Architecture and security for cooperative autonomous systems

Divi De Lacour

COLLEGES BRETAGNE LOIRE SCIENCES POUR L'INGENIEUR ET LE NUMERIQUE

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THÈSE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPÉRIEURE
MINES-TÉLÉCOM ATLANTIQUE BRETAGNE PAYS DE LA LOIRE –
IMT ATLANTIQUE

ECOLE DOCTORALE N° 648
Sciences pour l'Ingénieur et le Numérique
Spécialité : Informatique

Par

Divi DE LACOUR

Architecture et sécurité pour les systèmes coopératifs autonomes

Thèse présentée et soutenue à IMT Atlantique, Nantes, le 16 juin 2025
Unité de recherche : STACK, LS2N (IMT Atlantique, Inria)
Thèse N° : 2025IMTA0474

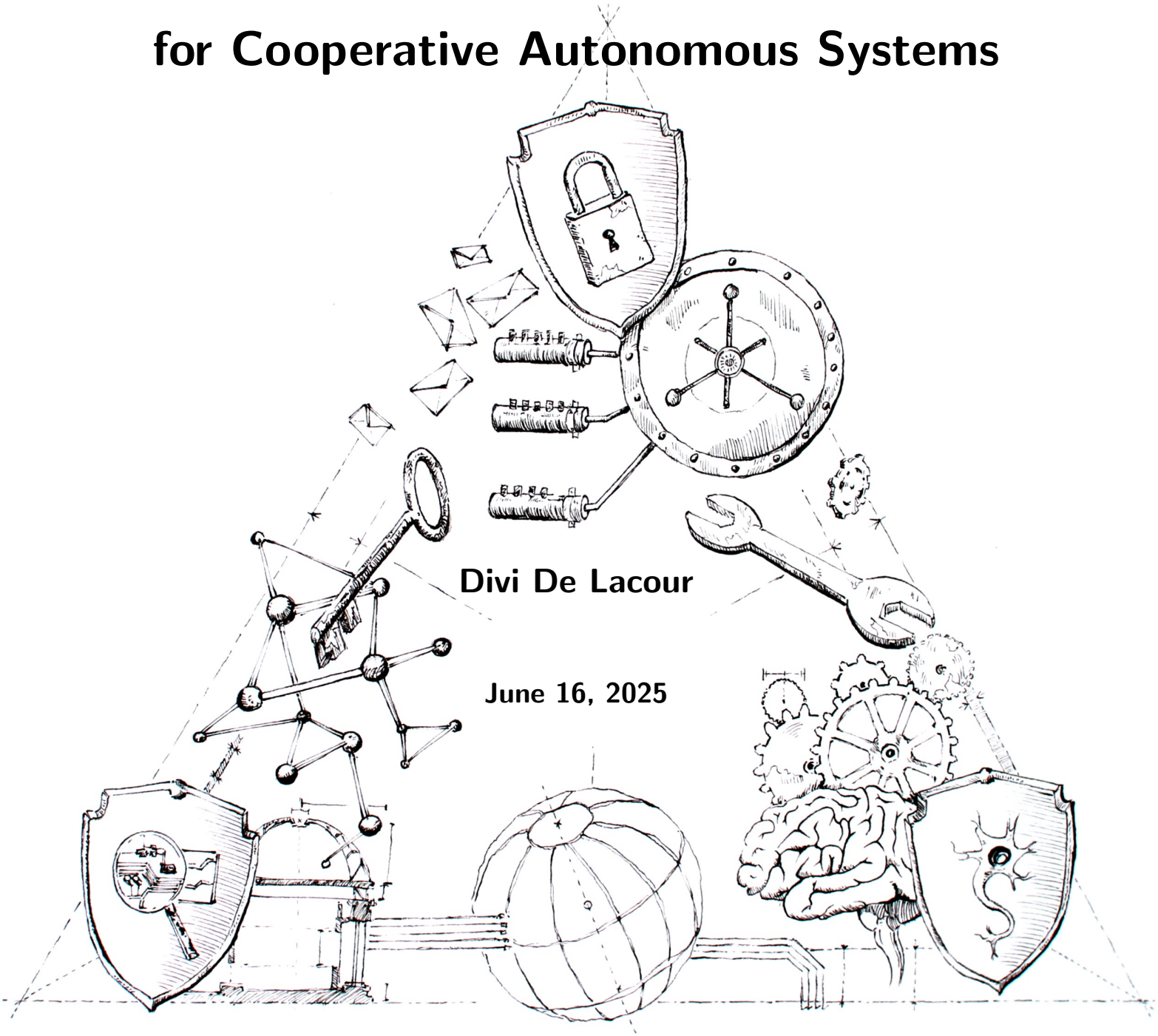
Rapporteurs avant soutenance :

Sophie CHABRIDON Professeure, Télécom SudParis
Eddy CARON Professeur, Université Claude Bernard Lyon1, ISFA

Composition du Jury :

Présidente :	Sophie CHABRIDON	Professeure, Télécom SudParis
Examinateurs :	Eddy CARON	Professeur, Université Claude Bernard Lyon1, ISFA
	Vivien QUÉMA	Professeur, ENSIMAG
Dir. de thèse :	Mario SÜDHOLT	Professeur, IMT-Atlantique
Co-encadrants :	Marc LACOSTE	Expert Recherche, Orange Innovation
	Jacques TRAORÉ	Chercheur, Orange Innovation

Architecture and Security for Cooperative Autonomous Systems



Divi De Lacour

June 16, 2025

Contents

Dedication

To my family, my parents and my brother Malo (thank you for the front-page illustration), whose unwavering love, encouragement, and sacrifices have shaped me into the person I am today. Your belief in me, even when I doubted myself, has been the cornerstone of this journey. Thank you for always being my rock.
To the Archery Club of Châtillon, especially Than and Léa, for teaching me the power of focus and precision. Your camaraderie and dedication have given me a sense of community and belonging, and have been a great source of support throughout my studies.
To the Jeudi Bière meetup, a group that has become more than just a space for sharing beers and laughter. Thank you for providing a unique sense of connection and camaraderie that reminded me to enjoy the journey, even during the most challenging times.
To the friends of Contre Jeudi Bière, for the unforgettable moments of fun and friendship, and for always being there to lift my spirits. Your humor and support have been invaluable, and I am forever grateful for our bond.
To the Danse aux Arts community (thank you Gilles for the recommendation), for introducing me to the joy of partner dance. Your passion for rock and salsa has not only been a source of relaxation and joy but also a reminder of the importance of balance in life.
To the Caves Alliées pub, where countless evenings of conversation, laughter, and reflection have taken place. Your welcoming atmosphere has been a place of refuge and a cherished spot for meaningful connections.
And to the PUC Ultimate Club, for teaching me the value of teamwork, and, resilience (especially to the rain). Through every game, every practice, and every throw, I have learned lessons that went far beyond the field. Your spirit of collaboration and friendship has been an essential part of my PhD journey.
To everyone I have encountered over the past three years.

Abstract

Cooperative Autonomous Systems (CAS) such as intelligent vehicles are harnessed in multiple fields of Industrial IoT systems (IIoT) to solve complex tasks, sharing information and task execution. They have gained significant importance, as illustrated by the autonomous vehicle use-case. Their performance requirements and the presence of multiple agents leads to a situation where they are integrated across a diversity of hardware accross the IoT-Cloud continuum. There, they handle private data and sensitive tasks whose security and privacy must be guaranteed. Security countermeasures exist, but there is no one size fits all countermeasure. This warrants their combination. However, security countermeasures can also have an impact on performance, requiring trade-offs between performance and security.

In this thesis we first review the relevant literature in artificial intelligence systems, distributed systems and security countermeasures. We propose a reference architecture for CAS (RACAS), identifying the ways autonomous systems can cooperate and studying how artificial intelligence can be distributed, with a focus on disaggregated AI. We then study the performance and security of the identified cooperation functions separately. We first focus on collaborative learning performance (FDFL) and on on-demand security countermeasures composition (Ti-skol). We then propose models for other means of cooperation, namely task delegation (D-Lambda) and status communication (ReDOSN) on both security and performance. Additionally, we propose a framework for the secure and efficient decentralized simulation of autonomous systems action environment (D-ECS).

We show that performance can be improved by distributing the autonomous systems and the cooperation across the Cloud-IoT continuum, identifying how CAS can be decentralised for improved scalability and resilience. We also identify the security properties to be protected in each proposed architecture and the associated security countermeasures.

We offer preliminary results on performance and security. This work also warrants the standardization of security countermeasures composition for common use-cases.

Acknowledgements

I would first like to express my deepest gratitude to my supervisor, Professor Mario Südholt, for his guidance, support, and invaluable insights throughout this research. His expertise was crucial to the success of this thesis.
I would also like to sincerely thank my supervisors at Orange, Marc Lacoste and Jacques Traoré for their critical input, constructive feedback, and encouragement during my time working with them. Their expertise and guidance were essential in shaping the direction of my research. I would like to especially thank Marc for his daily supervision and mentorship, teaching me the ins and outs of research.
A special thanks to Adam, Françoise, Gilles, Jean-Phillipe, Sok-yen, Nicolas, Ruby, Jean-François, Pascal, Kahina, Emmanuel, Ronan, Hichem, Maria, Benoit, Gaël, my colleagues at Orange, for their support and mentorship.
Finally, I would like to acknowledge the Inria/IMT Atlantique Stack team for their support, discussions, and the opportunity to be a part of such an innovative and inspiring environment.

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In this chapter we describe cooperative autonomous systems and their requirements. We then identify the techniques used to implement, optimize and secure them. Finally we identify the scientific challenges and the associated approach that we tackle in this thesis.

1.1 Context

Autonomous Systems (AS) are characterized by a high diversity of use cases, from driving cars and piloting drones to managing networks. They sense through various sensors (e.g. cameras, microphones) and act through actuators (e.g. motors, light bulbs) in their environment to reach their objective (e.g. reaching destination, guaranteeing network performance).
Multiple AS can be placed in the same environment (e.g. multiple cars on the same road), which requires cooperation between the ASs to better reach their own objectives. Cooperative Autonomous Systems (CAS) share data collected from their sensors (e.g., cars sharing alerts on collisions) or coordinate with one another to solve tasks together (e.g. multiple robots on a building construction).
CASs collect, analyse, store and share increasing amounts of data that are used for prediction and optimisation by AI and machine learning algorithms, which require more and more computing resources. To provide those resources, CASs feature different Execution Environments (EE) (e.g., VMs, containers, processes). These execution environments can provide more resources either by being replaced by more powerful ones, such as replacing a VM by a bigger one (vertical scaling) or by using multiple EEs at the same time (horizontal scaling). EEs can be distributed throughout the Cloud-IoT continuum (from the same device to the edge located in the neighborhood and the cloud data center). This impacts the system performance such as throughput, latency, and resilience. Cooperation amplifies this problem, as those considerations must be extended to the EEs of a large number of peers on which an AS has limited information and control.

ASs manipulate confidential data and perform sensitive tasks; this implies privacy and security issues. The distribution of EEs and cooperation with untrusted ASs increase the attack surface of ASs, as sensitive systems are connected to each other.
In some cases, compromises must be found between information integrity and information privacy. For example, navigation maps used by connected cars can be updated cooperatively with communications between cars. In this case, there is a trade-off between the amount of private information shared by a car to prove that data is valid and the privacy leakage due to sharing information.
Security countermeasures exist to guarantee the security and privacy properties. For example, Differential Privacy [37] gives mathematical guarantees on the anonymization of a dataset and secure aggregation [152] allows to compute the sum of data collected from multiple sources without disclosing the individual inputs. However, there are no global countermeasures that guarantee all properties. Countermeasures must be combined, but they are not necessarily compatible with each other. Furthermore, countermeasures often impact the performance of the system, trade-offs must be found between security and performance.
ASs need to satisfy three classes of requirements, each one featuring multiple metrics:

- **Functionality/Correctness**: The AS accomplishes what it was designed to do. For example, a car will drive autonomously without having an accident. Metrics evaluate its ability to fulfill its intended purpose, including the accuracy of the AI model on the task, the quality of task accomplishment, and the usability of the system across multiple use cases.

- **Security**: The AS ensures the integrity, privacy, and availability of its data and processing, protecting itself and other ASs from malicious behaviors. For example, a car may ensure the privacy of its exact location. Metrics can include the accuracy of protection mechanisms and specific guaranteed properties such as data encryption standards, authentication protocols, and system resilience against attacks.

- **Performance**: The AS optimizes its resource usage (e.g., computational power) while providing the necessary resources to achieve its functionality. Metrics can include resource usage such as bandwidth or memory, response latency, and energy efficiency.

Those requirements for ASs warrant the combination of the scientific fields:

- Artificial Intelligence and autonomous systems for functionality.

- Distributed and decentralised systems for performance.

- Security countermeasures and privacy-enhancing technologies for security.

1.2 State of the art

1.2.1 Autonomous systems functional requirements

Autonomous systems depend on diverse functionalities, including navigation, decision-making, and task execution. They follow a control loop, observe their environment and then act on it. This control loop has been formalised using the MAPE-k standard [14] applied in modern autonomous systems in various forms.
In addition to functional requirements, autonomous systems must also meet non-functional requirements, often referred to as S-CHOP (Self-Configuration, Self-Healing, Self-Optimization, and Self-Protection) properties.
Autonomous systems can be extended to Cooperative Autonomous Systems (CAS). Notable means of cooperation include collaborative learning, in which CASs jointly train a machine learning model using their own data [105], or sharing their status like their position with the other CASs (e.g. anticipated cooperative collision avoidance [76]). However, there is currently no universally accepted framework or set of guidelines that clearly defines the various methods for cooperation between Cooperative Autonomous Systems (CASs).
ASs actions can be placed in a real or simulated environment such as driving simulators [42]. There are standard simulations for a single AS [168], but there is no standard for the simulation of multiple AS environments.
To guarantee the functionality, there is a need to formalize the different means of cooperation and formalize the means of cooperation themselves.

1.2.2 Tools and technologies

To provide functionality, performance and security we identify here the tools and techniques that answers them: artificial intelligence, distributed systems and security countermeasures.

Artificial Intelligence

Functionality/correctness of autonomous systems is provided by Artificial Intelligence (AI) approaches. We distinguish two main families of AI techniques:
Machine Learning approaches where a machine performs a statistical analysis of data to establish a model that is used for prediction and decisions. They include Deep Learning approaches where a neural network is trained, as well as federated and decentralized learning approaches which are extensively discussed in this thesis. Federated Learning allows multiple autonomous systems to collaboratively train a shared model while keeping their data localized, thus enhancing privacy and reducing data transfer overhead. Decentralized learning further extends this concept by eliminating the need for a central coordinator, distributing the learning process across multiple nodes to improve scalability and resilience.
Expert systems where the algorithm is fully designed by experts. This includes state machines and rule based systems.

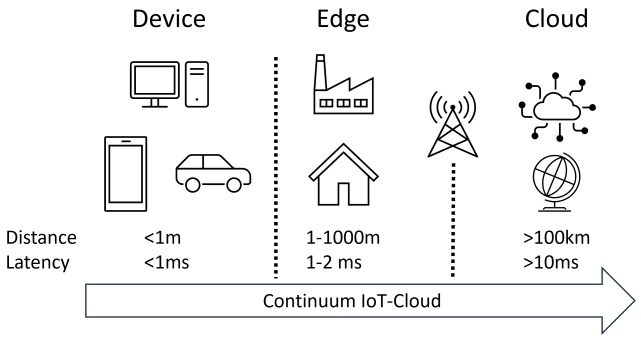


Figure 1.1: IoT-Cloud continuum

Distributed and decentralised systems

The need for distribution is crucial for achieving enhanced performance and supporting multi-tenancy in AI algorithms. To be applied in real-life use-cases, the AI algorithms that provide the functionalities have to run on an infrastructure that meets security and performance requirements.
To scale an algorithm for better performances, two approaches are available: vertical scaling where the infrastructure is replaced by a more performant execution environment, and, horizontal scaling where the number of execution environments is increased to handle the load [185]. Both are combined at the same time, for example to process deep learning models, the Deep Learning community switched from CPU to GPU that shows better performances and switched from single GPU systems to multiple GPU systems.
The infrastructure can be separated at different physical distances (Figure 1.1): in the same machine (1m), at a point of presence situated in the neighborhood($1-$1000m), in a cloud datacenter located in the country($>$ 100km). There is a trade-off between close infrastructure that features faster communications and farther infrastructure that provide more computing power.
Distributing systems and handling their communications is usually done using a single coordinator. This poses resilience and scalability issues as the coordinator can fail and must handle an increasing number of systems. To handle those issues decentralised protocols have been proposed [34] like blockchains to handle monetary exchanges [61] and P2P networks to share files [123].

Security Countermeasures

Security countermeasures can take various forms (e.g. cryptographic algorithms, hardware protections). They provide integrity, availability or privacy guarantees on the execution or processed data. Each countermeasure provides different guarantees but at a cost (e.g. performance overhead, degraded privacy)
Trusted execution environments are a type of hardware protection integrated in the CPU (e.g. AMD-SEV, Intel TDX, ARM Trustzone) [77]. They guar-

antee the integrity and privacy of computations and data they handle with an encrypted memory. They provide remote attestation schemes that allow to remotely verify that a workload is on a TEE. Workloads may take up to 5% more time on TEE than on their normal world counterpart [113].
Other countermeasures include encryption schemes such as secure aggregation [152] and Homomorphic Encryption [53] that allow to compute on encrypted data, noising schemes such as Differential Privacy [37] that guarantee the anonymization of data and consensus algorithms such as byzantine fault tolerance [11].

1.2.3 Combining the solutions

Techniques have been identified to answer to the requirements of the three axes: functional, performance and security. This warrants their combination. There are multiple works to combine two of these axes: Federated and Distributed Learning [106] combine machine learning schemes and distribution schemes. Ekiden [56] combines TEEs for security and blockchains for distribution. Combinations involving three axes are more rare but exist (e.g. secure federated learning [110]).
Those combinations may be specific to the use case. We could want to have slightly different properties on the different axes (e.g; different performance trade-off between latency and throughput). This warrants properties on demand as in aspect-oriented programming [6], where the developer selects the desired properties and a weaver makes the technical choices and adapts the program.
Current architectures are also often difficult to adapt to specific performance and security constraints. For example, the byzantine protection scheme FLAME [130] for federated learning guarantees the integrity of the training, but does not provide privacy of the trained model and is not compatible with secure aggregation [152] schemes.

1.3 Challenges

In the state of the art, we identified the lack formalization of cooperation means and their architectures that combine functionality, performance and security.
The research questions of this thesis are:

1. How can we define a reference architecture for autonomous systems that supports both cooperation and performance? What are the different means of cooperation that need to be identified and incorporated into this architecture?

2. How can we propose more performant versions of cooperative functionalities through distribution and decentralization? What specific techniques and methodologies can be employed to enhance the performance of these cooperative functionalities?

3. In the proposed architectures, what are the essential security properties that need to be guaranteed? What countermeasures can be implemented to ensure these security properties, and how can these countermeasures be combined on demand to address varying security and performance requirements?

1.4 Contributions

This thesis structure follows a functional approach, at each step we study a specific functional need. Figure 1.2 associates the research questions with the contributions. Security and performances are tangential, we identify at each step how the architecture can be distributed, the security properties and associated countermeasures:

- We formalise Cooperative Autonomous Systems with RACAS. We show how autonomous and AI systems can be disaggregated for better performance. We identify the different types of cooperation for ASs.
- We focus on cooperative learning of AI systems, first on the decentralisation for better performance (FDFL), then on security countermeasures and their composition (Ti-skol).
- We focus on other means of cooperation and their performance: task delegation with Function as a Service (D-lambda), status sharing with social networks (ReDOSN), and action environment simulation with video games (D-ECS). Each work being independent, we show how they can be decentralised and secured.

The background chapter reviews the basic knowledge of this thesis. Each chapter is independent enough that it necessitates its own related works section.

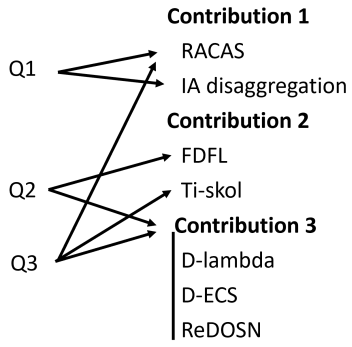


Figure 1.2: Association between scientific questions and contributions

Publications

- **Divi De Lacour, Marc Lacoste, Mario Südholt, Jacques Traoré**
  Towards a Reference Architecture for Secure Cooperative Autonomous Systems for IIoT
  1st Workshop on Cloud-edge Systems in Industrial IoT Applications (CE-IIA) - 28th Conference on Innovation in Clouds, Internet and Networks , mars 2025, Paris, France, ©2025 IEEE.

- **Mohamed Ghamri, Marc Lacoste, Divi De Lacour**
  Disaggregation Patterns for Secure AI Systems
  HPEC 2024 - 28th Annual IEEE High Performance Extreme Computing Virtual Conference, Sep 2024, Boston (MA), United States, ©2024 IEEE.

- **Divi De Lacour, Marc Lacoste, Mario Südholt, Jacques Traoré**
  Towards Scalable Resilient Federated Learning: A Fully Decentralised Approach
  PeRConAI 2023: 2nd IEEE Workshop on Pervasive and Resource-constrained Artificial Intelligence, IEEE International Conference on Pervasive Computing and Communications (PerCom Workshops), Mar 2023, Atlanta, GA, United States, ©2023 IEEE.

- **Divi De Lacour, Marc Lacoste, Mario Südholt, Jacques Traoré**
  Ti-skol: A Modular Federated Learning Framework Supporting Security Countermeasure Composition
  Big Data 2024 - Special Session on Federated Learning on Big Data, IEEE BigData 2024, Dec 2024, Washington, DC, United States, ©2024 IEEE.

- **Divi De Lacour**
  D-Lambda: a fully decentralised serverless model, Research Paper, `hal-04805193`, 2024

- **Divi De Lacour, Adam Gautier**
  ReDOSN a customizable Decentralised Social Network, Research Paper, `hal-04889747`, 2025

- **Divi De Lacour**
  D-ECS: Towards decentralising video games, Research Paper, `hal-04886531`, 2025

- **Divi De Lacour, Marc Lacoste, Mario Südholt, Jacques Traoré**
  Towards Volunteer Deep Learning: Security Challenges and Solutions
  HAL, `hal-04879559`, 2025

Chapter 2

Principles of Secure Intelligent Distributed Systems

In this chapter, we provide an overview of the foundational concepts pertinent to this thesis. The discussion is organized into three principal categories as identified in the introduction chapter. Section 2.1 covers the field of artificial intelligence and autonomous systems. In Section 2.2, we explore key distributed and decentralized systems. Section 2.3 addresses security properties and potential countermeasures.

2.1 Intelligent Systems

2.1.1 Autonomous Systems

Autonomous Systems [14] are self-governing systems capable of operating independently without human intervention. These systems can make decisions and perform tasks based on predefined rules, algorithms, and real-time data inputs. They are designed to adapt to changing environments and conditions, learning from their experiences to improve their performance over time.

Autonomous systems

Autonomic systems are often characterized by four key self-management properties [14, 19], which describe their ability to manage themselves:

- **Self-Configuring** means the system can automatically configure and reconfigure itself in response to changing conditions and requirements. This includes tasks such as installing software, setting parameters, and adjusting configurations without human intervention.

- **Self-Healing** means the system can detect, diagnose, and repair faults and issues autonomously. This involves identifying problems, isolating faulty components, and taking corrective actions to restore normal operation.
- **Self-Optimizing** means the system can monitor its performance and make adjustments to optimize resource utilization and efficiency. This includes tasks such as load balancing, tuning performance parameters, and reallocating resources to meet changing demands.
- **Self-Protecting** means the system can identify and protect itself against security threats and attacks. This involves monitoring for suspicious activities, detecting vulnerabilities, and taking preventive measures to safeguard the system's integrity and data.

These self-* properties enable autonomic systems to operate with minimal human intervention, improving their reliability, efficiency, and resilience.
The **MAPE-K loop** [14] is a framework used in autonomic computing to manage and control systems autonomously. It stands for **Monitor, Analyze, Plan, Execute, and Knowledge**:
Monitor collects data from the system and its environment, observing the current state and performance metrics. **Analyze** processes the collected data to identify patterns, trends, and anomalies. **Plan** develops strategies and actions to address any identified issues or to optimize performance. **Execute** implements the planned actions to adjust the system's behavior or configuration. **Knowledge** maintains a repository of information and experiences that the system can use to improve its decision-making processes over time. This includes historical data, policies, and learned behaviors.
Different architectures have been proposed for several AS use cases. The GANA architecture (Generic Autonomic Networking Architecture) [43] has been standardized at ETSI for autonomous networks. Le Cun [192] proposes a reference architecture for autonomous systems where all of the components are neural network based and can be trained with gradient descent. [127, 155] advocate for a foundation architecture for AS. [134] reviews emerging research directions in this area. However, a comprehensive overview of CAS architectures is lacking. Performance, security and trustworthiness insights remain hard to transfer between architectures and use cases due to the specificity of existing approaches and optimizations. AS cooperation is usually hidden in the architecture or limited to a specific use case (e.g., sharing localization) and remains difficult to identify and secure.

Cooperative Autonomous Systems

Cooperative Autonomous Systems (CAS) extend ASs as systems that interact to jointly solve tasks or maximize utility [16]. Cooperation may take different forms such as CAS broadcasting their status, delegating tasks to other CAS or training distributed machine learning models [160].

Cooperation can take different forms:

- **Consensus**: a subset of components can establish a contract on tasks and transactions to perform complex tasks [3]. Contracts can be established in a decentralized setting using blockchain protocols [90] or a centralized arbiter.
- **Collaborative learning**: AS can create new knowledge from their individual experiences, using their local data to train a common machine learning model [107, 160]. This includes distributed compute like Volunteer Computing[60] where each participant shares a part of its computing ressources to solve complex tasks.
- **Resource search**: AS can search for specific resources [165] (e.g., service, file) in the network of AS using their ID [13] or using semantic search [50].
- **Task delegation**: AS can delegate tasks to other AS such as executing functions [154].
- **Status communication**: AS can broadcast their status to one or multiple AS, or even to humans as in social networks [81, 136].

2.1.2 Artificial Intelligence

[204] provides two axes to define Artificial Intelligence: systems that act/reason like humans/rationally. This is a goal that current "AI systems" do not reach. So we would prefer to define AI here by the techniques that are called AI.
We distinguish two families of AI techniques : machine learning/data based and model/rule based.

Data based AI / Machine Learning

Data-based AI, commonly referred as Machine Learning (ML), involves learning from data. The typical process includes collecting data, training a model, evaluating the model, and using the model to make predictions. The quality of the data is critically important, as the adage "garbage in, garbage out" suggests; poor quality data leads to poor model performance.
There are various types of models used in machine learning, such as decision trees, linear regressions, and neural networks. These models are evaluated using metrics like accuracy, which measures the percentage of correct predictions.
By design, machine learning models cannot guarantee to always provide correct answers. However, they are particularly useful for tasks where it is impractical to program expert rules to handle all edge cases, but where data is available that could show examples of good or bad predictions.
Deep Learning is a sub-field of machine learning that focuses on using simulated neural networks as models. These neural networks have shown very good performance on tasks such as text, image, and audio processing.

During training of a deep learning model, the model makes predictions and calculates the error between the predicted and actual outcomes. This error is then propagated back through the network using a technique called backpropagation. The weights are updated using an optimization algorithm such as gradient descent, which minimizes the error by iteratively adjusting the weights to reduce the error. Over many iterations, the model learns to make more accurate predictions by minimizing the error.
Two phases are distinguished in a ML model: training and inference.

Training

Training in machine learning involves providing data to a model so that it can learn patterns and make accurate predictions. This process often requires significant computational resources (e.g. such as thousands of GPU instances for, and terabytes of data [73]), or highly advanced algorithms.
Training can be centralized on a single machine, but for compute intensive models, it is often distributed across specialized hardware or multiple machines to leverage parallelization and overcome the limitations of a single machine's computational power.
Additionally, training can be federated [44, 105], which allows the use of private data while preserving privacy. An extension is fully decentralized training [96, 119], which further enhances privacy and scalability by distributing the training process across a network of nodes.
Training algorithms deeply depend of the type of ML models. Fine-tuning is a type of training for specific cases large language models that has already been pre-trained.

Inference

Inference in Machine Learning refers to the process of using a trained model to make predictions or decisions based on new, unseen data. This process can occur in various settings. In the classic case, inference occurs in a centralised setting, where the model runs on a single machine or server. Alternatively, inference can be distributed across multiple machines, such as using multiple GPUs to handle large-scale or real-time predictions. Another approach involves splitting the inference process for privacy and efficiency reasons: the initial part of the prediction is done locally to handle sensitive information on the input, while the latter part is performed at the edge or in the cloud to leverage computational acceleration [137].

Model based AI

Model-based systems in AI refer to systems that use explicit models to represent knowledge about the world. They include a large variety of techniques such as graphs path-finding, rule-based systems, state machines and linear optimization algorithms. Unlike ML models, their whole behavior is explicit in the algorithm

and they do not learn from data. These systems are designed to be verifiable, ensuring that they consistently produce correct answers within their defined scope. However, they may struggle in complex environments where decision rules are not clearly defined such as computer vision.

2.1.3 Security of AI

Security of Machine Learning (ML) systems encompasses multiple security and privacy issues, extending beyond traditional computing security concerns, particularly in the context of Federated Learning (FL) [92, 132].
Attacks to alter the model involve modifying the model through data poisoning before training. This aims to reduce the model's performance or introduce backdoors (backdoor attacks) that trigger specific behaviours for certain inputs.
Attacks on an already trained model search for adversarial examples. These attacks involve crafting specific inputs that cause the model to behave incorrectly.
Attacks on privacy, such as membership inference and training data extraction, aim to extract information about the data used during training. Membership inference attacks determine whether a specific data point was part of the training set, while training data extraction attempts to reconstruct the training data itself.
Unlike data-based systems, model-based systems are not trained on private data. However, expert systems may still contain sensitive information, such as proprietary business knowledge. In these cases, security is treated similarly to traditional computing. This involves verifying the integrity and availability of computations and ensuring the privacy of inputs and outputs (and in some cases privacy of the code).

2.2 Distributed and decentralized systems

Distributed systems [191] are a collection of systems that appear to the users of the system as a single coherent system. These computers, or nodes, work together to achieve a common goal, sharing resources and coordinating their actions through network communication. Distributed systems aim to improve performance, reliability[1], and scalability by distributing tasks and data across multiple nodes. Examples include cloud computing platform, and distributed databases.
Decentralized systems [34] are a subset of distributed systems where control and decision-making are distributed across multiple nodes, rather than being managed by a central authority. Each node operates independently and can make its own decisions, contributing to the overall system's functionality. Decentralized systems enhance fault tolerance, eliminating the risk of single points of failure, and often improve security and privacy. Examples include blockchain networks and Peer-to-Peer (P2P) file-sharing systems.

[1] "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable." Leslie Lamport

Distributed and decentralized systems are designed to run on multiple execution environments simultaneously, communicating with each other to achieve a common goal. These systems distribute the load across various environments such as Virtual Machines (VMs), containers, and processes. By leveraging these environments, they can optimize resource utilization and improve overall performance.
To further enhance performance, these environments are distributed across multiple hardware resources. Specialized hardware, such as GPUs, is often employed for specific tasks that require high computational power. The hardware itself can be spread across the entire IoT-cloud continuum, which integrates Internet of Things (IoT) devices with cloud computing resources [185]. This continuum aims to allow for seamless data processing and storage, leveraging the computational power of the cloud while utilizing the real-time data collection capabilities of IoT devices.
Distribution of resources and tasks in these systems significantly improves performance metrics such as latency, throughput, storage capacity, and data sharing capabilities. This distributed architecture can be managed under a single administrative domain or across different management entities, providing flexibility and scalability to meet various operational requirements.

2.2.1 Definitions and types

We define here the different notions of distribution and decentralisation used in this thesis. We first identify the types of distribution. We then review the decentralised technologies used in this thesis.

Distribution

We identify the distribution on two axes, distribution into multiple execution environments and distribution onto multiple hardware.

Execution Environments

Execution Environments provide the necessary infrastructure for running applications and managing resources, they can take different forms each with its own overhead on performance and isolation guarantee:
Processes are the fundamental units of execution within an operating system, each with its own memory space and system resources. They enable multitasking by allowing multiple applications to run concurrently on a single machine.
Virtual Machines (VMs) [64] offer a more isolated execution environment by emulating entire operating systems on a host machine. Each VM operates independently, with its own virtualized hardware and software stack, providing strong isolation and the ability to run different operating systems on the same physical hardware.
Containers [54], on the other hand, provide a lightweight alternative to VMs by encapsulating applications and their dependencies within a single package.

Containers share the host operating system's kernel but maintain isolated user spaces, allowing for efficient resource utilization and rapid deployment.

Hardware distribution

Hardware distribution can occur both within a single machine and across multiple machines [185], each offering unique advantages and capabilities. Within a single machine, CPUs with multithreading capabilities enable parallel processing, allowing multiple tasks to be executed simultaneously on a single processor. GPUs are optimized for handling highly parallel tasks, making them ideal for graphics rendering and computationally intensive applications such as machine learning and scientific simulations. Additionally, specialized hardware like FPGAs (Field-Programmable Gate Arrays), TPUs (Tensor Processing Units), and various types of XPUs provide tailored solutions for specific computational tasks such as deep learning inference, offering significant performance improvements over general-purpose processors.
Hardware distribution can be extended across multiple machines: clusters of interconnected computers work together to perform large-scale computations, distributing the workload across multiple nodes to achieve higher throughput and redundancy. This setup is commonly used in High-Performance Computing (HPC) environments and data centers. Furthermore, the IoT-cloud continuum [133] represents the integration of Internet of Things (IoT) devices with cloud computing resources. This continuum allows for seamless data processing and storage, leveraging the computational power of the cloud while utilizing the real-time data collection capabilities of IoT devices.

Decentralization

Decentralization is essential for achieving resilience and scalability in systems [34], as it eliminates the reliance on a central server, which can become a single point of failure and a bottleneck. By distributing tasks and data across multiple nodes, decentralized systems can continue to operate effectively even if some nodes fail or become compromised. This approach also allows for better load distribution, enhancing the system's overall performance and scalability.
However, decentralization often involves machines that are connected via slower links and managed by different administrators, which introduces additional complexities. These machines may have varying levels of performance and reliability, and coordinating them requires robust protocols to ensure consistent and reliable operation.
We review here the main decentralised technologies used in this thesis.

P2P networks

Peer-to-Peer (P2P) networks [186] are decentralized networks where each participant, known as a peer, acts as both a client and a server. In P2P networks, peers share resources directly with each other without relying on a centralized

server. This architecture enhances scalability, fault tolerance, and resource distribution.
Peer-to-Peer (P2P) networks can be structured in various ways to balance efficiency and complexity. **Structured** P2P networks connect peers in specific topologies to enable efficient and deterministic routing, though they are complex to maintain. **Unstructured** P2P networks connect peers randomly, making them easy to set up and resilient to failures but are more inefficient for search and resource discovery. **Hybrid** P2P networks combine centralized and decentralized elements, using a central server for managing connections and indexing while allowing direct peer-to-peer resource sharing, improving search efficiency but potentially introducing bottlenecks or single points of failure.
Gossip protocols [18] refers to a class of communication protocols used in P2P networks to disseminate information in a decentralized manner. Gossip protocols are inspired by the way gossip spreads in social networks, where information is shared randomly between peers until it reaches the entire network. In a gossip protocol, each peer periodically selects a random subset of other peers and shares information with them. These peers then repeat the process, leading to rapid and widespread dissemination. Gossip protocols efficiently spread information in large networks and are robust against peer failures due to redundant sharing. However, they can generate significant network traffic due to redundant message exchanges and ensuring all peers have consistent information can be challenging and may take a long time for the information to be fully disseminated. Gossip protocols are used for tasks such as updating routing tables, disseminating configuration changes, and maintaining consistency in distributed databases.

DHT

Distributed Hash Tables (DHTs) [104] are distributed key-value store architectures over P2P networks (Figure 2.1). Each node in a DHT is responsible for a portion of the data, the system ensures that the data is evenly distributed and that the system can scale efficiently as nodes join or leave the network.
In a DHT, each node maintains a routing table with information about a subset of other nodes, allowing it to forward queries to the appropriate node responsible for a given key. This enables efficient lookups, typically with a logarithmic time complexity relative to the number of nodes in the network.
DHTs are commonly used in structured P2P networks to manage resource locations and facilitate efficient search and retrieval. Examples of DHT implementations include Chord, Kademlia [13], and Pastry. These systems are designed to be robust, scalable, and fault-tolerant, making them suitable for large-scale distributed applications such as file sharing, distributed storage, and decentralized applications.
In modern unstructured P2P networks like IPFS [124], Kademlia [13] is used as they do not make assumptions on the network topology and need failure resilience. Kademlia allows to look for a value in a $log(n)$ (n size of the network) time, which makes it scalable.

Blockchains

Blockchains [61] are decentralized and distributed digital ledgers that record transactions across multiple computers in a way that ensures the transparency, and immutability of the data (Figure 2.1). Each record, or "block," contains a list of transactions and is linked to the previous block through cryptographic hashes, forming a "chain" of blocks. This structure ensures that once a block is added to the blockchain, its data cannot be altered without changing all subsequent blocks, which would require consensus from the majority of the network.

Blockchains operate on a P2P network where each participant, or "node", maintains a copy of the entire blockchain. Transactions are validated and added to the blockchain through a consensus mechanism, such as Proof of Work (PoW) or Proof of Stake (PoS) [52], which ensures that nodes agree on the validity of the transactions.

Blockchains can be extended run small programs that are called smart contracts [56] or decentralized apps where custom transactions are made.

Blockchains show scalability issues in the number of transactions they can handle as they must be disseminated in the whole network [94]. Transactions also take time to be considered validated (e.g. a bitcoin transaction is considered validated after one hour). They have privacy issues since everyone can see all of the transactions and the different consensus protocols [38, 52, 89] provide different security and performance guarantees.

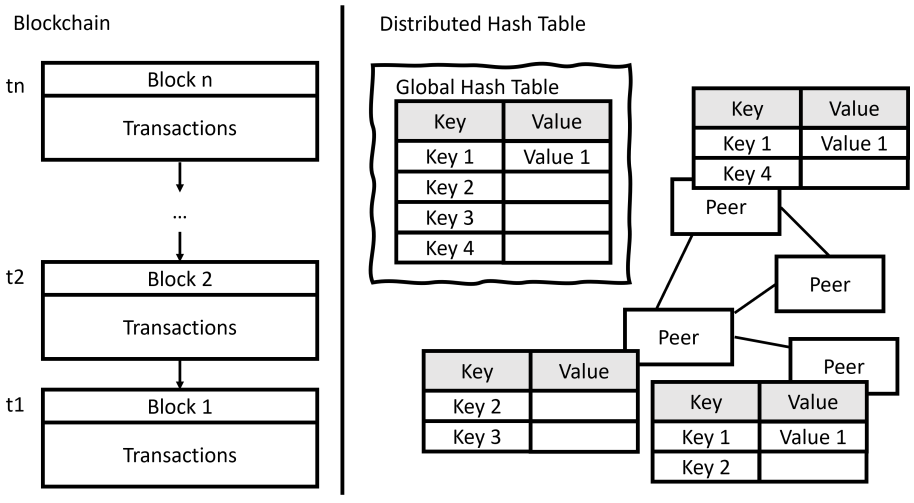


Figure 2.1: Blockchains and Distributed Hash Tables

2.2.2 Performances evaluation

Various dimensions are utilized to assess performances of distributed systems, such as scalability, resilience, latency, resource consumption (CPU, Memory), and throughput, each with its own set of metrics. Often, compromises must be found among these dimensions [185]. For instance, positioning further in the IoT cloud continuum enhances throughput, albeit at the expense of increased latency.

2.2.3 Security issues and properties

In distributed systems, entities typically trust each other, which simplifies many security concerns. However, this inherent trust is not present in decentralized settings, leading to a range of security challenges.
One major issue is protecting against Sybil attacks [12], where a single entity creates multiple identities to gain disproportionate influence or voting rights within the network. Byzantine attacks [11] pose a significant threat, as malicious entities can disrupt common decision-making processes by acting dishonestly.
Additionally, verifying the integrity of processing and guaranteeing the privacy of data becomes increasingly difficult when the execution environment is distant and not under direct control [200]. Privacy issues are further compounded by the need to ensure that sensitive information is not exposed or misused.
Moreover, decentralized systems are vulnerable to attacks against availability, such as coordinated failures and Distributed Denial of Service (DDoS) attacks, which can severely disrupt network operations.
Addressing these multifaceted security concerns requires robust mechanisms and protocols to ensure the reliability, integrity, and confidentiality of the system.

2.3 Security and privacy counter-measures

We identify here the different security countermeasures used in this thesis and their associated properties. They can be added to AI or distributed systems to provide security properties. Adding these security measures often incur performance overheads, be it in compute, memory or bandwidth usage. The combination of multiple countermeasures is often necessary to address different security properties, as no single measure can offer comprehensive protection. However, combining countermeasures depends greatly of the architecture of the system to be protected since they may not be compatible with each other.
We review the countermeasures for three security properties: privacy, integrity and availability. **Privacy** refers to the protection of data from unauthorized access, ensuring that only authorized individuals can access sensitive information. **Integrity** involves maintaining the accuracy and completeness of data, ensuring that unauthorized parties do not alter information. **Availability** ensures that information and resources are accessible to authorized users when needed and that systems function reliably.

2.3.1 Cryptography

In this subsection we list the main cryptography tools and their corresponding guarantees.

Symmetric encryption schemes allow to guarantee the privacy of the content sent from A to B. A and B have both the same private key k that is used to encrypt the message on the side of A and decrypt it on the side of B. AES is the most common algorithm, it is standardized and recommended by the NIST.

Asymmetric encryption schemes guarantee the privacy of the message sent from A to B. For that B generates two keys, a public key pub_k and and a private key $priv_k$ *generate_key*. It shares pub_k with A and keeps $priv_k$ secret. pub_k is used by A to encrypt the message sent to B with *encrypt(*pub_k*, message)*. $priv_k$ is used by B to decrypt the received message calling *decrypt(*$priv_k$*, message)*. This schemes allow to exchange private messages without the need for a common secret key to be established between A and B. Asymmetric encryption schemes require far more computing resources than symmetric scheme. They are combined into protocols such as SSL where they are used to exchange an encryption key used with a symmetric schemes. Current schemes are based on the difficulty to solve mathematical problems such as integer factorization and discreet logarithm problems in the case of RSA [1]. However, algorithms have been proposed to solve those problems using quantum computers. The NIST has opened a competition [41] to standardize new schemes that are quantum computer resistant. This warrants the change of the asymmetric schemes and the associated private keys in the existing infrastructures.

Cryptographic signatures are schemes to authenticate the author of a content and guarantees that it was not altered in the transit. They may be related to asymmetric encryption, but not necessarily. They provides two functions: *sign(content,* $priv_k$*)* that allows the author of a content to generate a signature acting as a proof of authorship and *verify(content, signature,* pub_k*)* that allows to verify that the signature is valid. Cryptographic signatures are often used on the hash of the content instead of the content itself for performance reasons.

Hash functions [2] such as SHA-256 are one-way way functions that transform an input into a fixed-size string of bytes. They allow to verify the integrity of the content by comparing the hash of a wanted content to the one of the received content. They are one-way, so they attempt to prevent to find possible inputs for a specific output and guarantee the privacy of the input. CPU instruction sets and hardware accelerators have been designed to accelerate the computation of hash functions.

Homomorphic Encryption (HE) [53] is a form of encryption that allows to perform compute tasks on the encrypted data. The results are then left in an encrypted state. This guarantees the privacy of the input and output. HE schemes may allow only a limited set of instructions, fully homomorphic encryption schemes allow for arbitrary computations. HE schemes are currently multiple order of magnitudes more compute intensive than their non-encrypted counterparts and encounter, thus are rarely deployed.

Multi-party computation [66] are schemes that allow for parties to jointly

compute a function over their inputs while keeping those inputs private.
Secure aggregation [152] is computing a function of the inputs of multiple sources without disclosing the individual inputs. They feature approaches such as secure multi party computation or hardware protections like TEEs. They are used in cases like federated learning [40] and aggregating IoT devices results.

2.3.2 Election protocoles and consensus

In distributed systems, there is a need for consensus between multiple component systems to take a common decision. However, some of the systems taking part in the decision may be malicious or fail. So we want to guarantee the integrity of the common decision.
We identify two families of approaches to reach consensus [89]:
Byzantine Fault Tolerance (BFT) [11] schemes are mechanisms designed to achieve consensus in distributed systems despite the presence of faulty or malicious nodes. They ensure the system can continue to operate correctly and provide consistent results even when some components fail or act maliciously, usually when their number is under 1/3 of participants.
In **Proof of X (PoX)** approaches [12, 89], every systems proves that it has a specific resource (e.g. compute, money). The voting rights are proportional to the amount of a specific resource. Proof of Work, used in blockchains like bitcoin, uses the device computing power has the proof. Intel has proposed proof of luck [38] that uses the TEE featured in its CPUs as the proof, this allows to give exactly one vote to each device without wasting ressources such as computing power.
Consensus protocols are susceptible to **sybil attacks** [12] where a single entity creates multiple identities to get more voting rights, impacting to the integrity of the consensus. Douceur [12] identified two ways to tackle those attacks: featuring a centralised organisation that provides a single identity by entity, and, proof of X approaches.

2.3.3 Reputation systems and access control

It is considered a best practice to minimize the attack surface of systems by ensuring that components have access to the least amount of resources necessary (e.g., data, computational power).
A proactive approach involves the identification and minimization of access rights required for resources during the system design phase using **Access control policies** [7]. This results in lists of authorized or blocked accesses for users or components, which can be represented in a matrix format. The challenge lies in identifying the required accesses for each user and limiting access accordingly, especially when there is a large number of users and resources. Those approaches can be analyzed using formal systems to detect incoherences in the policies.
A reactive approach involves restricting access based on user behavior. This can be implemented through systems such as intrusion detection systems (IDS)

in charge of monitoring and reacting to attack in real-time. The detection of malicious behaviors can be disseminated across systems using reputation systems [33] to evaluate and report agent behavior for improved performances.

A combination of both approaches is usually done, where access control policies serve to complicate potential attacks, while active monitoring integrated with reputation systems enhance the response once an attack has occurred.

2.3.4 Remote attestation

Remote attestation [193] verifies the integrity and authenticity of a remote software. It involves generating a cryptographic proof, often using a specific hardware [51] that attests the current state of the system's software and hardware configurations. The verification of the proof allows to verify the integrity and authenticity of the remote software.

Remote attestation can be done in swarms (swarm attestation) [149] where, for performances reasons, each system does not need to attest all of the other systems in the swarm.

2.3.5 Trusted hardware and isolation

To guarantee the integrity and privacy of compute and data, some solutions propose to isolate parts of the system from the rest, limiting their interactions and so their surface of attack. We identify here hardware and software isolation solutions.

TPMs (Trusted Platform Modules) [51] are secure cryptoprocessors. They help ensure the integrity and security of the system by protecting and verifying its boot process.

A **Trusted Execution Environment (TEE)** [77] is a secure area within a processor that ensures the integrity and confidentiality of code and data loaded inside. It provides an isolated environment where sensitive computations can be executed securely, protecting them from a potentially compromised main operating systems. They feature remote attestation functions to verify that a workload is running in a TEE. TEEs had very limited in memory at the beginning (20 MB for Intel SGX) but this limit tends to be overcome with new architectures such as Intel TDX. TEEs also show computing overheads that tend to be of 5% [113]. They are becoming available on GPUs, which make them even interesting for confidential machine learning workloads [99].

Isolation can also be done with software solutions like **virtual machines** [64] and **containers** [54] that do not need any specialized hardware. They guarantee the integrity and privacy between workloads running on the same machine. But they do not protect from attacks coming from the supervisor managing those workloads.

2.3.6 Anonymisation tools

Some countermeasures aim to guarantee privacy by reducing the amount of information observable by an attacker. We distinguish here anonymisation tools for networks that aim to hide the identity of a message sender, and data anonymisation tools that aim to limit the information exploitable from data to the minimum necessary for their use.

Network anonymisation

We distinguish two ways to guarantee the privacy of a message sender's identity on a network.
Mixnets (or mixing networks) [17] are cryptographic systems designed to anonymize communications by shuffling messages. They obscure the relationships between senders and recipients by passing messages through multiple intermediate nodes, where each node mixes and encrypts the messages before forwarding them.
Onion routing [15] is a technique for anonymous communication over a computer network. In an onion routing system, messages are encapsulated in layers of encryption, similar to the layers of an onion. The message is sent through a series of network nodes called onion routers, each of which peels away a single layer of encryption to reveal the next destination. This process continues until the message reaches its final destination, making it difficult for any intermediary to determine both the origin and the final destination of the message. Onion routing is used in the TOR project that enables anonymous communications over the internet.
Both of this approaches require the use of partially trusted intermediates to obfuscate the message source. An attacker would need to breach multiple intermediates. However, they limit the available bandwidth and increase latency in communications. Furthermore, tracking through behavior monitoring [24] remains possible, use information such as time of connection.

Data anonymisation

To guarantee privacy when sharing data such as sharing training data used for ML models, approaches have been proposed to limit the disclosed information with statical guarantees. We distinguish two main approaches based on the type of privacy guarantees:
k-Anonymity [20] is a privacy-preserving technique used to protect individual identities in a dataset. It ensures that each record is indistinguishable from at least $k-1$ other records with respect to certain identifying attributes, known as quasi-identifiers. This means that any given individual cannot be uniquely identified within a group of k individuals, thereby reducing the risk of re-identification. It is hard to apply as there is the need to process the data and to manually identify anonymisation techniques (such as removing the birth dates and leaving only the birth year) for each use-case.

Differential Privacy (DP) [37] provides a probabilistic indistinguishability guarantee for processing a specific data sample during training. This means that a specific training sample cannot be extracted from the trained model – the training data set probability distribution can still be leaked. It is used in training deep learning models [37]. However, the performance degrades when we increase the level of privacy.

2.3.7 Composition of security priorities and mechanisms

No counter-measure fits all security requirements and they often have an impact on the performances of the systems they protect. This warrants their combination to obtain the desired security properties.
Such combinations may be specific to the use case. We could want to have slightly different properties on the different axes (e.g; different performance trade-off between latency and throughput). This warrants the use of properties on demand as in aspect-oriented programming [6], where the developer selects the desired properties and a weaver makes the technical choices and adapts the program.
Security countermeasures may also be incompatible. For example, the byzantine protection scheme FLAME [130] for federated learning guarantees the integrity of the training, but does not provide privacy of the trained model and is not compatible with secure aggregation [152] schemes that provide this guarantee.

As we saw, techniques have been identified to answer to the requirements of the three axes that are functional, performance and security with respectively intelligent systems, distributed systems and security countermeasures. This warrants the combination of these three axes to build secure and efficient cooperative autonomous systems. There are multiple works two combine two of these axes: Federated and Distributed Learning [106] combine machine learning schemes and distribution schemes. Ekiden [56] combines TEEs for security and blockchains for distribution. Combinations involving three axes are more rare but exist (e.g. secure federated learning [110]). A combination of these axes on the general case of autonomous systems and their cooperation is lacking.
Furthermore, those combinations may be specific to the use case. We could want to have slightly different properties on the different axes (e.g; different performance trade-off between latency and throughput, different security guarantees). This warrants properties on demand as in aspect-oriented programming [6], where the developer selects the desired properties and a weaver makes the technical choices and adapts the program. Current architectures are also often difficult to adapt to specific performance and security constraints. For example, the byzantine protection scheme FLAME [130] for federated learning guarantees the integrity of the training, but does not provide privacy of the trained model and is not compatible with secure aggregation [152] schemes.

Part I

Cooperative Autonomous Systems

In this contribution, we focus on formalizing Autonomous Systems and identifying the patterns in the disaggregation of AI.
In Chapter 3 we formalise Cooperative Autonomous Systems with our RACAS reference architecture for cooperative systems. We study security and performances of cooperative systems. We identify cooperation means and how they are placed in the architecture.
In Chapter 4, we study AI disaggregation by highlighting three axes vertical/horizontal, data or model disaggregation, hardware continuum. We explore the trade-offs between those axes and illustrate with the case of disaggregated LLM performances.

Associated publications:

- **Divi De Lacour, Marc Lacoste, Mario Südholt, Jacques Traoré**
  Towards a Reference Architecture for Secure Cooperative Autonomous Systems for IIoT
  1st Workshop on Cloud-edge Systems in Industrial IoT Applications (CE-IIA) - 28th Conference on Innovation in Clouds, Internet and Networks , mars 2025, Paris, France,
- **Mohamed Ghamri, Marc Lacoste, Divi De Lacour**
  Disaggregation Patterns for Secure AI Systems
  HPEC 2024 - 28th Annual IEEE High Performance Extreme Computing Virtual Conference, Sep 2024, Boston (MA), United States,

Chapter 3

A Reference Architecture for Secure Cooperative Autonomous Systems for IIoT

Cooperative Autonomous Systems (CAS) are explored in multiple fields of Industrial IoT systems (IIoT) to solve complex tasks, sharing information and task execution. CAS have different objectives and are usually controlled by different administrators. Cooperation also comes with security and privacy challenges. Architectures have been proposed for limited use-cases or for specific security and performance issues. We argue that a unified reference architecture for CAS is needed for security, efficiency and customization. We propose such an architecture called RACAS to design efficient and modular CAS and show how it can be secured on-demand. We illustrate how RACAS can be applied to an intelligent transportation system including networks of autonomous and connected vehicles.

3.1 Introduction

Autonomous Systems (AS) have been increasingly popular for multiple Industrial IoT (IIoT) use cases, sensing the environment and reacting to meet system objectives [8, 161]. In the transportation area, intelligent vehicles have been reaching increasing levels of autonomy for self-driving applications. Autonomous networks is another use case to deploy self-configuring network architectures for Web 3.0 applications [183]. The MAPE-K framework [14] is widely used to formalize AS. It identifies 4 key properties for self-management related to configuration, optimization, healing and protection.
Cooperative Autonomous Systems (CAS) extend AS as systems that interact

to jointly solve tasks or maximize utility [16]. Cooperation may take different forms such as CAS broadcasting their status, delegating tasks to other CAS or training distributed machine learning models [160].

A typical connected cooperative and autonomous mobility use case is Anticipated Cooperative Collision Avoidance (ACCA): vehicles cooperate to detect and anticipate collisions in blind spots [76]. For such safety-demanding use cases, proving and verifying that AS behave according to requirements in any given situation remains difficult, notably to guarantee trustworthiness for machine learning based systems [155].

Cyber-resilience is a focal challenge for CAS security, with tight interplay between safety and security and multiple security and privacy vulnerabilities. Threats may come from AS components (e.g., LLM security [199]), communication between components (e.g., in-vehicle network security [35]) or cooperation protocols (e.g., security of collaborative learning [126]). CAS may have conflicting cyber-resilience objectives. For instance: mitigation of propagation of safety breaches or spurious data to other AS vs. protecting against access to private information, such as vehicle position monitoring.

CAS should meet various performance requirements, alone or in cooperation. *Latency* is critical for collision detection in vehicular systems. *Scalability* is also important, as autonomous networks need to scale with the number of participants.

Different architectures have been proposed for several CAS use cases. The GANA architecture (Generic Autonomic Networking Architecture) [43] has been standardized at ETSI for autonomous networks. [127, 155] advocate for a foundation architecture for AS. [134] reviews emerging research directions in this area. However, a comprehensive overview of CAS architectures is lacking. Performance, security and trustworthiness insights remain hard to transfer between architectures and use cases due to specific approaches and optimizations. AS cooperation is usually hidden in the architecture or limited to a specific use case (e.g., sharing localization) and remains difficult to identify and secure.

In this chapter, we propose RACAS, a reference architecture for CAS to formalize cooperation and enhance efficiency. We show how security and privacy can be integrated in the architecture. We illustrate how the RACAS architecture can be applied in practice, taking as example an intelligent transportation system use case.

The chapter is structured as follows. Section 3.2 reviews related work. Sections 3.3 and 3.4 present RACAS and security aspects respectively. Section 3.5 describes a typical RACAS application to intelligent transportation systems before concluding.

3.2 Autonomous systems

We review some landmark works on AS and challenges in terms of performance and cooperation.

3.2.1 The MAPE-K Framework

The MAPE-K [14] framework standardizes AS reasoning and interactions with its environment. It features a control loop including four components (Figure 3.1):

- **Monitor**: checks the AS status through sensors.
- **Analyze**: processes observations and derives insights.
- **Plan**: based on those insights, determines the most appropriate course of action given the AS goals and constraints.
- **Execute**: realizes the planned actions through actuators.

All components have access to a **Knowledge** component that represents the AS inner knowledge.

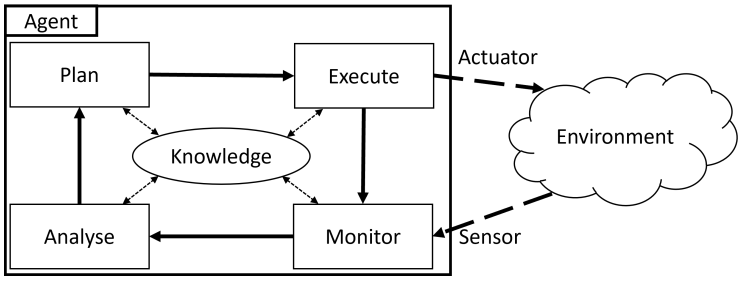


Figure 3.1: MAPE-K self-management loop

The MAPE-K design pattern does not explicitly account for cooperation, implicitly integrated in the different components. Most approaches related to cooperation adopt different formalisms that cannot be generalized. MAPE-K is also a single control loop that does not support the execution of sub-tasks that require low latency. This is challenging for performance as sub-tasks must wait for the whole loop to be executed. Failure of any component in the loop may induce that of the whole system. MAPE-K does not take into account security and privacy either. It considers components as trusted and does not support access control nor isolation.

3.2.2 Multi-Component Architecture

NIST proposed the 4D-RCS [10] architecture for unmanned vehicles. 4D-RCS is a hierarchical control system arranged in a tree structure. Parent systems are in charge of long-term objectives. Sub-systems implement short-term objectives with faster decision loops. This design is inspired from military organizations where each level of management follows the objective given by the upper layer while handling observations and feedback from lower layers. This design allows verification and analysis of each sub-system separately based on their specific goals. However, in 4D-RCS, every control system should be coordinated with sub-systems [155].

3.2.3 AS Challenges

In [155], the authors advocate a foundation for developing AS to recommend engineering practices and methods. They identify three challenges:

- **Behavior specification:** for each level (assuming a hierarchical AS architecture).
- **Analysis:** to verify and prove system trustworthiness.
- **Combining model- and data-driven approaches:** to benefit from both machine learning approaches, that offer high accuracy but lack explainability and trust, and from expert system approaches.

3.3 The RACAS Architecture

We now describe the RACAS architecture that extends MAPE-K for cooperation and improved performance. It standardizes the places of cooperation in CASes and shows how the components can be distributed for better performance.

3.3.1 Extending ASes for Cooperation

Cooperation between AS may fail due to errors in communications, failures from other AS or malicious behaviors related to shared information. Therefore, cooperation functionalities should not be trusted. In terms of system design, such functionalities should not be placed in the core of the system in order to protect the AS from cooperation hazards.

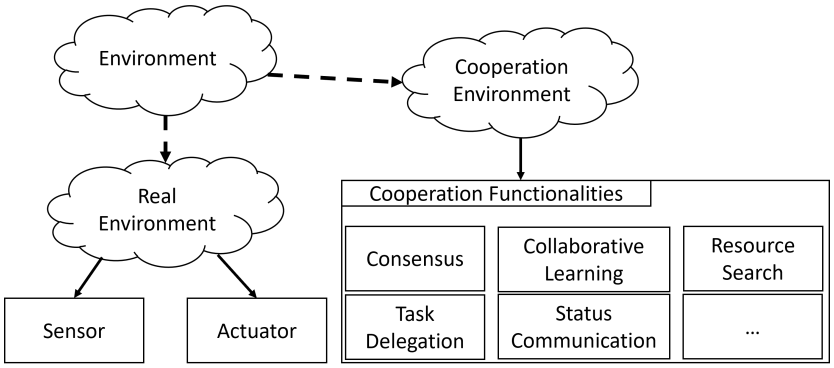


Figure 3.2: AS environment: cooperation functionalities

To extend MAPE-K for cooperation, we propose to place all cooperation functionalities in the environment (see Figure 3.2). The AS monitors and reacts in the cooperation environment as it would in its real environment.
More specifically, we consider the environment to be composed of:

- **The real environment:** the AS evolves in this setting to reach its goals (e.g., real-life, simulation, video game).
- **The cooperation environment:** this virtual environment is created between the ASes to communicate and provide cooperation functionalities.

This design enables to handle failures or errors in cooperative observations and actions similarly to a real environment. The cooperation environment is separated from the real environment to guarantee CAS availability, even when the cooperation environment is unreachable due to communication failure.
We identified the following key functionalities for a cooperation environment. They can be implemented in a centralized or decentralized setting depending on requirements (e.g., latency, privacy policy):

- **Consensus**: a subset of components can establish a contract on tasks and transactions to perform complex tasks. Contracts can be established in a decentralized setting using blockchain protocols [90] or a centralized arbiter. Consensus protocols may help addressing the case of presence of rogue nodes in the cooperation environment to enhance overall system robustness.
- **Collaborative learning**: AS can create new knowledge from their individual experiences, using their local data to train a common machine learning model [107, 160].
- **Resource search**: AS can search for specific resources (e.g., service, file) in the network of AS using their ID [13] or using semantic search [50].
- **Task delegation**: AS can delegate tasks to other AS such as executing functions [154].
- **Status communication**: AS can broadcast their status to one or multiple AS, or even to humans as in social networks [81, 136].

We consider that each cooperation function is run independently. Such functions are isolated in their own execution environment, and called by an AS through Monitor and Execute components.
The cooperation environment is agnostic to the type of connection that is under the responsibility of the cooperation function.
Connections may be: direct, low-latency using wireless communications (e.g., V2V); infrastructure-based at the Edge or in the Cloud (e.g., V2I); or hybrid (e.g., V2X), the type of network changing dynamically, or being different for each cooperation function.

3.3.2 Improving Resilience and Performance

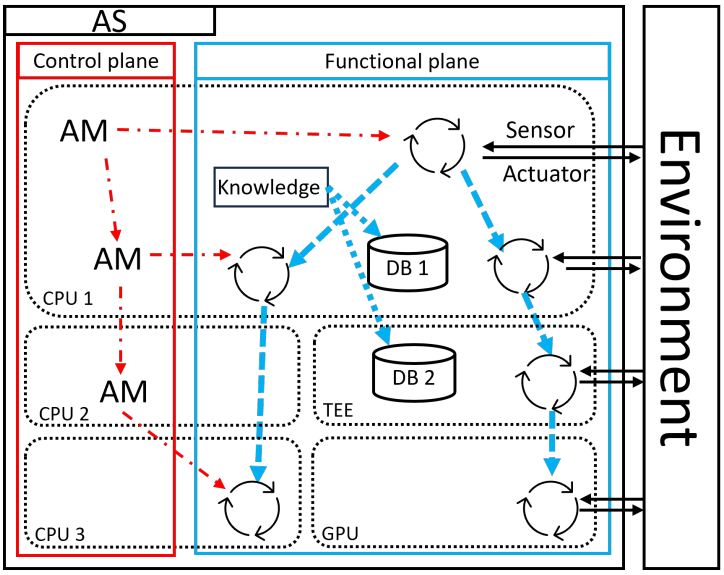


Figure 3.3: AS inner architecture
AM: Autonomic Manager

To overcome the MAPE-K model resilience and performance limitations due to the single-loop structure, we extend the autonomic design with a hierarchical architecture shown in Figure 3.3 [10, 43].
We take a closer look at the AS inner structure. The AS functional plane is built from a hierarchy of autonomic MAPE-K loops. Each loop fetches information from the external environment through sensors (Monitor component) and reacts on the Environment through actuators (Execute component). The loop decision logic may create sub-loops that can either be data or model driven.
Each loop has access to the same knowledge base (Knowledge component). Such knowledge may be implemented as multiple databases [153] (e.g., real-time, domain-specific knowledge graph) with no assumption on format, allowing for more adaptability.
A parent loop monitors its direct children loops. Sub-loops inherit the environment and knowledge access of their parents. We consider that by default the parent loop has the priority on its children. Exceptions can be defined for control loops handling emergency cases (e.g., emergency braking). Sub-loops are asynchronous to improve performance so that sub-tasks may meet more easily low latency requirements. Such a design also improves resilience, to prevent failure propagation from sub-loops to the whole system with a component based approach.
This architecture also allows more deployment flexibility, enabling to distribute tasks/components on different execution environments (e.g., process, CPU, GPU,

machine, edge, cloud). Execution Environments may interact through different types of network connections with different requirements (e.g., latency, bandwidth, resilience). Knowledge can be distributed in execution environments for better performance and reliability, e.g., placing real-time databases closer to critical sub-systems.

3.3.3 Self-Management

For AS self-management, an Autonomic Manager (AM) is usually added for monitoring and controlling the autonomic behavior of managed elements [9, 14]. In addition to the functional plane, we introduce a control plane in the AS architecture. We selectively encapsulate each decision loop or loop component in an AM to supervise those components (Figure 3.3). The AM monitors the inner state of components and their communications, either with the environment, other components or the Knowledge component. Each AM also manages the AMs of sub-loops. This results in the tree architecture for AMs shown in Figure 3.3. The AM includes primitives for a component to be supervised by a specific sub-AM on-demand.
An AM may cover selectively several self-* properties (e.g., self-protection and self-healing), each of them being handled by specific sub-AMs.

3.4 Security and privacy

We now delve into the security and privacy challenges of our architecture. We review the security properties to guarantee and analyze threats and approaches for mitigation.

3.4.1 Security Properties

We consider three security properties to guarantee for a CAS:

- **Availability**: the CAS continues working as expected.
- **Integrity**: no tampering with CAS data/execution flows.
- **Privacy**: no illegitimate access to system data.

Attacks include manipulating an AS component to access private data (from Knowledge or Sensor components) or tamper with data (knowledge) or actions (execution flows, actuators). They can also originate from outside the AS regarding cooperation (e.g., collaborative learning threats [126]) or communications.

3.4.2 Access Control and Isolation

Access Control – To enforce AS security, we apply an access control system to components, following the tree architecture. This approach facilitates

analysis of authorizations and logging of access to resources. Component authorizations include permissions to send messages to other components or to access the Environment and the Knowledge component. Components inherit their authorizations from their parents following the functional part of the architecture. Like access rights management, this enable to enforce the principle of least privilege for leaf components, more customized, and less worthy of trust.
Isolation – To improve security, components can be isolated in different execution environments. To guarantee integrity and privacy, a component can be placed inside a Trusted Execution Environment (TEE) [77]. TEE guarantee hardware-level isolation using encrypted memory to guarantee run-time privacy and integrity of code and data. TEE trustworthiness can be verified externally using remote attestation.

3.4.3 Autonomic Security Manager

Access control and isolation are *proactive approaches* to limit malicious behaviors of AS components. To go beyond, we introduce a *reactive* mechanism in the AM to detect and mitigate malicious behaviors in the AS components. It realizes component self-protection by actively monitoring the behavior, state and communications of each component in the AS. It can be implemented as an intrusion protection system monitoring AS inter-component communications and isolating malicious components. A typical management policy could be inspired from simplex architectures [9] that replace an optimized component by a simpler and more trusted one in case of security crisis. Note that a parent AM monitors its sub-AMs in case they behave maliciously.

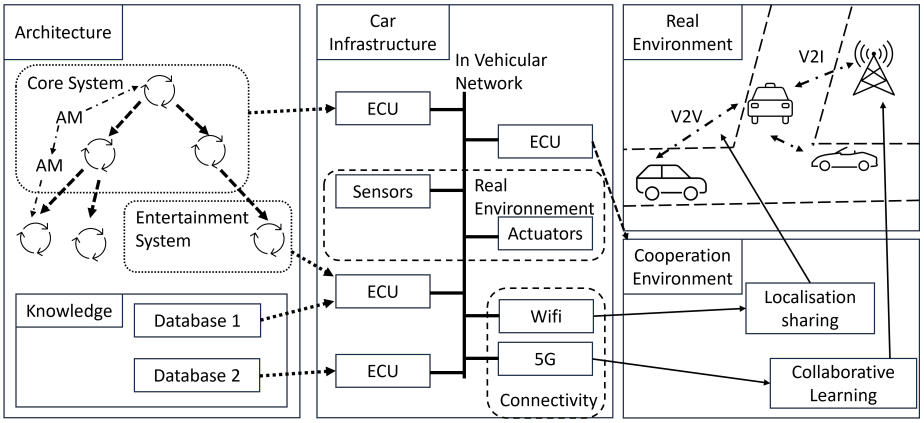
3.4.4 Cooperation Security

We isolate cooperation functions from AS inner components by placing them in the cooperation environment. This enables to monitor information entering and leaving the AS. Cooperation may only occur by an AS taking an explicit decision (Execute Component) to share information outwards or to receive information (Monitor Component). Threats originating from outside of the AS such as attacks on collaborative learning [126] are handled by the cooperation environment. Each cooperation function is in charge of its security due to the diversity of security properties to guarantee the variety of existing use cases (e.g., centralized, decentralized).

3.4.5 Security, Privacy vs. Performance

Adding security and privacy features in most cases decreases performance (e.g., TEE overhead [77]). Security and privacy may also be at odds with one another (e.g., protecting data makes it harder to detect malicious behaviors). Better integrity may require more information to be shared than directly needed. The AM may also create latency overheads due to communication interception and component introspection.

(a)



(b)

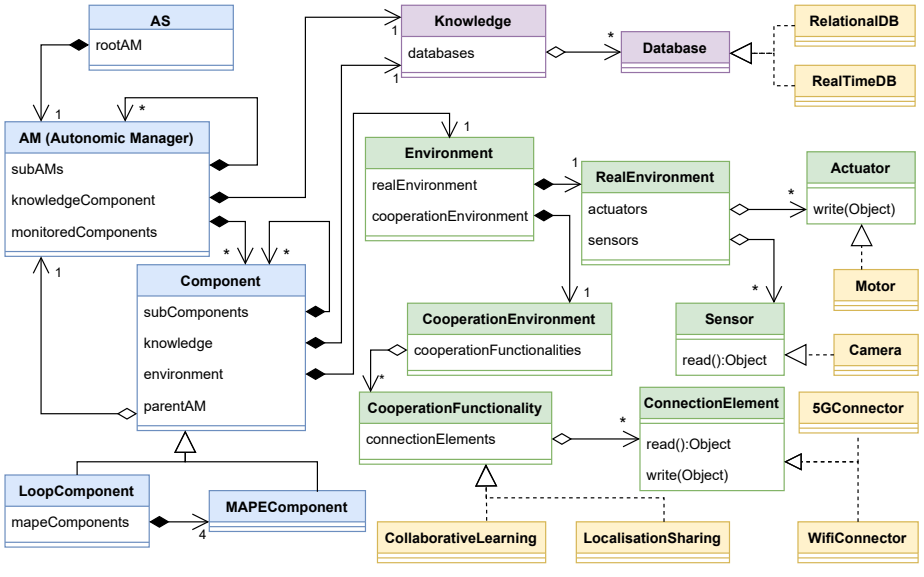


Figure 3.4: Applying RACAS to an intelligent transportation system: (a) architecture; (b) meta-model

3.5 Use Case: Intelligent Transportation System

We now illustrate the application of the RACAS architecture for an intelligent transportation system. Figure 3.4(a) shows how decision components, AMs and knowledge base can be distributed across multiple Electronic Control Units (ECUs) in an in-vehicle network. The components that interact with the real environment (sensors and actuators) and with the cooperation environment (Wi-Fi and 5G connectors) are also connected to the in-vehicle network. Critical and entertainment systems are isolated on different ECUs with different authorizations. Regarding the Knowledge component, databases are distributed on different ECUs to be closer to their users. A separate ECU handles cooperation functionalities through connectivity components such as location sharing through V2V Wi-Fi and collaborative learning through V2I 5G networks.
Figure 3.4(b) illustrates the software architecture of RACAS for this use case. In the architecture, only a small set of components are use case specific (shown in yellow), tending to show that RACAS could easily be extended to other verticals.
The AS features a root AM which monitors the root component and its sub-components. The AM may create a new sub-AM to manage sub-components. This sub-AM is under supervision of the parent AM. A decision loop component features the MAPE components – only Execution and Monitor having access to the environment. The Knowledge component is implemented as a set of databases. It can be selectively given access to a subset of databases to sub-components. Similarly to the Knowledge component, the Environment component can be invoked to limit authorizations. It features a real environment with sensors to gather information from the outdoor environment and actuators to react on it. The cooperation environment features cooperation functions with specific interfaces – including to observe and act on the cooperation environment. Such functions may use connection elements to communicate with the outdoor environment.

3.6 Conclusion

In this chapter, we introduced RACAS, a reference architecture to enhance CAS efficiency, security, and customization. We extend the MAPE-K model considering cooperation as a feature of the environment. This approach allows addressing the integration challenges of multiple AS with different objectives. We identified the security challenges and applicable countermeasures. We illustrated how RACAS could be applied to IIoT systems using a typical intelligent transportation system architecture, but other verticals are also possible.
As future work, we intend to explore the definition of standard APIs for cooperation functionalities. We will also investigate the trade-offs between security, privacy, and performance to provide comprehensive design guidelines. A performance evaluation in particular is necessary to quantify the low-latency

dimension of RACAS. Broader research avenues include RACAS architecture refinement towards implementation and deployment, and proving safety and security guarantees, with potential automation, e.g., using AI or formal verification techniques.

Chapter 4

Disaggregation Patterns for Secure AI Systems

In Chapter 3, we proposed to separate Autonomous Systems in multiple decision loops, allowing to distribute them to improve the performance. The decision components of these loops (often AI based) were considered atomic, however, they may also encounter resource limitations. An approach to solve that would be to also distribute or disaggregate them.
Disaggregation is a growing trend in large-scale Artificial Intelligence (AI) systems to overcome hardware and software resource limitations and improve performance while preserving security and privacy. This chapter takes a closer look at different dimensions of the concept, in AI, security and hardware. We identify two key design patterns (horizontal and vertical disaggregation) that may be combined to build optimized disaggregated AI architectures and discuss benefits and limitations for AI and security. Using a large language model use case, we also highlight some key trade-offs between performance, resource allocation and security for different disaggregation strategies in hardware and in software.

4.1 Disaggregation in Computing

Researchers and developers have consistently faced the challenge of resource limitations in hardware and in software. Extensively explored, e.g., in software engineering and in networking, *disaggregation* is emerging as a key solution for large-scale artificial intelligence systems. This concept may be defined as separating a system into smaller elements or *components*. Foreseen benefits include optimization of resources, security and performance. Disaggregation may be applied to *AI*, *hardware* and *security* (see Figure 4.1).

AI Disaggregation. Software disaggregation distributes workloads across multiple computing units. This approach enhances computational capabilities to handle complex AI models and large datasets.

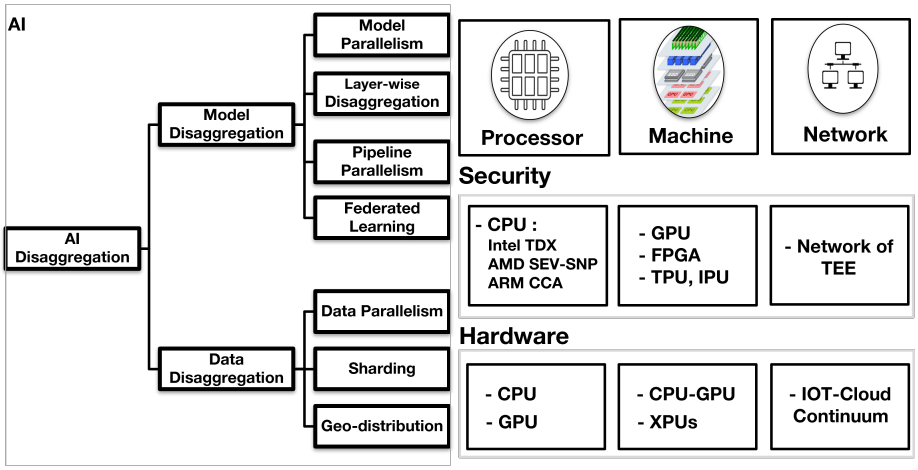


Figure 4.1: Overview of disaggregation: AI, hardware and security

1) Model disaggregation distributes computations performed in neural networks to improve efficiency and scalability. This set of techniques applies but is not limited to inference, training and fine-tuning.

For instance, *model parallelism* [206] distributes model parameters across multiple devices – enabling simultaneous processing of different parts of models too large to fit into single-device memory. *Layer-wise disaggregation* adopts different parallelization configurations for different layers of a neural network, e.g., for convolutional layers and fully-connected layers. *Pipeline parallelism* [187] divides a model into segments processed sequentially across hardware units, reducing significantly training time. Finally, *Federated Learning* [194] collaboratively trains a model across decentralized devices and is expected to improve data privacy.

2) Data disaggregation is another approach that distributes data across storage and computations for model training and deployment, with similar efficiency and scalability benefits.

Data parallelism replicates data across processors or machines, each copy handling a subset of the data. Training may scale out with faster convergence and management of larger datasets. *Sharding* partitions a large dataset into smaller elements that can be processed in parallel – improving performance, e.g., for database management systems. Finally, *geo-distribution* policies specify spatial data placement across data centers to reduce latency, increase fault tolerance and ensure compliance with data sovereignty regulations.

Hardware Disaggregation. Disaggregation may occur at various hardware levels, from single accelerators to groups of accelerators (XPUs) [207] up to High-Performance Computing (HPC) and cloud infrastructures. Such hierarchical disaggregation enables more scalable and tailored allocation of resources.

Within the processor (multiple cores, execution environments), computing tasks

may be run concurrently, optimizing processor capabilities. *On a single machine*, workloads may also be distributed across different processing units (e.g., CPUs, GPUs, other accelerators) to handle Computations may finally be expanded *between multiple machines* (in a data center, geographically dispersed) to process large-scale datasets and train large models.

Confidential Computing/AI. Security and privacy are key properties to guarantee as AI systems are increasingly distributed across multiple platforms. *At application-level*, techniques such as Secure Multi-Party Computation (SMPC) and homomorphic encryption enable private collaborative computation and encrypted data operations respectively. *At middleware level*, secure Kubernetes frameworks such as CNCF Confidential Containers (CoCo) or Constellation preserve data integrity for distributed AI workloads. *At hardware level*, Trusted Execution Environments (TEE) [206] enhance AI security, integrating TEE secure execution with GPU acceleration.

4.2 AI Disaggregation Patterns

We identify two main patterns for disaggregation: *Horizontal Disaggregation (HD)* and *Vertical Disaggregation (VD)* shown in Figure 4.2. Those patterns capture different strategies to organize and optimize processes and resources in AI systems, both at software and hardware levels. HD and VD may be used to scale out and scale up respectively.

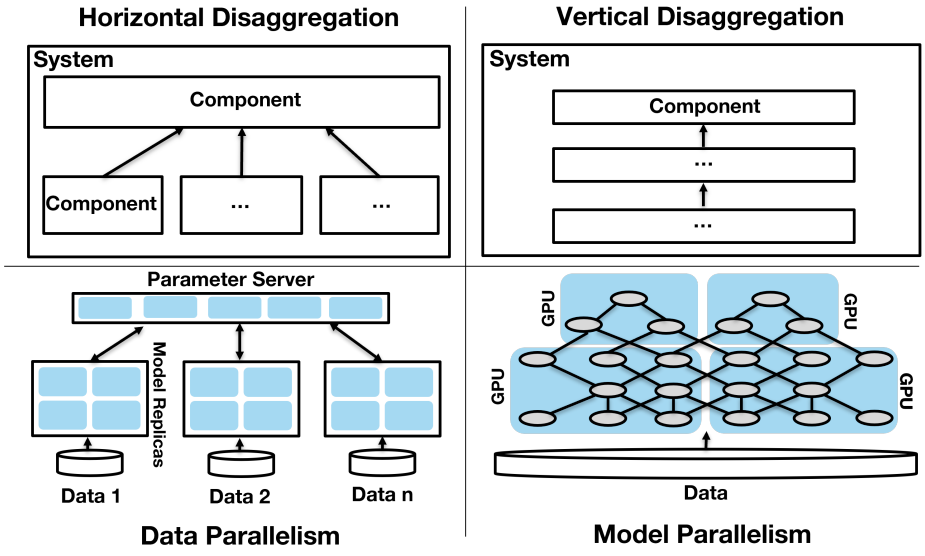


Figure 4.2: Disaggregation patterns and application to data/model parallelism

Horizontal Disaggregation. In HD, tasks are distributed across multiple

components, and run concurrently or in parallel, cooperating towards a common goal. HD is found in neural networks and in multi-core CPU architectures or hardware accelerators such as GPUs.
The main benefit of HD is scalability for distributed computations, reducing latency or increasing throughput. Component segmentation and distribution of responsibilities together with malicious behavior detection in aggregators can limit the impact of security breaches. Challenges include robust synchronization, as component coordination increases complexity. The attack surface may be enlarged as multiple components are interconnected.

Vertical Disaggregation. In VD, tasks are processed sequentially across different layers. Each layer is specialized in a particular function that must be completed before passing on to the next. VD is notably found in layer-by-layer propagation algorithms (forward and backward) for neural network architectures.
Benefits include layer-level optimization and upgrade of components. Isolation between layers is straightforward as the placement of protection mechanisms may be directly derived from the system architecture. The challenge is performance as each layer has to wait for the previous one for task completion.

VD vs. HD? HD and VD are usually used together. For instance, in Large Language Models (LLM), HD is used for data preprocessing on CPU cores, and HD/VD for training acceleration with GPUs. In federated learning, VD is used for propagation through neural network layers, and HD for model aggregation.

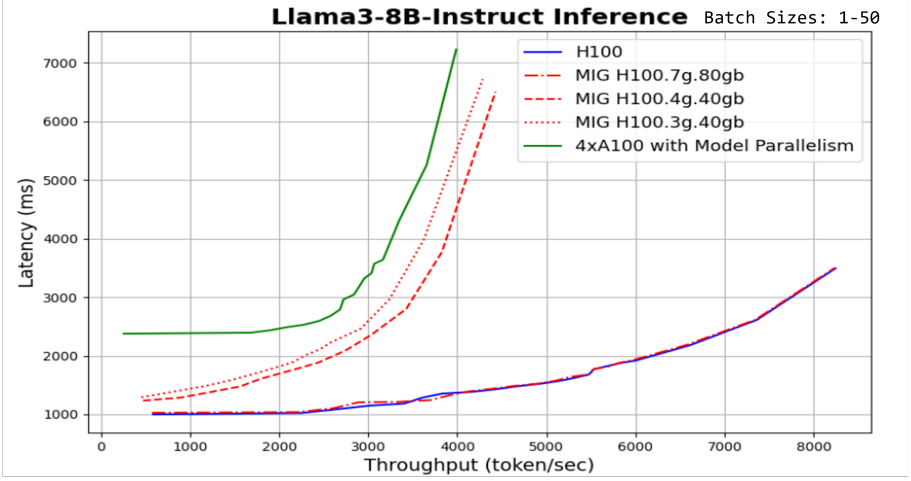


Figure 4.3: Throughput w.r.t. latency for single GPU, MIG GPU slicing and networked-GPUs hardware disaggregation configurations

4.3 Initial Experiments and Next Steps

Hardware Disaggregation: LLMs. We developed a use case to explore the impact of AI disaggregation in hardware, focusing on LLM inference. We assess

hardware performance (e.g., throughput, latency) and influence of batch size to reach optimal, secure and efficient deployment. The platform runs containerized LLM workloads using secure Kubernetes scheduling and supports several levels of disaggregation: 1) a single powerful GPU, 2) GPU partitioned into slices using NVIDIA Multi-Instance GPU (MIG) and 3) networked GPUs.

Preliminary scalability results are shown in Figure 4.3. The single GPU provides the best performance. Disaggregating the GPU using MIG provides flexibility to run multiple AI workloads concurrently but with performance degradation, which increases as GPU slices get smaller. Pushing hardware disaggregation further, networked GPUs over the cloud continuum using model parallelism yield even lower performance due to network communication. Increasing batch size illustrates HD performance benefits on throughput, but increasing latency. Regarding vertical scaling, the use of newer technology (H100 vs. A100 for GPUs) increases performance, in terms of latency and throughput. Such results could help finding the right disaggregation configuration depending on application requirements (e.g., latency, bandwidth, locality).

Conclusion. In this chapter, we identified and described the axes on which AI can be disaggregated, be it horizontally or vertically, on data or model or on software, hardware and security. We highlighted the trade-offs in performance with an LLM use-case for different disaggregation strategies in software and hardware.

Next Steps. We plan to confirm such findings on another use case on software disaggregation in Federated Learning. We intend to train several neural network architectures both locally and on cloud platforms to get more insight on disaggregation regarding security, performance and AI efficiency.

Part II

Collaborative learning

In this part we focus on the specific means of collaboration for Autonomous Systems that is collaborative learning. We confound it with Federated Learning which is a technique for systems to collectively train deep learning models with each client using its own training data without sharing those data with the other. We focus on the performance of FL through decentralisation and its security countermeasures combination.
In Chapter 5 we study the performance of FL on scalability and resilience and propose to decentralise it using a P2P network with the FDFL model [142]. Experiments show improved scalability the same resilience.
In Chapter 6 we review the security properties of FL, the available countermeasures and propose the Ti-skol architecture to combine those security countermeasures on demand. Experiments show that adding security countermeasures impact performance, even more when they are combined, and that there are incompatibilities in countermeasure composition.

Associated publications:

- **Divi De Lacour, Marc Lacoste, Mario Südholt, Jacques Traoré**
  Towards Scalable Resilient Federated Learning: A Fully Decentralised Approach
  PeRConAI 2023: 2nd IEEE Workshop on Pervasive and Resource-constrained Artificial Intelligence, IEEE International Conference on Pervasive Computing and Communications (PerCom Workshops), Mar 2023, Atlanta, GA, United States,
- **Divi De Lacour, Marc Lacoste, Mario Südholt, Jacques Traoré**
  Ti-skol: A Modular Federated Learning Framework Supporting Security Countermeasure Composition
  Big Data 2024 - Special Session on Federated Learning on Big Data, IEEE BigData 2024, Dec 2024, Washington, DC, United States, ,

Chapter 5

Towards Scalable Resilient Federated Learning: A Fully Decentralised Approach

Federated Learning (FL) can be defined as collaboratively training machine learning models on the data of local devices without having to move the data itself: a central server aggregates models. However, we observe that even though FL shows privacy and performance benefits, It also shows scalability and resilience challenges. In this chapter we present FDFL, a new fully decentralized FL model and architecture that improves standard FL scalability and resilience with no loss of convergence speed. FDFL provides an aggregator-based model that enables scalability benefits and features an election process to tolerate node failures. Simulation results show that FDFL scales well with network size in terms of computing, memory, and communication compared to related FL approaches such as standard FL, FL with aggregators, or FL with election, with also good resilience to node failures.

5.1 Introduction

Recent years have witnessed the emergence of a cloud-edge continuum providing efficient connectivity to the Cloud for massive numbers of edge devices. This new paradigm also ambitions the pervasive training of Machine Learning (ML) models for a wide spectrum of data processing applications. While traditional ML uses fully centralized data sets, it requires important communication resources and is subject to security and privacy concerns.
Federated Learning (FL) [44] is a first step to overcome these challenges. FL features a central server that sends the model to clients holding the training data.

Clients independently train the model on their local data and send back model updates to the server for aggregation to produce a new model. Unlike centralized learning, FL-related approaches are based on *decentralized architectures* – some of them being *fully decentralized* in the absence of a central node [96].

FL has demonstrated its value for the IoT [179]. But many limitations remain regarding performance, security, privacy, and fairness. Kairouz *et al.* provide a comprehensive review of the corresponding challenges and solutions [106]. In this chapter, we focus on FL *scalability* and *resilience* challenges. Scalability means that the server should adapt memory, bandwidth, and computation resources to support an increasing number of client nodes (scaling up). However, a central node limits the number of supported clients, scaling out not being an option due to the network communication overhead. Resilience aims to avoid interruption of training upon failure of the central server or of a significant number of clients. FL architectures may support the failure of client devices, but the central server remains a *single point of failure* [85, 106]. All those elements call for more *decentralized approaches* beyond FL.

Many designs have been explored at the level of models [44, 55, 57, 85] and frameworks [28, 32]. These approaches investigate different trade-offs between properties such as scalability, resilience, and convergence speed. A key factor is data locality, i.e., the amount of data dependency between neighboring nodes that strongly impacts communication overhead and model accuracy. Unlike strongly-coupled models, fully decentralized or local aggregator-based ones tend to show better resilience and scalability but with slower convergence. Models with less locality may have better performance but are harder to scale or are less likely to tolerate failures.

In this chapter, we propose FDFL, a new fully decentralized FL model and architecture that improves standard FL *scalability* and *resilience* based on a peer-to-peer network with no loss of convergence speed. We have chosen to extend the design of Bonawitz *et al.* [55] which adds aggregators to standard FL as proxies between clients and the server for scalability. FDFL features an election process for aggregators and the central server to better tolerate node failures.

We also propose a new distributed network architecture supporting the model and identify the security properties to be guaranteed and the associated security countermeasures.

We evaluate our architecture using the `OMNeT++` simulation framework on a $\sim$ 400k parameter convolutional neural network using CIFAR-10, FMNIST and MNIST datasets show that FDFL scales well compared to other models (e.g., standard FL, FL with aggregators) with constant computing and memory overheads, while preserving convergence speed. FDFL achieves similar resilience to crash failures than standard FL with reasonable failure rates (up to 20 % of nodes).

The chapter is structured as follows. Section 5.2 introduces distributed learning. Section 5.3 reviews related work. Sections 5.4 and 5.5 present the FDFL model and supporting architecture. Sections 5.6 and 5.7 report on experiments and

discuss results. Finally, Section 5.8 concludes and sketches directions for future work.

5.2 Distributed Learning

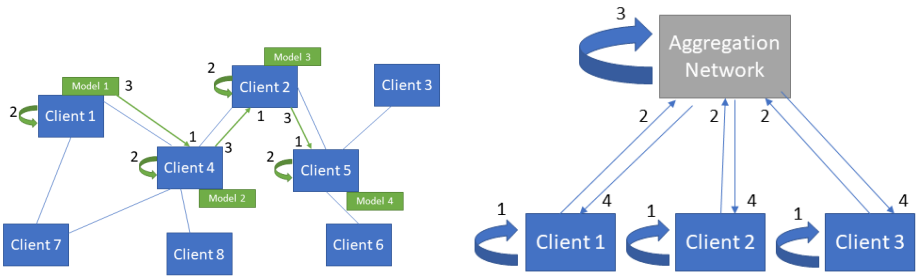
ML is traditionally performed on a single computer hosting data and computing power. Some models (e.g., deep learning) require more resources (e.g., memory) beyond those limits. Training may then be performed concurrently.
Distributed learning [114] relies on multiple computing elements (e.g., physical or virtual machines, CPU cores) running concurrently to accelerate training. Beyond consistency and failures challenges, a main hurdle is the communication speed between the entities that are part of the training process.
Distributed ML assumes that all computing resource providers taking part in the training may access the full training dataset. A simple security model is to consider all such providers as trusted by the data providers. In practice, the security model is often more complex. It may include untrusted or mutually distrustful nodes. FL [44] enables distributed learning without requiring data providers to disclose their data: data providers behave as trainers on their own datasets, sharing model updates with a central server.

Some main challenges for distributed learning include:

- **Scalability:** performance should remain high when increasing the number of clients rather than node resources.
- **Resilience to failures:** the system should tolerate single or coordinated failures of a reasonable fraction of clients without impact on training.
- **Neighbor dependency**: the topological proximity of network nodes may have various kinds of influence on the evolution of the model during training.
- **Convergence speed:** communication rounds (or steps) to reach a given accuracy should be minimized.
- **Training speed:** latency to reach high accuracy should remain low [176].

Neighbor dependency is closely related to the distribution level and to data locality. Aggregation strategies range from centralized (FL) to decentralized, e.g., tree-based [28], ring-based using parameter servers – partitioning the model over servers holding subsets of the dataset – or fully distributed [114].
Moving from centralized learning to a decentralized setting much widens the design space for models and architectures. Some key dimensions include:

- **Architecture:** communication topology (e.g., star-shaped, peer-to-peer); adaptability (e.g., devices joining/leaving); scalability.
- **Performance:** system resources (computing, memory, bandwidth); convergence speed.



(a) Gossip models (b) Global gradient models

Figure 5.1: Gradient-based decentralised learning

- **Model quality:** high accuracy; low training time.
- **Resilience:** tolerance to failures: single vs. multiple; accidental vs. malicious.
- **Security:** resistance to hostile entities (e.g., devices, routers); confidentiality, integrity, availability threats [132].
- **Privacy:** sensitive/private training data should be protected; inference should not disclose further information.

5.3 Related Work

Gradient-based decentralized learning architectures include:

- *Gossip-based architectures* (see Figure 5.1(a)): models are propagated in the network using a *gossip protocol* [57, 68]. A node updates its local model **(1)**, aggregates the models received from neighbors with its own model **(2)**, and sends the result to some of its neighbors **(3)**.
- *Aggregation-based architectures* (see Figure 5.1(b)): for each training round, nodes train their model **(1)**, send it to an *aggregator* **(2)** that merges the received models **(3)**, and sends the new model to clients **(4)**.
  - *Local gradient architectures* [63, 85]: each client considers all its neighbors to be aggregators to which it broadcasts its updates. The aggregation result becomes the new client model.
  - *Global gradient architectures* [44, 111]: all clients send their updates to an aggregator which evaluates a global new model.

Table 5.1 shows that no architecture gives fully satisfying results. Models with high resilience and scalability tend to show slow convergence, while models with high convergence speeds have moderate scalability and resilience.

Table 5.1: Overview of decentralised learning models
[1]Theoretical analysis [2]Benchmarking

Model	Scalability[1]	Convergence speed[2]	Resilience[1]	Neighbor dependency[1]	Training speed[1]
Global gradient models					
FL [44]	Moderate	High	Moderate	No	High
Scalable FL [55]	High	High	Moderate	No	Moderate
IPLS [111]	Moderate	High	Moderate	No	Moderate
Local gradient models					
CFA [85]	High	Low	High	Yes	High
BrainTorrent [63]	Low	Low	High	Yes	High
Gossip models					
Gossip models [57]	High	Low	High	Yes	Low
Moshpit sgd[178]	High	Moderate	High	No	Moderate

Up to our knowledge, there is no model providing at the same time the fast convergence of a global gradient model and the resilience and scalability of fully decentralized approaches. A new learning model is therefore required.

Fault-tolerance vs. Scalability. During training, participants may encounter errors causing loss of network connectivity and node drop out (*crash failures*). Non-malicious failures may happen randomly [106], e.g., a device running out of battery. Failures may also be malicious (*byzantine failures*): a coalition of nodes may tamper with training [112] with many applicable counter-measures [69].

The resilience vs. scalability trade-off has been actively investigated by frameworks. For instance, distributed computing platforms have explored the benefits of extending to ML their architectures for distributed computations [174] or data. With cloud ML hyperscaler solutions, distributed learning is also becoming mainstream. Native distributed ML frameworks [28] generally do not support fault tolerance or have limited scalability due to node availability – some staleness [32] may enhance resilience without affecting convergence. A few frameworks offer sophisticated resilience strategies against stragglers [175] or failing nodes [177], but remain expensive in terms of computational complexity.

Enhancing Security and Privacy. At the hardware level, *Trusted Execution Environments (TEE)* [110] guarantee strong computation and data confidentiality and integrity based on processor-based memory encryption.

At the software level, cryptographic techniques such as *Secure Multi-Party Computation (SMPC)* [40] and *Homomorphic Encryption (HE)* [93] enable respectively multiple agents to compute the result of a common function without sharing local inputs and to compute on encrypted data. Perturbative techniques such as *Differential Privacy (DP)* [37] may also improve privacy by adding noise on nodes during training. Finally, the immutability of *distributed ledgers* [82] is helpful to guarantee distributed integrity in a history of dataset updates.

Such components can be used throughout the architecture to provide different guarantees. The TEE can help to protect client data [112] or to prevent access

to the model for clients. Those techniques usually slow down training and may alter the performance of the trained model.
Pervasive FL training will often be deployed on resource-limited devices. The training *Execution Environment (EE)* can take different forms including virtual machines, containers, or lightweight containers [173], with many tools available [184]. The EE should take into account the diversity of hardware and software, notably hardware accelerators (e.g., GPUs, TPUs) or security components (e.g., TEEs).

5.4 Fully Decentralized Federated Learning

We propose the Fully Decentralized Federated Learning (FDFL) model to improve FL scalability and resilience. The idea is to do FL with a hierarchy of aggregators for scalability, where the roles of server or aggregator is attributed by the participants which are connected through a P2P network for resilience. The model is suitable for P2P networks of edge devices connected to cloud infrastructures.

Model Overview. We adopt the following design principles:

- *Standard FL-inspired* model to preserve the training speed benefits of FL.
- *Aggregator-based model* to increase scalability in terms of computation and communication.
- *No central node in the network* to improve resilience by removing single points of failures in the architecture.
- *Global gradient approach* to enable the best convergence speed without any neighbor dependency.

FDFL applies FL to fully decentralized P2P networks, where nodes know only their neighbors. We apply aggregators to standard FL with no central node to make the model scalable and resilient while preserving training speed. We adopt a global gradient approach for its benefits of convergence speed, model unicity, and absence of neighbor dependency.

The model includes two phases:

- *Role assignment:* three different roles may be assigned to a network node, *client*, *aggregator*, and *server*.
- *Training:* the assigned roles are used to train the model.

Election Process. Regularly (e.g., at constant time intervals), the network elects *aggregators* and a *server* among network nodes. An election should not happen too often to limit communications. But it should allow the network to adapt quickly to failing nodes (e.g., especially the central server).

Training Process. The training process (see Algorithm 1 and Figure 5.2) adapts and extends that of standard FL [44] similarly to Scalable FL [55] by

adding aggregators that behave as proxies between clients and the central server: **(1)** the server dispatches the model to train to clients through the aggregators. **(2)** Clients train the model on their local data. **(3)** Clients send their gradient update propositions to their aggregators. **(4)** The aggregators merge the received updates. **(5)** The results are sent upwards to the server. **(6)** The server merges the updates received from its aggregators. **(7)** The server sends back the new model to clients through the aggregators. Step **7** allows all clients to know the current model in case of a new election.

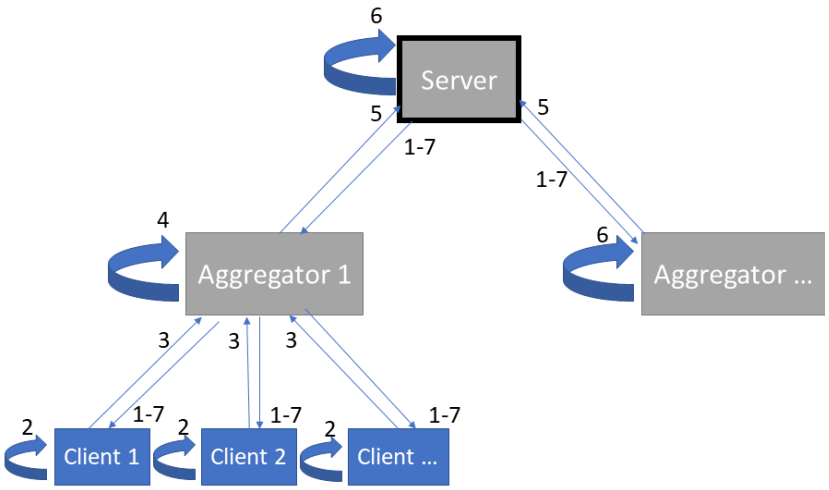


Figure 5.2: FDFL training process

In FDFL, using a central server (changing over time according to the election process), instead of a consensus between top-level aggregators (parameter server approach [114]) has benefits in terms of simplicity of implementation and network communications, but decreases resilience. With this design, security and privacy components (e.g., DP, SMPC) may be easily added.

It has been shown that sharing only gradient updates is not enough to guarantee full privacy [132]. In standard FL [44], privacy management is centralized in the server, and some trust should be granted to the server. However, with FDFL, constantly changing aggregators and servers makes the situation different. No node has access to as much data as the central server of standard FL, but more nodes get access to some data on a given node. This decreases the attack surface on a single node, but may increase the number of vulnerable nodes. To address those challenges, cryptographic (e.g., SMPC, HE), hardware-based (e.g., TEE), or perturbative (e.g., client-level or server-level DP) approaches can be used [132].

Algorithm 1: Training process (adapted from [44]).
n_c: # samples in client c, n_k: # samples per aggregator k,
n: dataset size, S_{agg} : set of aggregators, S^k_{client} : sets of clients,
μ : learning rate, w: weights

Server:
initialize w_0
for each round t **do**
  for each aggregator $k \in S_{agg}$ *in* || **do**
    $(w^k_{t+1}, n_k) \leftarrow$ AggregatorUpdate(k, w_t)
  $w_{t+1} \leftarrow \sum_{k \in S_{agg}} \frac{n_k}{n} \cdot w^k_{t+1}$
  // Send back new model
  // to clients through
  // the aggregators

AggregatorUpdate(k, w):
// Run on aggregator k
for each client $c \in S^k_{client}$ *in* || **do**
  $(w^c_{t+1}, n_c) \leftarrow$ ClientUpdate(c, w_t)
$w_{t+1} \leftarrow \sum_{c \in S^k_{client}} n_c \cdot w^c_{t+1}$
$n_k \leftarrow \sum_{c \in S^k_{client}} n_c$
return (w_{t+1}, n_k) to server

ClientUpdate(c, w):
// Run on client c
for each local epoch **do**
  for each batch b **do**
    $w \leftarrow w - \mu \nabla \ell(w; b)$
return (w, n_c) to aggregator

5.5 The FDFL Architecture

We now describe the FDFL architecture. We present the components that constitute a network node, the supporting election protocol, and the network and failure models.

Node Architecture. We consider the components shown in Figure 5.3. *Routing* supervises the other node components, and routes messages. *Aggregation* stores and aggregates the proposed models from other nodes. *Training* trains the ML model. *Election* participates in the election of the aggregators and of the server.

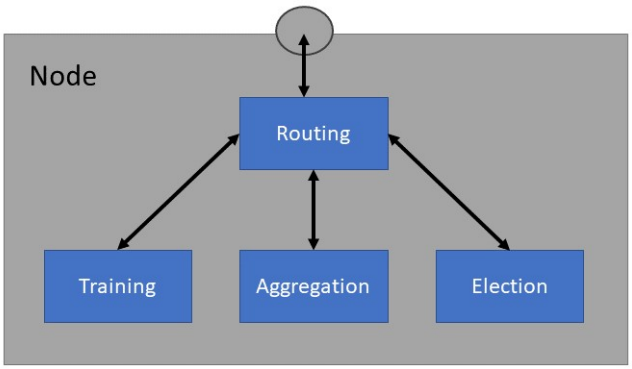


Figure 5.3: Node architecture

Such components can be isolated from one another and deployed on different machines. For instance, the training component could be isolated using a TEE to protect the model from curious clients and from gradient poisoning threats.

Election Protocol. *Node election* in P2P networks has been extensively explored, with many protocols providing guarantees of latency, scalability and security [89]. In FDFL, we consider a simple protocol inspired by the Proof of Luck (PoL) scheme [38] – without the TEE which may be added in a second step [39] – to elect a single server and a percentage of aggregators in the network:

1. *Aggregator election:* each client uses a random number generator to elect itself with a tunable α probability. α is set to 2%. The elected aggregators broadcast their PoL score to network nodes.

2. *Server election:* after a pre-defined time, the aggregator with the best PoL score is elected as the server.

Network and Failure Model. We consider a P2P network where nodes only know their direct neighbors. Non-neighboring nodes may communicate by static routing through the network. We consider short-term failures (typically 1s) on the node side. Byzantine failures are not considered at this stage. FDFL

makes no further assumption regarding communication between nodes, making it applicable to many communication patterns of distributed ML [114].

Security Modules. Several security modules may be added to protect the components of the architecture. They aim to provide the following protections:

- **Confidentiality** – *Data Confidentiality (DC):* Privacy of client local training dataset. No information or limited information can be acquired on the dataset held by the client device and used to train the model. // *Model Confidentiality (MC):* Confidentiality of architecture/weights of trained model.
- **Protection against Byzantines** –Byzantine attacks are attacks where a fraction of the participants act maliciously in coordination.// *Aggregation Integrity (AggI):* The aggregation process is not tampered with.// *Byzantine Tolerance (BT):* Model protection against tampering with fake updates.// *Backdoor Protection (BackP):* Protection against introduction of backdoors in the model (special Byzantine case)
- **Protection against Sybils** – *Sybil tolerance ST:* Protection against Sybil attacks where a single entity creates multiple identities to increase its voting rights in a distributed systems. Sybil attacks can be used to reach the fraction of malicious participant in a distributed system necessary to launch Byzantine attacks.

Table 5.2(a) provides an overview of the architecture components on which each class of attack can be applied. Table 5.2(b) reviews for each attack type some possible countermeasures. Implementing some of those countermeasures will be a next step beyond this work.

5.6 Experimental Results

We evaluate FDFL scalability and resilience w.r.t. other models such as standard FL [44], FL with server election, and FL with aggregators [55].

5.6.1 Experimental Setup

We validate FDFL using `OMNeT++`, a `C++` framework for network simulations, e.g. for vehicular systems.We simulate P2P networks with 100 and 1000 nodes and 300 and 3000 links on an Intel Pentium E5300 dual-core 2.6 GHz and 2GB RAM running Linux Ubuntu 20.04. Training of deep learning models is done separately on Google Colab with TensorFlow Federated v0.20 for FL. Models are trained on CIFAR-10, NMIST and FMNIST datasets and evaluated on their accuracy. These are datasets consisting of images labeled in 10 distinct categories (e.g. MNIST handles handwritten digits). They are commonly used to test Machine Learning models. The accuracy is here defined as the percentage of images rightly identified (e.g. a drawn 1 is identified as “1”). The model is a

Table 5.2: Security – (a) Attack types vs. architecture components; (b) Counter-measures vs. attack types

(a)

Security modules **Attack type**	**Router**	**Election**	**Aggregation**	**Training**
Data Confidentiality			✓	✓
Model Confidentiality			✓	✓
Aggregation Integrity			✓	
Backdoor Protection			✓	
Byzantine Tolerance	✓		✓	✓
Sybil Tolerance	✓	✓		

(b)

Attack **Counter-measure**	**DC**	**MC**	**AggI**	**BT**	**BackP**	**ST**
Differential Privacy	✓				✓	
TEE	✓	✓	✓	✓		✓
SMPC	✓					
Reputation System				✓		✓
Filtering				✓		
Redundancy				✓		

convolutional neural network with `3 * (Conv2D` → `Maxpooling)` → `Dense` → `Dense` architecture, with 389,642 parameters. The SGD learning rate is set to 0.001.

5.6.2 Scalability

We first analyze the theoretical computing resources, memory, and bandwidth usage during training according to network size before reporting on experimental results. We also discuss the impact of network size on training time.

Resource usage

Table 5.3(a) summarizes the node resource usage for different models for transparent (e.g., single-hop) network communications. We assume the memory and the computing time for local training on client devices and the memory to store the model to be constants – the neural network structure is considered static, without compression or structure-specific optimisations.
Our model shows constant computing and memory overheads for aggregation when the network scales. Bandwidth usage outside routing and election is also independent from the network size unlike the other models. We see that the total bandwidth usage for a training step is more important in FDFL because of the use of aggregators behaving as proxies.

Table 5.3: Scalability – (a) resource usage; (b) impact on training time

(a)

	Std FL [44]	**Bonawitz [55]**	**FDFL**
Aggregation performance			
Computing	$O(N)$	$O(N/n_{agg})$	$O(A)$
Memory	$O(N)$	$O(N/n_{agg})$	$O(A)$
Bandwidth usage			
Max bandwidth	$O(N)$	$O(N/n_{agg})$	$O(A)$
Total bandwidth	$O(N)$	$O(N/n_{agg})$	$O(N + N/A + N/A^2 \ldots)$
Election performance			
# messages	-	-	$O(N/A)$

(b)

Model	Training time
Std FL [44]	$2.T_{com} + T_{train} + T_{agg}(N)$
Bonawitz [55]	$4.T_{com} + T_{train} + T_{agg}(N/n_{agg}) + T_{agg}(n_{agg})$
FDFL	$\lceil log_A(N) \rceil.(2.T_{com} + T_{agg}(A)) + T_{train}$

N: # network nodes, A: # clients per aggregator, n_{agg}: # aggregators
T_{train}: training step time on device,
$T_{agg}(A)$: aggregation time for A models,
T_{com}: time for a message to reach its destination

Communication overhead

We evaluate the network overhead for multi-hop communications, taking into account elections. We consider a single aggregation layer and elections occurring regularly (every 30s). Figure 5.4 shows the communication overhead for different fractions of aggregators.
We see that outliers get fewer messages when the number of aggregators increases: no node is flooded by messages compared to others. This means that aggregators lift some of the load off the server. Moreover, the median number of messages increases with aggregators. This is due to aggregators behaving as proxies. The election process also has impact on network communications. Finally, the number of messages increases again above a threshold. The network overhead due to the election process then exceeds the benefits of the presence of aggregators. Those trends are confirmed as the network size increases from 100 to 1000 nodes.

Impact of network size on training time

Table 5.3(b) shows the impact of the network size on training time. The training time is more important in FDFL than for other models due to the presence of aggregators – although for some cases such as FL where $T_{agg}(N) = O(N)$, logarithmic speedups may be achieved.

A trade-off may be found between training time, computing and memory overheads of a single node. Reducing the number of aggregation layers improves training time and reduces the communication overhead and the cost of the election process, but puts a higher toll on the aggregators, increasing significantly the upper bound on network traffic.

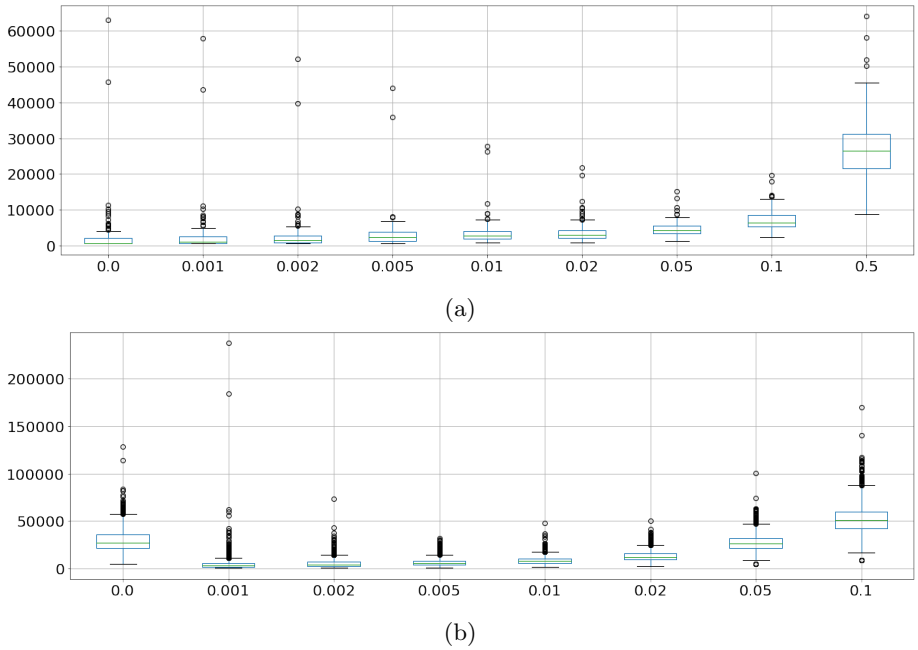


Figure 5.4: Scalability: number of exchanged messages vs. fraction of aggregators – (a) 100 nodes; (b) 1000 nodes
bar: median, box: quartiles, whiskers: data range (excl. outliers), dots: outliers

5.6.3 Resilience

We evaluate as well the resilience of FDFL to non-byzantine failures: nodes may fail randomly for short, random periods of time. Resilience is assessed for a single layer of aggregators representing 2% of network nodes by training models and computing their accuracy. We simulate short failures of around 1s for the models with and without aggregation and election processes shown in Table 5.4.

Influence of aggregators and election

Results for CIFAR-10 are shown in Figure 5.5 for $f = 0.1$, where f is the average fraction of failing nodes. We observe that FDFL has about the same resilience as standard FL (see Figure 5.5(a)). Also, we find that separately, aggregation and election protocols have an impact on resilience, but together, they enhance it (see Figures 5.5(b) and 5.5(c)).

Table 5.4: Considered Models

Model	Aggregators	Election
FL [44]		
Bonawitz [55]	✓	
Election FL		✓
FDFL	✓	✓

The presence of aggregators reduces training performance in the presence of failures (see Figure 5.5(b)). This is due to aggregators behaving as proxies, which increases the distance to the server and the likelihood of messages being dropped.
With an election process, some nodes may not be aware of the identity of the newly elected server, this information being lost due to failures. This reduces the number of active participants and model accuracy (see Figure 5.5(c)).
However, when using aggregators and election together, FDFL improves resilience compared to using them separately. With aggregators, it is easier for a client to know at least one aggregation node and to participate in training. With an election, the regular change of aggregators increases the diversity of participating clients and training samples, improving accuracy. Table 5.5(a) provides accuracy benchmarks for other datasets, but the conclusions are broadly similar.

Table 5.5: Accuracy for different datasets – (a) model comparison (b) impact of failures

(a)

Model / Dataset	FL f=0	FL	FDFL	Bonawitz	Election FL
CIFAR-10	0.6361	0.6172	0.6104	0.6084	0.6071
MNIST	0.9891[1]	0.9894	0.9871	0.9879	0.9863
FMNIST	0.8828[2]	0.8954	0.8858	0.8914	0.8873

(b)

Model	FL				FDFL		
Dataset	f=0	f=0.1	f=0.2	f=0.3	f=0.1	f=0.2	f=0.3
CIFAR-10	0.6361	0.6172	0.5965	0.5668	0.6104	0.5623	0.4584
MNIST	0.9891[1]	0.9894	0.9882	0.9837	0.9871	0.9858	0.9718
FMNIST	0.8828[2]	0.8954	0.8876	0.869	0.8858	0.8816	0.8346

[1] 131 epochs [2] 104 epochs

Influence of failure rate

We see that performance is slightly lower for our model than for FL as the failure rate increases (see Figure 5.5(d) and Table 5.5(b)) but remains quite acceptable. Beyond $f = 0.2$, performance degrades further, although still good

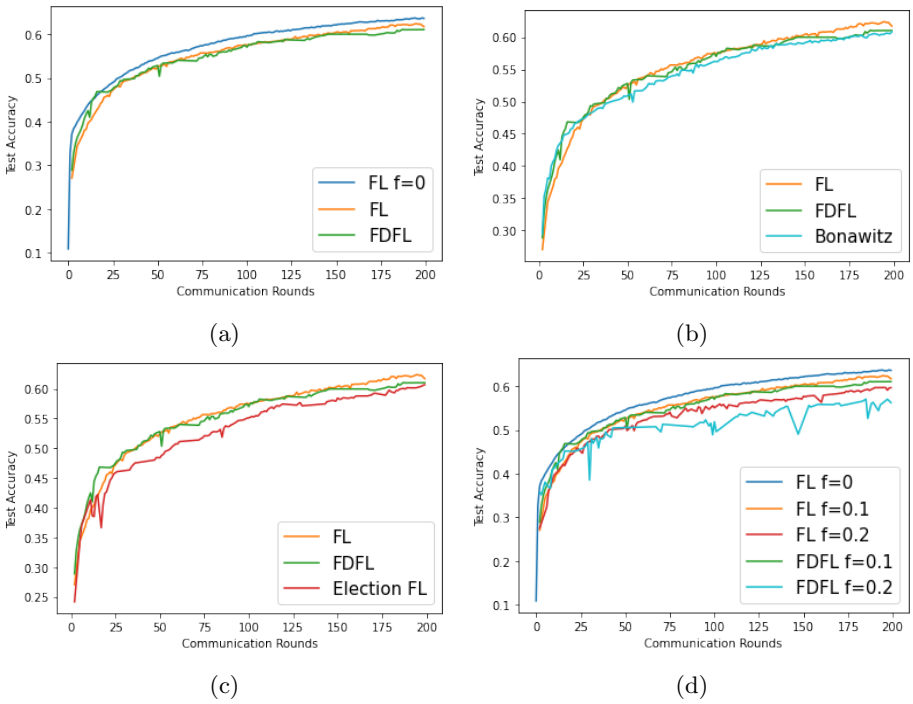


Figure 5.5: Resilience (CIFAR-10) – influence of (a) model (b) aggregators (c) election (d) number of failures

for MNIST and FMNIST, perhaps due to less complex datasets. FDFL seems thus to be well adapted to network environments with reasonable failure rates.

Scalability vs. Resilience Trade-off

Benchmarks highlighted that elections tend to increase bandwidth usage. A trade-off should be found between having regular elections to improve resilience but within reasonable bounds to preserve bandwidth usage and scalability.

5.7 Discussion

We observed that FDFL improves FL scalability, as shown by simulation results for networks with 100 and 1000 nodes. Further experiments are needed for networks with multiple layers of aggregators and when considering an evolution of the number of participants during training.
FDFL shows good resilience against short-term and non malicious failures for simple datasets. Additional research is needed to investigate more realistic failure models, such as long-term and malicious (i.e., Byzantine) failures [117, 118].

Several reactions of the network to a node failure are possible, depending on its role:

- **Server failure:** training of the entire network is stopped until the server recovers or another server is elected in the next election round.
- **Aggregator failure:** data of the aggregator's clients are not taken into account until the aggregator recovers or clients select another aggregator in the next election round.
- **Client failure:** client data are not taken into account until the first training step after recovery. If a new election happens during the failure period, the client has to wait for the next election to participate again.

When they fail, aggregators have more impact than a normal client on the quality of the model: they may drop gradient updates of multiple clients and the network may fork into two branches training concurrently, merging back only at the next election round. This reduces the number of clients active during training, and the diversity of training data.
Though, the model remains very simple and datasets such as CIFAR-10 not particularly demanding. Training datasets with more complex data would allow to highlight the influence on accuracy associated with a loss in training data diversity [205].

5.8 Conclusion

This chapter presented FDFL, a new fully decentralized FL model and architecture combining aggregators and election process that improves the scalability and the resilience of standard FL while preserving convergence speed.
Future work includes enhancing security and privacy [125] by adding new components to the architecture, e.g., SMPC, HE, TEE [116], and DP [122]. For instance, SMPC and TEE could be efficiently and simply integrated, e.g., the SMPC protocol of [39] combines well with the PoL election process used in FDFL assuming TEE-protected nodes.
Another direction is to improve performance, e.g., by exploring the influence of multiple aggregator layers w.r.t. the number of clients or by relaxing model unicity [91].

Chapter 6

Ti-skol: A Modular Federated Learning Framework Supporting Security Countermeasure Composition

Federated Learning (FL) is a growing technology that enables training of Deep Learning models on private data. Many FL enhancements have been proposed, notably for better security and privacy. Current architectures and frameworks focus on specific sets of enhancements with little extensibility and do not support composition of enhancements. In this chapter, we introduce Ti-skol, an architecture and framework that supports composition of security and privacy countermeasures, including countermeasure incompatibilities. Ti-skol also enables modular management of FL enhancements beyond security, being compatible with most enhancements. Ti-skol is promising to assess the cost of countermeasures, individually or in combination. We evaluate our framework on a use-case of Volunteer Deep Learning – applying Volunteer Computing to reduce the cost of large model training by harnessing idle resources of single machines into the required massive distributed computing power. Experimental results show that Ti-skol is scalable as the network size increases. While adding security countermeasures such as Byzantine protections or secure aggregation substantially increase computing overheads, they do not change their order of magnitude, individually or in combination. This tends to show the practicality of the Ti-skol framework for on-demand FL security.

6.1 Introduction

In recent years, IoT devices have produced a deluge of data, enabling to train Machine Learning (ML) models for prediction and classification in a large range of applications [157]. While centralized model training raises privacy concerns, the promise of Federated Learning (FL) [45] is to train joint models without the data leaving the device. A server sends the model to the devices holding the data. Data holders train the model on their local data and send back model updates to the server for aggregation. Only model updates are shared, not raw data.

FL systems have been extensively studied [105]. Yet, many security and privacy vulnerabilities have been reported [146]. Threats target mainly privacy, integrity and identity: 1) extracting private information from training datasets [46, 59] or reconstructing raw data [67]; 2) poisoning data samples [87], polluting models through malicious updates [74, 101] or introducing backdoors [88]; and 3) Sybil attacks [12].

An even larger range of countermeasures has been proposed to mitigate such threats with different security guarantees and limitations [146] – with no countermeasure covering the full set of protection requirements. Countermeasures remain difficult to compare and are often not compatible with one another. While a few benchmarking frameworks have been proposed [163], they do not address countermeasure composition – referring to the case when countermeasures are combined on-demand to complement one another.

To enhance FL security, *composition of countermeasures* appears as a focal challenge to: 1) select on-demand the countermeasures meeting the security requirements; 2) handle countermeasure incompatibilities; and 3) assess the cost of countermeasures, individually or in combination. Current frameworks do not meet those goals and do not support modular management of FL security enhancements.

In this chapter, we introduce Ti-skol[1], a modular architecture for FL supporting highly expressive customization policies for security countermeasure composition, enabling benchmarking of countermeasures. We propose a framework implementing the architecture and show how security countermeasures can be composed within this architecture, taking as examples Byzantine protection and aggregation integrity. Such flexibility to assess and deploy the just-needed level of security makes Ti-skol a novel solution to enhance FL security.

Ti-skol also enables modular management of composition of FL enhancements beyond security. Ti-skol is a solution to enhance FL flexibility in this area, as many FL enhancements have been proposed regarding model quality or performance, but with little extensibility in current deployment frameworks.

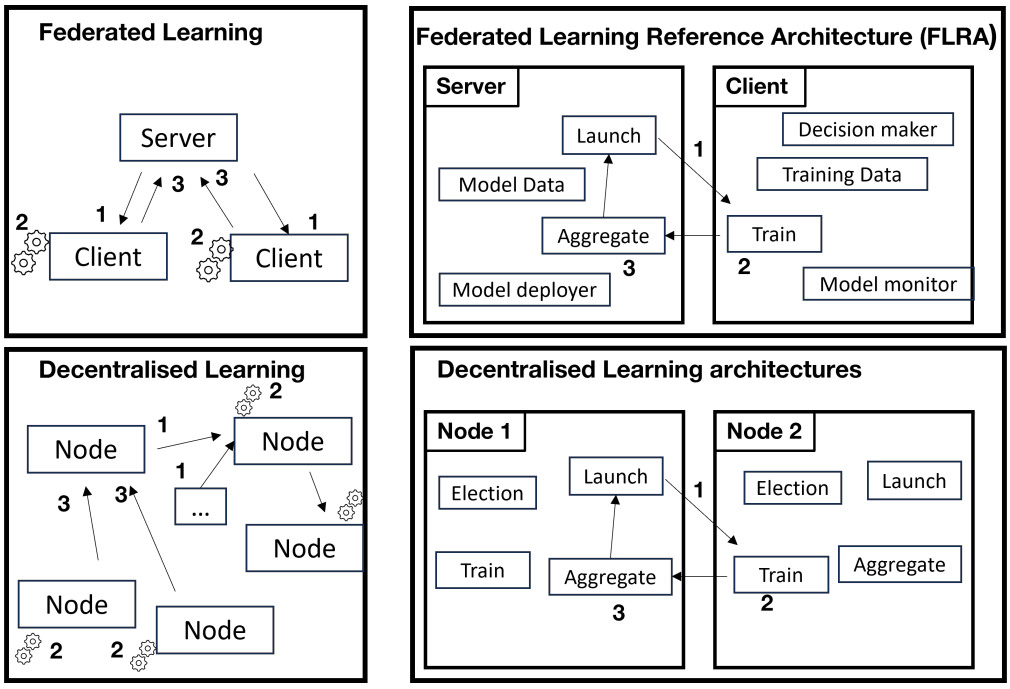
We evaluate Ti-skol framework on a Volunteer Deep Learning (VDL) use-case. VDL is an application of FL based on the Volunteer Computing paradigm [60]

[1]Ti-skol is the breton word for *school building*. It provides the base infrastructure to organize lectures – centralized, around a teacher or decentralized, rearranging the classroom. Additional security can be added if needed.

to train DL models. The idea is to use idle computing resources to reduce the cost of large distributed computing tasks.

This chapter provides the following contributions: 1) we propose a modular FL architecture supporting on-demand composition of security countermeasures and effective benchmarking in terms of impact on performance; 2) we show that our framework effectively supports countermeasure incompatibilities with good performance and scalability.

The chapter is organized as follows. Section 6.2 introduces our reference architecture for FL compatible with legacy enhancements. Section 6.3 explains the composition of security countermeasures in our architecture. Section 6.4 presents the Ti-skol framework. Section 6.5 evaluates Ti-skol scalability and security composition on a VDL use-case. Section 6.6 reviews related work. Section 6.7 concludes.



a) b)

Figure 6.1: Previous approaches: (a) training methods and (b) reference architectures for federated and decentralized learning

6.2 The Ti-skol Architecture

This section presents the design of our architecture, in terms of key properties, design principles and structure.

6.2.1 Key Properties

We consider the following properties for a FL architecture applicable to real-world systems, regarding *performance* and *customization*.

Performance – The architecture should not hamper application performance nor reliability.

- The architecture should be **scalable**: enhancing performance should be possible by increasing the computing power of nodes (*vertical scaling*); or by supporting a high number of nodes (*horizontal scaling*). FL has shown weaknesses in horizontal scaling [142]. The central server is a bottleneck, limiting client scalability due to available bandwidth and computing resources.
- The architecture should be **resilient** to node failures: FL has shown resilience on the client side, but remains weak on the server side (single point of failure).

Customization – The architecture should be easy to improve by integrating research results and composing them.

- The architecture should be **compatible with legacy** FL systems: it should be expressive enough to support most previous works on FL. It should be compatible with the identified FL design patterns [108] (e.g. client selection, decentralised aggregation).
- **Security countermeasure composition** should be supported to provide *on demand* security and privacy to meet use-case requirements. New countermeasures should be easy to integrate in the architecture.

6.2.2 Design Principles

We adopt the following design principles.

Performance – We adopt an architecture which is *aggregator-based* and *fully decentralized*.

- **Scalability ⇒ Aggregator-based architecture**: the training process should allow multiple concurrent nodes in the aggregation process to reduce the load on the server, as more clients participate [55, 142]. This may be achieved by placing intermediate *aggregators* between the server and clients [55, 142] or with *fully decentralized designs* [57], where participants aggregate the models of their neighbors in the network.

- **Resilience ⇒ Decentralized architecture**: to tolerate server failures, the architecture should have *no central node* and be fully decentralized [57, 142].

Customization – We adopt an architecture which is *compatible with a reference FL architecture* to enable extensibility and is *modular* to support the composition of countermeasures.

- **Legacy ⇒ Reference architecture compliance**: the architecture should follow a *reference architecture* such as FLRA [109] to be compatible with most FL enhancements and design patterns [108].
- **Countermeasure composition ⇒ Modularity**: families of security countermeasures should be identified and managed as components, with specific hooks defined in the architecture. Their incompatibilities should be identified. Seamless replacement of countermeasures with new versions should be supported.

6.2.3 Ti-skol Architecture

We propose Ti-skol, an architecture based on the previous design principles.
FL and decentralized learning both feature the same training loop (Figure 6.1(a)): the model to train is sent to clients (1), trained on local client data (2), and sent back to an aggregator to update the model (3). In FL, the server sends the model and aggregates the response. In decentralized approaches, any participant may perform those tasks.
FLRA [109] is a reference architecture for FL describing the sub-components on client-side and server-side and how they communicate with each other (Figure 6.1(b)). Three components are required for training: (1) *Launch* sends the models to train; (2) *Train* trains the models on local data; and (3) *Aggregate* aggregates the received trained models. In decentralized approaches [142], each participant includes all such components, i.e., any component may perform aggregations.

Compatibility with legacy – We design our architecture to be compliant with FLRA [109] and assume the same core components. For decentralized learning, we add the *Election* component to assign to nodes the relevant role in the training process (i.e., client, aggregator, server).
A sample decentralized learning architecture [142] is shown in Figure 6.2. Similarly to a FL server, the *Aggregation* component features a *Launch* component that starts the training rounds and an *Aggregate* component that receives and aggregates trained models. The *Train* component trains the models similarly to FL clients. Communications between components are performed by a *Network* component, part of each node sub-component. A *Data* component is in charge of storing *Model* data and *Training Data*.

Countermeasure composition – In our architecture, the ownership relationship between components is structured as a tree. Each branch is dedicated to

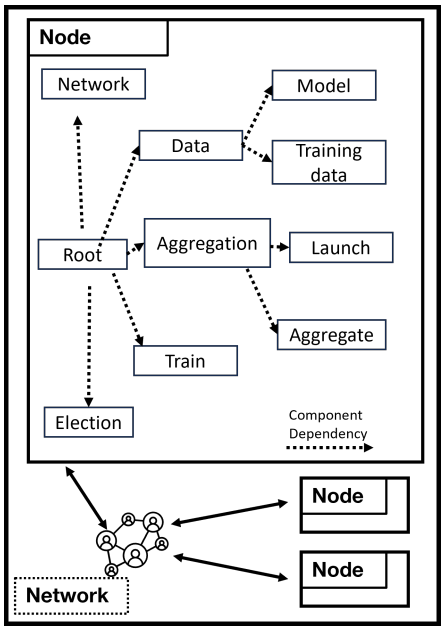


Figure 6.2: Ti-skol architecture

one of the core system functions. This design ensures that each component is responsible for a specific function without overlap in terms of responsibilities, i.e., only a single component is in charge of aggregation. The independence of components allows them to be distributed into multiple execution environments in order to improve performance or isolation.

Scalability and resilience – We extend our architecture to support decentralized learning [57, 142]. Each node can be a client or a server in the training task. Therefore, each node should feature client and server components at any time. An *Election* component is added to reach consensus with peer nodes on their roles [142]. It provides the current role of a node to other components (e.g., allow the *Aggregate* component to launch a training round as server node) and discover other nodes (e.g., find an aggregator in the network for a client).

6.3 Security Architecture

Our tree-based architecture enables to easily add legacy enhancements by compliance with a FL reference architecture. We now focus on FL security and privacy challenges and applicable countermeasures. We review the challenges, countermeasures and their composition in our architecture.

Ti-skol improves FL security by allowing the implementation and combination of security countermeasures and the isolation of FL software components.

6.3.1 Security Challenges

Many threats on DL [92] also apply to FL [146]. We focus on the following key FL security challenges: 1) *integrity*, i.e., training should produce a correct model; and 2) *privacy*, i.e., training data should be kept private.
Threats can be categorized as: *malicious*, where participants may deviate from the protocol; or *honest but curious*, where participants respect the protocol, but try to gain information from legitimate messages.

Privacy. Keeping training data private is a main driver for FL adoption [45, 105]. But assessing the privacy guarantees of FL schemes remains difficult.
A common approach for FL privacy is based on *Differential Privacy (DP)* [37]. DP provides a probabilistic indistinguishability guarantee for processing a specific data sample during training. This means that a specific training sample cannot be extracted from the trained model – the training data set probability distribution can still be leaked. Privacy requirements may also include confidentiality of the neural network architecture and of weights.

Integrity. FL does not require an exact model. It can even be hard to get the same model every time due to floating-point operations roundings that can vary between different systems. Models can be the target of *backdoor attacks* [88], where model performance is degraded on a specific class of samples. Countermeasures may only filter part of malicious updates sent by clients. Different levels of integrity may be achieved depending of the update impact on the overall model. The challenge is then to evaluate model integrity – model accuracy is a first broad metric.

Identity threats. In *Sybil attacks*, an attacker creates multiple identities to increase its voting power [12]. For FL systems, such attacks are generally not considered, as mitigations such as IP address bans provide a sufficient level of security.

6.3.2 Security and Privacy Countermeasures

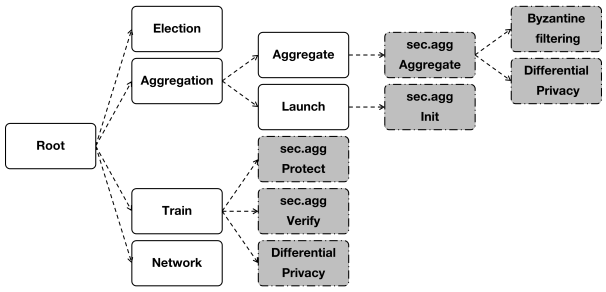


Figure 6.3: Countermeasure composition in a node

Privacy countermeasures. Privacy countermeasures include: 1) *differential*

privacy to prevent leakage of private data from the model weights; and 2) *secure aggregation protocols* to prevent the aggregating server from accessing the weights of individual models.

DP [37] improves FL privacy by adding random noise to the model weights. DP can be applied by clients to protect their training data from aggregators; or by aggregators to protect training data from other clients during the next training round. DP increases privacy but reduces model accuracy.

Secure aggregation protocols [152] are cryptographic schemes where clients encrypt their model before sending it to the server for aggregation. The server is only able to access the weights of the aggregated model and not of individual models sent by clients. Secure aggregation schemes require high processing power. An algorithm that is scalable, fault tolerant, while supporting Byzantine protection and DP-enabled privacy remains to be found.

Integrity countermeasures. For distributed machine learning, training integrity is closely related to *Byzantine resilience* [144], as malicious models may be introduced in such a decentralized setting. Most solutions against Byzantine clients such as FLAME [130] are based on filtering the most important outliers. They convert the model weights into vectors and compare them, e.g., keeping the median of vectors instead of the average. FLAME also adds noise (DP) to further reduce the effectiveness of backdoor attacks.

Other approaches are based on redundancy by making multiple clients perform the same tasks or storing intermediate results. However, they breach training data privacy, as clients have to share their training data for safety checks.

To guarantee *aggregation integrity*, most solutions are secure aggregation schemes [152] that may get in conflict with Byzantine protections. For FL, this challenge is little explored as the central server is considered trusted, under direct control of the organizers.

Privacy and integrity. *Trusted Execution Environments (TEE)* are also part of applicable counter-measures to achieve both privacy and integrity [112, 128]. The TEE encrypted memory and integrity guarantees may help to analyze training data, train or aggregate models in a secure and privacy-preserving manner. TEE remote attestation may also be useful for devices to verify that tasks take place on secure nodes. How to compose TEE with other privacy-enhancing technologies such as distributed protocols, DP, and cryptographic schemes is still an open research challenge.

6.3.3 Identity Countermeasures

In [12], two families of protections against Sybil attacks are distinguished: a *central identity management authority* that attributes an identity to each stakeholder, and *Proof of X protocols* that grant voting rights according to the capacity of the smallest participants – e.g., Proof of Work, where voting rights are according to the computing power of a stakeholder.

6.3.4 Ti-skol Security Architecture

The tree-shaped organization of components inside a node enables to easily manage *authorizations* and enforce isolation. A unified system for specifying and enforcing component authorizations may be defined from the tree path, i.e., sub-components inherit their authorizations from their parents. This hierarchical approach helps enforcing the principle of least privilege.
Sub-components are more likely to change than their parents, due to more frequent system reconfigurations in the lower-levels. They are therefore considered as less trustworthy. Other sub-components behave as helpers for their parents for core functional sub-tasks. They require similar levels of authorizations as their parents.
This hierarchical approach to component identity management, e.g., for enrolment, helps tracking the component initialization status and increases accountability.

6.3.5 Countermeasure Composition in Ti-skol

Figure 6.3 shows an extended architecture with the security components (security components in grey) considered inside each node as part of the training process.

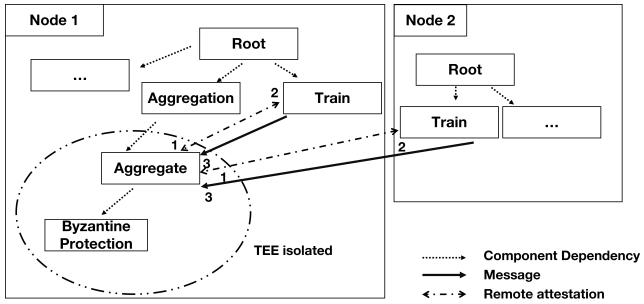


Figure 6.4: TEE isolation for the aggregation process

We consider the following security countermeasures: Byzantine protection, secure aggregation and DP. As [152], we consider that a secure aggregation protocol features four components: *init*, *protect*, *aggregate* and *verify*. Byzantine protection replaces the aggregation function, with the models to aggregate as input and the aggregated model as output. DP updates the model weights after aggregation or during training.
The Ti-skol tree is established by a human expert and captures incompatibilities between countermeasures. For instance, a secure aggregation process may not tolerate Byzantine protections due to privacy guarantees on individual trained models. Conversely, it may allow both DP and Byzantine protection at the same time.

DP can always be added after training or aggregation. Some approaches include DP directly into their aggregation process to improve privacy [152]. In that case, only the aggregated model with DP is visible to the aggregator.
When secure aggregation is compatible with Byzantine protections (e.g., mean aggregation), a training round follows the process shown in Algorithm 2. It is an extension of the FL training loop where the server sends a model to be trained to the clients who train it on their local data and send them back to the server to be aggregated. The server and clients set up a secure aggregation process using the four phases identified in [152] (Setup, Protect, Aggregate, Verify) to aggregate the trained models sent back by the clients. Byzantine protection is done in the server aggregation when the aggregation protocol allows for it.

Algorithm 2: Ti-skol training round

- **Init**: initialize secure aggregation with **verify** of clients;
- *Aggregation/Launcher*: send `totrain()` to *train* of clients;
- Clients: *train* the model, send it to client **verify**;
- Clients: **protect** trained model, send result to *Aggregation*;
- *Aggregation/Process*: run **Aggregate**;
- **Aggregate**: run **Byzantine protection** if possible;
- **Aggregate**: send result to **verify** of clients.

normal component, **security component**

A specific security composition (e.g., with/without DP and at which scale, secure aggregation algorithm) aims to meet specific security requirements. Such configuration is independent from the training task and can be re-used for other training tasks or projects. It can either be human or machine generated, taking into account the component compatibilities, performance and security properties for the combination to answer the training requirements.

6.3.6 TEE Component Isolation

While TEE are not captured as a specific component of the architecture, they may be simply applied to multiple components. TEEs may help to isolate specific branches in the tree-based Ti-skol architecture to guarantee integrity and confidentiality. Due to branch independence, this approach enables to limit the Trusted Computing Base (TCB) and enforce selective component isolation. It also prevents dependence of components on untrusted sub-components. Remote attestation may be used to verify that a component is running in a TEE, performed by other components communicating with it, inside the same node or outside.
Figure 6.4 illustrates the case where the *Aggregation Process* component is isolated in a TEE to improve aggregation integrity and privacy. Its sub-component in charge of *Byzantine filtering* is also isolated in the TEE. The two training components (in the same node and in another node) both first send a challenge

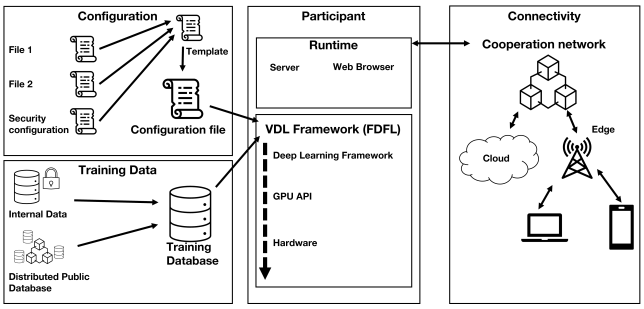


Figure 6.5: High-level framework overview

to the aggregation process to verify that it is running in a TEE (1). The aggregation process sends back a proof (2). Once they have verified the proof, the training components notify the aggregation process (3). Remote attestation can be performed only during the first communication. Swarm attestation approaches may help to reduce the attestation overhead.

6.4 Implementation

We now describe our implementation. We also develop a use-case of volunteer deep learning for federated learning.

6.4.1 Ti-skol Implementation

Figure 6.5 gives a high-level overview of the Ti-skol framework. Each participant runs the framework and loads a configuration file and the training data. The application then connects to the cooperation network to participate in the training process – to connect to the server (centralized FL) or to other peers (decentralized learning).
We configure a node using a nested JSON file. A component is described by its configuration and that of its sub-components. This limits the configuration information shared between components to the strict necessary. The initialisation process follows the tree shown in Figure 6.3, where each component is a js object that creates and configures its sub-components. A component is fully initialized when its sub-components are also initialized. Listing 6.1 illustrates the configurations of the *Aggregation* component and of its two sub-components: *Launch* and *Aggregate*. The configuration file can include other files for specific parts, e.g., the `Election` component. This design allows reusing well-specified and well-tested configuration files for different projects or to create configuration file templates.

Listing 6.1: Sample aggregator configuration

```
aggregation_launch_config: {
  number_training_steps: 10,
  training_instructions: {
```

```
      fit: {
        epochs: 1
      },
      model: "CNN1"
    }
  },
  aggregation_aggregate_config: {
    secure_aggregation_config: {
      sec_agg: "None",
      byzantine_protection: "FLAME"
    }
  }
```

As in most FL systems, training data can be loaded internally from a node or from a public database, e.g., distributed in P2P networks such as IPFS. Nodes that do not have training data may participate in training by providing computing resources.
The framework and components are implemented in JavaScript for simplicity of deployment – it can be run on any device with a JavaScript run-time (e.g., `Node.js` server, smartphone web browser, video game console). Thus, a single framework may be developed with user-level rights. The components are re-implemented to be compatbile with the tree structure of ti-skol and provide hooks for inter-component communications.
JavaScript allows us to run compute-intensive tasks using WASM for improved performance and safety. WASM offers close to native performance. It also improves safety, through type-safe memory isolation for code sandboxing to minimize impact on the operating system or the hypervisor.
Device GPUs may help to accelerate DL model training using frameworks such as TensorflowJS based on CUDA APIs (server side) or WebGL APIs (web browser side).
Communications between nodes are based on `libp2p`[2]. This library for P2P networks supports direct communications between nodes using multiple network protocols (e.g., TCP, WebRTC). It provides a Kademlia Distributed Hash Table to find resources in the network.
Each component includes a *Network* component in charge of communications with other components, inside the same node or between different nodes.
As shown in Listing 6.2, each component interface includes the component name and a *multiaddr*. These are respectively the path of the component in the tree and the network identifier of the libp2p node. The network identifier includes the communication protocol, IP address, port number, and a hash of the node public key. The interface also includes the methods which may be invoked on a component.

Listing 6.2: Sample component interface

```
{
  component: '/root/aggregation/[...]',
```

[2]https://libp2p.io/

```
  multiaddr: '/ip4/127.0.0.1/tcp/52728/[...]',
  handler: 'become_client' // component methods
}
```

This unified communication model facilitates enforcement of component access control policies. For instance, in the *Network* component, a policy could specify that the component is authorized to accept invocations originating only from another specific component of the same node.

6.5 Experimental Results

We evaluate the Ti-skol framework scalability and the impact of addition of security countermeasures on performance. We illustrate the framework capabilities with a VDL use-case, including the required security guarantees.

6.5.1 Testbed

We run our experiments on a Dell Precision 5750 laptop with an Intel i7, 10th generation, 8 cores, 2.30 GHz CPU and 32 GB RAM, running Windows 10. Framework nodes are run as concurrent `Node.js` processes on the same machine. CPU usage is monitored with the Python library `psutil`. Of course, a single local machine cannot be considered as the optimal setup for a full assessment of scalability, but nonetheless provides insightful results as a first step before moving to a cloud platform.
We train a dense neural network of 101770 parameters in 2 dense layers on the MNIST dataset. This dataset includes 60000 28x28 gray-scale images of handwritten digits to be classified.

6.5.2 VDL Use-case

We focus our evaluation on a Volunteer Deep Learning (VDL) use-case. VDL applies the Volunteer Computing (VC) paradigm to train deep learning models to reduce costs. We shortly explain below the main ideas of the use-case.

Volunteer Computing. To reduce costs of large distributed computing tasks, academia introduced *Volunteer Computing* [60]. Explored in projects such as Folding@home [21] or BOINC [70], VC uses idle resources of computers (e.g., CPU scavenging) for scientific tasks (e.g., protein folding simulations, prime number research) that require massive computing power or large amounts of data. Crowdsourcing [22] is a related approach. It uses volunteers to produce data, e.g., to label data sets. The Amazon Mechanical Turk (MTurk)[3] is a commercial example of crowdsourcing against a remuneration.

Volunteer Deep Learning: combining VC and DL. We use the term *Volunteer Deep Learning (VDL)* to refer to configurations when VC is applied

[3]https://www.mturk.com/

to train DL models [95, 181, 201]. VDL leverages federated learning [45] to train DL models on confidential data directly on the devices holding the data. Compared with centralized model training, VDL presents several critical challenges:

- **Synchronization** requires efficient communications and to take into account device heterogeneity.
- **Security** countermeasures are needed to guarantee training integrity and privacy.
- **Deployment** guidelines should be defined to support a high diversity of hardware and software.

While current works have focused on improving performance, security and deployment remain little studied. The VDL use-case requires scalability to support the maximal number of participants. In a first approach, only training integrity is needed. This implies to guarantee aggregation integrity and to protect the system against Byzantine threats.

6.5.3 Security Countermeasures

We assessed the composition of countermeasures related to aggregation integrity and Byzantine protection.
More specifically, we evaluated the following countermeasures for Byzantine protection:

- `None`: Aggregation is performed by averaging the models sent by clients as in traditional FL.
- `Median`: Aggregation outputs the element-wise median of weights of clients.
- `FLAME`: Following the approach of [130], aggregation is based on clustering to eliminate outliers, e.g., using geometric filtering. DP is also used to protect against backdoors.

Regarding the last approach, our implementation did not implement DBSCAN filtering, and as such underestimates CPU usage.
We also evaluated the following countermeasures regarding aggregation integrity:

- `None`: the aggregator averages models without any proof sent to clients.
- `None_Sec`: the aggregator sends an empty proof to clients, automatically accepted.
- `Whole`: each node receives the hashes of the aggregated model and of models from other clients, downloads them and performs again the whole aggregation operation.

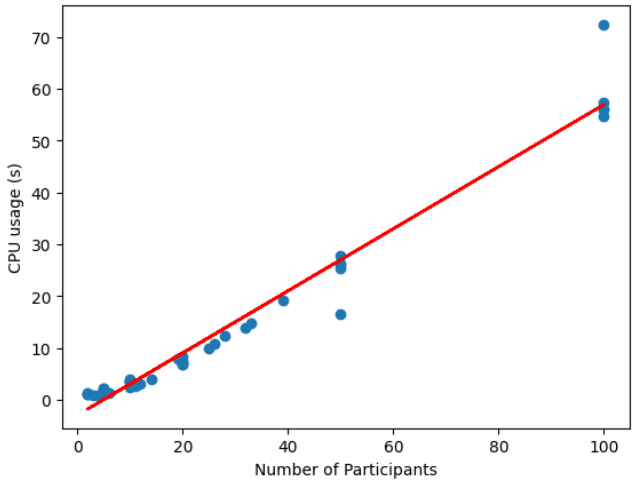
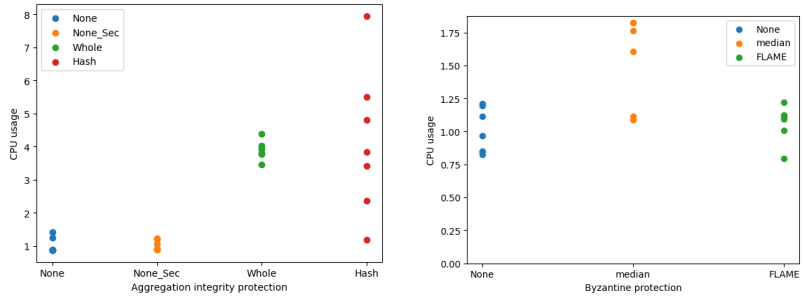


Figure 6.6: CPU usage vs. number of training participants

- `Hash`: this countermeasure is based on "incremental" *(homomorphic) hashing* [5]. The models are "hashed" by being projected on a smaller space. The verifier receives the hash of the aggregated model and of other client models. It checks the means of the hashes of clients models is identical to the hash of the aggregated model.

The `Hash` aggregation integrity countermeasure is not compatible with `Median` and `FLAME` Byzantine protections as they do not output the means of a set of models.
In this chapter, we do not evaluate protection efficiency or compare the security guarantees of countermeasures. We refer the reader to [130] for comparisons for Byzantine protections and to [103, 152] for communication and computation efficiency evaluations for advanced aggregation integrity protections.



(a) (b)

Figure 6.7: CPU usage (normalised): (a) aggregation integrity; (b) Byzantine protections
`Whole` and `Hash` protections use IPFS to share a model. Thus, every node has

already downloaded a model and broadcasts it. This reduces the server load by preventing the server from sending all models to all nodes.

6.5.4 Scalability

We evaluate the scalability of Ti-skol when increasing the number of participants in the training process, spawning multiple processes, each acting as an independent client. We split the dataset so that each client is allocated the same amount of training data. The total amount of data remains the same when increasing the total number of clients. The size of local training datasets only results in being smaller.
Figure 6.6 shows the total CPU usage of server and clients (in seconds) needed for a training step over the whole dataset. In the case of the MNIST dataset, the training time itself is negligible. We observe a linear increase of the overall CPU overhead of the distribution with the number of participants. Such results tend to show that the framework scales well with the number of participants, as the overhead per participant remains constant [4].

6.5.5 Security Countermeasure Composition

We evaluate the CPU overhead when countermeasures are added. For each composition, we run multiple experiments with 10 clients participating in the training process. We compare to a baseline – a configuration with no security countermeasures.
Figure 6.7a evaluates the total CPU usage for different aggregation integrity protections. We observe a comparable CPU usage with `None_Sec` when an empty proof is sent. This is expected and may reflect a lightweight base cost for remote attestation. `Whole` and `Hash` protections have much higher CPU overheads – up to X4 on average, but remaining practical nonetheless.
Figure 6.7b shows the total CPU usage for different Byzantine protections. We observe an increased CPU usage for Byzantine protections that remains in the same order of magnitude – up to X2, which tends to show their practicality when used with Ti-skol.

byz_protection_type **sec_agg_type**	**None**	**median**	**FLAME**
None	1.00	1.37	0.88
None_Sec	1.24	1.51	0.72
Whole	3.60	4.95	2.97
Hash	3.41	x	x

Table 6.1: CPU usage (normalized) vs. countermeasure composition

Table 6.1 shows the mean CPU usage when combining Byzantine protections and secure aggregation countermeasures. In the case of homomorphic hashing,

only configurations with no Byzantine protection are evaluated, since Byzantine protections are not compatible with the `Hash` integrity protection.
We observe that verifying the integrity of aggregation with that scheme induces far more computing overheads than Byzantine protections – e.g., for `Whole` and `median` countermeasures, the overhead overhead of their combination (3.9) is greater than the result of individual overheads (0.37, 2.6). This confirms that: 1) countermeasures are usually not compositional; and 2) non-negligible overheads may result from composition.

6.5.6 Discussion

Scalability and VDL feasibility. We observe a linear increase in the total computing overhead when adding clients. This tends to show that Ti-skol is scalable when adding FL enhancements.
Such results also encourage the use of decentralized or hierarchical approaches for large-scale model training. We may wonder how this approach scales when increasing the size of the model. Such investigations are deferred for future work.
Security countermeasures composition. Experimental results also show that the security features necessary for practical VDL induce non-negligible CPU overhead, especially secure aggregation and that their combination may exceed the sum of individual costs. The security countermeasures composition and management of incompatibility goals for Ti-skol may thus be considered achieved in a first step.
Results also show that the different protections stay within the same order of magnitude in terms of computing time (but still with non negligible overhead). Those first experimental results suggest that VDL still remains feasible with an upper bound not exceeding 5 for the overhead – to be confirmed on a testbed VDL platform, at a higher scale and with bigger training datasets.
Ti-skol would show its benefits when there are more computing between synchronizations, e.g., for bigger datasets and with more epochs per training round. Ti-skol then paves the way for VDL at scale using decentralized learning approaches and neural networks with more parameters.

6.6 Related Work

The Ti-skol approach could be applied to a significant body of work in two broad research areas: FL and VDL.

6.6.1 FL Frameworks

Existing architectures and frameworks for FL explore properties such as legacy compatibility, scalability, resilience and security composition. They include FL simulation and production frameworks and decentralized learning frameworks.

FL simulation frameworks. Middleware such as Tensorflow Federated or

PyTorch provide highly customizable environments to simulate FL training and add enhancements in a modular manner. Security components can be implemented but their combination often requires a redesign of the whole system. Such middleware are not specifically designed to be deployed for real-life use-cases. As such, scalability and resilience concerns are not usually considered.

FL production frameworks. Middleware such as Flower [72], FedML [75] and ModularFed [141] aim to provide fully deployable frameworks. By design, they handle resilience and scalability for centralized FL and may provide security countermeasures – usually not freely configurable and that need to be placed explicitly in the architecture. Unlike Ti-skol, such frameworks lack expressivity for enhancing FL due to incompatibility with a reference FL architecture such as FLRA [109].

Decentralized Learning. Architectures and frameworks such as Gossip Learning [25], FDFL [142] and Hivemind [78, 121] provide resilience and scalability. They do not explore the implementation of security countermeasures. Addition of legacy components is more difficult because they diverge from traditional FL. Therefore, such architectures are not compatible with all enhancements. Ti-skol allows for the extension of FDFL for security composition.

Unlike existing FL frameworks, Ti-skol provides the customization expressivity of simulation frameworks, including for fully decentralized learning approaches while being deployable. It also remains scalable and supports composition of security countermeasures and management of their incompatibilities.

6.6.2 VDL Frameworks

VDL projects include [49, 95, 156, 181, 201, 202]. [95] includes resilience mechanisms and AWS spot instances to reduce computing costs (cloud optimization). Its client-server architecture is implemented with the VC framework BOINC.
For training, two VDL projects are up to our knowledge implemented and ready-to-deploy. Hivemind [78, 121] is used to train LLMs[4]. Disco [143] proposes to participate in the training of DL models directly in the web browser, in a federated or decentralized manner – without need to install a software. For inference, Petals [120] runs in a cooperative manner models too large to fit on a single computer.
Unlike current VDL frameworks, Ti-skol supports composition of security countermeasures to guarantee training integrity of VDL tasks.

6.7 Conclusion and Next Steps

In this chapter, we proposed an architecture and framework supporting the composition of FL security and privacy countermeasures and countermeasure incompatibilities. The framework is scalable and compatible with most FL

[4]https://training-transformers-together.github.io/

enhancements. We showed how the framework can be applied to a VDL use-case, highlighting the overhead of countermeasures, individually or in combination, tending to show the framework practicality for on-demand FL security. Ti-skol enables benchmarking of security countermeasures and their composition for FL frameworks.
Future work includes extending Ti-skol with a library of standardized security configurations for common FL use-cases. This would be a first step towards an autonomic FL security framework to dynamically adapt defenses to application security requirements.

Part III

Other means of cooperation for Cooperative Autonomous Systems

Chapter 7

Other means of cooperation for CAS

In part II, we explored cooperation for autonomous systems through collaborative learning with Federated Learning. We showed how we can improve its performance through decentralisation. Furthermore we identified the security properties to be guaranteed in collaborative learning, what are the available security countermeasures, and, how these countermeasures can be implemented and combined to address varying security and performance requirements.

This contribution reviews some other means of cooperation identified in Chapter 3 using similar approaches in decentralisation to improve performance and countermeasure composition for security, each section being independent. We study here Function as a Service (FaaS) for task delegation and Online Social Networks for status sharing. Online video games is a special case as it does not directly fit as a mean of cooperation but aims to provide an environment for the simulation of multiple autonomous systems.

FaaS is a key enabler of cloud computing, particularly for low-latency tasks like machine learning inference. Traditional models rely on centralised cloud providers, which can introduce latency and trust issues. In Section 7.1 we present D-lambda a decentralised model for FaaS. It is proposed as a simplified case of task delegation. D-lambda extends FaaS to the edge-cloud continuum with a fully decentralised, multi-provider approach, improving flexibility, scalability, and resilience. We analyse its performance and security requirements, and propose mechanisms to meet them.

In Section 7.2 we present ReDOSN an architecture for designing and building decentralised online social networks. Taking inspiration in the cooperation of humans using social networks, It allows autonomous systems to collaborate by sharing their status. ReDOSN addresses the limitations of centralised and federated social networks such as privacy concerns and performance bottlenecks by removing the single point of failure and identifying security countermeasures. We demonstrate how ReDOSN can be built with existing decentralised tech-

nologies, arguing that data availability is not necessary to build an online social network.
Current multiplayer video games struggle with scalability and rely on tightly coupled client-server architectures. Section 7.3 presents D-ECS, a model to convert any ECS based games to multiplayer and especially to decentralise multiplayer, accounting for the security issues. D-ECS decouples game logic from network logic, allowing seamless transitions between client-server and peer-to-peer setups. We examine the performance improvements it brings and outline the security challenges along with their respective countermeasures. It is not a mean of cooperation but instead aims to provide a secure and efficient simulated action environment for autonomous systems.
Together, these contributions extend the study of decentralised cooperation in autonomous systems beyond collaborative learning. They demonstrate how decentralisation and security countermeasure composition can be applied to various domains such as task delegation, status sharing, and simulation—to build efficient and secure cooperative systems.

Associated publications (Research Papers available on HAL):

- **Divi De Lacour**
  D-Lambda: a fully decentralised serverless model, Research Paper, `hal-04805193`, 2024
- **Divi De Lacour, Adam Gautier**
  ReDOSN a customizable Decentralised Social Network, Research Paper, `hal-04889747`, 2025
- **Divi De Lacour**
  D-ECS: Towards decentralising video games, Research Paper, `hal-04886531`, 2025

7.1 D-Lambda: a fully decentralised serverless model

Functions as a Service (FaaS) is a crucial element of cloud computing, particularly for machine learning inferences. To reduce latency, cloud offloading has been proposed, positioning the FaaS provider closer to the client on the Edge-Cloud continuum. We extend this approach by proposing a multiprovider, fully decentralised model to enhance the flexibility, scalability, and resilience of FaaS. Additionally, we identify the necessary security properties and available countermeasures.

7.1.1 Introduction

Functions as a Service (FaaS) is a cloud computing approach that allows for the execution of functions without the need to manage the underlying infrastructure. This model offers a pay-per-use billing system and provides elasticity, making it particularly useful for Machine Learning (ML) model inference [135, 159].

Current FaaS approaches typically rely on a single cloud provider, this introduces latency issues due to the distance between the client and the data centre. In addition, this approach faces challenges in scalability and resilience due to the dependence on a single provider.

Multicloud approaches are used to call multiple providers at the same time, looking for the best performance/cost equilibrium and improve the scalability and resilience by handling multiple providers [140].

To improve FaaS latency, it has been proposed to execute loads closer to client on Edge-Cloud continuum. When Edge capacity is reached, cloud offloading transfers the load to the cloud to maintain availability, but at the cost of higher latency [83, 154].

P2P approaches have been proposed to handle cloud offloading [154] and load distribution [97, 98] in FaaS settings, although these approaches often leave security considerations underdeveloped.

Other works focus on the security of FaaS [150], especially on the use of Trusted Execution Environments (TEE) [65] to guarantee the computation integrity and data privacy.

In this chapter, we propose a new model for decentralised FaaS, allowing for flexible scheduling and offloading. We identify the necessary security properties and the available countermeasures.

The chapter is organised as follows. Subsection 7.1.2 reviews the FaaS and decentralisation approaches. Subsection 7.1.3 describes our proposed model. Subsection 7.1.4 identifies the FaaS security challenges our model faces and the possible countermeasures. Subsection 7.1.5 discusses the paths to be explored. Subsection 7.1.6 concludes.

7.1.2 Related work

Functions as a Service (FaaS) is a cloud computing model that allows developers to execute individual functions or pieces of code in response to specific events without the need to manage the underlying infrastructure. In this model, the cloud provider automatically provisions, scales, and manages the servers required to run the code [151]. Users are billed based on the actual execution time and resources consumed by their functions, rather than on pre-allocated server capacity.

FaaS challenges and solutions

Surveys such as [135, 151] have reviewed the challenges of serverless computing. We focus here on three primary challenges: scalability, resilience, and latency.
For scalability and resilience, two complementary approaches are possible: using multiple providers (the multicloud approach [140]) and improving FaaS scheduling. Utilizing multiple providers enhances elasticity and resilience by distributing the load across various cloud services [26]. Scheduling involves the efficient allocation of computational resources to execute functions in response to events. It optimizes performance, ensuring scalability and resilience. Effective scheduling dynamically distributes resources to handle varying workloads, maintaining system responsiveness and reliability.
Latency is another significant challenge, particularly in scenarios involving cold and warm starts. A cold start involves initializing a new execution environment, causing significant latency (can add more than 2 seconds of latency [131]), while a warm start uses an already-initialized environment for near-instant execution. Solutions to mitigate latency include scheduling to prevent cold starts such as pre-warming (keeping execution environments warm by periodically invoking them), provisioned concurrency (asking the cloud provider to keep a specified number of execution environments always warm) and event triggers (using an event trigger to pre-allocate resources). To reduce warm start latency, a possible approach is edge computing, placing the execution environment closer to the client in the IoT-cloud continuum.
To enhance resilience and scalability, decentralisation through peer-to-peer (P2P) approaches are the next step, as showed with IPFS [123] for file sharing. This allows multiple providers to be found and called by a client. This extension of the multicloud approach enables better distribution of providers, with some positioned closer in the continuum for improved latency. Additionally, the volunteer computing approach [60, 71] can be leveraged to exploit unused computing resources, particularly in consumer devices, further enhancing the system's scalability, resilience and latency.

Decentralised FaaS

Current works on decentralised FaaS, such as Serverledge [154], aim to offload computation to edge nodes and cloud regions. Serverledge organizes edge and cloud nodes into different regions and zones, allowing users to send requests

to any edge node. When resource consumption becomes too high in a node, scaling is handled automatically. Nodes can offload tasks vertically to the cloud or horizontally to another edge node.
Another approach, DFaaS [97], uses a peer-to-peer (P2P) network for edge nodes to exchange status information and offload tasks if needed. Each 5G/WiFi station has an associated edge node, and a proxy within the node decides whether to offload tasks.
Existing approaches have limitations. They do not address security concerns. DFaaS and Serverledge require edge nodes with datacenter-like capabilities to handle numerous functions simultaneously. Serverledge also relies on a global registry, which poses resilience issue, and lacks customisability such as thematic nodes (e.g. LLM specialised nodes). Serverledge operates within a single cloud ecosystem with a single scheduler, limiting resilience and creating vendor lock-in.

7.1.3 Proposed model

In this subsection we present our D-Lambda architecture (Figure 7.1). We propose to create a virtual broker on a peer-to-peer network. Clients and providers meet and negotiate the execution of functions on this P2P network.
We propose to feature four components, separated in the client and provider infrastructures. On the client side: The **client** calls a function. The **querier** handles the search for a provider and the negotiation of the provisioning. On the provider side, the **provider** looks for potential clients and handles the negotiation. The **executor** executes the function.
This architecture allows for a client to use multiple providers at the same time for better scalability and resilience. It also allows to look for closer providers on the IoT-Cloud continuum for a lower latency.
We detail in Figure 7.2 the execution process of our architecture. For the initialization part (cold start). The client first asks the querier to look for potential providers on the P2P network. The querier then contacts the potential providers and negotiates the provision with them. Once the negotiation is done, the provider loads the function in the executor. The functions are then called by the client through the querier that transmits them to the providers executing them with their executor. On reception of the output, the querier sends an acknowledgement to the provider.
The search for a provider using a P2P network can be done in multiple ways. It can be done using a topical pubsub [115] or a specific key on a DHT [23]. Those rendez-vous points allow for a pre-selection of the type of provider for some topic (e.g. compute power, geographical zones). The search can be either active with the client broadcasting its requirement or passive with the providers advertising themselves.
The Querier has the responsibility to choose a reliable provider (e.g. Reputation systems [33], testing latency with a ping).
The Querier verifies that the clients requirements are compatible with the providers specifications (e.g. Listing 7.1). It is also in charge of anticipating the perfor-

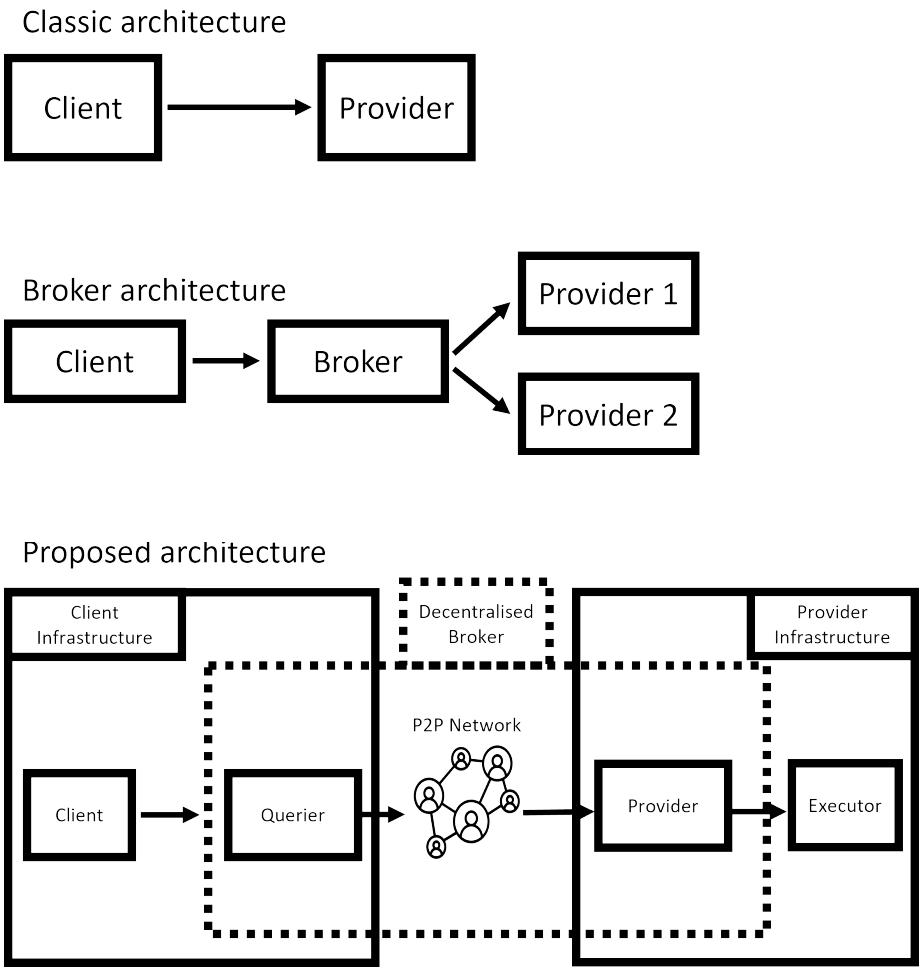


Figure 7.1: Decentralising the broker

mance issues by offloading from the edge to the cloud or contacting multiple providers.

Listing 7.1: Example of provider specifications

```
"cpu": {"cores": 2},
"memory": {"size": "512MB"},
"gpu": {"vram": "2GB"},
"tee": {"enabled": true}
```

Once the Querier has found a possible provider, it negotiates a contract for the execution of the function. It is an agreement signed by both parties that specifies

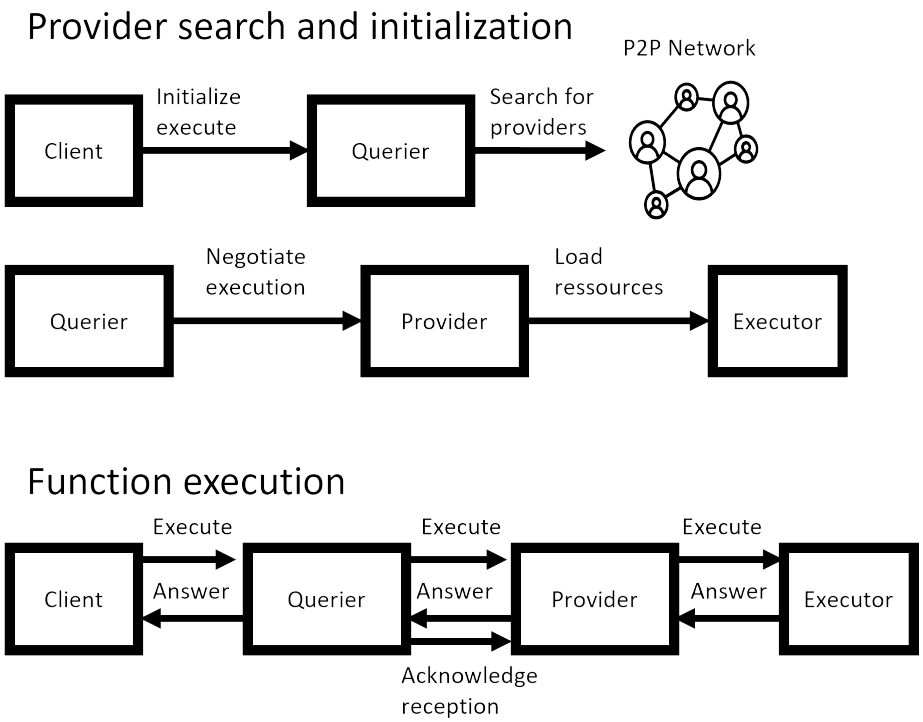


Figure 7.2: D-Lambda execution process

the conditions of execution (e.g. number of calls, cost, specific technologies like TEE, quality of Service).
In case of conflict between the client and provider on the contract execution, the arbitrage can be done using a trusted third party or a smart contract on a blockchain.

7.1.4 Security

In Subsection 7.1.3, we presented our model of decentralised FaaS. In this subsection we delve in the security of our model. We study how each actor (clients and providers) can protect themselves from malicious behaviors. We consider that components of the same infrastructure trust each other. We first identify the security properties to guarantee from the clients and providers point of view. We then identify the possible countermeasures.

Security properties to guarantee

From the *client point of view*, **Input/Output privacy** is the main privacy challenge: the system should prevent the provider from having access to the

functions inputs and outputs. It may also require **Function Privacy** where the provider does not have access to the function executed on its infrastructure. The client requires infrastructure integrity and availability, the **Integrity** of the function output is needed, the FaaS provider should also guarantee the **Availability** of executors to limit the impact on clients of failures and scaling. The client also expects **Billing Honesty** from the provider where only the resources used are billed and **Behavior Privacy** where the provider and members of the network do not track abusively the client behavior in time and number of requests.
From the *provider point of view*, the main security challenges are **Execution Isolation** and **Billing Honesty**. Execution Isolation ensures that requests are isolated from the rest of the infrastructure. This means that they gain no reading or writing access to other functions being executed or to the provider's underlying infrastructure. Billing Honesty ensures that clients pay accurately for the execution of their requests.

Security countermeasures

We distinguish the following countermeasures to meet the security guarantees. There is no countermeasure covering the full set of protection requirements. They must be combined to provide the required security properties. However, their usage may induce overheads in compute and latency.

Execution Isolation – TEE/ Hardware isolation: TEEs like Intel TDX, ADM-SEV, ARM Trustzone are hardware isolated Execution Environments with encrypted memory that guarantees integrity of the execution and privacy of its data (Function and Input/Ouput Privacy). They feature remote attestation schemes that allows to remotely verify that a program is being executed in a TEE [65]. In our proposed model, the querier establishes an encrypted connection with the executor's TEE and remotely attests that it is connected to a TEE. It sends the workload (code and data) directly to the TEE and receives the output using the encrypted connection.
Software isolation include containers or WASM, which isolate the workload from the OS and other processes [86], preventing malicious workloads from interfering with other workloads.
Cryptographic countermeasures – Homomorphic Encryption (HE) allows to compute on encrypted data, this guarantees the privacy of the Input and Output data [53]. It can be added to our model by encapsulating the function to be called. Current schemes are too costly to be applied.
Consensus techniques like blockchains or trusted third parties can act as arbitrators by reviewing the execution results to guarantee the integrity of execution. They can also review the exchanged messages to arbitrate the billing in case of conflict [89]. They are added in the querier and provider components. A reputation system [33] can be featured in the querier and provider to encourage availability, execution integrity of providers and billing honesty of both providers and clients by signaling bad behaviors.

Networking Counter-measures – Distributed Hash Tables (DHT) [23] help by maintain the availability by providing more possible providers to the client in case of failures or scaling. Networking anonymity techniques like onion routing [15] and Mixnets [17] can help keep the client's behavior private by hiding its identity and its requests behaviour [24].

7.1.5 Discussion

The proposed model has multiple paths to be explored to improve performance. The first path to explore is the combination of security countermeasures and their impact on performance. Another approach is function storage, by hosting and sharing functions on IPFS for better resilience and scalability. Scheduling improvements can be made by having providers detect the most demanded functions and pre-load them in anticipation, as well as by clients broadcast their anticipated load. Performance can also be enhanced by providers offering multiple functions simultaneously, as seen in models like tinyfaas. Additionally, "private networks" can be established to offer better security and performance guarantees, with their own reputation systems and search mechanisms. Finally, this FaaS model could also be extended to microservices, allowing for stateful loads.

7.1.6 Conclusion

We have presented the base architecture for a fully decentralised FaaS, designed to enhance scalability, resilience and latency. Additionally, we have identified the security aspects of this model and the associated countermeasures with their place in the architecture. Future work will focus on implementation and benchmarking, with particular attention to evaluating latency during both hot and cold starts.

7.2 ReDOSN a customizable Decentralised Social Network

In Section 7.1, we saw task delegation as a means of cooperation with the subcase of function execution. We focus here on another mean of cooperation, status sharing, explored with Social Networks.
Social Networks have become a major part of people life, used to communicate and cooperate between humans and even with connected objects. Current social networks are server based, either centralised or federated. This poses security, privacy and performance issues as the server administrators must be trusted and the server is a single point of failures. Decentralised social networks proposed to tackle those issues lack customization. We propose a model to create customizable social network, focusing on the decentralisation and its security. We show how it can be implemented using existing decentralized technologies, arguing that data availability is not a hard constraint.

7.2.1 Introduction

Online Social Networks (OSN) have been a major innovation of the 21th century. They are used for efficient cooperation and information sharing between humans, for example for sharing alerts [189], or collaborating at work [170].
OSNs are a medium for human-machine interactions and cooperation (symbiotic) [129], they are also useful for cooperation between autonomous agents (social Internet of Things approach [79]) where connected objects imitate human social networks to better cooperate.
Current OSNs are mainly centralised, either with a single server or with a federation of servers. This poses performance and security issues as users are dependent of a server that can see their content and is a single point of failures [195].
Decentralised OSN (DOSN) [30, 81] have been proposed to tackle those issues, however they are designed to reproduce a specific centralised social network and as such lack the ability to be customized. For example Mastodon [62] is designed to imitate twitter and so only allows for small public messages. It cannot scale for video hosting that is handled by PeerTube, the federated alternative to YouTube.
Content hosting and guaranteeing data availability at scale is the main technical challenge for OSN. In centralised OSN, content hosting costs are handled by showing ads to users and using their data for marketing or AI training purposes. However this is hardly feasible in DOSN, culturally and technically, since there is not central authority guaranteeing data availability and DOSN are designed for an improved privacy. Data replication schemes with cryptocurrencies incentives [84, 148] have been proposed for the collaborative hosting of data, but they remain complex to implement and secure.
Data availability is not guaranteed by centralised OSN [198], they may arbitrarily remove any content (e.g. Google has the rights to delete inactive ac-

counts [188] to limit hosting cost). Since data availability is not guaranteed in centralised OSN, we argue that it is not necessary for decentralised OSN. In our approach, content that is not popular enough to be hosted by anyone can legitimaly be removed from the OSN.
Securing DOSN is also an important challenge as they can be victim of various attacks such as communication privacy breaches [24], attacks on decentralisation [23] or spam. The integrity and privacy of data and communication must be guaranteed without the help of a centralised trusted third-party while protecting from toxic content.
This chapter provides the following contributions: We propose a customizable model for social networks and show how it can be decentralised. We identify the security properties of our model and the counter-measures that can be added. We propose an architecture and its technological stack for an implementation of our model.
This chapter is structured as follow: In Subsection 7.2.2 we review the current OSN approaches. Subsection 7.2.3 presents our proposed model for social networks. Subsection 7.2.4 presents our architecture and the associated technology stack.

7.2.2 Related Work

In this subsection we review the current approaches to decentralising OSN. We identify the core functionalities and security properties an OSN may feature.

Decentralising OSNs

We distinguish three types of OSN in terms of architecture (Figure 7.3):

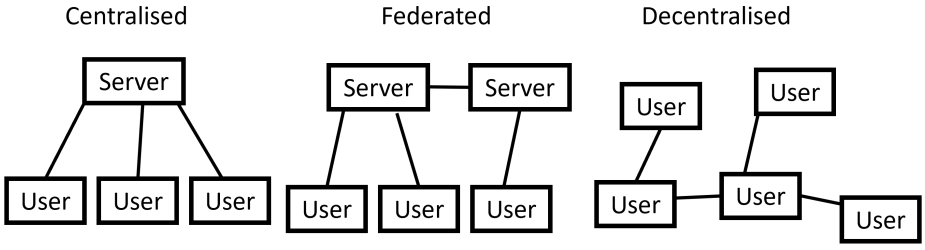


Figure 7.3: Types of OSN

Centralised – OSN hosted and administrated by a single entity [81] (e.g. Twitter, Facebook). They may be distributed on multiple data-centers for better scalability and resilience as they are limited by the providers infrastructure. They show privacy and security issues as the administrators have a full control on the OSN and may block or read any content.
Federated – Multiple OSN are hosted and administrated by different entities following the same protocol [47] (e.g. mail protocol, Mastodon). Users of an

instance can interact with users of another, for that they are identified with a *user@server* identifier. The W3C [203] has worked on the normalization of such protocols.

Decentralized – The users interact directly with each other without using centralized hosting instances [81]. Each peer is in charge of hosting a part of the social network.

Data hosting is the main challenge of decentralized social network as it must guarantee data availability of the content with a high client churn. Distributing hosting is sensitive to Sybil attacks [12] where a single entity creates multiple identities to get more storage. Decentralized OSN usually limit Sybil attacks on free hosting and spam by using monetary incentives in forms of cryptocurrency [148].

To design DOSN, two technologies are used in existing projects (Figure 2.1):

Blockchain [58, 94, 158] approaches write all of the content in a distributed ledger copied over all of the OSN users. Users share their content (transaction) by propagating it by gossip in the network. Every x amount of time, the new content is added to the ledger as a block pointing to the previous block of the blockchain. All of the hosted content is copied by each peer, this drastically limits the scalability of the OSN in the number and size of messages.

Distributed Hash Tables (DHT) [13] approaches allow to search for clients sharing a specific content (user, file, service) using its identifier (e.g. file hash in IPFS [138]). The hash table is distributed between the users to allow for resilient and fast look ups. DHT based OSN often feature a protocol to shard the user content into multiple clients for redundancy [102].

Functionalities of OSN

OSN allow for different types of interactions that may be themselves a composition of other interactions. We identified in OSN the following functionalities:

Communities/recursive: grouping the users in sub-communities with their own set of rules and functionalities. A user may be part of multiple sub-communities as in Discord with its servers. Sub-communities may be created in a recursive way, inherinting the properties of their parent.

Profile: a public and unique file is associated to a user, only he can modify. Anyone can find the profile knowing the user (e.g. Facebook wall, profile picture).

Direct messages: sending direct messages be it text or streams between two users (e.g. text, phone/video call).

Groups: direct messages are sent to a predefined group of users (e.g. chat groups). They follow the publish-subscribe (PubSub) [115] pattern where users can subscribe to a subject of interest and receive messages pertaining to this subject.

Inter-OSN interactions: interacting in an OSN with the content of another (e.g. using Facebook authentication to log in another website, sharing a YouTube video on Facebook). The W3C Fedpub standard [172] aims to standardize inter-OSN interactions.

Boards: are spaces where users can write. All of the content written on a board can be found by consulting it (e.g. product comments on e-commerce sites, forum discussions, 4chan boards).
Content Search: searching for a content either by identifier or with semantic search. (e.g. DHT to search content by hash [13, 138], search engines for semantic search [169])
Transactions: establishing contracts with other users, especially to buy or sell content. (e.g. Facebook Marketplace, Ethereum transactions and smart contracts [158])
Other functionalities that improve the user experience include:
Username: provide the possibility to look for user and content through a human readable name (e.g. usernames instead of public key, domain names instead of IP address for websites).
Timestamp: provide a proof of the content existing at a specific time [100].
Other specific functionalities can be added for specific cases by combining existing functionalities (e.g. making votes and polls using boards where each user can only write once).

Security of OSN

In this subsection, we identify the security properties needed in OSNs.
Integrity – *Content integrity*: the recipient of a content should be able to verify it was produced by the right person and not altered in the transfer.
Functionality isolation: functionalities should not impact the working of other (e.g. the board functionality should not interfere with direct messages). An automatism on a specific functionality should be isolated from the rest of the functions (e.g. a spelling checker not interfering with audio communications).
Availability – *Function availability*: functionalities should keep working in case of system failure and scale in the number of users or exchanged content. *Data availability*: The user content on the OSN should be available at anytime.
Privacy – *Exchanged content*: content should be accessible only to those authorized by the author. *Behaviour*: user behaviour like content consulted and times of connections should not be accessible to others [24].
Malicious users – *Toxic and illegal content* should be hidden from users. The OSN should be able to detect and limit the impact of spam.

7.2.3 Proposed model

We propose ReDOSN (Recursive Decentralised Online Social Networks) a model to create customizable decentralised social networks. It is a middleware (Figure 7.4) that allows to reproduce most social networks with an API for either a human user's GUI or for autonomous systems to cooperate. It provides the functions identified in Subsection 7.2.2 based either on a decentralised or centralised organisation of the infrastructure, connecting the members of the organisation using different network protocols.

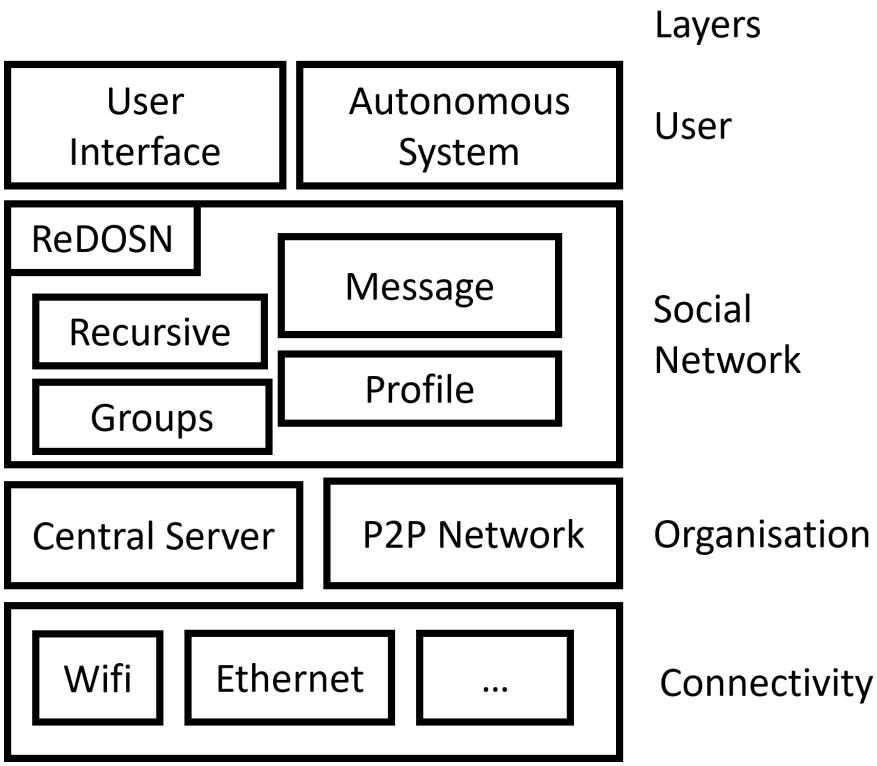


Figure 7.4: ReDOSN software stack

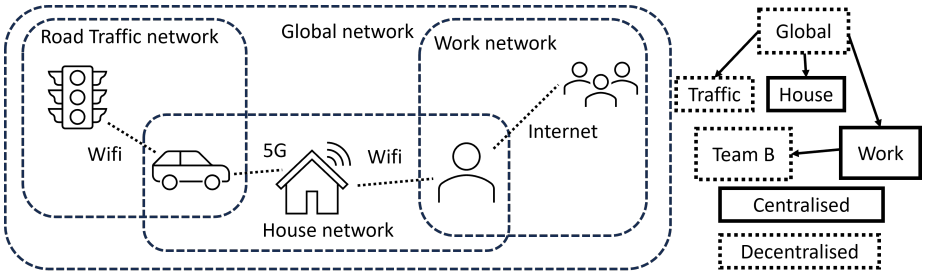


Figure 7.5: Composition of OSN

Recursive OSN

As illustrated in Figure 7.5, we consider in ReDOSN that a social network is a composition of sub social networks or communities with their own rules. A user participates in multiple OSN as in Helios [167]. Each OSN has its own functionalities (customization), access rights (e.g. a specific house network) and connectivity (e.g. Wifi, 5G). Social Networks can be centralised or decentralised depending of the security and performance needs, for example a road traffic network may be decentralised for better latency and a house network can be centralised around the WiFi router.

To handle multiple OSN and the possibilities of sub-OSN, we propose a recursive approach where the OSN are composed as a tree (Figure 7.5). This account for the possibility to participate in multiple independent OSN at the same time, or to participate in a sub-OSN inheriting its parent properties (e.g. security policy).

Figure 7.6 shows how ReDOSN is organised as a tree where functionalities are attached to OSNs, those OSN may handle the functionalities in a centralised or decentralised way. OSNs feature a recursive functionality in charge of handling the sub-OSNs. This defines a path to reach the functionality of a sub-OSN from the root OSN. Calling a remote client follows then the format *functionality:user@OSN1/OSN2.*

Figure 7.6 illustrates simplified visions of WhatsApp, Microsoft Teams and Reddit with the ReDOSN model. WhatsApp features direct messages and group messages, the profile is used to set a profile picture. Teams features direct messages, users can join teams which are sub-OSNs with their own access rules and discussion channels (groups). Reddit features communities called subreddit which work as sub-OSN with their own rules, they use a main board to list the post made in the community, each post is a board that features the post and its associated comments. This shows that the ReDOSN model allows for expresivity and customisation in OSNs. Contrary to other DOSN, it can be used to imitate different centralised OSN.

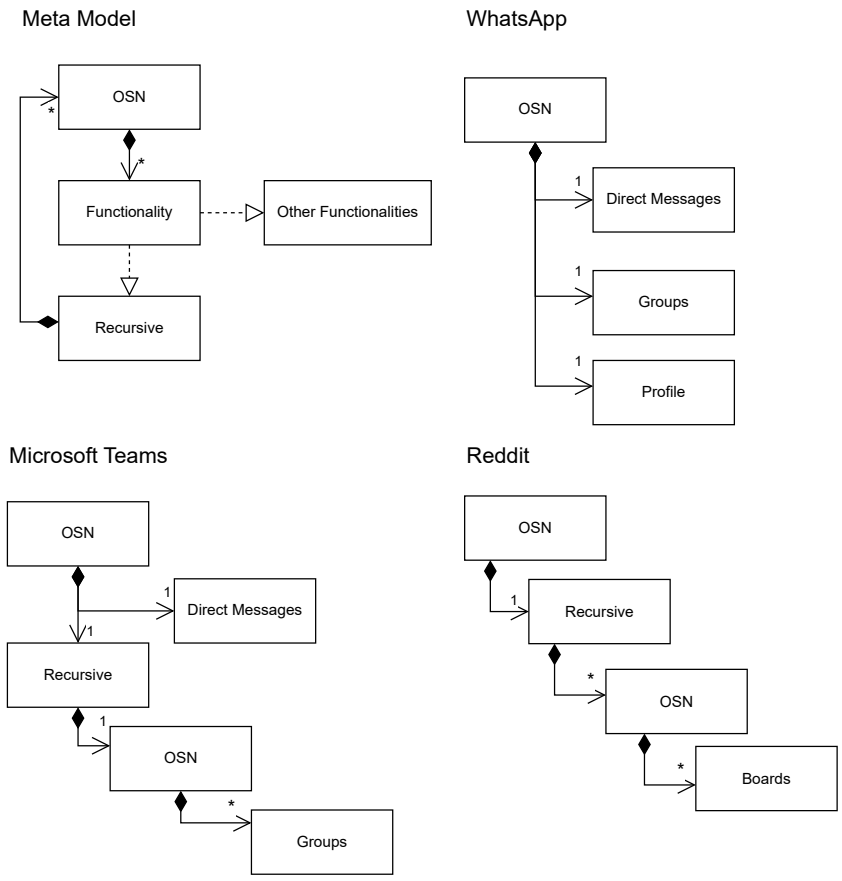


Figure 7.6: ReDOSN model and illustrations

Decentralising ReDOSN

We observed in the main decentralized content sharing tools like IPFS [124] that data availability is not guaranteed. Hosting depends of the will of the participants to keep content available. So, contrary to other DOSN, we do not aim to guarantee availability of user data. To decentralise ReDOSN, we consider here that content not popular enough to be hosted by anyone can legitimately be lost with limited impact on user experience.
As in Bluesky [166], we consider that each user is identified by a Decentralised identifier (DID) [147]. It can be an ID provided by a third party or a public key working as an ID. This allows to establish encrypted and authenticated communications between users. A sub network can either use its own certificate with a hierarchy of certificates as in https or use the certificate through its parent network.
Providing the functionalities described in Subsection 7.2.2 for centralised OSNs is a mature field. We describe here how they can be implemented in a decentralised way using existing P2P solutions. Those technologies are resilient and scalable but at the cost of reduced data availability. Username, Timestamp and Transaction functions are exceptions which use blockchains at their core, but at the cost of limited scalability in the number of calls [94].
We use the Kademlia [13] DHT to find the resources on the network in a $log(n)$ time where n is the number of users. This means that finding a ressource can be done in a quasi-constant time as the OSN scales. Kademlia has been extensively tested in terms of efficiency and resilience by its implementation in IPFS [138].
Profile: We use an IPNS [190] approach, the hash of the user public key is used to point to the static content. The changes are broadcasted on the network and signed using the user private key to guarantee integrity and prove that the update is the most recent. To provide data availability, policies such as hosting friends profiles when the user is offline can be implemented.
Direct messages: to send messages to other users, the DHT can be used to find the IP address of a user from its public key. Indirect messages can be sent by asking a tier to host and forward a message.
Groups: groups are implemented using IPFS Pub-Subs [115], the group is a topic to which users subscribe. Other subscribers to a specific topic are found through the DHT. The messages are broadcasted through gossiping, this reduces the load on the sender: it will not have to send a message to each member individually. Access rights to groups can be set by sharing a common encryption key between allowed members and setting a list of participants to prevent message gossiping from leaving the group.
Inter-OSN interactions: the user's DID can be used to identify a user on multiple different OSN. Content from other OSN can be shared using their URI. Other OSN may access content through a trusted user acting as a gateway to the p2p network.
Boards: decentralised databases like orbitDB [153, 196]can be used to store messages of boards. Each board (e.g. forum discussion) should be considered as its own database that is collaboratively hosted by those that interacted with

it. Meta-boards can be created to list all of the sub-boards.
Content Search: a DHT can be used to search for content by its identifier (e.g. searching with a hash). Decentralised semantic search is an active field of research with tools like YaCy and De-DSI [169].
For functionalities like **usernames** and **timestamping** there is a need for a trusted third party, either centralised or with blockchains approaches (e.g. Ethereum name service [182], smart contracts for time stamps [100]), but at a scalability cost.

Security countermeasures in a decentralized setting

We described in the previous subsection how our model can be implemented in a decentralised way using existing technologies. We identify here the security countermeasures applicable to guarantee the security properties identified in Subsection 7.2.2.
DHT security as been explored separately [23], we consider that they are secure against attacks aiming to prevent honest nodes from discovering and interacting with each other. The use of hashes to identify content and public keys for users guarantees the integrity of content and the user identity.
Integrity – **Content integrity**: exchanged content is signed by its sender using its private key. Boards and groups can set access rights to control which user can write and what they can write. **Functionality isolation**: the OSN and functionalities are organised as a tree this allows to define access rights between functionalities and isolate them (e.g. a functionality can call sub OSN functionalities but not the opposite). **Consensus**: to prevent Sybil attacks [12] where a single entity creates multiple identity to get more vote rights. The core functionalities are based on a DHT that does not need for consensus. Blockchain based functionalities use Proof of X approaches. A list of identities allowed to vote can also be set in other cases.
Availability – DHT based functionalities remain available by exploiting the Kademlia DHT's resilience and scalability. Data availability is guaranteed by replication [31]. For example using hosting policies where users automatically participate in hosting the content they liked or from their friends. A user can also use paid tiers (e.g. Pinata or Filecoin for IPFS) to host their content. Finding content that is provided by someone in the OSN is guaranteed by the DHT. Blockchain based functionalities have their availability guaranteed by the blockchain.
Privacy – **Exchanged content**: message content is encrypted using the recipient public key. Access control policies on groups and boards can be set where updates are only shared between a list of allowed users that also share a common encryption key. As in centralised OSN, were a single member to be corrupted, the whole content of the groups and boards could be leaked. Cryptographic schemes can be set for specific cases (e.g. polls and votes, computing the mean of a private value [152]) to guarantee the privacy.
Behaviour: network anonymization tools like TOR or mixnets [17, 180] and the use of temporary identities can limit network traffic monitoring. Other ob-

fuscation methods like dummy traffic, creating fake content requests and content padding can be added to limit privacy leakages [24].
Malicious user – To tackle toxic content and spam, reactive approaches can be applied where a trusted authority that publishes lists of whitelisted/blacklisted content/user. This has been implemented in the ublock origin browser extension to block adds through filtering lists, or in the Bad Bits Denylist for IPFS files. This is also the approach adopted in Bluesky with labels emitted by moderation services[1]. Proactive approaches can be applied by a user or a group of users using a reputation system [33], setting real-time detectors using rule-based systems or machine learning to detect and block users having unusual behaviors (e.g. sending thousands of messages, words featured in message content).
Confidential workloads – In cases where a trusted central authority is needed to run confidential workloads, confidential and trusted compute can be delegated to networks of TEE (Trusted Execution Environment) like Ekiden [56]. TEEs are hardware secured execution environments with encrypted memory that guarantees the privacy and integrity of data an execution. They feature remote attestation schemes that allow to remotely verify that a specific code is running in a TEE.
Countermeasure composition and impact on performance – There is no one size fits all security countermeasure, this warrants their composition. Contrary to [162], we expect here the countermeasures to compatible with each other. As in [171], adding security and privacy features may in most cases decrease performance. Security and privacy may also be at odds with one another (e.g., protecting data privacy makes it harder to detect malicious behaviors such as spam). Benchmarks are needed to verify that the combination of these countermeasures remains efficient.

7.2.4 Implementation

In Subsection 7.2.3 we described the ReDOSN model and identified how the functionalities can be decentralised and secured.
In this subsection we review how it can be implemented using existing technologies.

Architecture

Figure 7.7 shows an UML representation or our proposed architecture. Each functionality has its own methods that are independent of their implementation (e.g. sending a direct message will provide the same interface in a centralised and decentralised implementation). An OSN has a policy defining access rights and the configuration of the functionalities. The private key for communications associated to the DID is held by the OSN object which grants its usage to functionalities and sub-OSNs. The recursive functionality allows to join multiple sub-OSNs that can have different implementations of functionalities (centralised or decentralised).

[1] `https://docs.bsky.app/docs/advanced-guides/moderation`

Receiving messages, either in direct messages or in groups is done by setting a handler function that will be called at each received message. Groups and Boards feature a policy which is used to define specific behaviors like access rights. A user may join a group by subscribing, it can create new boards or read/write on existing ones.
Inter-OSN interactions feature the **GET** and **POST** calls defined in the ActivityPub standard [172], they allow to pull interactions received from other OSN and push content outside to another OSN.
Search can either be done by looking for the content associated to an ID or semantic search, looking for content with a certain meaning. New content can be set to be searchable using the index method, with a policy to set access rights.
A user has a profile with content it may modify, it can also find the content of other users using their identifier.
A user can set a unique human readable name for itself and others ID from their name. A timestamp of a content or a transaction can be generated and existing timestamps/transactions can be verified.

Technology stack

Figure 7.7 shows how each functionality can be implemented using existing decentralised technologies and how those technologies linked to each other. We choose technologies that can run directly in the browser, allowing to run on a multitude of environments and specially to run on user device without any software installation. We consider here that users are identified on the OSNs by their public key, this can be extended using veramo for DID.
They are two main families of decentralised technologies proposed:
Blockchain based technologies are built with the Ethereum blockchain at the core. Operations induce a monetary cost to use the smart contracts but guarantee data availability. The library web3.js is used for interaction with the Ethereum blockchain. Etherum name allows to associate a human readable name to a public key, Ethereum attestation service [2] provides time-stamping based on the Ethereum blockchain.
DHT based technologies induce no monetary cost but provide no data availability guarantee. The libp2p library provides a DHT to find other peers and content, it allows authenticated and encrypted direct communication for direct messages. It is fully available to run directly in web browsers using WebRTC and works at the core for IPFS implementations. For groups it provides a pub-sub that broadcast messages to the users following a specific subject. IPNS is a sub-system allowing for dynamic pointers in IPFS instead of static ones by file hash, this allows to build a profile that can only be modified by the user. OrbitDB is a key-value database built on top of IPFS that provides a global ledger users can read and write on to build boards. Semantic search can be done using solutions such as YaCy and De-DSI.

[2] `https://docs.attest.org/docs/tutorials/timestamping-attestations`

In both approaches, the DOSN are not directly accessible with an URL, they need a gateway for interactions with other OSN like cloudflare gateway acts for IPFS.

7.2.5 Conclusion

We proposed a model to create decentralised social networks that is completely customizable. We identified the key functionalities working as building blocks for any OSN. Arguing that data availability is not a hard requirement, we identified the decentralisation techniques that allow for efficient, scalable and secure OSN functionalities. We showed how our model can be implemented, identifying existing technologies for its decentralisation and highlighting their ability to be deployed directly in varied environments and especially in web browsers. However, an implementation or ReDOSN and benchmarks of the technologies proposed are necessary to verify that beyond their architecture guarantees, the implementations answer to the performance requirements of the OSN use-case.

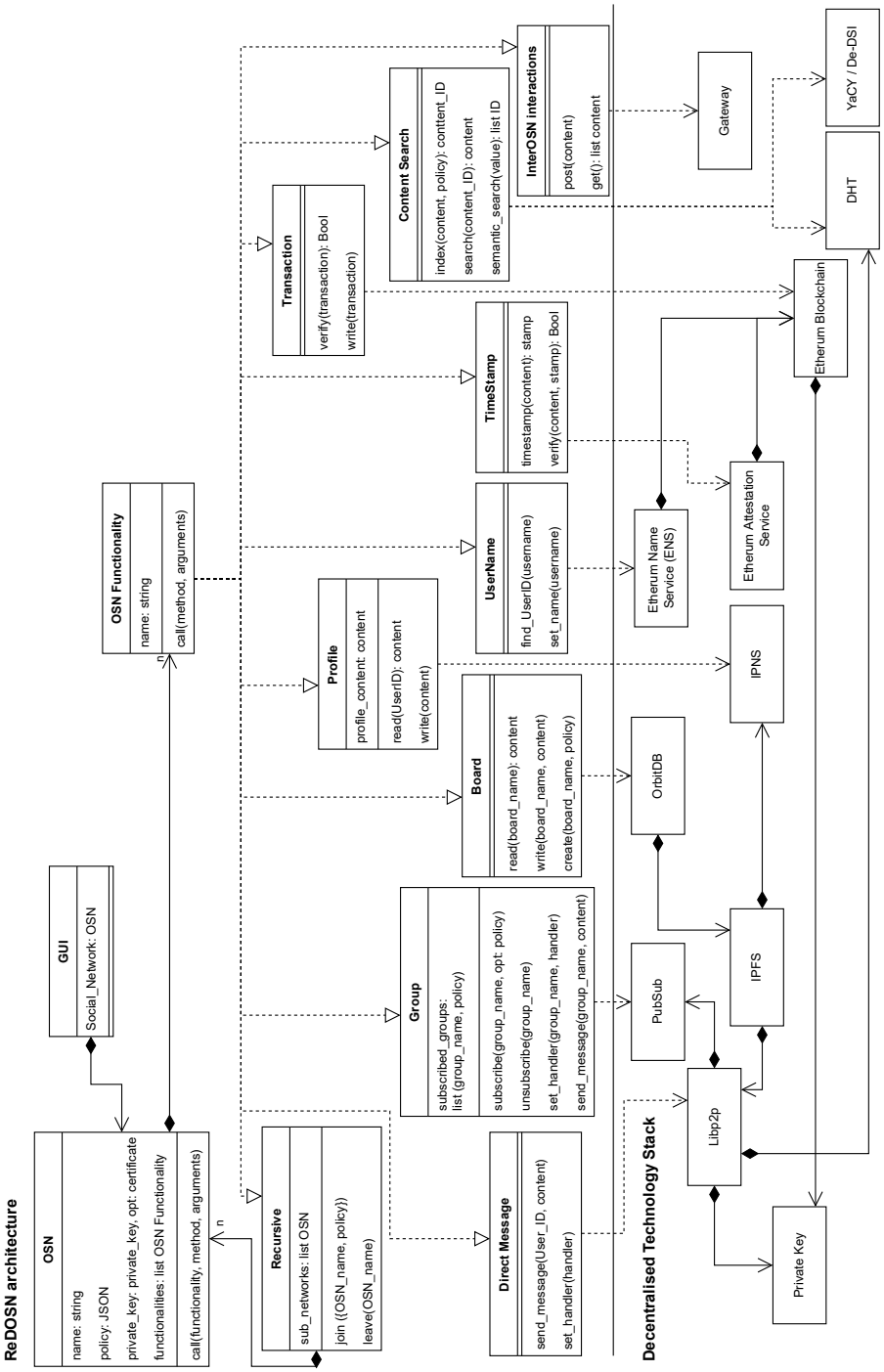


Figure 7.7: ReDOSN architecture and technology stack

7.3 D-ECS: Towards decentralising video games

Current multiplayer video games are hard to scale in the number of players and simulated entities. Developers are forced to create federations of semi-independent virtual worlds instead of a single simulated world. Multiplayer architectures beyond basic client-server architectures are not provided by game engines and must be re-implemented for each game. Peer-to-peer approaches have been proposed to improve scalability, either to reduce the server network load created by broadcasts, or to create a scalable server infrastructure based on multiple machines. However, this approach increases the complexity of video game architectures. We provide a model to adapt games for P2P multiplayer, separating game and network logic. It allows for a game to be in a client-server or peer-to-peer setting seamlessly for any kind of game. We show how it can be applied to improve the performances and identify its security requirements and associated countermeasures.

7.3.1 Introduction

The market for online games is experiencing significant growth, with revenues projected to reach 27.97 billion USD by 2024. This drives the growth of various applications, including metaverse use cases [145] and the simulation of multi-agent systems [42]. Multiplayer online games and massively multiplayer online games (MMOGs) are at the forefront of this expansion, offering complex and immersive experiences to players worldwide. They are to be distinguished from online social networks [81] which aim to provide communications without the ability to simulate a virtual environment.

Multiplayer games can be designed using either centralized or decentralized networking architectures [27]. Centralized architectures, such as the traditional client-server model, are easier to design and offer robust protection against cheating. In these configurations, a central server manages game state and player interactions, ensuring consistency and security. However, centralized architectures come with higher hosting costs and can become bottlenecks as the number of players increases [27].

In contrast, decentralized architectures aim to reduce hosting costs by distributing the computational and networking load across multiple machines or even directly among players. While this approach can enhance scalability and reduce server costs, sensitive tasks are often delegated to untrusted devices, introducing significant challenges related to security and cheating [208].

One of the primary challenges in developing multiplayer games is scaling the number of simultaneous players and the number of entities that need to be simulated. A common solution to this problem is to distribute both execution and data. For instance, games like World of Warcraft (WoW) divide the game world into distinct areas or zones (Interest Management [29]), each managed by different servers. This approach allows for horizontal scaling, enabling the game to support a large number of players by distributing the load across multiple servers. However, such games are typically built using client-server architec-

tures that can become complex and cumbersome. The mixing of game logic with distribution logic often results in a complicated server infrastructure, with for example the use of separate inventory, authentication, and area of interest servers.
Another significant challenge is the lack of a standard formalization for games, which complicates research and development efforts focused on game distribution. Most games are developed using object-oriented programming (OOP) with serialization, which can be difficult to synchronize and may introduce security vulnerabilities by introducing payloads. The Entity-Component-System (ECS) paradigm [36] offers a solution to these issues, separating data and code to be executed. ECS allows for efficient management of a large number of entities and is used in game engines such as Bevy and Unity DOTS.
In this chapter, we propose a model for flexible online video games that is adaptable to any ECS-based game and can operate in both centralized and decentralized architectures. Inspired by aspect oriented programming [6], our model separates game logic from networking logic, making it easier to scale and secure. We also study the security implications of our model and propose various optimizations to enhance performance.
The structure of this chapter is as follows: Subsection 7.3.2 reviews the related works on ECS and decentralising video games, accounting for performances and security. Subsection 7.3.3 presents our proposed architecture. Subsection 7.3.4 identifies the security countermeasures available and how they can be added to our architecture.

7.3.2 Related Work

Formalising video games

There is no standard formalization for video games, which complicates research and development. Traditionally, most games are developed using Object-Oriented Programming (OOP). However, this approach limits the ability to use multithreading to improve performances and makes synchronization harder, as it involves both data and code.
Data-oriented game design treats the game as a database, focusing on efficient data management. A specific paradigm in data-oriented game design is the Entity-Component-System (ECS) architecture [36], which is used for managing a large number of entities, as seen in games like *Cities: Skylines 2*. Better performances have been observed in games developed with a data-oriented approach [139], which is enhanced by its ability to parallelize execution into multiple threads [80].
The ECS paradigm consists of three elements (Figure 7.8):
Entities: Unique identifiers representing game objects. Entities themselves do not contain data or behavior; they are simply references that can be associated with various components.
Components: Data containers that store properties of entities, such as position, velocity, or health. Components are simple data structures and do not

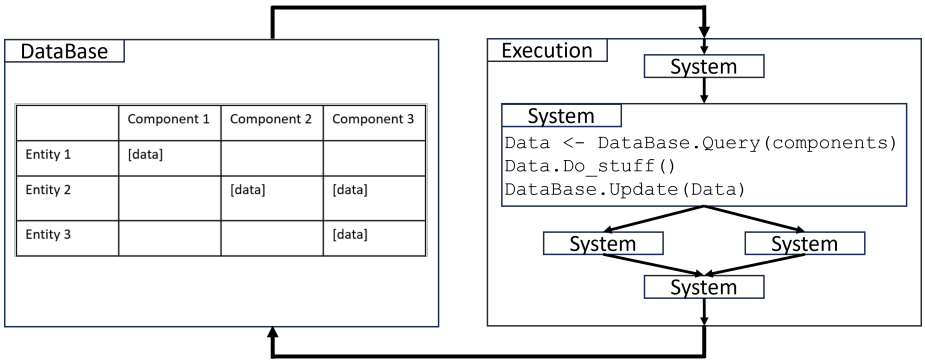


Figure 7.8: ECS illustration

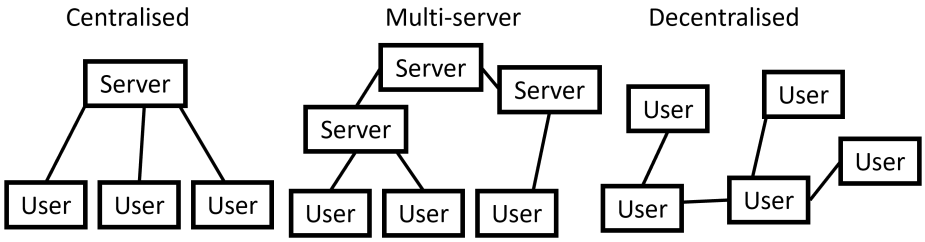


Figure 7.9: Multiplayer architectures
Client-Server Model: Centralised, Multi-Server
P2P Model: Decentralised

integrate any game logic.
Systems: Logic processors that execute game logic on entities based on their components. Systems iterate over entities with specific components and update their state accordingly.

Multiplayer video games

In multiplayer online games, clients exchange information about game objects in two approaches (Figure 7.9):
Client-Server Model: a central server or a multi-server infrastructure verifies actions and access rights. It is responsible for receiving updates from clients and broadcasting the new game status to all connected clients, ensuring consistency and security. The server infrastructure can be distributed into multiple servers communicating with each other.
Peer-to-Peer (P2P) Model: clients communicate directly with one another. They share information about the game environment without the need for a central server.
Scalability in multiplayer games encompass several critical challenges. Scalability of execution is a significant concern when the computational requirements

of game logic exceed available resources, resulting in performance degradation. Additionally, the scalability in the number of simulated entities poses a challenge, as the management of an excessive volume of data related to game objects can overwhelm system capabilities and hinder responsiveness. Furthermore, the scalability in the number of players complicates the architecture, as broadcasting status updates within constrained timeframes becomes increasingly difficult with a growing player base. A promising solution to these scalability issues is distribution, specifically through the segmentation of the game into multiple semi-independent worlds of interest, a methodology referred to as Interest Management. This strategy enables horizontal scaling, allowing the system to accommodate increased loads by distributing the processing across multiple servers.

Peer-to-Peer approaches

Peer-to-peer (P2P) approaches, as surveyed in [27], offer a cost-effective alternative to traditional client-server models by distributing the computational and networking load among multiple peers. These approaches can either involve a full P2P model with no central server or a hybrid model where a federation of servers handles different parts of the game world. An example of the latter is Second Life, which operates on a vast network of servers. While P2P architectures can significantly reduce server costs and improve scalability, they require the game to be designed with specific distribution strategies in mind, making them difficult to change and adapt. Additionally, P2P models pose significant security challenges, as any peer can potentially illegally read and modify game variables.

Interest management

Interest management [29] improves scalability in multiplayer online games by dividing the game world into distinct zones, each managed by different servers, this reduces the number of entities each player or server needs to track. This distribution of data, bandwidth, and execution load reduces the load on individual servers.

As illustrated in Figure 7.10, players are dispatched into different zones, each handled by a different server. Servers communicate with each other to handle the transfer of players between zones. Distinct servers handle the players authentication and manage the zone servers.

Strategies for interest management include static zoning, with predefined regions, and dynamic zoning, which adapts to the current distribution of players and entities [27].

However, adapting an already existing game to interest management is challenging as the game logic and distribution logic are mixed in the game architecture.

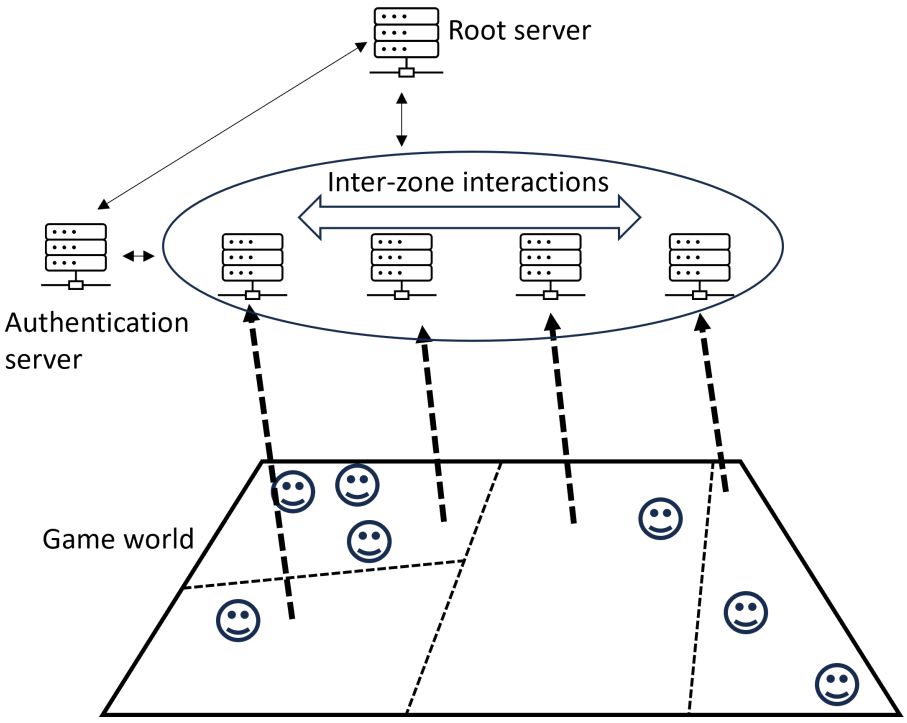


Figure 7.10: Interest management illustration

Security of online games

Online games are vulnerable to various types of cheating and security threats [208]. Unauthorized writing of values can lead to incorrect game execution and corrupted game data, undermining the integrity of the game. Unauthorized reading of values can compromise player privacy and provide unfair advantages.
Some other cheating techniques such as aim bots and modified controllers exploit automation on the user side, they use legal game actions to gain an edge over other players. Players can also use design errors in the game logic (bugs) to gain an unfair advantage.
Network attacks are also a significant concern. Withholding information during status dissemination can disrupt the game's state synchronization, and finding other players' IP addresses can lead to targeted attacks.

7.3.3 The D-ECS model

We extend the ECS model for multiplayer games with the D-ECS model, making it flexible for either client-server or full P2P online games. It can be compared to real-time database synchronisation [153]. This allows to easily port existing games (written with the ECS paradigm) into multiplayer and to separate game and networking logic for any game engine; To allow developers to focus on the core game development first and on multiplayer security and optimizations later, as it often happens in game development processes.

Proposed model

Figure 7.11 shows our proposed architecture:

- **ECS engine**: is the engine used when creating the game.
- **Executor**: executes systems in a function like approach: *function (current_state, action): new_world_state*, this makes it compatible with FaaS approaches [151].
- **Network component**: is in charge of the game state synchronization and network security.
- **Distribution engine**: defines the policy on the distribution and synchronization and replication, it handles conflicts in values. In practice, it sets the synchronization as specific systems run in the ECS Engine, and regularly calls them.

We keep the existing ECS engine as it may have its own database optimizations [153].
Developers can set behavior for distribution (e.g. scope for interest management areas, regularity of synchronization) and security (e.g. data access rights) of entities by adding components (meta-components) which are parsed by the decentralization systems. This allows to change the multiplayer policy without

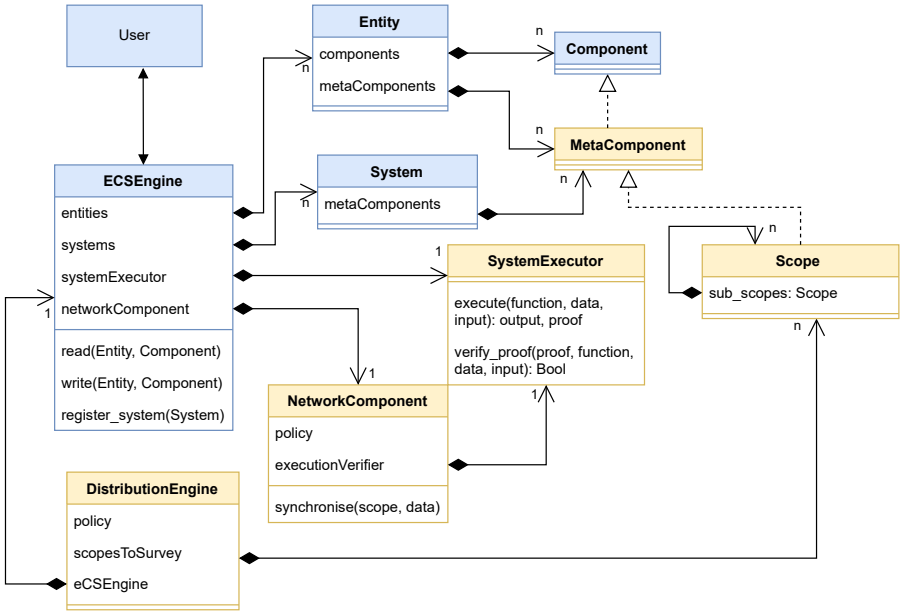


Figure 7.11: Proposed architecture

changing the logic of the game handled by normal components. This way, systems can be limited to a specific scope to reduce execution load. Specific values can also be anticipated by a client before the network update as in dead reckoning [48] which is used to anticipate the trajectories of objects in distributed virtual environments.

Performance optimization

The presented architecture can be extended for improved performances
Decentralizing the architecture of multiplayer games can help improve the performances by helping scaling and resilience.
For that, the network component can utilize either a centralized server or decentralized pub/sub systems like Libp2p pubsubs [115] to propagate updates or decentralized databases such as OrbitDB [197] hosting the entity-component data.
Scope composition allows to dynamically handle the load change in the different scopes. It can be done hierarchically with scopes featuring sub-scopes. The composition does not rely on spatial assumptions (such as a world being in 2D or 3D world), this allows for greater flexibility.
The distribution manager is in charge of fusing or spliting the scopes. It updates the entities scopes. It must make sure no entity is lost because no peer monitored it.
Inside/outside interface – In some cases, interactions between two scopes can

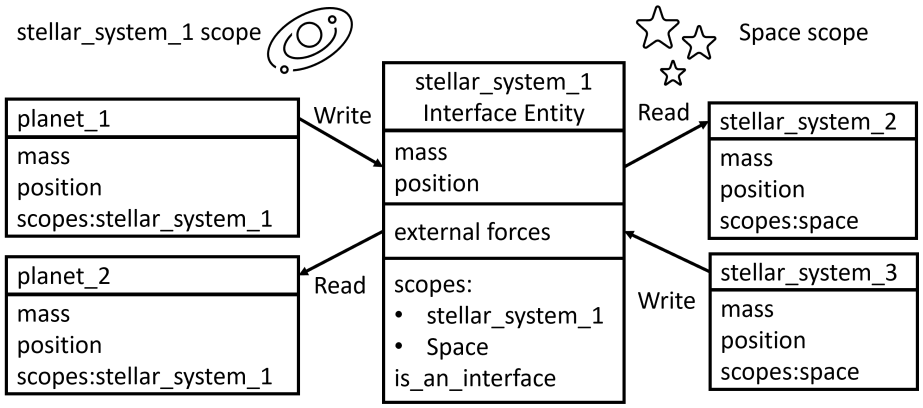


Figure 7.12: Interface for stellar system isolation

be very limited. For instance, the precise positions of planets within a stellar system can be disregarded when viewed from another stellar system, which can be treated as a single point with all its mass concentrated.
As illustrated in Figure 7.12, developers can provide a specific API to interact with a scope as a whole. This requires creating an entity symbolizing the entire scope, which has an *is_an_interface* component. Only one such entity can represent a scope, it can be present in multiple scopes and must be present in the scope it represents.
Specific systems must be added to update the interface entity, both from inside and outside. For example, in a stellar system simulated using mechanics: **From inside**: Update its mass and position components by processing all planets. **From outside**: receive and aggregate the external forces from other stellar systems.

7.3.4 Security countermeasures

We review here the available security countermeasures for our architecture:
Proactive approaches aim to prevent attacks: Access rights policies are applied to the game status, for reading and writing. They are applied by the network component that may refuse to share certain data (entities or components) or restrict writing permissions [164]. Additionally, it can verify the result of a system execution with the system executor.
Network security – Techniques such as mixnets and onion routing [180] can help hide IP addresses to protect user privacy. Network resilience can be bolstered by employing P2P networks, which distribute data across multiple nodes, reducing the risk of failure and improving overall connectivity.
TEEs [65] provide a secure execution environment within a processor, ensuring that code and data loaded inside are protected with respect to confidentiality and integrity. They can be used in the system executor to guarantee the integrity

of system execution and game data privacy.
Reactive countermeasures aim to detect and limit the impact of on-going attacks:
Reputation systems [33] can be set in the network component where peers verify systems executions to guarantee the game status integrity.
Behavior monitoring based on AI systems can be set to detect cheating that is legitimate from the execution flow point of view (e.g. aim bots, modified controllers).
Bug exploitation can be limited by regular monitoring by the game developpers, patching the game to keep it equilibrate.

7.3.5 Conclusion

We proposed here a model for customizable online games. We showed how performances can be improved by segmenting the game trough interest management and distributed in a decentralised way. We identified the security properties and countermeasures. Remains the question of trade-offs between performances and security. For that, an implementation an benchmarking of D-ECS is needed.

General Conclusion

7.4 Main Results

In this thesis we proposed architectures to distribute and secure Cooperative Autonomous Systems and their means of cooperation. Figure 1.2 recaps the association between research questions and proposed models.

In Part I we formalised Cooperative Autonomous Systems, showing how we can define a reference architectures for CASes. We showed how autonomous and AI systems can be disaggregated for better performance. We identified the different types of cooperation for ASs. We proposed RACAS, a reference architectures for Cooperative Autonomous Systems that accounts for cooperation, performance and security in CASs. This allowed us to identify the means of cooperation that should be explored separately. We focused on the disaggregation of AI on the IoT-cloud continuum and its impact on performance with a need to explore security.

In Part II, we explored cooperation through collaborative learning with Federated Learning. We showed how we can improve its performance through decentralisation. Furthermore we identified the security properties to be guaranteed in collaborative learning, what are the available security countermeasures, and, how these countermeasures can be implemented and combined to address varying security and performance requirements. We aimed to improve FL performance in terms of resilience and scalability by decentralizing it on a peer-to-peer (P2P) network with the FDFL model and architecture. We also studied the security of FL by proposing the Ti-skol architecture to combine security countermeasures on demand. Preliminary results showed its ability to compose countermeasures, but with performance overheads.

In Part III, we addressed other means of cooperation following a similar approach as we did with collaborative learning in Part II. We showed how we can improve performance, in scalability and resilience, by decentralizing them on P2P networks. Morevover, we identified their security requirements and available security countermeasures. We first explored task delegation with delegating function execution though D-lambda, then we explored status sharing with the decentralised social network ReDOSN. Finally we explored the simulation of the CASs action environment and its decentralization and security with D-ECS.

Through all of these contributions we proposed to improve the models perfor-

mance and resilience through decentralisation. We identified similar security countermeasures and highlighted the need for their combination to answer each model security requirements. We also highlighted the trade-offs that can be found between security and performance.

7.5 Limits

Only federated learning works were implemented and benchmarked. Implementation and benchmarking of other cooperation functions are needed to verify their performances. These benchmarks should be based on real-life conditions to prove feasibility. The combination of security countermeasures in federated learning highlighted the need to establish good practice guidelines when deciding on the security properties to guarantee and security/performance trade-off.
Other identified means of cooperation, such as semantic search, have not been explored.

7.6 Future Works

This work opens the ways for future work that would focus on implementing and benchmarking the different means of cooperation. The proposed models should be configured and evaluated on realistic use-cases, be it on the scale, processed data and attacks. They should be assessed at the same time on functionality, performance (e.g. scalability, resilience) and security. This would allow to establish decision rules on disaggregation and countermeasure composition. The identified rules should also take into account the trade-offs between performance and security and determine their acceptable level (e.g. level of Differential Privacy). Additionally, efforts are needed to merge the different architectures into a single unique prototype.

Acronyms

AI Artificial Intelligence

AM Autonomic Manager

AS Autonomous System

BFT Byzantine Fault Tolerance

CAS Cooperative Autonomous System

CPU Central Processing Unit

D-ECS Decentralised Entity Component System

DHT Distributed Hash Table

DID Decentralised identifier

DL Deep Learning

DP Differential Privacy

DOSN Decentralized Online Social Network

ECS Entity Component System

ECU Electronic Control Unit

EE Execution Environment

FaaS Function as a Service

FDFL Fully Decentralised Federated Learning

FL Federated Learning

FLRA Federated Learning Reference Architecture

FMNIST Fashion MNIST

FPGA Field-Programmable Gate Array

GANA Generic Autonomic Networking Architecture

GPU Graphics Processing Unit

GUI Graphical User Interface

HD Horizontal Disaggregation

HE Homomorphic Encryption

HPC High-Performance Computing

IoT Internet of Things

IPFS InterPlanetary File System

IPNS InterPlanetary Naming System

LLM Large Language Model

MAPE-k Monitor, Analyze, Plan, Execute over a shared Knowledge

ML Machine Learning

OOP Object-Oriented Programming

OSN Online Social Network

P2P Peer-to-Peer

PoL Proof of Luck

PoS Proof of Stake

PoX Proof of Anything

RACAS Reference Architecture for Cooperative Autonomous Systems

ReDOSN Recursive Decentralised Online Social Network

S-CHOP Self- Configuring, Healing, Optimizing, Protecting

SMPC Secure Multi-Party Computation

TEE Trusted Execution Environment

TOR The Onion Router

TPM Trusted Platform Module

TPU Tensor Processing Unit

VC Volunteer Computing

VD Vertical Disaggregation

VDL Volunteer Deep Learning

VM Virtual Machine

V2I Vehicle-to-Infrastructure

V2V Vehicle-to-Vehicle

V2X Vehicle-to-Everything

Résumé de la thèse en français

Les Systèmes Coopératifs Autonomes (CAS), tels que les véhicules intelligents, sont utilisés dans de nombreux domaines des systèmes IoT industriels (IIoT) pour résoudre des tâches complexes, cela en partageant informations et exécution des tâches. Ils ont gagné en importance, comme l'illustre le cas d'usage des véhicules autonomes. Leurs exigences de performance et la présence de multiples agents conduisent à une intégration sur une diversité de matériels à travers le continuum IoT-Cloud. Dans ces cas, ils manipulent des données privées et exécutent des tâches sensibles dont la sécurité et la confidentialité doivent être garanties. Des contre-mesures de sécurité existent, mais aucune n'est universelle. Cela justifie leur combinaison. Cependant, ces contre-mesures peuvent aussi impacter les performances, imposant des compromis entre sécurité et performance.

Dans cette thèse, nous passons d'abord en revue la littérature pertinente sur les systèmes d'intelligence artificielle, les systèmes distribués et les contre-mesures de sécurité. Nous proposons une architecture de référence pour les CAS (RACAS), en identifiant les modes de coopération entre systèmes autonomes et en étudiant la distribution de l'intelligence artificielle, avec un focus sur l'IA désagrégée. Nous étudions ensuite séparément les performances et la sécurité des fonctions de coopération identifiées. Nous nous concentrons d'abord sur les performances de l'apprentissage collaboratif (FDFL) et sur la composition à la demande des contre-mesures de sécurité (Ti-skol). Nous proposons ensuite des modèles pour d'autres formes de coopération, notamment la délégation de tâches (D-Lambda) et la communication d'état (ReDOSN), en étudiant à la fois la sécurité et les performances. De plus, nous proposons un cadre pour la simulation décentralisée, sécurisée et efficace de l'environnement d'action des systèmes autonomes (D-ECS).

Nous montrons que les performances peuvent être améliorées en distribuant les systèmes autonomes et leur coopération à travers le continuum Cloud-IoT, en identifiant comment les CAS peuvent être décentralisés pour améliorer leur passage à l'échelle et leur résilience. Nous identifions également les propriétés de sécurité à protéger dans chaque architecture proposée et les contre-mesures de sécurité associées.

Nous présentons des résultats préliminaires sur la performance et la sécurité.

Ce travail justifie également la standardisation de la composition des contre-mesures de sécurité pour des cas d'usage courants.

7.7 Contexte

Les Systèmes Autonomes (SA) se caractérisent par une grande diversité de cas d'usage, de la conduite automobile au pilotage de drones en passant par la gestion de réseaux. Ils perçoivent leur environnement via divers capteurs (ex. : caméras, microphones) et agissent via des actionneurs (ex. : moteurs, ampoules) pour atteindre leur objectif (ex. : arriver à destination, garantir les performances d'un réseau).
Plusieurs SA peuvent être présents dans le même environnement (ex. : plusieurs voitures sur une même route), ce qui nécessite une coopération entre eux pour mieux atteindre leurs objectifs. Les Systèmes Coopératifs Autonomes (CAS) partagent des données collectées par leurs capteurs (ex. : des alertes de collision) ou se coordonnent pour résoudre ensemble des tâches (ex. : plusieurs robots sur un chantier de construction).
Les CAS collectent, analysent, stockent et partagent des quantités croissantes de données utilisées pour la prédiction et l'optimisation par des algorithmes d'IA et d'apprentissage automatique, nécessitant toujours plus de ressources informatiques. Pour fournir ces ressources, les SAC utilisent différents Environnements d'Exécution (EE) (ex. : VM, conteneurs, processus). Ces environnements peuvent être renforcés par un passage à l'échelle vertical (VM plus puissante) ou horizontale (multiplication des EE). Les EE peuvent être répartis sur le continuum Cloud-IoT (du même appareil au cloud distant en passant par la périphérie locale). Cela impacte les performances du système comme le débit, la latence et la résilience. La coopération amplifie ce problème, car ces considérations doivent être étendues à de nombreux pairs sur lesquels un SA a peu de contrôle.
Les SA manipulent des données confidentielles et exécutent des tâches sensibles ; cela pose des problèmes de confidentialité et de sécurité. La distribution des EE et la coopération avec des SA non fiables augmentent la surface d'attaque, car les systèmes sensibles sont connectés entre eux.
Des compromis sont parfois nécessaires entre l'intégrité et la confidentialité des informations. Par exemple, des cartes de navigation mises à jour coopérativement par des voitures connectées exigent de partager des données personnelles pour vérifier leur validité, au prix d'une potentielle perte de confidentialité.
Des contre-mesures existent pour garantir sécurité et confidentialité. Par exemple, la confidentialité différentielle [37] garantit l'anonymisation des données, et l'agrégation sécurisée [152] permet de sommer des données sans révéler les valeurs individuelles. Cependant, aucune contre-mesure ne garantit toutes les propriétés. Elles doivent être combinées, mais elles ne sont pas toujours compatibles. De plus, elles ont souvent un coût en performance, d'où des compromis nécessaires.
Les SA doivent répondre à trois types d'exigences, chacune associée à plusieurs métriques :

- **Fonctionnalité/Exactitude** : Le SA accomplit ce pour quoi il est conçu. Par exemple, une voiture se déplace sans accident. Les métriques évaluent sa capacité à atteindre son objectif, incluant la précision du modèle d'IA, la qualité de la tâche accomplie, et l'utilisabilité sur différents cas d'usage.

- **Sécurité** : Le SA garantit l'intégrité, la confidentialité et la disponibilité de ses données et de ses traitements, se protégeant lui-même et les autres contre les comportements malveillants. Exemple : une voiture protège la confidentialité de sa position. Les métriques incluent la précision des mécanismes de protection et les propriétés garanties telles que les standards de chiffrement, les protocoles d'authentification, ou la résilience face aux attaques.

- **Performance** : Le SA optimise l'utilisation de ses ressources (ex. : puissance de calcul) tout en fournissant les ressources nécessaires pour accomplir sa fonction. Les métriques incluent l'utilisation des ressources (bande passante, mémoire), la latence de réponse et l'efficacité énergétique.

Ces exigences justifient la combinaison de plusieurs domaines scientifiques :

- Intelligence Artificielle et systèmes autonomes pour la fonctionnalité.

- Systèmes distribués et décentralisés pour la performance.

- Contre-mesures de sécurité et technologies de protection de la vie privée pour la sécurité.

7.8 État de l'art

7.8.1 Exigences fonctionnelles des systèmes autonomes

Les systèmes autonomes reposent sur diverses fonctionnalités, notamment la navigation, la prise de décision et l'exécution de tâches. Ils suivent une boucle de contrôle : ils observent leur environnement, puis agissent en conséquence. Cette boucle de contrôle a été formalisée via le standard MAPE-k [14], appliqué sous différentes formes dans les systèmes autonomes modernes.
En plus des exigences fonctionnelles, les systèmes autonomes doivent répondre à des exigences non-fonctionnelles, souvent désignées par l'acronyme S-CHOP (Self-Configuration, Self-Healing, Self-Optimization, Self-Protection).
Les systèmes autonomes peuvent être étendus en Systèmes Coopératifs Autonomes (CAS). Parmi les méthodes de coopération notables figurent l'apprentissage collaboratif, où plusieurs CAS entraînent ensemble un modèle d'apprentissage automatique à partir de leurs données locales [105], ou encore le partage de statut, comme leur position (ex. : évitement coopératif anticipé des collisions [76]). Cependant, il n'existe actuellement aucun cadre ou ensemble de directives universellement reconnu qui définisse clairement les différentes méthodes de coopération entre CAS.

Les actions des SA peuvent être situées dans des environnements réels ou simulés, tels que les simulateurs de conduite [42]. Il existe des simulations standards pour un SA unique [168], mais aucun standard pour simuler plusieurs SA dans un même environnement.
Pour garantir la fonctionnalité, il est nécessaire de formaliser les différents moyens de coopération et de structurer les architectures les utilisant.

7.8.2 Outils et technologies

Pour répondre aux besoins de fonctionnalité, de performance et de sécurité, nous identifions ici les outils et techniques pertinents : intelligence artificielle, systèmes distribués, et contre-mesures de sécurité.

Intelligence artificielle

La fonctionnalité/exactitude des systèmes autonomes est assurée par des approches d'Intelligence Artificielle (IA). On distingue deux grandes familles de techniques :
Apprentissage machine : une machine effectue une analyse statistique des données pour établir un modèle utilisé pour la prédiction ou la prise de décision. Cela inclut l'apprentissage profond (Deep Learning), où un réseau de neurones est entraîné, ainsi que l'apprentissage fédéré et décentralisé, étudiés dans cette thèse. L'apprentissage fédéré permet à plusieurs SA de former ensemble un modèle tout en gardant leurs données locales, réduisant ainsi les transferts de données et améliorant la confidentialité. L'apprentissage décentralisé étend ce concept en éliminant le besoin d'un coordinateur central, distribuant le processus d'apprentissage sur plusieurs nœuds pour u meilleur passage à l'échlle et une meilleure résilience.
Systèmes experts : l'algorithme est entièrement défini par des experts. Cela inclut les automates à états finis et les systèmes à base de règles.

Systèmes distribués et décentralisés

Le besoin de distribution est crucial pour atteindre de meilleures performances et supporter la multi-localité des algorithmes d'IA. Pour être utilisables dans des cas d'usage réels, les algorithmes d'IA doivent s'exécuter sur une infrastructure répondant à des contraintes de sécurité et de performance.
Pour améliorer les performances, deux approches existent : le *passage à l'échelle vertical* (remplacement par un environnement plus performant) et le *passage à l'échelle horizontal* (augmentation du nombre d'environnements pour répartir la charge) [185]. Les deux sont souvent combinés. Par exemple, les modèles de Deep Learning sont passés des CPU aux GPU pour de meilleures performances, puis à des systèmes multi-GPU.
L'infrastructure peut être répartie selon plusieurs distances physiques (Figure 1.1) : dans la même machine (1m), à proximité (point de présence dans le quartier, $1 - 1000$m), ou dans un centre de données cloud (> 100 km). Cela

implique un compromis entre des infrastructures proches (communication plus rapide) et des infrastructures éloignées (puissance de calcul supérieure).
La gestion des systèmes distribués et de leurs communications repose souvent sur un coordinateur unique. Cela pose des défis de résilience et de passage à l'échelle, le coordinateur pouvant tomber en panne ou devenir un goulet d'étranglement. Des protocoles décentralisés ont été proposés pour répondre à ces enjeux [34], comme les blockchains pour les échanges monétaires [61] ou les réseaux pair-à-pair (P2P) pour le partage de fichiers [123].

Contre-mesures de sécurité

Les contre-mesures de sécurité peuvent prendre différentes formes (algorithmes cryptographiques, protections matérielles, etc.). Elles assurent l'intégrité, la confidentialité ou la disponibilité des traitements et des données. Chaque contre-mesure apporte des garanties spécifiques mais avec un coût (par exemple surcharge de performance ou dégradation de la confidentialité).
Les Environnements d'Exécution de Confiance (Trusted Execution Environments, TEE) sont des protections matérielles intégrées au processeur (ex. : AMD-SEV, Intel TDX, ARM Trustzone) [77]. Ils garantissent l'intégrité et la confidentialité des données et calculs via une mémoire chiffrée. Ils offrent également des mécanismes d'attestation à distance permettant de vérifier qu'une tâche s'exécute bien dans un TEE. L'exécution peut être jusqu'à 5% plus lente qu'en environnement classique [113].
D'autres contre-mesures incluent le chiffrement sécurisé comme l'agrégation sécurisée [152], le chiffrement homomorphe [53] qui permet de calculer sur des données chiffrées, les mécanismes d'anonymisation comme la confidentialité différentielle [37], ou les algorithmes de consensus tolérants aux fautes byzantines [11].

7.8.3 Combinaison des solutions

Des techniques ont été identifiées pour répondre aux exigences de fonctionnalité, de performance et de sécurité. Cela justifie leur combinaison. Des travaux combinent deux de ces axes : l'apprentissage fédéré et distribué [106] associe IA et systèmes distribués ; Ekiden [56] combine TEE et blockchain. Les combinaisons des trois axes sont plus rares mais existent (ex. : apprentissage fédéré sécurisé [110]).
Ces combinaisons peuvent être spécifiques au cas d'usage. On peut souhaiter des propriétés légèrement différentes sur chaque axe (ex. : compromis différent entre latence et débit). Cela justifie une approche à la demande comme en programmation orientée aspect [6], où le développeur choisit les propriétés désirées et un tisseur adapte le programme.
Les architectures actuelles sont souvent difficiles à adapter à des contraintes spécifiques de performance ou de sécurité. Par exemple, FLAME [130] protège l'intégrité de l'apprentissage fédéré, mais ne garantit pas la confidentialité du modèle et n'est pas compatible avec l'agrégation sécurisée [152].

7.9 Défis

Dans l'état de l'art, nous avons identifié un manque de formalisation des moyens de coopération et des architectures combinant fonctionnalité, performance et sécurité.
Les questions de recherche de cette thèse sont les suivantes :

1. Comment définir une architecture de référence pour les systèmes autonomes qui prenne en compte à la fois la coopération et la performance ? Quels sont les différents moyens de coopération à identifier et intégrer dans cette architecture ?

2. Comment proposer des versions plus performantes des fonctionnalités de coopération via la distribution et la décentralisation ? Quelles techniques et méthodologies spécifiques peuvent être employées pour améliorer la performance de ces fonctions coopératives ?

3. Dans les architectures proposées, quelles sont les propriétés de sécurité essentielles à garantir ? Quelles contre-mesures peuvent être mises en œuvre pour garantir ces propriétés, et comment les combiner à la demande en fonction des besoins en sécurité et en performance ?

7.10 Contributions

La structure de cette thèse suit une approche fonctionnelle : à chaque étape, nous étudions un besoin fonctionnel spécifique. La Figure 1.2 associe les questions de recherche aux contributions. La sécurité et les performances sont transversales : à chaque étape, nous identifions comment l'architecture peut être distribuée, quelles sont les propriétés de sécurité à garantir, et les contre-mesures associées.

- Nous formalisont les Systèmes Coopératifs Autonomes avec le modèle RACAS. Nous montrons comment les systèmes autonomes et les systèmes d'IA peuvent être désagrégés pour de meilleures performances. Nous identifions les différents types de coopération entre SA.

- Nous nous concentrons sur l'apprentissage coopératif des systèmes d'IA : d'abord la décentralisation pour de meilleures performances (FDFL), puis la composition des contre-mesures de sécurité (Ti-skol).

- Nous abordons d'autres moyens de coopération et leurs performances : la délégation de tâches avec un modèle de Function-as-a-Service (D-lambda), le partage de statut via les réseaux sociaux (ReDOSN), et la simulation de l'environnement d'action avec les jeux vidéo (D-ECS). Chaque travail est indépendant, nous montrons comment ils peuvent être décentralisés et sécurisés.

Nous avons montré comment améliorer la performance (scalabilité et résilience) via leur décentralisation sur des réseaux pair-à-pair (P2P). Nous avons également identifié leurs exigences en matière de sécurité et les contre-mesures disponibles.

Enfin, nous avons exploré la simulation de l'environnement d'action des SAC et sa décentralisation avec D-ECS.

7.11 Résultats principaux

Dans cette thèse, nous avons proposé des architectures pour distribuer et sécuriser les Systèmes Autonomes Coopératifs et leurs moyens de coopération. La Figure 1.2 récapitule l'association entre questions de recherche et modèles proposés.
Dans la Partie I, nous avons formalisé les Systèmes Coopératifs Autonomes, en montrant comment définir une architecture de référence pour ces systèmes. Nous avons montré comment les systèmes autonomes et d'IA peuvent être désagrégés pour de meilleures performances. Nous avons proposé RACAS, une architecture de référence pour les CAS, prenant en compte coopération, performance et sécurité. Cela nous a permis d'identifier les types de coopération à étudier séparément, en nous focalisant sur la désagrégation de l'IA sur le continuum IoT-cloud et son impact sur la performance, avec un besoin d'exploration des aspects de sécurité.
Dans la Partie II, nous avons étudié la coopération via l'apprentissage collaboratif avec l'apprentissage fédéré. Nous avons montré comment améliorer ses performances par la décentralisation. Nous avons identifié les propriétés de sécurité à garantir dans l'apprentissage collaboratif, les contre-mesures disponibles, et comment les mettre en œuvre et les combiner pour répondre aux besoins variés de sécurité et de performance.
Nous avons proposé le modèle FDFL et son architecture pour améliorer la résilience et la scalabilité de l'apprentissage fédéré via un réseau P2P. Nous avons aussi étudié la sécurité via l'architecture Ti-skol permettant la composition à la demande de contre-mesures de sécurité. Les résultats préliminaires ont montré sa capacité à composer ces contre-mesures, au prix d'une surcharge en performance.
À travers toutes ces contributions, nous avons proposé d'améliorer les performances et la résilience des modèles via la décentralisation. Nous avons identifié les contre-mesures de sécurité récurrentes et souligné la nécessité de leur combinaison pour répondre aux exigences spécifiques de chaque modèle. Nous avons également mis en évidence les compromis à faire entre performance et sécurité.

7.12 Limites

Seuls les travaux portant sur l'apprentissage fédéré ont été implémentés et évalués expérimentalement. La mise en œuvre et l'évaluation expérimentale des autres fonctions de coopération sont nécessaires pour vérifier leurs performances. Ces évaluations doivent se baser sur des conditions réalistes pour prouver leur

faisabilité. La combinaison de contre-mesures de sécurité dans l'apprentissage fédéré a mis en évidence la nécessité d'établir des bonnes pratiques pour choisir les propriétés de sécurité à garantir et les compromis sécurité/performance.
D'autres moyens de coopération identifiés, comme la recherche sémantique, n'ont pas été explorés.

7.13 Perspectives

Ce travail ouvre la voie à des perspectives de recherche axées sur la mise en œuvre et l'évaluation des différents moyens de coopération. Ces modèles doivent être évalués dans des cas d'usage réalistes, selon les deux axes de performance (e.g. scalabilité, résilience) et de sécurité. Cela inclut l'établissement de règles de décision sur la désagrégation et la composition des contre-mesures. Les règles doivent prendre en compte les compromis entre performance et sécurité, et déterminer des seuils acceptables (ex. : niveau de confidentialité différentielle). Par ailleurs, des efforts seront nécessaires pour fusionner les différentes architectures proposées en un prototype unique.

Bibliography

[1] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: https://dl.acm.org/doi/10.1145/359340.359342 (visited on 01/08/2025).

[2] M. Naor and M. Yung. "Universal one-way hash functions and their cryptographic applications". en. In: *Proceedings of the twenty-first annual ACM symposium on Theory of computing - STOC '89*. Seattle, Washington, United States: ACM Press, 1989, pp. 33–43. ISBN: 978-0-89791-307-2. DOI: 10.1145/73007.73011. URL: http://portal.acm.org/citation.cfm?doid=73007.73011 (visited on 01/08/2025).

[3] J. Turek and D. Shasha. "The many faces of consensus in distributed systems". In: *Computer* 25.6 (1992), pp. 8–17. DOI: 10.1109/2.153253.

[4] Edward A Luke. "Defining and measuring scalability". In: *Scalable Parallel Libraries Conference (SPLC)*. IEEE. 1993.

[5] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. "Incremental Cryptography: The Case of Hashing and Signing". In: *Annual International Cryptology Conference (CRYPTO)*. 1994.

[6] G. Kiczales. "Aspect-oriented programming". en. In: *ACM Computing Surveys* 28.4es (Dec. 1996), p. 154. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/242224.242420. URL: https://dl.acm.org/doi/10.1145/242224.242420 (visited on 10/18/2024).

[7] Vincent Nicomette. "La protection dans les systèmes à objets répartis". fr. PhD thesis. Institut National Polytechnique de Toulouse - INPT, Dec. 1996. URL: https://theses.hal.science/tel-00175252 (visited on 02/06/2023).

[8] Stan Franklin and Art Graesser. "Is It an agent, or just a program?: A taxonomy for autonomous agents". en. In: *Intelligent Agents III Agent Theories, Architectures, and Languages*. Ed. by Jörg P. Müller, Michael J. Wooldridge, and Nicholas R. Jennings. Berlin, Heidelberg: Springer, 1997, pp. 21–35. ISBN: 978-3-540-68057-4. DOI: 10.1007/BFb0013570.

[9] D. Seto et al. "The Simplex architecture for safe online control system upgrades". en. In: *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*. Philadelphia, PA, USA: IEEE, 1998, 3504–3508 vol.6. ISBN: 978-0-7803-4530-0. DOI: 10.1109/ACC.1998.703255. URL: http://ieeexplore.ieee.org/document/703255/ (visited on 03/28/2024).

[10] James S. Albus. "4D/RCS: a reference model architecture for intelligent unmanned ground vehicles". en. In: ed. by Grant R. Gerhart, Chuck M. Shoemaker, and Douglas W. Gage. Orlando, FL, July 2002, pp. 303–310. DOI: 10.1117/12.474462. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=885594 (visited on 03/22/2024).

[11] Miguel Castro and Barbara Liskov. "Practical byzantine fault tolerance and proactive recovery". en. In: *ACM Transactions on Computer Systems* 20.4 (Nov. 2002), pp. 398–461. ISSN: 0734-2071, 1557-7333. DOI: 10.1145/571637.571640. URL: https://dl.acm.org/doi/10.1145/571637.571640 (visited on 07/22/2022).

[12] John R. Douceur. "The Sybil Attack". In: *Peer-to-Peer Systems*. Ed. by Gerhard Goos et al. Vol. 2429. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260. ISBN: 978-3-540-44179-3 978-3-540-45748-0. DOI: 10.1007/3-540-45748-8_24. URL: http://link.springer.com/10.1007/3-540-45748-8_24 (visited on 08/22/2022).

[13] Petar Maymounkov and David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". en. In: *Peer-to-Peer Systems*. Ed. by Gerhard Goos et al. Vol. 2429. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65. ISBN: 978-3-540-44179-3 978-3-540-45748-0. DOI: 10.1007/3-540-45748-8_5. URL: http://link.springer.com/10.1007/3-540-45748-8_5 (visited on 08/24/2022).

[14] J.O. Kephart and D.M. Chess. "The vision of autonomic computing". en. In: *Computer* 36.1 (Jan. 2003), pp. 41–50. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1160055. URL: http://ieeexplore.ieee.org/document/1160055/ (visited on 11/23/2022).

[15] Paul Syverson, Roger Dingledine, and Nick Mathewson. "Tor: The secondgeneration onion router". In: *Usenix Security*. USENIX Association Berkeley, CA. 2004, pp. 303–320.

[16] Liviu Panait and Sean Luke. "Cooperative Multi-Agent Learning: The State of the Art". en. In: *Autonomous Agents and Multi-Agent Systems* 11.3 (Nov. 2005), pp. 387–434. ISSN: 1573-7454. DOI: 10.1007/s10458-005-2631-2. URL: https://doi.org/10.1007/s10458-005-2631-2 (visited on 06/10/2024).

[17] K. Sampigethaya and R. Poovendran. "A Survey on Mix Networks and Their Secure Applications". en. In: *Proceedings of the IEEE* 94.12 (Dec. 2006), pp. 2142–2181. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2006.889687. URL: http://ieeexplore.ieee.org/document/4077263/ (visited on 10/10/2022).

[18] Ken Birman. "The promise, and limitations, of gossip protocols". en. In: *ACM SIGOPS Operating Systems Review* 41.5 (Oct. 2007), pp. 8–13. ISSN: 0163-5980. DOI: 10.1145/1317379.1317382. URL: https://dl.acm.org/doi/10.1145/1317379.1317382 (visited on 02/07/2022).

[19] Markus C. Huebscher and Julie A. McCann. "A survey of autonomic computing—degrees, models, and applications". en. In: *ACM Computing Surveys* 40.3 (Aug. 2008), pp. 1–28. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/1380584.1380585. URL: https://dl.acm.org/doi/10.1145/1380584.1380585 (visited on 11/24/2022).

[20] Douglas J. Kelly et al. "A survey of state-of-the-art in anonymity metrics". en. In: *Proceedings of the 1st ACM workshop on Network data anonymization*. Alexandria Virginia USA: ACM, Oct. 2008, pp. 31–40. ISBN: 978-1-60558-301-3. DOI: 10.1145/1456441.1456453. URL: https://dl.acm.org/doi/10.1145/1456441.1456453 (visited on 01/09/2025).

[21] Adam L. Beberg et al. "Folding@home: Lessons from eight years of volunteer distributed computing". en. In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. Rome, Italy: IEEE, May 2009, pp. 1–8. ISBN: 978-1-4244-3751-1. DOI: 10.1109/IPDPS.2009.5160922. URL: http://ieeexplore.ieee.org/document/5160922/ (visited on 08/19/2022).

[22] M. Hirth et al. "Anatomy of a Crowdsourcing Platform - Using the Example of Microworkers.com". In: *IMIS*. 2011.

[23] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. "A survey of DHT security techniques". In: *ACM Computing Surveys* 43.2 (Feb. 2011), 8:1–8:49. ISSN: 0360-0300. DOI: 10.1145/1883612.1883615. URL: https://doi.org/10.1145/1883612.1883615 (visited on 02/03/2023).

[24] Benjamin Greschbach, Gunnar Kreitz, and Sonja Buchegger. "The devil is in the metadata — New privacy challenges in Decentralised Online Social Networks". In: *2012 IEEE International Conference on Pervasive Computing and Communications Workshops*. Mar. 2012, pp. 333–339. DOI: 10.1109/PerComW.2012.6197506.

[25] Róbert Ormándi, István Hegedűs, and Márk Jelasity. "Gossip learning with linear models on fully distributed data". In: *Concurrency and Computation: Practice and Experience* 25.4 (2013), pp. 556–571.

[26] Fawaz Paraiso, Philippe Merle, and Lionel Seinturier. "Managing elasticity across multiple cloud providers". en. In: *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*. Prague Czech Republic: ACM, Apr. 2013, pp. 53–60. ISBN: 978-1-4503-2050-4. DOI: 10.1145/2462326.2462338. URL: https://dl.acm.org/doi/10.1145/2462326.2462338 (visited on 10/14/2024).

[27] Amir Yahyavi and Bettina Kemme. "Peer-to-peer architectures for massively multiplayer online games: A Survey". en. In: *ACM Computing Surveys* 46.1 (Oct. 2013), pp. 1–51. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/2522968.2522977. URL: https://dl.acm.org/doi/10.1145/2522968.2522977 (visited on 09/18/2023).

[28] A. Agarwal et al. "A Reliable Effective Terascale Linear Learning System". In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1111–1133.

[29] Elvis S. Liu and Georgios K. Theodoropoulos. "Interest management for distributed virtual environments: A survey". en. In: *ACM Computing Surveys* 46.4 (Apr. 2014), pp. 1–42. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/2535417. URL: https://dl.acm.org/doi/10.1145/2535417 (visited on 07/05/2024).

[30] Thomas Paul, Antonino Famulari, and Thorsten Strufe. "A survey on decentralized Online Social Networks". en. In: *Computer Networks* 75 (Dec. 2014), pp. 437–452. ISSN: 13891286. DOI: 10.1016/j.comnet.2014.10.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S1389128614003600 (visited on 02/06/2023).

[31] Evjola Spaho, Leonard Barolli, and Fatos Xhafa. "Data Replication Strategies in P2P Systems: A Survey". In: *2014 17th International Conference on Network-Based Information Systems*. ISSN: 2157-0426. Sept. 2014, pp. 302–309. DOI: 10.1109/NBiS.2014.74.

[32] E. P. Xing et al. "Petuum: A New Platform for Distributed Machine Learning on Big Data". In: *IEEE Trans. on Big Data* 1.2 (2015), pp. 49–67.

[33] Ferry Hendrikx, Kris Bubendorfer, and Ryan Chard. "Reputation systems: A survey and taxonomy". en. In: *Journal of Parallel and Distributed Computing* 75 (Jan. 2015), pp. 184–197. ISSN: 07437315. DOI: 10.1016/j.jpdc.2014.08.004. URL: https://linkinghub.elsevier.com/retrieve/pii/S0743731514001464 (visited on 09/13/2022).

[34] Anne-Marie Kermarrec and François Taïani. "Want to scale in centralized systems? Think P2P". In: *Journal of Internet Services and Applications* 6.1 (Aug. 2015), p. 16. ISSN: 1869-0238. DOI: 10.1186/s13174-015-0029-1. URL: https://doi.org/10.1186/s13174-015-0029-1 (visited on 09/15/2023).

[35] Lotfi Ben Othmane et al. "A Survey of Security and Privacy in Connected Vehicles". In: *Wireless Sensor and Mobile Ad-Hoc Networks: Vehicular and Space Applications*. Ed. by Driss Benhaddou and Ala Al-Fuqaha. New York, NY: Springer New York, 2015, pp. 217–247. ISBN: 978-1-4939-2468-4. DOI: 10.1007/978-1-4939-2468-4_10. URL: https://doi.org/10.1007/978-1-4939-2468-4_10.

[36] Dennis Wiebusch and Marc Erich Latoschik. "Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems". en. In: *2015 IEEE 8th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. Arles, France: IEEE, Mar. 2015, pp. 25–32. ISBN: 978-1-4673-6881-0. DOI: 10.1109/SEARIS.2015.7854098. URL: http://ieeexplore.ieee.org/document/7854098/ (visited on 03/14/2024).

[37] Martín Abadi et al. "Deep Learning with Differential Privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Oct. 2016). arXiv: 1607.00133, pp. 308–318. DOI: 10.1145/2976749.2978318. URL: http://arxiv.org/abs/1607.00133 (visited on 01/20/2022).

[38] Mitar Milutinovic et al. "Proof of Luck: an Efficient Blockchain Consensus Protocol". In: *Proceedings of the 1st Workshop on System Software for Trusted Execution*. arXiv:1703.05435 [cs]. Dec. 2016, pp. 1–6. DOI: 10.1145/3007788.3007790. URL: http://arxiv.org/abs/1703.05435 (visited on 05/18/2022).

[39] R. Bahmani et al. "Secure Multiparty Computation from SGX". In: *Financial Cryptography and Data Security*. 2017.

[40] Keith Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". en. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas Texas USA: ACM, Oct. 2017, pp. 1175–1191. ISBN: 978-1-4503-4946-8. DOI: 10.1145/3133956.3133982. URL: https://dl.acm.org/doi/10.1145/3133956.3133982 (visited on 02/03/2022).

[41] Information Technology Laboratory Computer Security Division. *Post-Quantum Cryptography — CSRC — CSRC*. EN-US. Jan. 2017. URL: https://csrc.nist.gov/projects/post-quantum-cryptography (visited on 01/08/2025).

[42] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". en. In: *Proceedings of the 1st Annual Conference on Robot Learning*. ISSN: 2640-3498. PMLR, Oct. 2017, pp. 1–16. URL: https://proceedings.mlr.press/v78/dosovitskiy17a.html (visited on 10/18/2024).

[43] Xinjian Long et al. "Autonomic Networking: Architecture Design and Standardization". In: *IEEE Internet Computing* 21.5 (2017). Conference Name: IEEE Internet Computing, pp. 48–53. ISSN: 1941-0131. DOI: 10.1109/MIC.2017.3481338. URL: https://ieeexplore.ieee.org/

document/8039304/citations?tabFilter=papers#citations (visited on 04/02/2024).

[44] H. Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *arXiv:1602.05629 [cs]* (Feb. 2017). arXiv: 1602.05629. URL: http://arxiv.org/abs/1602.05629 (visited on 02/03/2022).

[45] Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *20th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2017, pp. 1273–1282.

[46] R. Shokri et al. "Membership inference attacks against machine learning models". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 3–18.

[47] Gabriel Silva et al. "Implementing Federated Social Networking: Report from the Trenches". en. In: *Proceedings of the 13th International Symposium on Open Collaboration*. Galway Ireland: ACM, Aug. 2017, pp. 1–10. ISBN: 978-1-4503-5187-4. DOI: 10.1145/3125433.3125455. URL: https://dl.acm.org/doi/10.1145/3125433.3125455 (visited on 07/04/2024).

[48] Youfu Chen and Elvis S. Liu. "Comparing Dead Reckoning Algorithms for Distributed Car Simulations". en. In: *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. Rome Italy: ACM, May 2018, pp. 105–111. ISBN: 978-1-4503-5092-1. DOI: 10.1145/3200921.3200939. URL: https://dl.acm.org/doi/10.1145/3200921.3200939 (visited on 07/08/2024).

[49] Ekasit Kijsipongse, Apivadee Piyatumrong, and Suriya U-ruekolan. "A hybrid GPU cluster and volunteer computing platform for scalable deep learning". en. In: *The Journal of Supercomputing* 74.7 (July 2018), pp. 3236–3263. ISSN: 1573-0484. DOI: 10.1007/s11227-018-2375-9. URL: https://doi.org/10.1007/s11227-018-2375-9 (visited on 09/01/2022).

[50] Mario M. Kubek and Herwig Unger. "The WebEngine: A Fully Integrated, Decentralised Web Search Engine". en. In: *Proceedings of the 2nd International Conference on Natural Language Processing and Information Retrieval*. Bangkok Thailand: ACM, Sept. 2018, pp. 26–31. ISBN: 978-1-4503-6551-2. DOI: 10.1145/3278293.3278294. URL: https://dl.acm.org/doi/10.1145/3278293.3278294 (visited on 01/29/2024).

[51] Pieter Maene et al. "Hardware-Based Trusted Computing Architectures for Isolation and Attestation". In: *IEEE Transactions on Computers* 67.3 (Mar. 2018). Conference Name: IEEE Transactions on Computers, pp. 361–374. ISSN: 1557-9956. DOI: 10.1109/TC.2017.2647955.

[52] Fahad Saleh. "Blockchain Without Waste: Proof-of-Stake". en. In: *SSRN Electronic Journal* (2018). ISSN: 1556-5068. DOI: 10.2139/ssrn.3183935. URL: https://www.ssrn.com/abstract=3183935 (visited on 09/26/2022).

[53] Abbas Acar et al. "A Survey on Homomorphic Encryption Schemes: Theory and Implementation". en. In: *ACM Computing Surveys* 51.4 (July 2019), pp. 1–35. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3214303. URL: https://dl.acm.org/doi/10.1145/3214303 (visited on 10/17/2024).

[54] Maxime Bélair, Sylvie Laniepce, and Jean-Marc Menaud. "Leveraging Kernel Security Mechanisms to Improve Container Security: a Survey". en. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*. Canterbury CA United Kingdom: ACM, Aug. 2019, pp. 1–6. ISBN: 978-1-4503-7164-3. DOI: 10.1145/3339252.3340502. URL: https://dl.acm.org/doi/10.1145/3339252.3340502 (visited on 01/09/2025).

[55] Keith Bonawitz et al. "Towards Federated Learning at Scale: System Design". en. In: *arXiv:1902.01046 [cs, stat]* (Mar. 2019). arXiv: 1902.01046. URL: http://arxiv.org/abs/1902.01046 (visited on 01/18/2022).

[56] Raymond Cheng et al. "Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts". In: *2019 IEEE European Symposium on Security and Privacy (EuroS P)*. June 2019, pp. 185–200. DOI: 10.1109/EuroSP.2019.00023.

[57] István Hegedűs, Gábor Danner, and Márk Jelasity. "Gossip Learning as a Decentralized Alternative to Federated Learning". en. In: vol. LNCS-11534. Springer International Publishing, June 2019, p. 74. DOI: 10.1007/978-3-030-22496-7_5. URL: https://hal.inria.fr/hal-02319574 (visited on 01/21/2022).

[58] Le Jiang and Xinglin Zhang. "BCOSN: A Blockchain-Based Decentralized Online Social Network". en. In: *IEEE Transactions on Computational Social Systems* 6.6 (Dec. 2019), pp. 1454–1466. ISSN: 2329-924X, 2373-7476. DOI: 10.1109/TCSS.2019.2941650. URL: https://ieeexplore.ieee.org/document/8855083/ (visited on 06/28/2024).

[59] Luca Melis et al. "Exploiting Unintended Feature Leakage in Collaborative Learning". In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 691–706.

[60] Tessema M. Mengistu and Dunren Che. "Survey and Taxonomy of Volunteer Computing". In: *ACM Computing Surveys* 52.3 (July 2019), 59:1–59:35. ISSN: 0360-0300. DOI: 10.1145/3320073. URL: https://doi.org/10.1145/3320073 (visited on 09/09/2022).

[61] Ahmed Afif Monrat, Olov Schelén, and Karl Andersson. "A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities". In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 117134–117151. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2936094.

[62] Aravindh Raman et al. "Challenges in the Decentralised Web: The Mastodon Case". en. In: *Proceedings of the Internet Measurement Conference*. Amsterdam Netherlands: ACM, Oct. 2019, pp. 217–229. ISBN: 978-1-4503-6948-0. DOI: 10.1145/3355369.3355572. URL: https://dl.acm.org/doi/10.1145/3355369.3355572 (visited on 07/05/2024).

[63] Abhijit Guha Roy et al. "BrainTorrent: A Peer-to-Peer Environment for Decentralized Federated Learning". en. In: *arXiv:1905.06731 [cs, stat]* (May 2019). arXiv: 1905.06731. URL: http://arxiv.org/abs/1905.06731 (visited on 02/14/2022).

[64] Zeyi Tao et al. "A Survey of Virtual Machine Management in Edge Computing". en. In: *Proceedings of the IEEE* 107.8 (Aug. 2019), pp. 1482–1499. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2019.2927919. URL: https://ieeexplore.ieee.org/document/8772112/ (visited on 01/09/2025).

[65] Bohdan Trach et al. "Clemmys: towards secure remote execution in FaaS". en. In: *Proceedings of the 12th ACM International Conference on Systems and Storage*. Haifa Israel: ACM, May 2019, pp. 44–54. ISBN: 978-1-4503-6749-3. DOI: 10.1145/3319647.3325835. URL: https://dl.acm.org/doi/10.1145/3319647.3325835 (visited on 04/30/2024).

[66] Chuan Zhao et al. "Secure Multi-Party Computation: Theory, practice and applications". In: *Information Sciences* 476 (Feb. 2019), pp. 357–372. ISSN: 0020-0255. DOI: 10.1016/j.ins.2018.10.024. URL: https://www.sciencedirect.com/science/article/pii/S0020025518308338 (visited on 10/08/2024).

[67] Lig Zhu, Zhijian Liu, and Song Han. "Deep leakage from gradients". In: *NeurIPS*. 2019.

[68] Akul Agrawal, Divya D Kulkarni, and Shivashankar B. Nair. "On Decentralizing Federated Learning". In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. ISSN: 2577-1655. Oct. 2020, pp. 1590–1595. DOI: 10.1109/SMC42975.2020.9282830.

[69] L. Lyu et al. "Privacy and Robustness in Federated Learning: Attacks and Defenses". In: *arXiv:2012.06337* (2020).

[70] D.P. Anderson. "BOINC: A Platform for Volunteer Computing". In: *J. Grid Computing* 18 (2020), pp. 99–122.

[71] David P. Anderson. "BOINC: A Platform for Volunteer Computing". en. In: *Journal of Grid Computing* 18.1 (Mar. 2020), pp. 99–122. ISSN: 1570-7873, 1572-9184. DOI: 10.1007/s10723-019-09497-9. URL: http://link.springer.com/10.1007/s10723-019-09497-9 (visited on 01/20/2023).

[72] Daniel J Beutel et al. "Flower: A Friendly Federated Learning Research Framework". In: *arXiv:2007.14390* (2020).

[73] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[74] Minghong Fang et al. "Local model poisoning attacks to Byzantine-Robust federated learning". In: *29th USENIX Security Symposium*. 2020, pp. 1605–1622.

[75] Chaoyang He et al. "FedML: A research library and benchmark for federated machine learning". In: *arXiv:2007.13518* (2020).

[76] Dirk Hetzer et al. *5GCroCo Use Cases and Key Performance Indicators for Cross-border Trials*. Sept. 2020. DOI: 10.5281/zenodo.4036550. URL: https://zenodo.org/record/4036550 (visited on 02/08/2022).

[77] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. "Trusted Execution Environments: Properties, Applications, and Challenges". In: *IEEE Security Privacy* 18.2 (Mar. 2020). Conference Name: IEEE Security Privacy, pp. 56–60. ISSN: 1558-4046. DOI: 10.1109/MSEC.2019.2947124.

[78] Learning@home team. *Hivemind: a Library for Decentralized Deep Learning*. Accessed: 2023-09-22. 2020. URL: https://github.com/learning-at-home/hivemind.

[79] Mozhgan Malekshahi Rad et al. "Social Internet of Things: vision, challenges, and trends". In: *Human-centric Computing and Information Sciences* 10.1 (Dec. 2020), p. 52. ISSN: 2192-1962. DOI: 10.1186/s13673-020-00254-6. URL: https://doi.org/10.1186/s13673-020-00254-6 (visited on 06/19/2024).

[80] Moreno Marzolla and Gabriele D'Angelo. "Parallel Data Distribution Management on Shared-memory Multiprocessors". en. In: *ACM Transactions on Modeling and Computer Simulation* 30.1 (Jan. 2020), pp. 1–25. ISSN: 1049-3301, 1558-1195. DOI: 10.1145/3369759. URL: https://dl.acm.org/doi/10.1145/3369759 (visited on 07/08/2024).

[81] Newton Masinde and Kalman Graffi. "Peer-to-Peer-Based Social Networks: A Comprehensive Survey". en. In: *SN Computer Science* 1.5 (Sept. 2020), p. 299. ISSN: 2661-8907. DOI: 10.1007/s42979-020-00315-8. URL: https://doi.org/10.1007/s42979-020-00315-8 (visited on 10/10/2022).

[82] Konstantin D. Pandl et al. "On the Convergence of Artificial Intelligence and Distributed Ledger Technology: A Scoping Review and Future Research Agenda". In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 57075–57095. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2981447.

[83] Tobias Pfandzelter and David Bermbach. "tinyFaaS: A Lightweight FaaS Platform for Edge Environments". In: *2020 IEEE International Conference on Fog Computing (ICFC)*. Apr. 2020, pp. 17–24. DOI: 10.1109/ICFC49376.2020.00011.

[84] Yiannis Psaras and David Dias. "The InterPlanetary File System and the Filecoin Network". In: *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. June 2020, pp. 80–80. DOI: 10.1109/DSN-S50200.2020.00043.

[85] Stefano Savazzi, Monica Nicoli, and Vittorio Rampa. "Federated Learning with Cooperating Devices: A Consensus Approach for Massive IoT Networks". In: *IEEE Internet of Things Journal* 7.5 (May 2020). arXiv: 1912.13163, pp. 4641–4654. ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2020.2964162. URL: http://arxiv.org/abs/1912.13163 (visited on 02/11/2022).

[86] Simon Shillaker and Peter Pietzuch. "Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing". en. In: 2020, pp. 419–433. ISBN: 978-1-939133-14-4. URL: https://www.usenix.org/conference/atc20/presentation/shillaker (visited on 05/12/2023).

[87] Vale Tolpegin et al. "Data poisoning attacks against federated learning systems". In: *European Symposium on Research in Computer Security*. Springer, 2020, pp. 480–501.

[88] H. Wang et al. "Attack of the Tails: Yes, You Really Can Backdoor Federated Learning". In: *NeurIPS*. 2020.

[89] Yang Xiao et al. "A Survey of Distributed Consensus Protocols for Blockchain Networks". en. In: *IEEE Communications Surveys & Tutorials* 22.2 (2020). arXiv: 1904.04098, pp. 1432–1465. ISSN: 1553-877X, 2373-745X. DOI: 10.1109/COMST.2020.2969706. URL: http://arxiv.org/abs/1904.04098 (visited on 01/31/2022).

[90] Yang Xiao et al. "A Survey of Distributed Consensus Protocols for Blockchain Networks". In: *IEEE Comm. Surveys & Tutorials* 22.2 (2020), pp. 1432–1465.

[91] Cong Xie, Sanmi Koyejo, and Indranil Gupta. "Asynchronous Federated Optimization". en. In: *arXiv:1903.03934 [cs]* (Dec. 2020). arXiv: 1903.03934. URL: http://arxiv.org/abs/1903.03934 (visited on 02/11/2022).

[92] M. Xue et al. "Machine Learning Security: Threats, Countermeasures, and Evaluations". In: *IEEE Access* 8 (2020), pp. 74720–74742.

[93] Chunyi Zhou et al. "Privacy-Preserving Federated Learning in Fog Computing". In: *IEEE Internet of Things Journal* 7.11 (Nov. 2020). Conference Name: IEEE Internet of Things Journal, pp. 10782–10793. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2987958.

[94] Qiheng Zhou et al. "Solutions to Scalability of Blockchain: A Survey". en. In: *IEEE Access* 8 (2020), pp. 16440–16455. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2967218. URL: https://ieeexplore.ieee.org/document/8962150/ (visited on 06/28/2024).

[95] Medha Atre, Birendra Jha, and Ashwini Rao. "Distributed Deep Learning Using Volunteer Computing-Like Paradigm". en. In: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. arXiv:2103.08894 [cs]. June 2021, pp. 933–942. DOI: 10.1109/IPDPSW52791.2021.00144. URL: http://arxiv.org/abs/2103.08894 (visited on 09/01/2022).

[96] Paolo Bellavista, Luca Foschini, and Alessio Mora. "Decentralised Learning in Federated Deployment Environments: A System-Level Survey". en. In: *ACM Computing Surveys* 54.1 (Apr. 2021), pp. 1–38. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3429252. URL: https://dl.acm.org/doi/10.1145/3429252 (visited on 01/25/2022).

[97] Michele Ciavotta et al. "DFaaS: Decentralized Function-as-a-Service for Federated Edge Computing". In: *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*. Nov. 2021, pp. 1–4. DOI: 10.1109/CloudNet53349.2021.9657141.

[98] Claudio Cicconetti, Marco Conti, and Andrea Passarella. "A Decentralized Framework for Serverless Edge Computing in the Internet of Things". en. In: *IEEE Transactions on Network and Service Management* 18.2 (June 2021). arXiv:2110.10974 [cs], pp. 2166–2180. ISSN: 1932-4537, 2373-7379. DOI: 10.1109/TNSM.2020.3023305. URL: http://arxiv.org/abs/2110.10974 (visited on 06/08/2023).

[99] Kha Dinh Duy et al. "Confidential Machine Learning Computation in Untrusted Environments: A Systems Security Perspective". In: *IEEE Access* 9 (2021). Conference Name: IEEE Access, pp. 168656–168677. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3136889.

[100] Gabriel Estevam et al. "Accurate and decentralized timestamping using smart contracts on the Ethereum blockchain". In: *Information Processing & Management* 58.3 (May 2021), p. 102471. ISSN: 0306-4573. DOI: 10.1016/j.ipm.2020.102471. URL: https://www.sciencedirect.com/science/article/pii/S0306457320309602 (visited on 06/27/2024).

[101] Yann Fraboni, Richard Vidal, and Marco Lorenzi. "Free-rider attacks on model aggregation in federated learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PLMR, 2021, pp. 1846–1854.

[102] Kalman Graffi and Newton Masinde. "LibreSocial: A peer-to-peer framework for online social networks". en. In: *Concurrency and Computation: Practice and Experience* 33.8 (2021). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6150, e6150. ISSN: 1532-0634. DOI: 10.1002/cpe.6150. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6150 (visited on 11/18/2022).

[103] Xiaojie Guo et al. "VeriFL: Communication-Efficient and Fast Verifiable Aggregation for Federated Learning". In: *IEEE Transactions on Information Forensics and Security* 16 (2021), pp. 1736–1751.

[104] Yahya Hassanzadeh-Nazarabadi et al. *DHT-based Communications Survey: Architectures and Use Cases*. en. arXiv:2109.10787 [cs]. Sept. 2021. URL: http://arxiv.org/abs/2109.10787 (visited on 03/08/2023).

[105] Peter Kairouz et al. "Advances and Open Problems in Federated Learning". In: *arXiv:1912.04977 [cs, stat]* (Mar. 2021). arXiv: 1912.04977. URL: http://arxiv.org/abs/1912.04977 (visited on 01/18/2022).

[106] Peter Kairouz et al. "Advances and Open Problems in Federated Learning". English. In: *Foundations and Trends® in Machine Learning* 14.1–2 (June 2021). Publisher: Now Publishers, Inc., pp. 1–210. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000083. URL: https://www.nowpublishers.com/article/Details/MAL-083 (visited on 05/16/2022).

[107] Peter Kairouz et al. "Advances and Open Problems in Federated Learning". In: *Foundations and Trends in Machine Learning* 14.1–2 (2021), pp. 1–210.

[108] Sin Kit Lo et al. *Architectural Patterns for the Design of Federated Learning Systems*. en. arXiv:2101.02373 [cs]. June 2021. URL: http://arxiv.org/abs/2101.02373 (visited on 09/29/2023).

[109] Sin Kit Lo et al. "FLRA: A reference architecture for federated learning systems". In: *European Conference on Software Architecture*. 2021.

[110] Fan Mo et al. "PPFL: Privacy-preserving Federated Learning with Trusted Execution Environments". In: *arXiv:2104.14380 [cs]* (June 2021). arXiv: 2104.14380. URL: http://arxiv.org/abs/2104.14380 (visited on 01/18/2022).

[111] Christodoulos Pappas et al. "IPLS : A Framework for Decentralized Federated Learning". en. In: *arXiv:2101.01901 [cs]* (Jan. 2021). arXiv: 2101.01901. URL: http://arxiv.org/abs/2101.01901 (visited on 02/14/2022).

[112] Saurav Prakash et al. "Byzantine-Resilient Federated Learning with Heterogeneous Data Distribution". In: *arXiv:2010.07541 [cs]* (July 2021). arXiv: 2010.07541. URL: http://arxiv.org/abs/2010.07541 (visited on 01/17/2022).

[113] Kuniyasu Suzaki et al. "TS-Perf: General Performance Measurement of Trusted Execution Environment and Rich Execution Environment on Intel SGX, Arm TrustZone, and RISC-V Keystone". In: *IEEE Access* 9 (2021). Conference Name: IEEE Access, pp. 133520–133530. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3112202. URL: https://ieeexplore.ieee.org/document/9536719/?arnumber=9536719 (visited on 01/07/2025).

[114] Joost Verbraeken et al. "A Survey on Distributed Machine Learning". en. In: *ACM Computing Surveys* 53.2 (Mar. 2021), pp. 1–33. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3377454. URL: https://dl.acm.org/doi/10.1145/3377454 (visited on 01/18/2022).

[115] Pedro Agostinho, David Dias, and Luís Veiga. "SmartPubSub: Content-based Pub-Sub on IPFS". In: *2022 IEEE 47th Conference on Local Computer Networks (LCN)*. ISSN: 0742-1303. Sept. 2022, pp. 327–330. DOI: 10.1109/LCN53696.2022.9843795. URL: https://ieeexplore.ieee.org/abstract/document/9843795 (visited on 06/27/2024).

[116] A. A. Messaoud et al. "Shielding Federated Learning Systems against Inference Attacks with ARM TrustZone". In: *ACM/IFIP International Middleware Conference (MIDDLEWARE)*. 2022.

[117] F. Elhattab et al. "Robust Federated Learning for Ubiquitous Computing through Mitigation of Edge-Case Backdoor Attacks". In: *Proc. ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6.4 (2022), pp. 1–27.

[118] I. Messadi et al. "SplitBFT: Improving Byzantine Fault Tolerance Safety Using Trusted Compartments". In: *ACM/IFIP International Middleware Conference (MIDDLEWARE)*. 2022.

[119] Enrique Tomás Martínez Beltrán et al. *Decentralized Federated Learning: Fundamentals, State-of-the-art, Frameworks, Trends, and Challenges*. en. arXiv:2211.08413 [cs]. Nov. 2022. URL: http://arxiv.org/abs/2211.08413 (visited on 11/18/2022).

[120] A. Borzunov et al. "Petals: Collaborative Inference and Fine-tuning of Large Models". In: *61st Annual Meeting of the Association for Computational Linguistics*. 2022.

[121] A. Borzunov et al. "Training transformers together". In: *NeurIPS 2021 Competition and Demonstration Track*. PMLR. 2022.

[122] E. Cyffers and A. Bellet. "Privacy Amplification by Decentralization". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2022.

[123] Erik Daniel and Florian Tschorsch. "IPFS and Friends: A Qualitative Comparison of Next Generation Peer-to-Peer Data Networks". In: *IEEE Communications Surveys & Tutorials* 24.1 (2022). Conference Name: IEEE Communications Surveys & Tutorials, pp. 31–52. ISSN: 1553-877X. DOI: 10.1109/COMST.2022.3143147.

[124] Erik Daniel and Florian Tschorsch. *IPFS and Friends: A Qualitative Comparison of Next Generation Peer-to-Peer Data Networks*. en. arXiv:2102.12737 [cs]. Jan. 2022. URL: http://arxiv.org/abs/2102.12737 (visited on 09/08/2022).

[125] Z. Tarkhani F. Mo and H. Haddadi. "SoK: Machine Learning with Confidential Computing". In: *arXiv:2208.10134* (2022).

[126] Ehsan Hallaji, Roozbeh Razavi-Far, and Mehrdad Saif. *Federated and Transfer Learning: A Survey on Adversaries and Defense Mechanisms*. en. arXiv:2207.02337 [cs]. July 2022. URL: http://arxiv.org/abs/2207.02337 (visited on 07/12/2022).

[127] David Harel, Assaf Marron, and Joseph Sifakis. "Creating a Foundation for Next-Generation Autonomous Systems". In: *IEEE Design & Test* 39.1 (Feb. 2022). Conference Name: IEEE Design & Test, pp. 49–56. ISSN: 2168-2364. DOI: 10.1109/MDAT.2021.3069959. URL: https://ieeexplore.ieee.org/abstract/document/9391694 (visited on 03/22/2024).

[128] I. Messadi et al. "SplitBFT: Improving Byzantine Fault Tolerance Safety Using Trusted Compartments". In: *Middleware*. 2022.

[129] Chiara Valentina Misischia, Flora Poecze, and Christine Strauss. "Chatbots in customer service: Their relevance and impact on service quality". In: *Procedia Computer Science*. The 13th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 5th International Conference on Emerging Data and Industry 4.0 (EDI40) 201 (Jan. 2022), pp. 421–428. ISSN: 1877-0509. DOI: 10.1016/j.procs.2022.03.055. URL: https://www.sciencedirect.com/science/article/pii/S1877050922004689 (visited on 07/17/2024).

[130] Thien Duc Nguyen et al. "{FLAME}: Taming Backdoors in Federated Learning". en. In: 2022, pp. 1415–1432. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/nguyen (visited on 02/13/2023).

[131] Sashko Ristov, Christian Hollaus, and Mika Hautz. "Colder Than the Warm Start and Warmer Than the Cold Start! Experience the Spawn Start in FaaS Providers". en. In: *Proceedings of the 2022 Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*. Salerno Italy: ACM, July 2022, pp. 35–39. ISBN: 978-1-4503-9280-8. DOI: 10.1145/3524053.3542751. URL: https://dl.acm.org/doi/10.1145/3524053.3542751 (visited on 10/15/2024).

[132] Nuria Rodríguez-Barroso et al. "Survey on Federated Learning Threats: concepts, taxonomy on attacks and defences, experimental study and challenges". In: *arXiv:2201.08135 [cs]* (Jan. 2022). arXiv: 2201.08135. URL: http://arxiv.org/abs/2201.08135 (visited on 01/26/2022).

[133] Daniel Rosendo et al. "Distributed intelligence on the Edge-to-Cloud Continuum: A systematic literature review". en. In: *Journal of Parallel and Distributed Computing* 166 (Aug. 2022), pp. 71–94. ISSN: 07437315. DOI: 10.1016/j.jpdc.2022.04.004. URL: https://linkinghub.elsevier.com/retrieve/pii/S0743731522000843 (visited on 01/10/2023).

[134] Selma Saidi et al. "Autonomous Systems Design: Charting a New Discipline". In: *IEEE Design & Test* 39.1 (Feb. 2022). Conference Name: IEEE Design & Test, pp. 8–23. ISSN: 2168-2364. DOI: 10.1109/MDAT.2021.3128434. URL: https://ieeexplore.ieee.org/document/9615198 (visited on 03/22/2024).

[135] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. "Serverless Computing: A Survey of Opportunities, Challenges, and Applications". In: *ACM Computing Surveys* 54.11s (Nov. 2022), 239:1–239:32. ISSN: 0360-0300. DOI: 10.1145/3510611. URL: https://dl.acm.org/doi/10.1145/3510611 (visited on 05/12/2023).

[136] Saima Shahab et al. "SIoT (Social Internet of Things): A Review". en. In: *ICT Analysis and Applications*. Ed. by Simon Fong, Nilanjan Dey, and Amit Joshi. Lecture Notes in Networks and Systems. Singapore: Springer Nature, 2022, pp. 289–297. ISBN: 9789811656552. DOI: 10.1007/978-981-16-5655-2_28.

[137] Chandra Thapa et al. "SplitFed: When Federated Learning Meets Split Learning". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.8 (June 2022). Number: 8, pp. 8485–8493. ISSN: 2374-3468. DOI: 10.1609/aaai.v36i8.20825. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20825 (visited on 09/02/2022).

[138] Dennis Trautwein et al. "Design and evaluation of IPFS: a storage layer for the decentralized web". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, Aug. 2022, pp. 739–752. ISBN: 978-1-4503-9420-8. DOI: 10.1145/3544216.3544232. URL: https://doi.org/10.1145/3544216.3544232 (visited on 02/15/2023).

[139] David Wingqvist, Filip Wickström, and Suejb Memeti. "Evaluating the performance of object-oriented and data-oriented design with multi-threading in game development". In: *2022 IEEE Games, Entertainment, Media Conference (GEM)*. 2022, pp. 1–6. DOI: 10.1109/GEM56474.2022.10017610.

[140] Haidong Zhao et al. "Supporting Multi-Cloud in Serverless Computing". In: *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. Dec. 2022, pp. 285–290. DOI: 10.1109/UCC56403.2022.00051. URL: https://ieeexplore.ieee.org/document/10061782/?arnumber=10061782 (visited on 10/14/2024).

[141] Mohamad Arafeh et al. "ModularFed: Leveraging modularity in federated learning frameworks". In: *Internet of Things* 22 (2023), p. 100694.

[142] Divi De Lacour et al. "Towards Scalable Resilient Federated Learning: A Fully Decentralised Approach". In: *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. ISSN: 2766-8576. Mar. 2023, pp. 621–627. DOI: 10.1109/PerComWorkshops56833.2023.10150295.

[143] EPFL Machine Learning and Optimization Laboratory. *DISCO - Distributed Collaborative Machine Learning*. Accessed: 2023-09-22. 2023. URL: https://github.com/epfml/disco.

[144] Rachid Guerraoui, Nirupam Gupta, and Rafael Pinot. "Byzantine Machine Learning: A Primer". en. In: *ACM Computing Surveys* (Aug. 2023), p. 3616537. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3616537. URL: https://dl.acm.org/doi/10.1145/3616537 (visited on 08/22/2023).

[145] Andreas Haeberlen, Linh Thi Xuan Phan, and Morgan McGuire. "Metaverse as a Service: Megascale Social 3D on the Cloud". en. In: *Proceedings of the 2023 ACM Symposium on Cloud Computing*. Santa Cruz CA USA: ACM, Oct. 2023, pp. 298–307. ISBN: 9798400703874. DOI: 10.1145/3620678.3624662. URL: https://dl.acm.org/doi/10.1145/3620678.3624662 (visited on 11/28/2023).

[146] E. Hallaji et al. "Federated and Transfer Learning: A Survey on Adversaries and Defense Mechanisms". In: *Federated and Transfer Learning*. Springer, 2023, pp. 29–55.

[147] Felix Hoops et al. "A Taxonomy of Decentralized Identifier Methods for Practitioners". In: *2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. ISSN: 2835-3498. July 2023, pp. 57–65. DOI: 10.1109/DAPPS57946.2023.00017. URL: https://ieeexplore.ieee.org/document/10237038/?arnumber=10237038 (visited on 11/22/2024).

[148] Cornelius Ihle et al. "Incentive Mechanisms in Peer-to-Peer Networks — A Systematic Literature Review". In: *ACM Computing Surveys* (Jan. 2023). Just Accepted. ISSN: 0360-0300. DOI: 10.1145/3578581. URL: https://dl.acm.org/doi/10.1145/3578581 (visited on 05/12/2023).

[149] Boyu Kuang et al. "FeSA: Automatic Federated Swarm Attestation on Dynamic Large-Scale IoT Devices". In: *IEEE Transactions on Dependable and Secure Computing* 20.4 (July 2023). Conference Name: IEEE Transactions on Dependable and Secure Computing, pp. 2954–2969. ISSN: 1941-0018. DOI: 10.1109/TDSC.2022.3193106. URL: https://ieeexplore.ieee.org/abstract/document/9837456 (visited on 01/09/2025).

[150] Xing Li, Xue Leng, and Yan Chen. "Securing Serverless Computing: Challenges, Solutions, and Opportunities". In: *IEEE Network* 37.2 (Mar. 2023). Conference Name: IEEE Network, pp. 166–173. ISSN: 1558-156X. DOI: 10.1109/MNET.005.2100335. URL: https://ieeexplore.ieee.org/document/9933509/?arnumber=9933509 (visited on 10/07/2024).

[151] Yongkang Li et al. "Serverless Computing: State-of-the-Art, Challenges and Opportunities". en. In: *IEEE Transactions on Services Computing* 16.2 (Mar. 2023), pp. 1522–1539. ISSN: 1939-1374, 2372-0204. DOI: 10.1109/TSC.2022.3166553. URL: https://ieeexplore.ieee.org/document/9756233/ (visited on 04/10/2024).

[152] Mohamad Mansouri et al. "SoK: Secure Aggregation Based on Cryptographic Schemes for Federated Learning". en. In: *Proceedings on Privacy Enhancing Technologies* 2023.1 (Jan. 2023), pp. 140–157. ISSN: 2299-0984. DOI: 10.56553/popets-2023-0009. URL: https://petsymposium.org/popets/2023/popets-2023-0009.php (visited on 04/27/2023).

[153] Alessandro Margara et al. "A Model and Survey of Distributed Data-Intensive Systems". In: *ACM Computing Surveys* 56.1 (Aug. 2023), 16:1–16:69. ISSN: 0360-0300. DOI: 10.1145/3604801. URL: https://dl.acm.org/doi/10.1145/3604801 (visited on 09/25/2023).

[154] Gabriele Russo Russo et al. "Serverledge: Decentralized Function-as-a-Service for the Edge-Cloud Continuum". In: *2023 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. ISSN: 2474-249X. Mar. 2023, pp. 131–140. DOI: 10.1109/PERCOM56429.2023.10099372.

[155] Joseph Sifakis and David Harel. "Trustworthy Autonomous System Development". en. In: *ACM Transactions on Embedded Computing Systems* 22.3 (May 2023), pp. 1–24. ISSN: 1539-9087, 1558-3465. DOI: 10.1145/3545178. URL: https://dl.acm.org/doi/10.1145/3545178 (visited on 03/22/2024).

[156] Z. Tang et al. "FusionAI: Decentralized Training and Deploying LLMs with Massive Consumer-Level GPUs". In: *LLM-IJCAI*. 2023.

[157] H. Wang et al. "Scientific discovery in the age of artificial intelligence". In: *Nature* 620.7972 (2023), pp. 47–60.

[158] Jie Xu, Cong Wang, and Xiaohua Jia. "A Survey of Blockchain Consensus Protocols". In: *ACM Computing Surveys* (Jan. 2023). Just Accepted. ISSN: 0360-0300. DOI: 10.1145/3579845. URL: https://dl.acm.org/doi/10.1145/3579845 (visited on 05/12/2023).

[159] Ming Zhao, Kritshekhar Jha, and Sungho Hong. "GPU-enabled Function-as-a-Service for Machine Learning Inference". In: *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. ISSN: 1530-2075. May 2023, pp. 918–928. DOI: 10.1109/IPDPS54959.2023.00096. URL: https://ieeexplore.ieee.org/document/10177435/?arnumber=10177435 (visited on 10/17/2024).

[160] Julio C. S. Dos Anjos et al. "A Survey on Collaborative Learning for Intelligent Autonomous Systems". en. In: *ACM Computing Surveys* 56.4 (Apr. 2024), pp. 1–37. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3625544. URL: https://dl.acm.org/doi/10.1145/3625544 (visited on 03/25/2024).

[161] Ons Aouedi et al. "A Survey on Intelligent Internet of Things: Applications, Security, Privacy, and Future Directions". In: *IEEE Communications Surveys & Tutorials* (2024).

[162] Divi De Lacour et al. "Ti-skol: A Modular Federated Learning Framework Supporting Security Countermeasure Composition". In: *2024 IEEE International Conference on Big Data (BigData)*. IEEE. 2024, pp. 7860–7869.

[163] Shanshan Han et al. "FedMLSecurity: A Benchmark for Attacks and Defenses in Federated Learning and Federated LLMs". In: *30th SIGKDD Conference on Knowledge Discovery and Data Mining*. 2024.

[164] Vincent C Hu. *Access control on NoSQL databases*. en. Tech. rep. NIST IR 8504. Gaithersburg, MD: National Institute of Standards and Technology (U.S.), May 2024, NIST IR 8504. DOI: 10.6028/NIST.IR.8504. URL: https://nvlpubs.nist.gov/nistpubs/ir/2024/NIST.IR.8504.pdf (visited on 09/11/2024).

[165] Navin Keizer et al. "A Survey on Content Retrieval on the Decentralised Web". en. In: *ACM Computing Surveys* 56.8 (Aug. 2024), pp. 1–39. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3649132. URL: https://dl.acm.org/doi/10.1145/3649132 (visited on 01/08/2025).

[166] Martin Kleppmann et al. *Bluesky and the AT Protocol: Usable Decentralized Social Media*. en. arXiv:2402.03239 [cs]. Oct. 2024. DOI: 10.1145/3694809.3700740. URL: http://arxiv.org/abs/2402.03239 (visited on 11/19/2024).

[167] Kevin Koidl, Ville Ollikainen, and Jarkko Kuusijärvi. "HELIOS a Decentralized Online Social Network Framework". In: *2024 IEEE International Symposium on Technology and Society (ISTAS)*. ISSN: 2158-3412. Sept. 2024, pp. 1–8. DOI: 10.1109/ISTAS61960.2024.10732337. URL: https://ieeexplore.ieee.org/document/10732337/?arnumber=10732337 (visited on 11/07/2024).

[168] Ariel Kwiatkowski et al. *Gymnasium: A Standard Interface for Reinforcement Learning Environments*. arXiv:2407.17032. Oct. 2024. DOI: 10.48550/arXiv.2407.17032. URL: http://arxiv.org/abs/2407.17032 (visited on 10/22/2024).

[169] Petru Neague, Marcel Gregoriadis, and Johan Pouwelse. "De-DSI: Decentralised Differentiable Search Index". en. In: *Proceedings of the 4th Workshop on Machine Learning and Systems*. arXiv:2404.12237 [cs]. Apr. 2024, pp. 134–143. DOI: 10.1145/3642970.3655837. URL: http://arxiv.org/abs/2404.12237 (visited on 04/25/2024).

[170] Inside Track staff. *Microsoft Teams increases collaboration in the modern workplace at Microsoft*. en-US. June 2024. URL: https://www.microsoft.com/insidetrack/blog/microsoft-teams-increases-collaboration-in-the-modern-workplace-at-microsoft/ (visited on 07/17/2024).

[171] Divi De Lacour et al. "Towards a Reference Architecture for Secure Cooperative Autonomous Systems for IIoT". In: *2025 28th Conference on Innovation in Clouds, Internet and Networks (ICIN)*. IEEE. 2025, pp. 181–185.

[172] *ActivityPub*. https://www.w3.org/TR/activitypub/.

[173] A. Haas et al. "Bringing the Web up to Speed with WebAssembly". In: *PLDI'17*.

[174] Chu et al. "Map-Reduce for Machine Learning on Multicore". In: *NIPS'06*.

[175] H. Cui et al. "Exploiting Bounded Staleness to Speed up Big Data Analytics". In: *USENIX ATC'14*.

[176] H. Zhang et al. “Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters”. In: *USENIX ATC'17*.

[177] M. Li et al. “Scaling Distributed Machine Learning with the Parameter Server”. In: *ODSI'14*.

[178] M. Ryabinin et al. “Moshpit sgd: Communication-efficient decentralized training on heterogeneous unreliable devices”. In: *NIPS'21*.

[179] T. D. Nguyen et al. “DÏoT: A Federated Self-learning Anomaly Detection System for IoT”. In: *ICDCS'19*.

[180] Roger Dingledine, Nick Mathewson, and Paul Syverson. “Tor: The Second-Generation Onion Router”. en. In: (), p. 18.

[181] Michael Diskin et al. “Distributed Deep Learning In Open Collaborations”. en. In: (), p. 19.

[182] *ENS*. `https://ens.domains`.

[183] ETSI White Paper. *Unlocking Digital Transformation with Autonomous Networks*.

[184] TensorFlow Federated. *TensorFlow Federated*. `https://www.tensorflow.org/federated/`.

[185] Mohamed Islam Ghamri, Marc Lacoste, and Divi De Lacour. “Disaggregation Patterns for Secure AI Systems”. en. In: ().

[186] *Handbook of Peer-to-Peer Networking*. en. URL: `https://link.springer.com/book/10.1007/978-0-387-09751-0` (visited on 02/06/2023).

[187] Yanping Huang et al. “GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism”. In: *NeurIPS 2019*.

[188] *Inactive Google Account Policy - Google Account Help*. `https://support.google.com/accounts/answer/12418290?hl=en&sjid=6046569782917084002-EU`.

[189] *Introducing Twitter Alerts*. `https://blog.x.com/en_us/a/2013/introducing-twitter-alerts`.

[190] *IPFS Concepts: IPNS*. `https://docs.ipfs.tech/concepts/ipns/`.

[191] Leonard Kleinrock. “Distributed systems”. en. In: ().

[192] Yann LeCun. “A Path Towards Autonomous Machine Intelligence Version 0.9.2, 2022-06-27”. en. In: (), p. 62.

[193] James Menetrey et al. “WaTZ: A Trusted WebAssembly Runtime Environment with Remote Attestation for TrustZone”. en. In: (), p. 13.

[194] Aghiles Ait Messaoud et al. “Shielding Federated Learning Systems against Inference Attacks with ARM TrustZone”. In: *Middleware 2022*.

[195] *More details about the October 4 outage*. `https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/`.

[196] *OrbitDB*. `https://orbitdb.org/`.

[197] *OrbitDB - Home*. URL: https://orbitdb.org/ (visited on 07/25/2024).

[198] *Our Digital History Is at Risk — Internet Archive Blogs*. https://blog.archive.org/2023/02/07/our-digital-history-is-at-risk/.

[199] *OWASP Top 10 for Large Language Model Applications — OWASP Foundation*. en. URL: https://owasp.org/www-project-top-10-for-large-language-model-applications/ (visited on 07/20/2023).

[200] Mark Russinovich et al. "Toward Confidential Cloud Computing". en. In: *hardware security* (), p. 28.

[201] Max Ryabinin and Anton Gusev. "Towards Crowdsourced Training of Large Neural Networks using Decentralized Mixture-of-Experts". en. In: (), p. 14.

[202] Max Ryabinin et al. "Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices". en. In: (), p. 17.

[203] *Social Web Protocols*. https://www.w3.org/TR/social-web-protocols/.

[204] Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach - Pearson France*. en. URL: https://www.pearson.fr/FR/book/?GCOI=27440100705580 (visited on 11/25/2022).

[205] J. O. du Terrail et al. "FLamby: Datasets and Benchmarks for Cross-Silo Federated Learning in Realistic Healthcare Settings". In: *NeurIPS'22*.

[206] Florian Tramèr and Dan Boneh. "Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware". In: *ICLR 2019*.

[207] Kapil Vaswani et al. "Confidential Computing within an AI Accelerator". In: *USENIX ATC 2023*.

[208] Steven Daniel Webb and Sieteng Soh. "Cheating in networked computer games – A review". en. In: ().

COLLEGES SCIENCES
BRETAGNE POUR L'INGENIEUR
LOIRE ET LE NUMERIQUE

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Architecture et sécurité pour les systèmes coopératifs autonomes

Mots clés : systèmes coopératifs autonomes, apprentissage fédéré, sécurité, décentralisation

Résumé : Les Systèmes Coopératifs Autonomes, tels que les véhicules intelligents, sont de plus en plus utilisés dans l'Internet Industriel des Objets (IIoT) pour partager des tâches et des informations. Leur complexité et leur nature distribuée nécessitent un déploiement sur une grande diversité d'environnements à travers le continuum IoT-Cloud. Ces systèmes traitent des données sensibles, ce qui rend la sécurité et la confidentialité essentielles. Bien que diverses contre-mesures existent, leur combinaison doit être soigneusement pensée, car elles peuvent affecter les performances et être incompatibles. Cette thèse passe en revue la littérature dans les domaines de l'intelligence artificielle, des systèmes distribués, de la sécurité ainsi que de la combinaison de ceux-ci. Elle propose une architecture de référence (RACAS) pour classifier les méthodes de coopération des systèmes autonomes et explore l'IA distribuée, notamment l'IA désagrégée. Nous évaluons la performance et la sécurité des fonctions de coopération clés : apprentissage collaboratif décentralisé (FDFL) et avec sécurité à la demande (Ti-skol), délégation de tâches (D-Lambda) et communication d'état (ReDOSN). Un cadre de simulation sécurisé et décentralisé pour des environnements de systèmes autonomes est également proposé (D-ECS). Nous montrons que la distribution et la décentralisation sur le continuum Cloud-IoT améliore les performances et la résilience. La thèse identifie les besoins en sécurité pour chaque fonction et propose la composition des contre-mesures de sécurité.

Title : Architecture and security for cooperative autonomous systems

Keywords : cooperative autonomous systems, federated learning, security, decentralisation

Abstract : Cooperative Autonomous Systems (CAS), such as intelligent vehicles, are increasingly used in Industrial IoT (IIoT) to share tasks and information. Their complexity and distributed nature require deployment across diverse hardware along the IoT-Cloud continuum. These systems handle sensitive data, making security and privacy essential. While various security countermeasures exist, they must be combined thoughtfully, as they can impact performance and be incompatible. This thesis reviews literature in AI, distributed systems, security and their combination. It proposes a reference architecture (RACAS), identifying CAS cooperation methods and explores distributed AI, especially disaggregated AI. We evaluate the performance and security of key cooperation functions: collaborative learning decentralisation (FDFL) and on-demand security (Ti-skol), task delegation (D-Lambda), and status communication (ReDOSN). A simulation framework (D-ECS) for secure, decentralized CAS environments is also introduced. We demonstrate that distribution and decentralization across the Cloud-IoT continuum enhances performance, scalability, and resilience. The thesis identifies security requirements for each function and suggests countermeasure composition for applicability.