world scenarios, such as Image classification [26] and Recommendation systems [9].

3. **Experimental evaluation and comparison with existing solutions:**

   The evaluation process will utilize specific metrics that will be discussed in a later section of this report. Additionally, a comparative analysis will be conducted with relevant solutions found in the literature.

1.3 Research Context

The work discussed in this document is conducted in the context of the European TaRDIS Project [62], this thesis aims to explore a new decentralized approach to implementing Split Learning within a fully decentralized Federated Learning framework. This work benefits from a collaboration with Telefónica Research, which has interest in the field of decentralized federated/split learning.

1.4 Document Structure (Roadmap)

The rest of the document is organized as follows:

1. Chapter 2 (Related work)

   This chapter provides a review of the foundational concepts of Federated Learning (FL) and Split Learning (SL), examining their architectures, strengths, and limitations. It also explores existing applications, delegation protocols for task distribution, and strategies for managing helper membership.

2. Chapter 3 (Planning)

   This chapter briefly presents a preliminary prototype that was developed already in the context of this work, details the strategy for explaining different alternatives to design delegation protocol and discusses the evaluation plan and the scheduling of future work.

2

Related Work

In this chapter, we discuss the limitations and negative aspects of Centralized Federated Learning Applications, highlighting the need for Decentralized Federated Learning Applications. We also explain how are we expecting to deal with the inherent challenges associates with the design and operation of solutions to support delegating work to helpers as to improve systems performance while preventing adverse effects on the functionality of the devices (the network nodes), due to network our device failures. However, we start by introducing fundamental concepts of Federated Learning (FL) and Split Learning (SL) to provide a clearer understanding of our proposal.

2.1 Decentralized Systems

Decentralized systems are relevant in the domain of distributed computing, enabling tasks to be executed without resorting to a central authority. This architectural approach enhances scalability, fault tolerance, and resilience by distributing responsibilities across multiple nodes, allowing them to function autonomously while maintaining coordination [74]. Such systems are particularly valuable in environments that require adaptive and robust communication between a multitude of devices, including Peer-To-Peer (P2P) networks [8], blockchain technologies [78, 38], and decentralized applications as the one introduced on N. Li et. al. paper [39], where they illustrate their work on decentralized self-adaptive systems using a case study [57] set in disaster-stricken cities where communication infrastructure is unavailable. Autonomous UAVs ensure district security by combining global knowledge, such as city maps, with local, dynamically shared information. This highlights the adaptability and resilience of decentralized architectures, where coordination emerges from localized interactions to address complex scenarios effectively.

In decentralized systems, participants can dynamically join, leave, or fail without significantly disrupting overall functionality. This is made possible by the use of adaptive protocols that ensure that the system remains operational. Data sharing, connectivity maintenance, and task distribution are facilitated by leveraging on different distributed algorithms and protocols. For example, membership protocols guarantee that nodes are

aware of each other's presence, whereas peer-to-peer communication abstraction facilitates direct interaction between participants. Membership protocols serve to ensure that nodes are aware of each other's presence, while P2P [2] communication models facilitate direct interaction between nodes.

Decentralized systems are a foundational element of modern technologies, including blockchain and decentralized P2P file-sharing platforms. These systems demonstrate the potential of decentralization to address scalability and reliability challenges in diverse domains including Finance [60], Data Sharing [35], Telecommunications [55], among others.

2.1.1 Membership Protocols

The establishment of membership protocols is of particular significance in the context of decentralized swarm systems, including federated learning and IoT networks, wherein nodes interact in a dynamic manner. These protocols delineate the nodes that constitute the system, thereby enabling each node to identify active participants. This is essential for optimizing resource allocation and avoiding communication failures. Furthermore, these protocols facilitate coordination and collaboration among nodes, enhancing their resilience.

2.1.2 Decentralized Membership Protocols

In larger systems (such as in swarm computing environments [61] or global p2p systems [8]), keeping track of every single node isn't practical, because of the dynamic node changes, high network overhead, resource constraints among others. Decentralized Membership Protocols simplify this by letting each node manage enough information to stay connected without needing to know all nodes. For example, the HyParView [36] protocol organizes nodes into two lists: an active view, which contains frequently updated links to a few critical nodes, and a passive view, a backup list of other nodes. The active view keeps each node actively connected, while the passive view acts as a fallback in case of node failure. This two-layered approach makes HyParView [36] resilient to failures or churn [24] candidates, and allows quick recovery if nodes unexpectedly leaves the system or fault. The protocol is often used within frameworks like Babel [15], which supports secure and automatic membership abbstractions to simplify building decentralized systems.

2.1.3 P2P Architecture

Peer-to-peer (P2P) [2, 8] communication is a decentralized system model in which each peer, or node, has equal capabilities and each participant has the same responsibilities. P2P allows direct communication between nodes without the need for a central authority to mediate the interaction, as in client-server architectures, as shown in Figure 2.1.

Nodes can connect and disconnect dynamically leading to an architecture that adapts as nodes join or leave the network (a process sometimes called churn [24]). These types of systems can scale up or down based on the number of peers. Increasing the number of nodes improves the network's capacity and resilience and decreasing them has limited impact, assuming enough nodes remain active to support the activity of the system. Each node can offer resources, such as processing power or storage capacity, contributing to the overall capacity of the network, this makes P2P an efficient approach for applications requiring vast amounts of computational resources.

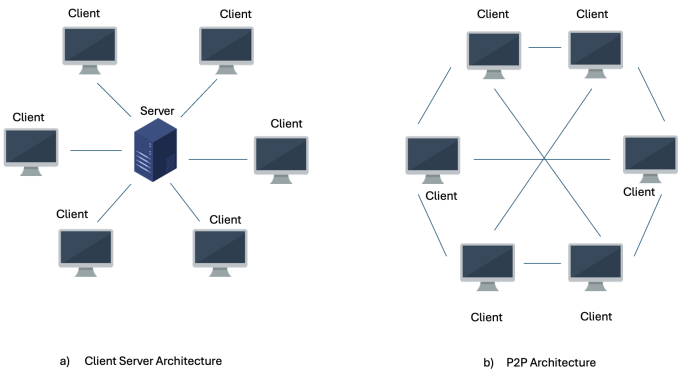


Figure 2.1: Illustration of client-server and P2P architectures

2.1.3.1 Broadcast

The term broadcast [33] refers to sending messages, or data, to all peers within the system, since P2P systems lack a centralized server, broadcasting relies on each node to assist in the process of propagating the message to others. This method allows information to spread across the network efficiently, but it also presents challenges related to network congestion, duplication, and ensuring all peers receive the broadcasted message.

Broadcasting can be implemented using various techniques:

Flooding [61, 45]: In flooding, a peer sends the message to all its directly connected peers (i.e. neighbours - provided by a membership protocol), who then forward it to their connected peers, and so on, until the message reaches every peer in the network. This method can lead to message duplication and network congestion, as each peer may receive multiple copies of the same message.

Gossip Protocols [61, 6]: Each node randomly selects a few nodes to send the message, and those nodes, select randomly another subset of their connections, and so on [6]. This method ensures that most or all nodes receive the message without overloading the network. In fact, careful configuration of fanout (the number of peers to which a node forwards a message) allows you to achieve full delivery with a high probability [4].

Multicasting [21]: In structured P2P networks, multicasting can be more efficient than broadcasting. Instead of randomly sending messages to every node, by sending

messages only to a specific group/subset of nodes who are most relevant to receive it. This is done by organizing nodes into groups or clusters and sending messages within those or between them.

2.1.4 Development Frameworks for Distributed and Decentralized systems

The design and implementation of distributed systems are inherently complex due to the necessity of addressing several challenges such as fault tolerance, concurrency, and efficient communication. To address these challenges, frameworks have been developed that abstract low-level details and simplify the creation of distributed systems. These tools allow developers to focus on the core logic of protocols and applications rather than being overwhelmed by the intricacies of fault tolerance, communication, and concurrency. From educational platforms that simulate algorithmic behavior to robust solutions for deployment in real-world scenarios, these frameworks serve as critical enablers. They not only accelerate development but also provide modularity, scalability, and adaptability in system design, aligning with the diverse needs of modern distributed applications.

This section presents a summarized review of several prominent decentralized development frameworks, namely Appia [49], Cactus [48], Yggdrasil [11], ViSiDia [1], and Babel [15], identifying features, strengths, and limitations of these frameworks in supporting the development of dependable distributed systems.

Appia: [49] A Java-based toolkit that take advantage of Java inheritance for creating custom protocols. It introduces the concept of channels and sessions as abstractions, which are used to bind protocols into services. However, this approach requires developers to manually stack protocols, which restricts the flexibility of the interaction. Appia uses a single execution thread for all protocols, which can cause significant performance bottlenecks.

Cactus: [48] A C++ framework designed to achieve optimal performance and expressiveness. It uses meta-protocols composed of smaller protocols that execute asynchronously, offering modularity. However, it does not manage concurrency, leaving developers responsible for addressing these challenges.

Yggdrasil: [11] A lightweight and efficient C framework, designed for use in both wireless ad hoc and traditional wired IP networks. The framework is designed to facilitate the processing of four distinct types of events: network messages, timed events, direct interactions within the protocol, and indirect interactions. It supports concurrent protocol execution while abstracting concurrency complexities. However, Yggdrasil only supports a single active network abstraction per instance (e.g., wireless or wired), and its reliance on C requires rigorous technical discipline to prevent issues such as invalid memory access.

ViSiDia: [1] A Java-based framework designed for understanding and visualizing distributed algorithms. It facilitates the design and simulation of these algorithms, making it an educational tool. The framework's primary objective is to facilitate the comprehension of algorithms, rather than to assess their performance in real-world scenarios or to analyze their interactions with protocols.

Babel: [15] Babel is a framework that promotes event driven programming, and it helps in the development of distributed protocols within a process that can execute any number of different protocols that can interact with each other or exchange messages within or outside the same process/machine. The system is designed with a generic architecture that allows for the implementation of multiple distributed protocols and applications, providing networking components that can capture interaction models (e.g., P2P, Client/Server), making the code mostly independent from the underlying communication aspects. It facilitates the development process by allowing the developer to focus on the fundamental aspects of the (distributed) protocol or application being developed, without having to address the inherent complexities associated with low-level details.

2.1.4.1 Discussion

The decision to utilize Babel was motivated by its development within the context of the TaRDIS project, in which this work is addressing the need for robust and effective decentralized collaborative learning framework. Babel's capabilities align with our requirements, particularly its integration with the Android platform. Additionally, our proximity to the development team facilitates steering its evolution also to make it adaptable to our own application.

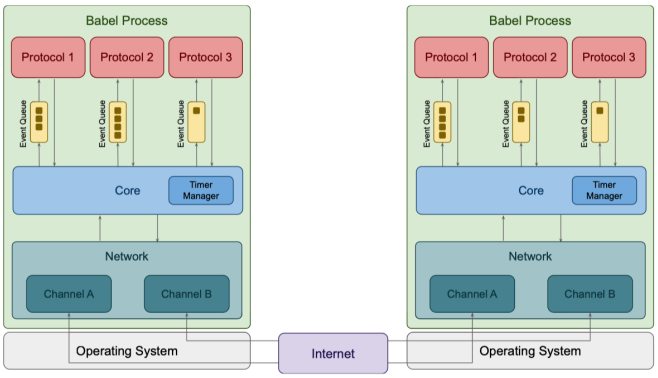


Figure 2.2: Architecture of Babel. Source: [15]

Figure 2.2 presents the architecture of Babel. In this example, two processes are executing Babel, with each process being composed of three protocols and two network channels for inter-process communication. It is important to note that any distributed system in the real world will generally consist of more than two processes. The Babel

framework is composed of three main components: Protocols, Core, and Network. It is recommended that interested readers consult the original paper [15] for further details.

2.1.4.2 Babel swarm

The Babel Swarm, a second-generation Babel framework currently under development, it adds advanced functionalities relevant to the development of new decentralized applications to address certain necessities of the previous version, including improvements in security, self-configuration, and self-management [16]. Once completed, our system will be migrated to the second generation of the Babel framework. This version has been already successfully ported to the android platform.

2.2 Learning Approaches

Machine learning approaches can be broadly categorized based on how data and computational resources are managed during the model training process [75]. The choice of approach significantly impacts scalability, privacy, resilience, and overall performance. Below, are mention some primary approaches.

2.2.1 Local Learning

Local Learning [75] refers to an independent approach where devices train models locally without any communication or exchange of information with other devices. This method eliminates the need to transmit data, preserving complete data privacy. However, it is limited by the non-existence of collaboration, which often leads to suboptimal models due to insufficient data diversity and volume. While useful in scenarios where data sharing is infeasible for legal or privacy reasons, Local Learning cannot leverage the collective potential of distributed datasets.

2.2.2 Centralized Learning

Centralized Learning [75, 47] relies on a server-centric approach, where data from multiple devices is transmitted to a central entity for model training. This method benefits from access to a server that has full access to the user data from different devices, which enables the creation of robust models. However, the high cost of central data storage and processing, combined with the potential for a single point of failure, renders centralized learning less resilient in certain contexts. Moreover, such an approach requires users who wish to collaborate on the learning problem to fully disclose their data, raising significant privacy concerns.

2.2.3 Decentralized Learning

Decentralized Learning [75, 47] focuses on removing the need for a single central entity by enabling devices to collaboratively train models in a distributed fashion. In this approach, devices communicate directly with each other or through a peer-to-peer network to exchange updates or aggregated model parameters. Decentralized Learning reduces latency, enhances fault tolerance by avoiding single points of failure, and increases trust by distributing control. The training process is delineated into a predetermined number of rounds, which are defined at the onset and may change during the course of the process if necessary. In each training round, the client performs a local training with its local data, subsequently shares model updates directly with other clients, and then receives model updates from the others. The client then aggregates these updates and incorporates them into the "main" model. Finally, the client proceeds to the next round. As our work becomes more focused on distributed learning, we will discuss this topic in more detail in sections 2.3.2 and 2.4.

2.3 Federated Learning (FL)

In recent years, Federated Learning [75] (FL) has emerged as an approach for training collaborative models without the need to share sensitive data. Since its emergence, FL or more precisely, (Centralized Federated Learning (CFL)) has been the predominant methodology in the literature, wherein a central entity is responsible for the creation of a global model. However, a centralized approach has the disadvantage of increasing latency due to bottlenecks, heightening vulnerability to system failures, and giving rise to concerns regarding the trustworthiness of the entity responsible for creating the global model. Decentralized Federated Learning (DFL) has been proposed as a way to address these concerns by promoting decentralized model aggregation and minimizing reliance on centralized architectures.

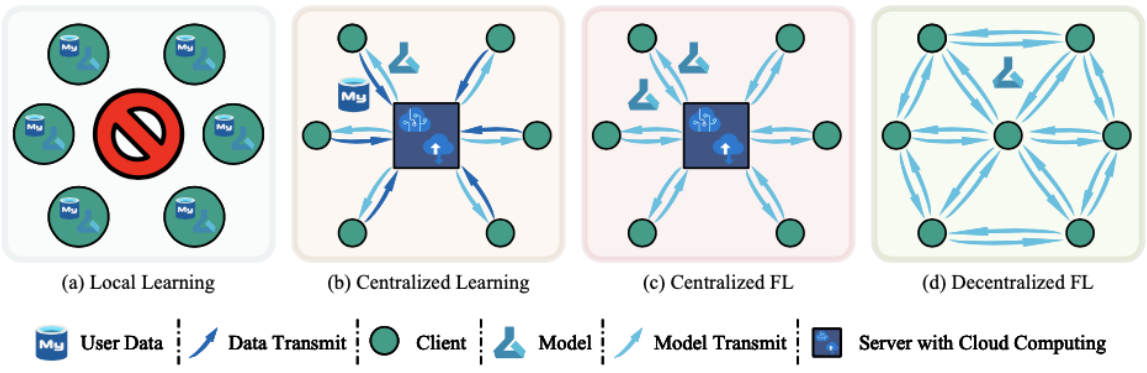


Figure 2.3: Illustration of local learning, centralized learning, CFL, and DFL. Source: [75]

2.3.1 Centralized approach

In CFL [75], a central entity is tasked with the models that are trained independently by and using each participant's data. By exchanging model parameters only and not raw data, this approach allows the data to remain on user devices. However, the dependency on a single aggregation server introduces several limitations. Firstly, as the number of participating devices increases, the central entity may become a bottleneck, resulting in increased latency and reduced scalability. Additionally, this architecture introduces a potential risk of single-point failure, where the central entity's failure can compromise the integrity of the entire training process. Security and trust are further concerns, as devices must trust the central entity with aggregating and distributing model updates.

Due to its simple design and well-established protocols for safe and effective aggregation, CFL is still widely used despite all of these disadvantages. A prominent example is Google's application of federated learning in Gboard [23], its virtual keyboard, that aims to give the best next-word prediction model. In order to minimize the possibility of data leakage, optimization efforts in CFL frequently concentrate on lowering communication costs between devices and the server, enhancing the aggregation algorithms, and guaranteeing strong data privacy techniques, such as differential privacy [69] and secure aggregation methods [75].

2.3.2 Decentralized approach

Decentralized Federated Learning (DFL) [47, 75] reduces dependency on a central entity, which helps to overcome some of the main limitations of CFL. Since model parameters are shared and aggregated in a distributed manner rather than via a single central aggregator, DFL spreads computational tasks more widely throughout the network.

However, even without a central entity, DFL still maintains a mechanism for updating a global model through decentralized aggregation strategies, often involving clusters or peer-to-peer networks where nodes collaborate to aggregate model parameters locally before updating the shared model.

This decentralized approach minimizes the emergence of bottlenecks and reduces the risk of single points of failure, making DFL inherently more fault tolerant and scalable. Additionally, it distributes trust across the entire network rather than concentrating it in a single entity, which can alleviate some concerns about trustworthiness in centralized environments. However, because node interactions are asynchronous, DFL makes it more difficult to maintain consistency in model updates, which can result in stale updates and slower convergence. It also increases communication overhead, as nodes must frequently exchange model parameters with their peers to keep the overall model aligned.

In DFL, the absence of a central authority complicates the detection of malicious devices, posing significant security and privacy challenges. To mitigate these risks, researchers have developed robust decentralized protocols focusing on secure aggregation [27], anomaly detection[50, 13], and model update verification [10].

Secure Aggregation: Protocols like ByzSecAgg [27] have been proposed to ensure the secure aggregation of model updates while being resistant to Byzantine attacks [71] (machines upload malicious information instead of the honest computational results to the server). ByzSecAgg integrates coded computing and vector commitment techniques to safeguard against adversarial behaviors and privacy breaches during the aggregation process.

Anomaly Detection: Protocols like Sentinel [13] leverages the accessibility of local data and defines a three-step aggregation protocol consisting of similarity filtering, bootstrap validation, and normalization to safeguard against malicious model updates. This approach leverages local data accessibility to detect and mitigate poisoning attacks effectively.

Model update verification: Ensuring the integrity of model updates without compromising privacy is crucial in DFL. The EIFFeL [10] system addresses this by enabling secure aggregation of verified updates, allowing for arbitrary integrity checks and the removal of malformed updates without violating privacy.

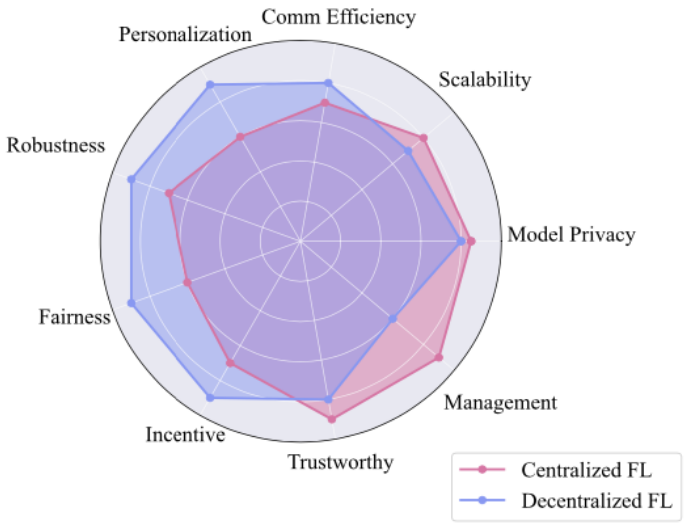


Figure 2.4: Comparative analysis between centralized FL and decentralized FL across various performance metrics. Each axis represents a metric with the plotted values indicating the relative strength of the respective FL approach in that domain. Source: [75]

2.3.3 Discussion

Considering these two approaches, our focus lies on Decentralized Federated Learning (DFL) to ensure that personal data stays on local devices and is not shared across the network. With DFL, only model updates such as parameters and gradients are shared, not the actual data, which helps protect privacy more effectively than a centralized approach. The Figure 2.4 permits you to do a overall comparative analysis between CFL and DFL in

different domains. By choosing DFL, our goal is to create a learning network that respects user privacy while remaining resilient and efficient, even as it grows. In the Section 2.4, we will go deeper into what DFL really is, how it works, some real-world applications, and existing challenges.

2.4 Decentralized Federated Learning (DFL)

DFL [47] represents an extension of FL that eliminates the necessity for a central coordinating server. In contrast, devices communicate directly with one another in a P2P network to train a global machine learning model.

2.4.1 DFL Architecture

DFL [75] is built on a P2P architecture where devices collaborate directly without a central entity or aggregator. Each device trains a model on its own data set and disseminates updates to other devices. This design enables the learning process while simultaneously preserving data privacy and avoiding the bottlenecks that are typical of centralized systems. Typically, devices are autonomous entities that determine their own training schedules and communications based on their available resources and specific requirements. Alternatively, devices may elect another device to act as the coordinating entity and aggregator.

Communication between devices plays an important role in DFL. Devices exchange model updates using protocols like one-to-one (direct communication), gossip [6] (randomized sharing), or broadcast [33] (one-to-many). These exchanges follow a network structure or topology, which defines how devices are connected. Common topologies [51, 67, 75] include linear (sequential sharing), ring (cyclical sharing), and mesh (all-to-all communication). More complex hybrid structures combine these approaches to balance efficiency and flexibility.

Two principal approaches inform the manner in which models evolve in DFL. The Continual Paradigm [47, 37, 75] involves the transfer of a model between devices, with each device updating the model using their own data. This approach is straightforward and resource-efficient, with a primary focus on personalization. In contrast, the Aggregate Paradigm [47] involves the collection and combination of updates from multiple devices before local training. This method results in the generation of a more generalized model, although it demands higher levels of computational resources.

The workflow of DFL is fundamentally iterative. As mention before, in centralized FL, a server coordinates training by aggregating model updates from multiple devices and redistributing the refined global model until convergence. In contrast, DFL eliminates the need for a central server by employing direct peer-to-peer interactions for model aggregation. The iterative workflow of DFL is outlined below:

1. **Initialization:**

Each device starts with an initial model, either received from a shared source or independently created.

2. **Local training:**

   Devices train the model locally using their respective datasets, improving model performance based on local knowledge.

3. **Model sharing:**

   Devices exchange their locally trained models with peers according to the established topology.

4. **Model aggregation:**

   Each device combines received models with its own using an agreed-upon strategy, such as weighted averaging.

5. **Iteration:**

   Steps 2 to 4 are repeated iteratively, with models being refined through continuous local updates and peer exchanges.

6. **Convergence:**

   The process is considered complete once the models across devices have converged, at which point the potential for further updates is negligible and the system's performance is maximized.

Dynamic features, like adaptive topologies [75, 47], allow DFL networks to evolve based on external factors, such as device participation or data shifts. Security measures, such as taking advantage of blockchain and differential privacy, are incorporated to ensure the trustworthiness of the system and to safeguard against potential threats, including data leakage and malicious attacks.

DFL provides a robust, scalable, and privacy-preserving framework for distributed learning. Its architecture makes it particularly suited for applications requiring secure and efficient collaboration, such as healthcare, Internet of Things (IoT), and social networks.

However, Model Inversion (MI) attacks have emerged as a serious privacy threat. MI attacks disclose private information about the training dataset by abusing access to the trained models. These attacks allow adversaries to reconstruct data with a high degree of fidelity, closely aligning with the private training samples, thereby posing significant privacy concerns [12, 77].

2.4.2 Challenges in federated learning

Both CFL and DFL approaches, while innovative in preserving data privacy during collaborative learning, face several critical challenges:

1. **Device and data heterogeneity:**

   Participating devices in FL differ significantly in terms of available memory, network connectivity, and processing power. Additionally, data from different devices is frequently non-IID (not independently and identically distributed), meaning that each device's data may exhibit distinct patterns [30]. As local data from each node can differ from the population as a whole, this lack of uniformity makes it more difficult to create an accurate global model [47]. In DFL, the added challenge of device diversity requires adaptive methods for synchronizing and aggregating model updates.

2. **Security and privacy:**

   Although FL naturally encourages data privacy by keeping data locally, it is still susceptible to a number of attacks because device´s shared parameters and gradients may unintentionally reveal sensitive information [30]. Techniques like model inversion [77] (machine learning security threat that involves using the output of a model to infer some of its parameters or architecture) or membership inference attacks [58] (allows an adversary to query a trained machine learning model to predict whether or not a particular example was contained in the model's training dataset) can exploit these shared updates to reconstruct aspects of the original data or identify if specific data points were involved in training.

   These vulnerabilities demonstrate the necessity of strong privacy-preserving techniques, like secure aggregation [5] or differential privacy [69], to reduce data leakage and preserve model performance.

3. **Scalability and fault tolerance:**

   While DFL enhances fault tolerance by eliminating single points of failure, scaling DFL networks remains challenging. As the network expands, the demands for communication and synchronization among nodes grow, which may result in a reduction in system performance. To manage large numbers of devices efficiently, strategies like effective node clustering [75] (groups devices based on criteria like geographical proximity or resource capacity, reducing communication overhead and ensuring faster synchronization), load balancing [47] (distributes computational tasks evenly across nodes, preventing bottlenecks and ensuring efficient use of network resources), and advanced fault tolerance techniques become essential.

   These methods help distribute workloads more equally across the network and prevent communication bottlenecks, ensuring that model accuracy and system efficiency are maintained as the network scales.

   Yuan et al. and Martinez Beltrán et al. highlight the strengths and weaknesses of centralized versus decentralized FL, focusing on scalability challenges [75, 47].

2.4.3 Real world Applications

DFL framework development depends on various key factors, such as relevant application scenarios, sources of information acquisition, information processing units and perceptual prediction modules, among others. And so DFL has been adopted in several areas, such as, automotive industry [53], healthcare [7], IoT industry [23], etc [75].

Connected and Automated Vehicles: Connected and Automated Vehicles (CAVs) serve as a robust hardware infrastructure for DFL, where each node uses integrated batteries, various sensors, computing units, storage devices, among others. Vehicle-to-vehicle (V2V) communication is a technology that already supports several applications in networks of independent and connected vehicles, creating a basis for the application of Distributed Federated Learning (DFL) in Connected Autonomous Vehicles (CAVs). Vehicle network frameworks, like V2V, have also laid the foundation for communication and networking experiences in DFL for CAV [53].

A V2V approach with FL allows vehicles to share updated knowledge with each other, which is crucial for the safe and efficient operation of vehicle networks. Studies such as that by Lu et al. [44] demonstrate a practical application using Roadside Units (RSUs) act as intermediaries for routing vehicle identities and essential data, facilitating the direct exchange of information between vehicles when necessary.

This system, in addition to protecting data privacy, helps reduce the risk of leaking information in vehicular cyber-physical environments (VCPS), enabling safe and efficient collaboration in real time.

Mobile Services: Mobile services based on IoT devices are an important application scenario for DFL, making use of the capabilities of smartphones, PCs, among others. These devices are equipped with several types of sensors and cameras that allow them to collect a big range of information sources. Unlike the relatively fixed connectivity of CAVs, mobile IoT devices offer more flexible systems and platforms to support a wide range of applications. Recent studies have focused on the development of DFL frameworks specifically designed for mobile IoT devices, with the goal of leveraging their computational power and sensor capabilities [70, 32]. While the traditional example of CFL, such as Google mobile keyboard prediction, is well-known [23], the transfer of such applications to DFLs is of significant interest. For instance, the development of DFLs among professionals with similar roles, such as doctors, lawyers, or engineers, has the potential to generate customized word recommendations that are tailored to their specific needs.

HealthCare: Hospital centers and clinics are beginning to shift from CFL to DFL due to their diverse private data and computing resources. Warnat-Herresthal et al. [68] presented the Swarm Learning, a DFL framework that deals with 4 heterogeneous disease use cases, including COVID-19, tuberculosis, leukaemia and lung pathology.

Healthcare institutions widely employ DFL frameworks in various studies [31, 7].

Satellites and Unmanned Aerial Vehicles: UAVs and satellites in dynamic computing environments have vast amounts of sensitive data for remote sensing, target recognition, and military-related tasks under constraints of limited resources. This makes them suited to extract the benefits of DFL [46, 73, 76]. Bandwidth is a valuable resource for mobile UAVs and satellites. A potential solution is that the application of a DFL framework based on broadcast gossip, customised to dynamic geographical locations, can significantly reduce bandwidth requirements and the consumption of communication resources [56].

Additionally, due to its dynamic nature and highly variable data, DFL can also improve real-time responsiveness, adaptability and efficiency of task execution.

2.5 Split Learning

SL [43, 63, 65] has been proposed as a method for enabling resource-constrained devices to train multi-parameter neural networks (NNs) and participate in FL. In essence, SL divides the Neural Network (NN) model into sections, enabling devices to transfer the largest section as a processing task to a computationally more powerful helper. In SL, it is possible to have multiple helpers that can process model parts of one or more devices, resulting in a notable reduction in the maximum training time across all devices.

In the field of collaborative machine learning, SL provides distinct advantages over traditional FL by addressing the resource constraints and privacy challenges of small devices such as IoT sensors and mobile devices. This model split reduces both the memory footprint and computational demand on client devices, which is essential for integrating learning frameworks in environments with limited resources.[1]

2.5.1 SL Architecture

In SL, the neural network model is divided into sections, at one or more points called "cut layers". Following we will briefly explore the architecture with 1 and 2 cut layers.

As illustrated in Figure 2.5, a 1-cut layer SL architecture involves the client device handling the initial layers, which process the raw data. In scenarios where a 1-cut layer is implemented, the helper requires access to part of the client's input data, typically the labels (assuming the server will be training the second part of the model). This configuration might represent a lower level of privacy compared to the 2-cut layer approach, which does not necessitate the sharing of input data.

[1]While the term "server" is often used in the literature, we adopt the term "helper" from [65]. We prefer this term as it is more general. It can be used to describe any entity that helps another device to perform the training task. It can be a cloud-based or edge-based server (in a centralized scenario) or any device (IoT device, for example) in the decentralized scenario.

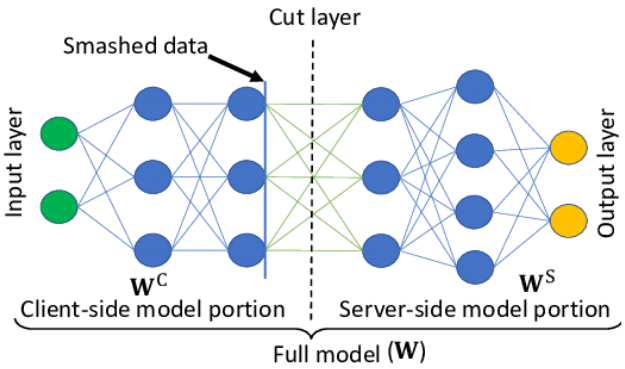


Figure 2.5: Illustration of split learning architecture - 1 cut layer. Source: [63]

On the other hand, Figure 2.6 represents a SL architecture with 2 cut layers, the helper trains the intermediate part, so the ownership of the input data can stay with the clients, since the model parts that require the samples and the labels are trained at the clients.

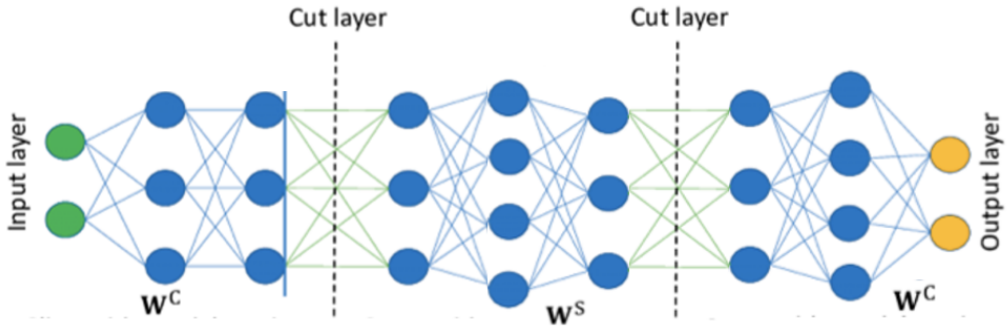


Figure 2.6: Illustration of split learning architecture - 2 cut layers. Adapted from [63]

	1 Cut Layer	2 Cut Layers
Architecture	Simple (two parts)	Complex (three parts)
Client Workload	Light	Moderate (final layers on client)
Helper Workload	Heavy	Light/Moderate (Shared with client)
Privacy	Lower (helper sees smashed data and labels of input data)	Higher (helper sees smashed data but the labels remains on client)

Table 2.1: Comparation between the use of different cut layers

Table 2.1 illustrates some distinctions between the two approaches. In environments characterized by resource-constrained edge devices, such as IoT networks, the 1 cut layer approach is frequently preferred due to its reduced client-side processing requirements. This approach is particularly well-suited for use cases such as real-time health monitoring with wearable devices, where processing capabilities are limited and privacy remains a critical concern. Alternatively, the 2 cut layer architecture offers enhanced privacy by

ensuring that sensitive features and final inferences remain on the client side. This feature is advantageous in applications such as financial analytics or personalized recommendations, where strict data ownership and privacy are paramount concerns. The added complexity of this architecture can be managed effectively in scenarios where client devices possess moderate computational capabilities.

2.5.1.1 Advancements in Cut Layer Selection

The decision on where to place the cut layer in Split Learning (SL) is critical for balancing computational effort and communication overhead. Lin et al. [43] explores the field of cut layer optimization in depth, proposing a stochastic optimization methods that consider resource allocation and workload distribution, particularly in resource-constrained environments like 6G edge networks. This work highlights the importance of adapting cut layers to improve scalability and efficiency in SL frameworks.

2.5.2 Advantages of Split Learning

SL's architecture inherently presents several key benefits

Resource Efficiency: By offloading the heaviest computations to a helper, SL allows smaller devices to contribute to model training without exhausting their limited computational resources. This makes SL appropriate for IoT environments and low-processing-power smart devices [43].

Device Heterogeneity [65]: SL's split structure allows it to work across various hardware configurations and network topologies. For example, SL´s heterogeneous structure enables the workload related to model training to be distributed among devices, helpers, and cloud, taking advantage of diverse computational resources to support scalable networks.

2.5.3 Applications of Split Learning in Emerging Fields

SL's privacy and resource-efficiency features make it suitable for various domains, including:

Healthcare: In collaborative medical AI, hospitals can share insights from their data without sharing actual data. For instance, in diagnosing conditions from imaging data, only processed features are sent to the helper, allowing each hospital to maintain patient confidentiality. SL enhance this capability by accelerating training across numerous institutions without centralizing data.

Currently, image processing applications in hospital imaging services use this type of mechanism and are already able to detect anomalies and, for example, monitor the development of nodules, comparing old images with current ones.

Vepakomma et al. [66] explore the healthcare application of Split Learning is detailed, specifically highlighting how it enables privacy-preserving collaborative learning in medical environments.

Smart Cities and 6G Networks: SL fits well with the distributed and resource-intensive requirements of future smart cities. SL can support real-time applications such as traffic monitoring and public safety by distributing the computational load across edge helper and mobile devices. In these dynamic configurations, SL's ability to utilize multiple edge nodes in a distributed manner is particularly advantageous.

Autonomous Systems: Autonomous vehicles and drones require frequent updates to machine learning models for tasks like object detection. SL enables these devices to offload intensive computations to nearby helpers, facilitating real-time updates without overloading on-board systems.

J. He et al. [25] highlights how cooperative systems can enhance smart cities and autonomous systems through innovations like cooperative sensing, intelligent networking, and edge computing. These technologies facilitate real-time applications such as collision avoidance, traffic monitoring, and vehicle-to-infrastructure communication. Bellow we mention some of this variants

2.5.4 SL variants

SL is a collaborative machine learning paradigm designed to address data ownership concerns, particularly in environments where data must remain decentralized. Over time, several SL variants have been developed to enhance scalability, efficiency, and model performance while adhering to the principle of maintaining data ownership. Below are some of these variants, which are discussed in the paper by Z. Lin et al [43]. These variants are also represented in Figure 2.7.

Vanilla or Sequential Split Learning: Vanilla or Sequential SL, which is the original form, operates sequentially, training the model for one client at a time [66, 40]. However, the sequential training process of vanilla SL incurs excessive training latency. In addition, under highly non-IID settings, the sequential training process may produce suboptimal learning outcomes, as the model tends to fit the data distribution of the last client more closely.

Parallel Split Learning (PSL): To overcome the inefficiencies of sequential training, Parallel Split Learning (PSL) was introduced. PSL allows multiple clients to train their sub-models in parallel, sharing activations and gradients with the helper independently. This approach reduces training latency and supports resource-constrained devices. The absence of aggregation on the clients' side, as in [28], may result in inconsistent parameters across clients, and thus, slowing down the convergence of the global model.

SplitFed Learning (SFL): Split Federated Learning (SFL) [64] combines SL and FL to address the trade-off between computational efficiency and model consistency. SFL introduces periodic synchronization through client-side model aggregation, ensuring alignment across distributed models.

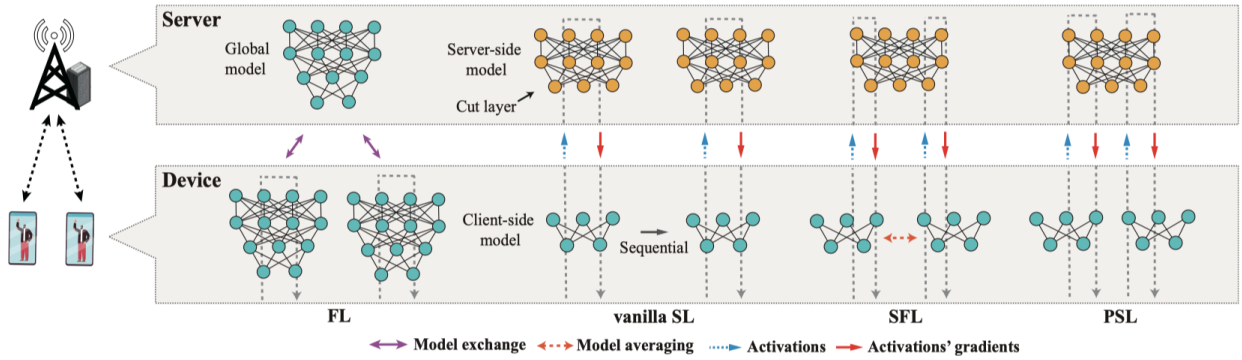


Figure 2.7: The following illustration provides a visual representation of the FL and SL variants referenced in this section. Source: [43]

2.6 Federated Learning vs Split Learning

A pertinent question that emerges from this context is how to choose between SL and FL, both of which are privacy-enhancing collaborative learning frameworks. A critical factor in this analysis is the computing capability of end devices. As previously discussed, SL emerges as a natural solution when end devices are constrained in terms of resources, particularly computing and memory, which hinders their ability to train large models effectively [42].

From a communication perspective, when the training dataset from clients is substantial, federated learning (FL) may be preferable to single-helper learning (SL) because SL incurs large volumes of smashed data that is proportional to the size of the dataset.

In the context of swarm computing, where device heterogeneity is usually high, an inherent approach combines both strategies in a decentralized environment.

2.7 Delegation protocols

As described before a client/node lacking sufficient resources must seek assistance to complete its training task as well as to avoid disrupting the collaborative learning process. It should identify another available or idle client with enough resources to which a portion of the training model can be delegated. For a client to delegate part of its task, certain prerequisites must be met, and several factors can influence the delegation decisions.

First, the client requiring assistance must have a mechanism to quickly identify available clients in the membership that are eligible to become helpers. It then evaluates each

candidate by analyzing their telemetry data - that should report their available resources - to evaluate their suitability. For a client to selected another one as a helper, it must take into account some conditions between both, such as the network conditions (Network latency and bandwidth), relative load between them (consider the overall system load to prevent overburdening and to optimize resource utilization across the network), and Resource Availability (computational power, memory without compromising the helper operations).

Once a suitable helper is identified, the client (delegator) should use a specific communication protocol to coordinate the offloading with the helper. During the establishment of this session, we plan to use communication protocols available in Babel as well as channel abstraction to enable this coordination process, including share the necessary information to proceed with the training.

Efficient management of clients and helpers involves not only dynamic updates of membership information but also identifying conditions that allow clients to be eligible to participate as helpers. This ensures a more efficient task delegation, and robust collaboration across the network. The optimization of resources has been identified as one of the main challenges in SL [41, 22].

2.8 Improvements on Split Learning

SL as already mentioned is a collaborative machine learning approach that partitions a model between clients and a central entity/aggregator, enabling joint training without sharing raw data. This method offers advantages in privacy preservation and resource efficiency. However, it also presents certain challenges and limitations.

Recent advances [72, 29] aim to improve SL by addressing the inherent limitations of the approach. Bellow are some of the recent advancements in SL.

Multi-Level Split Federated Learning (SFL): This approach combines the benefits of SL and FL to improve scalability and reduce communication delays. By introducing a multi-level aggregation architecture, it outperforms traditional SFL by improving model accuracy and convergence speed in large-scale, non-IID settings. The work authored by H. Xu et al. [72] identifies and proposes an architecture, its workflow, and its advantages in improving the robustness and scalability of artificial intelligence of things (AIoT) [59] systems in smart cities while preserving data privacy.

SplitFed Learning without Client-Side Synchronization: Traditional SplitFed Learning requires synchronization among devices, leading to communication and computation overhead. An alternative approach removes this requirement, resulting in a "Multi-head Split Learning" architecture [29].

In MHSL, each client operates independently, performing forward and backward passes on its portion of the split model without requiring frequent coordination with

other clients. Once computations are completed locally, each client sends activation data (output of its model) to a central server. The server processes these activations, updates the shared portion of the model, and returns gradients to the clients for further local updates. This approach eliminates the overhead of client-to-client synchronization, as each client works in parallel without requiring alignment with others during the training process. Empirical studies indicate that this method achieves performance comparable to traditional SplitFed Learning while reducing client-side overhead [29].

2.8.1 Open Challenges

Several studies have identified challenges in SL that remain unresolved, such as Privacy Leakage which despite SL's design to protect data privacy, research has demonstrated the existence of potential vulnerabilities that could allow adversaries to reconstruct the original data from gradients or shared activations [22]. This suggests that SL may not provide comprehensive protection for sensitive information.

2.8.2 Limitations of Existing Solutions

While enhancements have been proposed, certain limitations remain:

Communication Overhead: Some solutions, such as traditional SFL, introduce considerable communication overhead due to the need for client-side synchronization. This can prove to be a significant burden for devices with limited resources.

Scalability Issues: Despite the fact that multi-level SFL is designed to enhance scalability, it may still face challenges in the context of extremely large and heterogeneous networks, particularly when confronted with non-IID data distributions.

These factors described below, once combined, can make efficient scaling difficult in some settings:

Diverse Data Patterns: Non-IID data means each client has significantly different data, making it harder to train a unified model that performs well for everyone.

Resource Constraints: Large networks often involve devices with varying computational power, causing delays and inefficiencies during training.

Communication Bottlenecks: As the network grows, the amount of data exchanged increases, leading to delays and higher costs in synchronizing updates across devices.

2.8.3 Discussion

Our proposed solution addresses the aforementioned limitations by staying with regular SL rather than adopting SFL, due to its significantly simpler development and operational advantages:

Reduced Communication Overhead: Split Learning inherently involves fewer data exchanges compared to SplitFed Learning, as it avoids the frequent client-to-client synchronization that SFL requires. This results in a more efficient and streamlined communication process.

Simplified Development: The development of SplitFed Learning is notably more complex due to the need to coordinate and synchronize updates across multiple devices. By using SL, our solution maintains a straightforward and more manageable architecture.

By focusing on standard SL, our solution prioritizes simplicity and efficiency while addressing key challenges in collaborative machine learning, making it a robust and practical choice for real-world applications.

2.9 FL frameworks

Machine learning as a service (MLaaS) has become more popular in recent decades due to increased collection and processing of big data, availability of public APIs, new advanced Machine Learning (ML) methods, open source libraries, tools for ML analysis large-scale and cloud-based computing. These types of systems tended to be predominantly centralized, but with the emergence of Federated Learning due to concerns in terms of privacy, scalability and devices diversity, new services more dedicated to Federated Learning have started to appear, the so called Federated Learning as a Service. Below will be presented some of these services.

TensorFlow Federated (TFF): TFF [20] is a framework based on Tensorflow developed by Google for decentralized ML and other distributed computations. It includes three key components: models, federated computation builders, and datasets. TFF can be deployed on multiple machines, creating a rotating aggregator.

Flower: Flower [14] is a federated learning framework designed for cloud, mobile, and edge devices. It integrates with PyTorch, TensorFlow, and other ML libraries, supporting cross-device and cross-silo learning. Flower's modular architecture enables custom strategies and client simulations, making it ideal for both research and production. It offers extensive tutorials to guide users from basic setups to advanced customization.

PySyft: PySyft [79] , developed by OpenMined is a framework based on PyTorch for performing encrypted, privacy-preserving DL and implementations of related techniques, such as Secure Multiparty Computation (SMPC) and DP, in untrusted environments while protecting data. PySyft is developed such that it retains the native Torch interface for developers to implement their training algorithm and it can work well with PyTorch and TensorFlow.

OpenFL: OpenFL [52] is a federated learning framework, original developed by Intel that enables organizations to collaboratively train machine learning models on sensitive data without sharing it with a central server. It supports popular deep learning frameworks like PyTorch and TensorFlow, ensuring flexibility and ease of use. OpenFL incorporates privacy-preserving algorithms and trusted execution environments to enhance security. Its scalable design allows users to start with simulations and expand to large federations, making it suitable for various domains, including healthcare and financial services.

FLaaS: Federated Learning as a Service (FLaaS) [34] is a framework by Telefónica Research that provides an accessible platform for federated learning. It focuses on providing easy-to-use APIs for orchestrating federated learning scenarios across distributed devices or organizations. FLaaS enables collaborative and privacy-preserving federated learning across applications, operating on mobile devices, IoT nodes, and edge environments.

2.9.1 Practical Applications and Domains for Federated Learning

Federated Learning (FL) has been widely adopted for training machine learning models on distributed datasets, enabling collaborative learning without compromising data privacy. Below are some prominent applications where FL is utilized:

Image Classification: FL is applied to train models for image recognition tasks using datasets such as CIFAR-10, MNIST, and ImageNet. These setups often simulate cross-device federated learning scenarios, where devices hold non-identical and unbalanced data distributions, collaboratively improving model accuracy.

Healthcare: FL shows significant use cases in healthcare by enabling collaborative training of diagnostic models across hospitals. For example, models for detecting cancer from medical imaging data or predicting disease progression are trained without sharing patient data, ensuring compliance with privacy regulations.

IoT and Edge devices on collaborative training: FL supports edge computing scenarios where IoT devices collaboratively train models on decentralized data. Applications include predictive maintenance in industrial IoT, anomaly detection, and personalization for smart home devices.

Text and Language Processing: FL is used for training natural language processing (NLP) models like next-word prediction and sentiment analysis. Models are trained on text datasets distributed across mobile devices, preserving user privacy.

Financial Services: FL facilitates privacy-preserving applications in the financial sector. Institutions collaborate to build fraud detection or risk assessment models by combining local data insights without sharing sensitive information across borders.

Recommendation Systems: FL enables applications like personalized recommendations for e-commerce platforms and content delivery systems. These systems leverage federated training to maintain user data privacy while improving the recommendation algorithms based on distributed user behavior data.

2.9.2 Discussion

Table 2.2 offers a general overview of the previously mentioned frameworks.

Framework	Key Features	Target Platforms
TFF	Decentralized ML, Rotating Aggregator	Distributed Machines
Flower	Modular, Cloud/Mobile/Edge Support, Cross-Device/Silo Learning	Cloud, Mobile, Edge, Finance, Automotive
PySyft	Privacy-preserving, Encrypted Computation, Secure multiparty computation	Untrusted Environments
OpenFL	Privacy-preserving, Trusted Execution, Scalable Simulations	Healthcare, IoT
FLaaS	Easy-to-use APIs, Collaborative, IoT/Edge/Device Support	Mobile, IoT, Edge

Table 2.2: Comparison table between the presented frameworks

We decided to take advantage of the existing FLaaS framework to facilitate integration with mobile devices and use PyTorch for the integration with desktops and servers. This approach provides cross-platform compatibility while optimizing development resources.

2.10 PyTorch

PyTorch [54] is a widely used open-source machine learning library designed to prioritize usability and performance. It provides an imperative and Pythonic programming interface that supports code as a model, makes debugging easy and is consistent with other popular scientific computing libraries. PyTorch's integration with GPU acceleration makes it suitable for computationally intensive tasks. These features have contributed to its widespread adoption in both academic research and industrial applications

Federated Learning (FL) requires flexible and efficient frameworks to train models collaboratively across distributed clients while preserving data privacy. PyTorch's modular design and dynamic graph capabilities make it an excellent foundation for building FL systems. Notably, PyTorch is the backbone of several FL frameworks, such as PySyft, due to some of it keys features as efficient gradient optimization, automatic differentiation, modularity and scalability.

The scientific community moved away from closed proprietary software such as Matlab to open-source Python tools such as NumPy, SciPy, and Pandas, which met numerical analysis needs and encouraged collaboration. Python's ecosystem, with its openness and flexibility, fostered vibrant communities and rapid development. Its wide adoption since 2014 has made Python the standard interface for most deep learning frameworks.

3

Planning

In this chapter, we begin by briefly presenting the (current) prototype of our decentralized training system (Section 3.1), as the work that has been under development at the same time as this document preparation. We further detail how we can leverage our initial work as a tool to achieve efficient helper delegations, by proposing a new delegation protocol (Section 3.2).

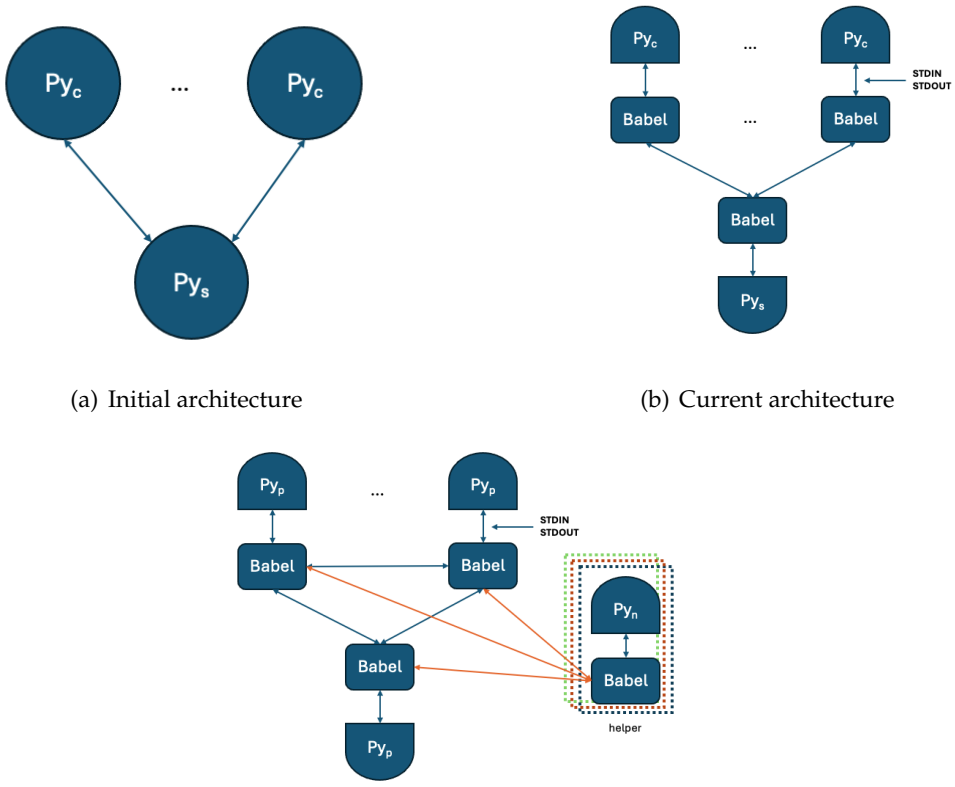
We continue by detailing the evaluation plan of the implemented solution (Section 3.3), as to suitably study these protocols according to reasonable and adequate performance metrics. Lastly, we present the schedule for the future work to be developed in the remainder of the thesis duration (Section 3.4).

3.1 Our Prototype

Presently, a prototype has been developed that utilizes Raspberry Pi, enabling collaborative model training via WLAN or LAN, by taking advantage of the Babel framework [15], which provides reliable and secure channels and facilitates the self-configuration of neighboring clients capable of initiating training. The necessity of efficient, fully decentralized, collaborative training between all kinds of client devices is imperative. This training integrates FL to ensure data ownership and facilitate workload distribution. The utilization of clients that are more powerful or in an idle state as helpers is crucial, as they can assist in the distribution of workload.

For now, the prototype is only performing federated learning and has already incorporated Babel for communication and self-configuration. The following step will be to generalize the Babel code so that it can be utilized across various Federated Learning projects. This will be followed by the integration of FLaaS code to support Android devices, enabling the orchestration of FL and SL on Android devices. Subsequently, a combination of both approaches (FL and SL) will be initiated. The system under consideration is designed to function with minimal human intervention. Specifically, it is not anticipated that central entity will need to manage clients training, or delegate helpers in a production environment. It is expected that these tasks will be fully automated.

Figure 3.1 illustrates the evolution of architecture, that is being pursued in this work from its initial state to its anticipated final form.



(a) Initial architecture

(b) Current architecture

(c) Desired architecture

Figure 3.1: (a) Centralized system without Babel (b) Centralized system with Babel (c) Decentralized system communicating using Babel

The initial architecture (Figure: 3.1 a) was initiated with the simple communication between Python processes, one acting as a server and the others as clients. The implementation of these processes relied on a centralized federated learning approach, necessitating the provision of the IP address of each participant to each process. The initial code was extracted from the source code of the Gao et al. paper [17], which is available on a public repository [18].

In the current architecture (Figure: 3.1 b), Python processes communicate through Babel processes, enabling all participants to automatically recognize each other upon entering its neighborhood. This approach relies on a centralized federated learning model, facilitated by protocols implemented on Babel.

The following step in this work (Figure: 3.1 c) is to move the current architecture to become a fully decentralized federated learning and split learning system. In this system, each client or node can assume the role of a normal client, helper, or both. The system is designed to self-organize and to delegate helpers as needed. All communications are conducted through Babel specific protocols that we will develop. At this stage, it is

anticipated that Babel swarm will be fully developed, and the system will be transitioned to operate on that rather than the current version.

As the reader can check on the Figures 3.1 (b) and (c) the communication between processes is achieved using stdin (standard input) and stdout (standard output). This mechanism allows one program to send data as output, which another program reads as input, enabling a seamless flow of information between the babel process and the python process spawned by it.

3.2 Delegation protocols

The delegation protocol will make decisions in the system that are related to the role of each participating device in the training. Specifically, a participating device may act as a client, and/or as a helper, and/or as an aggregator. The delegation protocol will be designed in a way that enables a balanced computing workload among all participants. Therefore, optimizing the delegation decisions will allow us to take advantage of underutilized computing resources and alleviate the communication overhead that the SL paradigm introduces. To this end, we identify two main directions:

1. Deciding which devices can act as helpers based on the system's characteristics, such as energy (this is particularly important for battery-powered devices) and memory capacity (since the model part that a client offloads to the helper has a memory footprint). This could result in a multi-objective optimization problem for which we aim to devise an efficient algorithm to solve it. This algorithm will be then integrated into the delegation protocol. Similarly, when a rotatory aggregator is employed, one needs to decide which node can act as the aggregator that will average all participant's model weights to compute the global model.

2. Designing incentive mechanisms for devices to act as helpers or as the aggregator. Given that a helper will need to perform additional processing tasks for the clients it assists, the incentives will aim to reward the helper. They may be monetary-based or reputation-based, for example. We plan to devise algorithms for the incentive mechanisms that will be part of the delegation protocol. In contrast to the decisions described above in 1), incentive mechanisms can be considered in less heterogeneous scenarios, where the system's capabilities may play a secondary role.

The initial direction will be addressed, necessitating the establishment of a mechanism to monitor participants and obtain information from them (e.g., CPU, memory). This will facilitate the decision-making process related to the delegation of work. Additionally, secure communication tunnels will be provided to ensure reliable client communication. The second direction will be explored following the completion of the initial phase, if time allows.

3.3 Evaluation Planning

To evaluate our work, we will use our test-bed composed of multiple Raspberry Pi and smartphones, so that we have device diversity, with different operating systems and resource characteristics. We plan to evaluate the implemented protocols using the following metrics: **cost, accuracy, time, usability and fault tolerance**. For **cost**, we will assess CPU and memory usage on each device, distinguishing their roles as either simple participants or as additional helpers, reflecting the varying computational demands of the decentralized system.

For **accuracy**, the focus will be on the performance of the collaboratively trained model. We will analyze metrics such as precision, recall, and F1-score [19] to assess trade-offs and ensure the process robustness. Additionally, validation against benchmark datasets will confirm the effectiveness of the integrated FL and SL approach in handling heterogeneous and decentralized data sources.

The evaluation of the protocols will take into account a variety of metrics, including training **time**, which is defined as the duration required to complete a training cycle under different configurations, and communication overhead, which is defined as the time spent on transmitting data between devices. The goal of this evaluations is to facilitate the identification of trade-offs and performance characteristics of different delegation protocols, thereby enabling us to fine-tune them for improved performance in different environments.

The evaluation of **usability** will prioritize the ease of deployment and operation of the protocols across a range of devices, including Raspberry Pis and smartphones. The metrics utilized to assess these protocols will include the simplicity of setup, the amount of user provided configuration, and the capacity to effortlessly incorporate new devices into the testbed. A particular emphasis will be placed on minimizing user intervention during the process, thereby enabling robust and autonomous operation.

For **fault tolerance**, the study will assess the system's resilience to device failures and communication disruptions. Key metrics include the system's capacity to sustain model training progress and accuracy in the face of failures in subsets of devices or loss of connectivity. To this end, the study will take into account several failure scenarios, including random device failures. Using these scenarios, we will assess the effectiveness of recovery mechanisms, ensuring that the distributed system can reconfigure and sustain functionality with minimal performance degradation.

The experiment will be divided into three phases of testing. Initially, the testing will be conducted exclusively on Raspberry Pis. Subsequently, the testing will be transitioned to smartphones. Finally, the test will include booth type of devices, creating a more heterogeneous environment.

3.4 Scheduling

In order to achieve our objectives, we organize the work plan in the following four main tasks and respective subtasks:

1. **Telemetry collection**

   The collection of relevant telemetry data is imperative for the optimization and improvement of the delegation of the right client/node.

2. **Delegation Protocols**

   a) Delegation decision making

   The establishment of a delegation protocol and a suitable delegation algorithm is crucial to ensure the optimal delegation of tasks in a given context.

   b) Optimization

   Optimize protocol communication

   c) Operationalization

   Coordination between clients/nodes across the parallel FL and SL processes.

   d) Deployment and optimization of the solution distributed protocols

   The integration of the proposed system with Babel is a natural progression, given Babel's relevance in the context of the TaRDIS project.

3. **System integration with Android**

   The system's integration into Android devices will be facilitated by Babel, which has already been ported into Android systems.

4. **Develop two sample applications**

   a) Image classification App:

   Image classification, which is widely used today in various fields such as health and security, and is already integrated in some galleries on our mobile devices.

   b) Recommendation App:

   Personalized recommendation, a feature that is prevalent across most streaming platforms.

5. **Experimental Evaluation**

   Performance tests will be conducted to ensure both correctness and adequate performance.

6. **Writing**

   a) Thesis writing

b) Preparation of paper submission for national conference

As previously stated, the focal point of this work is the Split learning component. The optimization of federated learning development is a subject that is addressed in another master's thesis. The time plan for these tasks is presented in the Figure 3.2 in the form of a Gantt diagram.

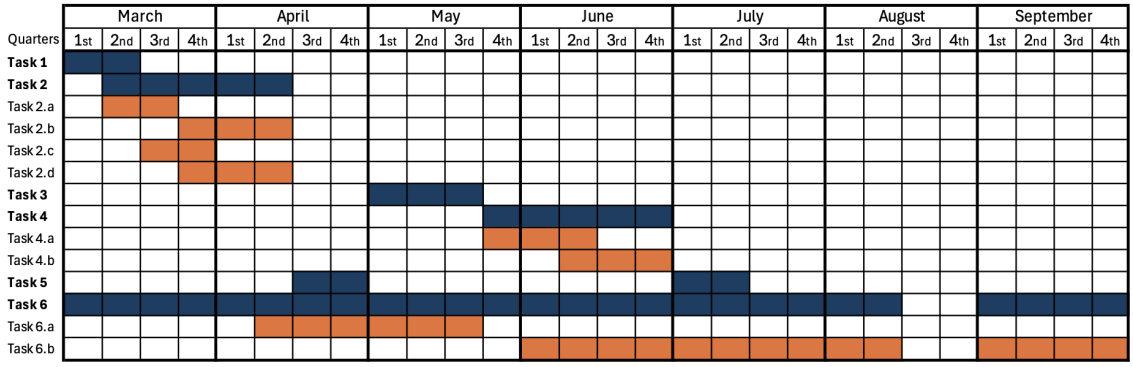


Figure 3.2: Work Plan

Bibliography

[1] W. Abdou, N. O. Abdallah, and M. Mosbah. "ViSiDiA: A Java Framework for Designing, Simulating, and Visualizing Distributed Algorithms". In: *2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*. 2014, pp. 43–46. DOI: 10.1109/DS-RT.2014.14 (cit. on pp. 8, 9).

[2] A. Anitha, J. JayaKumari, and G. Venifa Mini. "A survey of P2P overlays in various networks". In: *2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies*. 2011, pp. 277–281. DOI: 10.1109/ICSCCN.2011.6024559 (cit. on p. 6).

[3] S. Bharati et al. "Federated learning: Applications, challenges and future directions". In: *International Journal of Hybrid Intelligent Systems* 18.1–2 (2022-05), 19–35. ISSN: 1875-8819. DOI: 10.3233/his-220006. URL: http://dx.doi.org/10.3233/HIS-220006 (cit. on p. 1).

[4] K. P. Birman et al. "Bimodal multicast". In: *ACM Trans. Comput. Syst.* 17.2 (1999-05), 41–88. ISSN: 0734-2071. DOI: 10.1145/312203.312207. URL: https://doi.org/10.1145/312203.312207 (cit. on p. 7).

[5] K. Bonawitz et al. *Practical Secure Aggregation for Federated Learning on User-Held Data*. 2016. arXiv: 1611.04482 [cs.CR]. URL: https://arxiv.org/abs/1611.04482 (cit. on p. 16).

[6] S. Boyd et al. "Randomized gossip algorithms". In: *IEEE Transactions on Information Theory* 52.6 (2006), pp. 2508–2530. DOI: 10.1109/TIT.2006.874516 (cit. on pp. 7, 14).

[7] B. Camajori Tedeschini et al. "Decentralized Federated Learning for Healthcare Networks: A Case Study on Tumor Segmentation". In: *IEEE Access* 10 (2022), pp. 8693–8708. DOI: 10.1109/ACCESS.2022.3141913 (cit. on pp. 17, 18).

[8] S. Chaudhari et al. *Peer-to-Peer Learning Dynamics of Wide Neural Networks*. 2024. arXiv: 2409.15267 [cs.LG]. URL: https://arxiv.org/abs/2409.15267 (cit. on pp. 5, 6).

[9] H.-T. Cheng et al. *Wide and Deep Learning for Recommender Systems*. 2016. arXiv: 1606.07792 [cs.LG]. URL: https://arxiv.org/abs/1606.07792 (cit. on p. 3).

[10] A. R. Chowdhury et al. *EIFFeL: Ensuring Integrity for Federated Learning*. 2022. arXiv: 2112.12727 [cs.CR]. URL: https://arxiv.org/abs/2112.12727 (cit. on pp. 12, 13).

[11] P. A. Costa, A. Rosa, and J. a. Leitão. "Enabling wireless ad hoc edge systems with Yggdrasil". In: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. SAC '20. Brno, Czech Republic: Association for Computing Machinery, 2020, 2129–2136. ISBN: 9781450368667. DOI: 10.1145/3341105.3373908. URL: https://doi.org/10.1145/3341105.3373908 (cit. on p. 8).

[12] H. Fang et al. *Privacy Leakage on DNNs: A Survey of Model Inversion Attacks and Defenses*. 2024. arXiv: 2402.04013 [cs.CV]. URL: https://arxiv.org/abs/2402.04013 (cit. on p. 15).

[13] C. Feng et al. *Sentinel: An Aggregation Function to Secure Decentralized Federated Learning*. 2024. arXiv: 2310.08097 [cs.DC]. URL: https://arxiv.org/abs/2310.08097 (cit. on pp. 12, 13).

[14] Flower. "Flower A Friendly Federated AI Framework". In: *(Online)*. 2024. URL: https://flower.ai (cit. on p. 25).

[15] P. Fouto et al. "Babel: A Framework for Developing Performant and Dependable Distributed Protocols". In: *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. 2022, pp. 146–155. DOI: 10.1109/SRDS55811.2022.00022 (cit. on pp. 1, 3, 6, 8–10, 28).

[16] T. Galvão et al. *Suporte ao Desenvolvimento de Novas Aplicações Descentralizadas*. https://www.inforum.pt/static/files/papers/INForum_2024_paper_43.pdf. Accessed: 2025-01-22. 2024 (cit. on p. 10).

[17] Y. Gao et al. "End-to-End Evaluation of Federated Learning and Split Learning for Internet of Things". In: *2020 International Symposium on Reliable Distributed Systems (SRDS)*. 2020, pp. 91–100. DOI: 10.1109/SRDS51746.2020.00017 (cit. on p. 29).

[18] Y. Gao et al. *Federated-Learning-and-Split-Learning-with-raspberry-pi*. https://github.com/Minki-Kim95/Federated-Learning-and-Split-Learning-with-raspberry-pi/tree/master. 2020 (cit. on p. 29).

[19] Google. *Classification: Accuracy, recall, precision, and related metrics*. https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall. Accessed: 2025-01-15. 2024 (cit. on p. 31).

[20] Google. "TensorFlow Federated: Machine Learning on Decentralized Data". In: *(Online)*. 2022. URL: https://www.tensorflow.org/federated (cit. on p. 25).

[21] I. Gopal and R. Rom. "Multicasting to multiple groups over broadcast channels". In: *Proceedings. Computer Networking Symposium*. 1988, pp. 79–81. DOI: 10.1109/CNS.1988.4980 (cit. on p. 7).

[22] H. Hafi et al. *Split Federated Learning for 6G Enabled-Networks: Requirements, Challenges and Future Directions*. 2023. arXiv: 2309.09086 [cs.NI]. URL: https://arxiv.org/abs/2309.09086 (cit. on pp. 23, 24).

[23] A. Hard et al. *Federated Learning for Mobile Keyboard Prediction*. 2019. arXiv: 1811.03604 [cs.CL]. URL: https://arxiv.org/abs/1811.03604 (cit. on pp. 12, 17).

[24] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Öznur Özkasap. *Interlaced: Fully decentralized churn stabilization for Skip Graph-based DHTs*. 2019. arXiv: 1903.07289 [cs.DC]. URL: https://arxiv.org/abs/1903.07289 (cit. on pp. 6, 7).

[25] J. He et al. "Cooperative Connected Autonomous Vehicles (CAV): Research, Applications and Challenges". In: *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. 2019, pp. 1–6. DOI: 10.1109/ICNP.2019.8888126 (cit. on p. 21).

[26] K. He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385 (cit. on p. 3).

[27] T. Jahani-Nezhad, M. A. Maddah-Ali, and G. Caire. *ByzSecAgg: A Byzantine-Resistant Secure Aggregation Scheme for Federated Learning Based on Coded Computing and Vector Commitment*. 2023. arXiv: 2302.09913 [cs.CR]. URL: https://arxiv.org/abs/2302.09913 (cit. on pp. 12, 13).

[28] J. Jeon and J. Kim. "Privacy-Sensitive Parallel Split Learning". In: *2020 International Conference on Information Networking (ICOIN)*. 2020, pp. 7–9. DOI: 10.1109/ICOIN48656.2020.9016486 (cit. on p. 21).

[29] P. Joshi et al. *Splitfed learning without client-side synchronization: Analyzing client-side split network portion size to overall performance*. 2021. arXiv: 2109.09246 [cs.LG]. URL: https://arxiv.org/abs/2109.09246 (cit. on pp. 23, 24).

[30] P. Kairouz et al. *Advances and Open Problems in Federated Learning*. 2021. arXiv: 1912.04977 [cs.LG]. URL: https://arxiv.org/abs/1912.04977 (cit. on pp. 1, 16).

[31] H. Karnati and B. B. V. "Federated and Split Learning for Enhanced Breast Cancer Diagnosis". In: *2023 IEEE Engineering Informatics*. 2023, pp. 1–7. DOI: 10.1109/IEEECONF58110.2023.10520444 (cit. on p. 18).

[32] A. Koloskova et al. *Decentralized Deep Learning with Arbitrary Communication Compression*. 2020. arXiv: 1907.09356 [cs.LG]. URL: https://arxiv.org/abs/1907.09356 (cit. on p. 17).

[33] K. Kouno et al. "Broadcast Protocols in Wireless Networks". In: *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*. 2015, pp. 272–277. DOI: 10.1109/WAINA.2015.135 (cit. on pp. 7, 14).

[34] N. Kourtellis, K. Katevas, and D. Perino. "FLaaS: Federated Learning as a Service". In: *Proceedings of the 1st Workshop on Distributed Machine Learning*. CoNEXT '20. ACM, 2020-12. DOI: 10.1145/3426745.3431337. URL: http://dx.doi.org/10.1145/3426745.3431337 (cit. on p. 26).

[35] S. Lamichhane and P. Herbke. *Verifiable Decentralized IPFS Cluster: Unlocking Trustworthy Data Permanency for Off-Chain Storage*. 2024. arXiv: 2408.07023 [cs.DC]. URL: https://arxiv.org/abs/2408.07023 (cit. on p. 6).

[36] J. Leitao, J. Pereira, and L. Rodrigues. "HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast". In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. 2007, pp. 419–429. DOI: 10.1109/DSN.2007.56 (cit. on p. 6).

[37] T. Lesort et al. *Continual Learning for Robotics: Definition, Framework, Learning Strategies, Opportunities and Challenges*. 2019. arXiv: 1907.00182 [cs.LG]. URL: https://arxiv.org/abs/1907.00182 (cit. on p. 14).

[38] C. Li, Y. Yuan, and F.-Y. Wang. "Blockchain-enabled Federated Learning: A Survey". In: *2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI)*. 2021, pp. 286–289. DOI: 10.1109/DTPI52967.2021.9540163 (cit. on p. 5).

[39] N. Li et al. "Verifying Stochastic Behaviors of Decentralized Self-Adaptive Systems: A Formal Modeling and Simulation Based Approach". In: *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 2018, pp. 67–74. DOI: 10.1109/QRS.2018.00020 (cit. on p. 5).

[40] Y. Li and X. Lyu. *Convergence Analysis of Sequential Split Learning on Heterogeneous Data*. 2023. arXiv: 2302.01633 [cs.LG]. URL: https://arxiv.org/abs/2302.01633 (cit. on p. 21).

[41] W. Y. B. Lim et al. "Federated Learning in Mobile Edge Networks: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* 22.3 (2020), pp. 2031–2063. DOI: 10.1109/COMST.2020.2986024 (cit. on p. 23).

[42] Z. Lin et al. *Pushing Large Language Models to the 6G Edge: Vision, Challenges, and Opportunities*. 2024. arXiv: 2309.16739 [cs.LG]. URL: https://arxiv.org/abs/2309.16739 (cit. on p. 22).

[43] Z. Lin et al. *Split Learning in 6G Edge Networks*. 2024. arXiv: 2306.12194 [cs.LG]. URL: https://arxiv.org/abs/2306.12194 (cit. on pp. 1, 2, 18, 20–22).

[44] Y. Lu et al. "Federated Learning for Data Privacy Preservation in Vehicular Cyber-Physical Systems". In: *IEEE Network* 34.3 (2020), pp. 50–56. DOI: 10.1109/MNET.011.1900317 (cit. on p. 17).

[45] P. Mannersalo, A. Keshavarz-Haddad, and R. Riedi. "Broadcast Flooding Revisited: Survivability and Latency". In: *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*. 2007, pp. 652–660. DOI: 10.1109/INFCOM.2007.82 (cit. on p. 7).

[46] E. T. Martínez Beltrán et al. "Mitigating communications threats in decentralized federated learning through moving target defense". In: *Wirel. Netw.* 30.9 (2024-01), 7407–7421. ISSN: 1022-0038. DOI: 10.1007/s11276-024-03667-8. URL: https://doi.org/10.1007/s11276-024-03667-8 (cit. on p. 18).

[47] E. T. Martínez Beltrán et al. "Decentralized Federated Learning: Fundamentals, State of the Art, Frameworks, Trends, and Challenges". In: *IEEE Communications Surveys and amp; Tutorials* 25.4 (2023), 2983–3013. ISSN: 2373-745X. DOI: 10.1109/comst.2023.3315746. URL: http://dx.doi.org/10.1109/COMST.2023.3315746 (cit. on pp. 2, 10–12, 14–16).

[48] S. Mena et al. "Appia vs. Cactus: comparing protocol composition frameworks". In: *22nd International Symposium on Reliable Distributed Systems, 2003. Proceedings.* 2003, pp. 189–198. DOI: 10.1109/RELDIS.2003.1238068 (cit. on p. 8).

[49] H. Miranda, A. Pinto, and L. Rodrigues. "Appia, a flexible protocol kernel supporting multiple coordinated channels". In: *Proceedings 21st International Conference on Distributed Computing Systems*. 2001, pp. 707–710. DOI: 10.1109/ICDSC.2001.919005 (cit. on p. 8).

[50] V. Mothukuri et al. "Federated-Learning-Based Anomaly Detection for IoT Security Attacks". In: *IEEE Internet of Things Journal* 9.4 (2022), pp. 2545–2554. DOI: 10.1109/JIOT.2021.3077803 (cit. on p. 12).

[51] G. Neglia et al. "The Role of Network Topology for Distributed Machine Learning". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 2019, pp. 2350–2358. DOI: 10.1109/INFOCOM.2019.8737602 (cit. on p. 14).

[52] OPENFL. "OpenFL: an open source framework for Federated Learning". In: *(Online)*. 2024. URL: https://openfl.io (cit. on p. 26).

[53] V. Pandi et al. "Federated Learning for Connected and Automated Vehicles: A Survey of Existing Approaches and Challenges". In: *IEEE Transactions on Intelligent Vehicles* PP (2023-01), pp. 1–21. DOI: 10.1109/TIV.2023.3332675 (cit. on p. 17).

[54] A. Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG]. URL: https://arxiv.org/abs/1912.01703 (cit. on p. 27).

[55] G. Praveen et al. "Blockchain for 5G: A Prelude to Future Telecommunication". In: *IEEE Network* 34.6 (2020), pp. 106–113. DOI: 10.1109/MNET.001.2000005 (cit. on p. 6).

[56] Y. Qu et al. *Decentralized Federated Learning for UAV Networks: Architecture, Challenges, and Opportunities*. 2021. arXiv: 2104.07557 [cs.NI]. URL: https://arxiv.org/abs/2104.07557 (cit. on p. 18).

[57] P. Sawyer et al. "Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems". In: *2010 18th IEEE International Requirements Engineering Conference*. 2010, pp. 95–103. DOI: 10.1109/RE.2010.21 (cit. on p. 5).

[58] R. Shokri et al. *Membership Inference Attacks against Machine Learning Models*. 2017. arXiv: 1610.05820 [cs.CR]. URL: https://arxiv.org/abs/1610.05820 (cit. on p. 16).

[59] S. I. Siam et al. "Artificial Intelligence of Things: A Survey". In: *ACM Transactions on Sensor Networks* 21.1 (2025-01), 1–75. ISSN: 1550-4867. DOI: 10.1145/3690639. URL: http://dx.doi.org/10.1145/3690639 (cit. on p. 23).

[60] H. Song et al. *Unveiling Decentralization: A Comprehensive Review of Technologies, Comparison, Challenges in Bitcoin, Ethereum, and Solana Blockchain*. 2024. arXiv: 2404.04841 [cs.CR]. URL: https://arxiv.org/abs/2404.04841 (cit. on p. 6).

[61] TaRDIS. *TaRDIS architecture definition and technical specifications*. https://project-tardis.eu/download/d2-3-tardis-architecture-definition-and-technical-specifications/. 2024 (cit. on pp. 6, 7).

[62] TaRDIS. *Trustworthy And Resilient Decentralised Intelligence For Edge Systems*. https://project-tardis.eu. 2024 (cit. on p. 3).

[63] C. Thapa, M. A. P. Chamikara, and S. A. Camtepe. *Advancements of federated learning towards privacy preservation: from federated learning to split learning*. 2020. arXiv: 2011.14818 [cs.LG]. URL: https://arxiv.org/abs/2011.14818 (cit. on pp. 1, 18, 19).

[64] C. Thapa et al. *SplitFed: When Federated Learning Meets Split Learning*. 2022. arXiv: 2004.12088 [cs.LG]. URL: https://arxiv.org/abs/2004.12088 (cit. on p. 22).

[65] J. Tirana et al. *Workflow Optimization for Parallel Split Learning*. 2024. arXiv: 2402.10092 [cs.DC]. URL: https://arxiv.org/abs/2402.10092 (cit. on pp. 18, 20).

[66] P. Vepakomma et al. *Split learning for health: Distributed deep learning without sharing raw patient data*. 2018. arXiv: 1812.00564 [cs.LG]. URL: https://arxiv.org/abs/1812.00564 (cit. on p. 21).

[67] S. Wang et al. "Impact of Network Topology on the Performance of DML: Theoretical Analysis and Practical Factors". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 2019, pp. 1729–1737. DOI: 10.1109/INFOCOM.2019.8737595 (cit. on p. 14).

[68] S. Warnat-Herresthal et al. "Swarm Learning for decentralized and confidential clinical machine learning". In: *Nature* 594 (2021-06). DOI: 10.1038/s41586-021-03583-3 (cit. on p. 17).

[69] K. Wei et al. "Federated Learning With Differential Privacy: Algorithms and Performance Analysis". In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3454–3469. DOI: 10.1109/TIFS.2020.2988575 (cit. on pp. 12, 16).

[70] F. Wilhelmi, E. Guerra, and P. Dini. *On the Decentralization of Blockchain-enabled Asynchronous Federated Learning*. 2022. arXiv: 2205.10201 [cs.CR]. URL: https://arxiv.org/abs/2205.10201 (cit. on p. 17).

[71] Q. Xia, Z. Tao, and Q. Li. "Defending Against Byzantine Attacks in Quantum Federated Learning". In: *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*. 2021, pp. 145–152. DOI: 10.1109/MSN53354.2021.00035 (cit. on p. 13).

[72] H. Xu et al. "Multi-Level Split Federated Learning for Large-Scale AIoT System Based on Smart Cities". In: *Future Internet* 16.3 (2024). ISSN: 1999-5903. DOI: 10.3390/fi16030082. URL: https://www.mdpi.com/1999-5903/16/3/82 (cit. on p. 23).

[73] Z. Yan and D. Li. "Convergence Time Optimization for Decentralized Federated Learning With LEO Satellites via Number Control". In: *IEEE Transactions on Vehicular Technology* 73.3 (2024), pp. 4517–4522. DOI: 10.1109/TVT.2023.3322461 (cit. on p. 18).

[74] S. Yonbawi and R. Calinescu. "Towards Self-Adaptive Systems with Hierarchical Decentralised Control". In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2018, pp. 14–16. DOI: 10.1109/FAS-W.2018.00017 (cit. on p. 5).

[75] L. Yuan et al. *Decentralized Federated Learning: A Survey and Perspective*. 2024. arXiv: 2306.01603 [cs.LG]. URL: https://arxiv.org/abs/2306.01603 (cit. on pp. 10–17).

[76] Z. Zhai et al. "FedLEO: An Offloading-Assisted Decentralized Federated Learning Framework for Low Earth Orbit Satellite Networks". In: *IEEE Transactions on Mobile Computing* 23.5 (2024), pp. 5260–5279. DOI: 10.1109/TMC.2023.3304988 (cit. on p. 18).

[77] Y. Zhang et al. "The Secret Revealer: Generative Model-Inversion Attacks Against Deep Neural Networks". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 250–258. DOI: 10.1109/CVPR42600.2020.00033 (cit. on pp. 15, 16).

[78] Z. Zheng et al. "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends". In: *2017 IEEE International Congress on Big Data (BigData Congress)*. 2017, pp. 557–564. DOI: 10.1109/BigDataCongress.2017.85 (cit. on p. 5).

[79] A. Ziller et al. "PySyft: A Library for Easy Federated Learning". In: *Federated Learning Systems: Towards Next-Generation AI*. Ed. by M. H. u. Rehman and M. M. Gaber. Cham: Springer International Publishing, 2021, pp. 111–139. ISBN: 978-3-030-70604-3. DOI: 10.1007/978-3-030-70604-3_5. URL: https://doi.org/10.1007/978-3-030-70604-3_5 (cit. on p. 25).