D2.1: Report on the Initial Requirements Analysis from Co-Design

Revision: v.1.2

Work package	WP2
Task	T2.1, T2.2
Due date	30/Jun/2023
Submission date	30/Jun/2023
Deliverable lead	Aravindh Raman (TID)
Version	1.2
Authors	Aravindh Raman (TID); Roland Kuhn (ACT); Giovanni Granato (GMV); Sotirios Spantideas (NKUA)
Internal Reviewers	Joao Costa Seco (NOVA); Miroslav Zaric (UNS); Roland Kuhn (ACT)
Abstract	This document reports on the methodology used, the individual findings, and the derived synthesis for an initial set of functional requirements to be fulfilled by the TaRDIS toolbox. It is the basis and starting point for developing the full specification of models, analyses, and APIs to be incorporated in the toolbox.
Keywords	decentralised programming toolbox, co-design

Grant Agreement No.: 101093006
Call: HORIZON-CL4-2022-DATA-01

Topic: HORIZON-CL4-2022-DATA-01-03
Type of action: HORIZON- RIA

DISCLAIMER

Funded by the European Union

Funded by the European Union (TARDIS, 101093006). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

COPYRIGHT NOTICE

Project funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	**R**	
Dissemination Level		
PU	*Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)*	✓
SEN	*Sensitive, limited under the conditions of the Grant Agreement*	
Classified R-UE/ EU-R	*EU RESTRICTED under the Commission Decision No2015/ 444*	
Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

* R: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

DATA: Data sets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

OTHER: Software, technical diagram, algorithms, models, etc.

EXECUTIVE SUMMARY

The TaRDIS project aims to build a distributed programming toolbox to simplify the development of decentralized applications deployed in a diverse setting. To this end, in this report, we showcase the methodology and results from a comprehensive co-design process. Leveraging large-scale data from external platforms, we gain insights into trends, patterns, and issues in decentralized application development. We conduct targeted surveys to gather direct feedback from diverse programmers, capturing their challenges, preferences, and suggestions. By combining these insights with the specific needs of our industrial partners, we present the initial set of requirements that lays the foundation for TaRDIS. Our systematic and inclusive co-design approach ensures that the toolbox meets real-world challenges in the decentralized landscape.

Our key contributions for this report encompass various aspects:

Firstly, through large-scale data analysis we delve into the growth of decentralized applications, their associated challenges, and the diverse range of programming languages. This helps ensure that TaRDIS remains aligned with the evolving needs of programming and empowers developers in this dynamic landscape.

Secondly, we present the findings of a questionnaire tailored to gather insights from developer communities. This questionnaire covers various aspects such as participants' familiarity with decentralized programming techniques, their programming language preferences, infrastructure deployment patterns, and their expectations from a decentralized programming toolbox.

Finally, we examine different use cases and describe the challenges encountered in each scenario. Our objective is to effectively address these challenges during the development of the TaRDIS toolbox.
Some key highlights we want to present as outcomes of the M6 report are as follows:

- Industrial partners, paired with academic partners (EDP and NOVA, GMV and UNS, ACT and DTU, TID and NOVA) participated in individual workshops to discuss the use case and understand the requirements to build the distributed programming environment and aspects of distributed machine learning.
- TID has published a couple of works showing the growth in decentralized application adoption and the challenges at The Web Conference (WWW 2023), a seminal conference on the topic of the future direction of the World Wide Web.
- As a part of co-design, a questionnaire is prepared by all the partners and distributed across developers involved in decentralised application development, outside the core TaRDIS team. We got close to 150 responses and the results not only strongly suggest the need for TaRDIS-like toolbox but also help in identifying the initial requirements outside of the consortium.
- As a part of co-design, TID and NOVA participated in regular meetings with Protocol Labs, a key contributor to open-source development of decentralised applications and components, such as IPFS.
- NOVA and TID hosted two editions of an online workshop involving all the partners to define the heterogeneous swarms from the insights of use-case discussions.

Funded by the European Union

TABLE OF CONTENTS

Funded by the European Union

LIST OF FIGURES

Funded by the European Union

ABBREVIATIONS

ML	Machine Learning
DL	Deep Learning
FL	Federated Learning
IPFS	InterPlanetary File System
P2P	Peer-to-Peer
CDF	Cumulative Distribution Function
API	Application Programming Interface
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HTTP	Hypertext Transfer Protocol
LEO	Low Earth Orbit
ISL	Inter-Satellite-Link
ODTS	Orbit Determination and Time Synchronization
G2G	Galileo 2nd Generation of satellites
BDS-3	BeiDou 3rd Generation navigation satellite system
AGV	Automated Guided Vehicle
ERP	Enterprise Resource Planning
MES	Manufacturing Execution System
PNT	Position, Navigation and Timing
DER	Distributed Energy Resources
SGAM	Smart-Grid Architectural Model
IoT	Internet of Things
JS	JavaScript
DP	Differential Privacy

Funded by the European Union

1 INTRODUCTION

The recent years have witnessed significant growth in the development of decentralized applications, driven by their ability to eliminate intermediaries, leverage decentralized networks and infrastructures. Traditional applications often rely on centralized servers, whereas decentralized applications utilize peer-to-peer networks or distributed ledgers, offering advantages such as improved data privacy, censorship resistance, and user empowerment. However, this growth has brought forth *challenges related to heterogeneity within decentralized networks*, encompassing diverse devices, protocols, and technologies.

1.1 TOP-DOWN CO-DESIGN APPROACH

The TaRDIS toolbox is specifically designed to simplify the development process of decentralized applications. In order to effectively address the requirements and challenges of this domain, we have implemented a comprehensive co-design process. This process adopts a top-down approach, allowing us to gather insights from various sources and engage with a diverse range of programmers.

To begin with, we leverage large-scale data collected from external decentralised platforms. By analyzing this data, we gain a broader perspective on the trends, patterns, and issues prevalent in the development of decentralized applications. This external data serves as a valuable foundation for our co-design process, enabling us to identify common challenges and emerging needs in the field. Additionally, we actively engage in virtual meetings with the maintainers of an external platform (InterPlanetary File System) to ensure alignment in design decisions and to discuss the potential role of TaRDIS within their frameworks.

In addition to the data analysis, we conduct targeted surveys to engage with a diverse set of programmers. These surveys provide us with direct feedback and insights from individuals actively involved in decentralized programming. By involving programmers from different backgrounds and expertise levels, we aim to capture a comprehensive understanding of the challenges they face, their preferences, and their suggestions for improving the development process.

We combine the insights described above with the specific challenges faced by the industrial partners within their use cases. This ensures that the TaRDIS toolbox effectively addresses their concrete requirements and provides practical solutions.

Overall, the co-design process implemented by TaRDIS follows a systematic and inclusive approach, enabling us to gather valuable inputs from both external sources and programmers directly involved in decentralized application development. This process ensures that the TaRDIS toolbox is designed to meet the real-world requirements and challenges of the decentralized landscape.

1.2 Results Summary

Our key contributions are:

- We explore the growth of decentralized applications, their challenges, and the diversity of programming languages, through data analysis and virtual meetings with developers of decentralised platforms operating at scale, in order to ensure that TaRDIS addresses the evolving needs of programming and empowers developers in this dynamic landscape.
- We present the findings of a questionnaire designed to gather insights from programming experts for the development of TaRDIS. The questionnaire covers participants' familiarity with decentralized programming techniques, programming language preferences, infrastructure deployment patterns, and expectations from a decentralized programming toolbox.
- We examine the use cases and highlight the challenges encountered in each scenario. We aim to effectively address them in the development of the TaRDIS toolbox.

Overall, we consolidate the gathered insights, requirements and use them as a basis to derive a comprehensive set of initial technical requirements, ensuring that the TaRDIS toolbox meets the diverse needs of the users. Also, we arrive at an initial description of the heterogeneous swarm, which is *"A distributed system whose nodes (differing in hardware and/or software) jointly execute some decentralised algorithm (including AI and ML, e.g. for orchestration and optimisation), via structured communication between a dynamic set of known parties that may join or leave the swarm application."*

1.3 Deliverable Structure

We begin the report by presenting the participatory process (Section 2) which encompasses an analysis of external data sources, survey responses from external developers, and insights from communities in partner organizations. We then give an overview of use cases and highlight the associated challenges (Section 3). Subsequently, we consolidate the gathered information to derive comprehensive requirements and outline potential future work (Section 4). Finally, in Section 5, we review related works in the field, exploring both the state-of-the-art and upcoming advancements in relation to these requirements. We provide the data management policy and the questionnaire used for the distributed programming survey in APPENDIX A.

Funded by the European Union

2 REQUIREMENTS THROUGH PARTICIPATORY DESIGN

We are witnessing a significant growth in the development of decentralised applications in recent years. This growth has been mainly driven by the applications' ability to eliminate the need for any intermediaries. Traditional applications often rely on centralized servers or authorities to manage data and transactions. In contrast, decentralised applications leverage decentralised networks, such as peer-to-peer networks or blockchain, where multiple participants contribute to the operation of the application, providing various benefits such as improved data privacy, censorship resistance, and user empowerment. However, this growth has also brought forth challenges related to heterogeneity, which refers to the diverse nature of devices, protocols, and technologies within decentralised networks.

As a part of co-designing the TaRDIS requirements, we discuss here the growth in decentralised applications outside the consortium and highlight diversity in the requirement of these applications. We then explore the methodology we followed to build our list of requirements, both by exploring the use cases in depth and through participatory design within the community of decentralized applications and distributed systems.

2.1 DECENTRALISED APPLICATIONS

2.1.1 Growth in decentralised file sharing: A case of IPFS

The InterPlanetary File System (IPFS) [1] [2] is part of this decentralization effort. IPFS is a storage layer for the Decentralized Web. It is an entirely decentralized content-addressable object storage and retrieval platform. IPFS is a community-driven, open-source project [3], which covers 200 git repositories with 67809 stars and 12407 forks. In total, there are 83.5K commits by 1352 code contributors, covering 400+ organizations including universities, start-ups and large corporations.

IPFS has seen widespread uptake with more than 1B web client accesses and more than 25k unique nodes participating in its peer-to-peer (P2P) network every day, with up to 200M requests handled in peak times (Figure 1). Critically, IPFS underpins various other Decentralized Web applications, including social networking and discussion platforms (Discussify, Matters News), data storage solutions (Space, Peergos, Temporal), content search (Almonit, Deece), messaging (Berty), content streaming (Audius, Watchit), and e-commerce (Ethlance, dClimate).

It is worth nothing from Figure 1 that the *(i)* IPFS gateways are popular and consistently serving the content requests arriving from most parts of the world *(ii)* At the same time, the number of peers in the network, which serves content, has an increasing adoption, meaning more nodes are joining the network. These figures point to the success of the IPFS. We collaborate with the key developers of the IPFS ecosystem to show that this decentralization comes at a cost [4].

Funded by the European Union

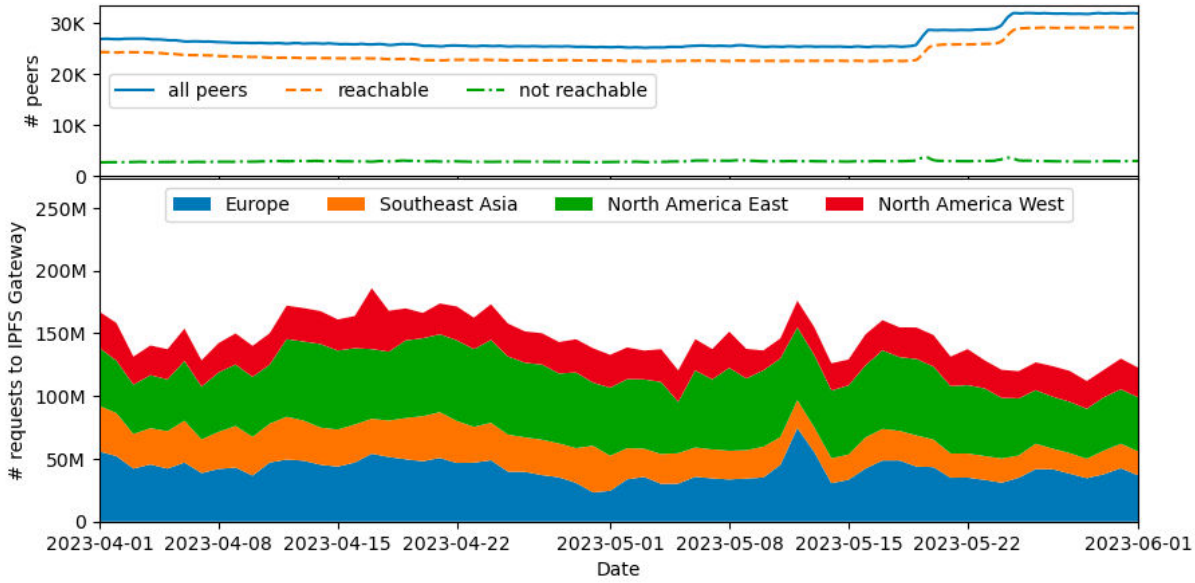

Figure 1: (top) Growth of IPFS peers in the Distributed Hash Table. (bottom) Daily number of requests to IPFS Gateways.

Decentralization increases complexity and overhead due to the need to coordinate many decentralized entities. Consequently, performance can be compromised, scalability can be challenging, and this can lead to a less performant system. Measuring, managing, and debugging such systems is also more challenging.

Thus, as a part of TaRDIS, we envision an ease-of-development with optimised data replication for IPFS-like decentralised platforms.

2.1.2 Interoperability in decentralised social networks: A case of Fediverse

As an alternative to Twitter and other centralized social networks, the Fediverse is growing in popularity. The recent, and polemical, takeover of Twitter by Elon Musk has exacerbated this trend. The Fediverse includes a growing number of decentralized social networks, such as Pleroma or Mastodon, that share the same subscription protocol (ActivityPub). Each of these decentralized social networks is composed of independent instances that are run by different administrators. Users, however, can interact with other users across the Fediverse regardless of the instance they are signed up to. The growing user base of the Fediverse creates key challenges for the administrators, who may experience a growing burden. We explore how large that overhead is, and whether there are solutions to alleviate the burden.

Mastodon is part of the wider Fediverse, in which any person can create and operate their own Mastodon server (aka *instance*). Each Mastodon instance operates as an independent microblogging service, where users can create local accounts and enjoy similar functions to Twitter (e.g. posting, following). Importantly, these instances can also federate together, allowing users on one instance to follow users on another. This means that Mastodon operates in a decentralized fashion (with people joining independent instances), while retaining the ability to interact across the entire globe. Moreover, the instances can be deployed in a low-resource setting.

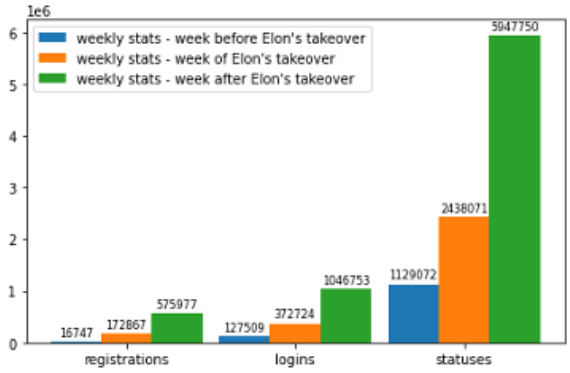

Figure 2: Weekly activity on Mastodon instances.

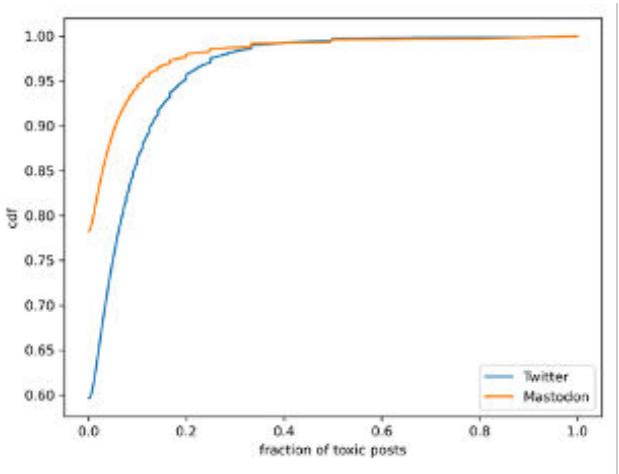

Figure 3: CDFs of fraction of each migrated user's toxic posts on Twitter and Mastodon.

This new paradigm has attracted significant attention (Figure 3) and has made it an obvious candidate for users who are unhappy with Musk's acquisition (and the associated centralization of power in the hands of one individual).

To understand the impact of this social synchrony event we perform a study to collect publicly available data from 136,009 unique twitter users who moved to 2,879 unique Mastodon instances [5]. Based on the migrated Twitter users, we find that such events have resulted in challenges in terms of platform scalability and content moderation. Scalability challenges arise when users move to the Fediverse (for instance, Figure 2 shows 5X growth in terms of number of user statuses after the migration) and users move across Fediverse instances. The latter happens when users migrate from one to another instance. Figure 4 shows the chord plot of switches from each user's first Mastodon instance to their second. A common pattern across these switches is that users move from general purpose/ flagship instances (e.g. mastodon.social, mastodon.online) to more topic specific instances, e.g. sigmoid.social (a Mastodon instance for people researching and working in Artificial Intelligence), historians.social (a Mastodon server for people interested in history) and infosec.exchange (a Mastodon instance for info/cyber security-minded people).

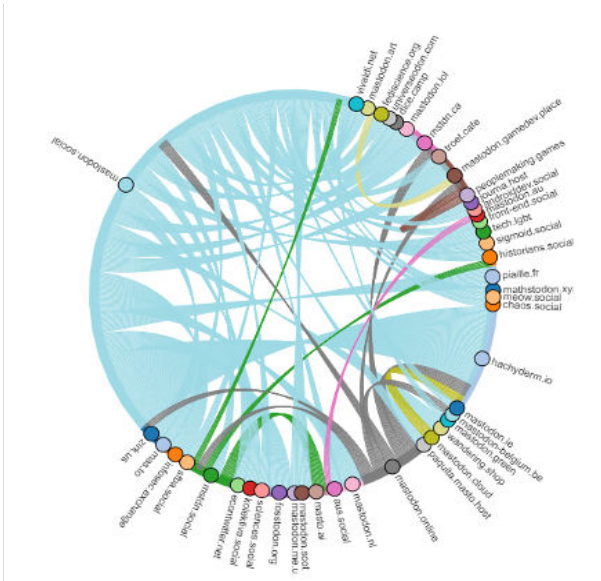

Figure 4: Chord plot showing user migration within the Mastodon instances.

As users migrate to the Fediverse, moderation challenges arise when they start posting toxic content. To understand the extent of this toxicity, we label all tweets and statuses using Google Jigsaw's Perspective API. For a given post, Perspective returns a score between 0 and 1 for its toxicity (0 = non-toxic). Specifically, we use the API's toxicity attribute that defines toxicity as a rude, disrespectful, or unreasonable comment that is likely to make people leave a discussion. Note that, the tool is centralized and may not work in a federated setting as each instance can have their own policy and definition for toxicity. However, we begin with this metric as a baseline to compare with other centralized platforms. Figure 5 shows the CDFs of the percentage of each migrated user's toxic posts on Twitter and Mastodon. Overall, 5.49% of Twitter tweets are toxic. In contrast to media speculation, Mastodon is substantially less toxic, with 2.80%. On average, each user posts 4.02% toxic tweets on Twitter vs. just 2.07% toxic statuses on Mastodon. Even though the discourse is non-toxic over both platforms, we notice that 14.26% of migrated users post at least one toxic post on both the platforms. While this may not be problematic for Twitter which has its own moderation team, it might present challenges for Mastodon, where volunteer administrators are responsible for content moderation. Overall, our study points to the requirement for federated learning for the scalability and moderation aspects under a low-resource setting of instances in the Fediverse. We point the interested readers to related works from the TaRDIS consortium [5–7] on management-related challenges of Mastodon-like decentralised platforms,

Thus, as a part of TaRDIS, we envision an ease-of-development of features like scaling management, help with development of effective moderation suggestions mechanism, as well as similar emerging services, with distributed machine/deep learning for Fediverse-like decentralised applications.

2.1.3 Diversity in Programming Languages: A case of Stack Overflow

Every year, Stack Overflow, a widely-used question-and-answer website for programmers, conducts a comprehensive survey within the programmer community [8]. This survey attracts a substantial number of responses from developers of diverse backgrounds and levels of

experience. For instance, in May 2022, approximately 70,000 developers participated, providing valuable insights into the tools, technologies, and areas of interest in the programming field. To support TaRDIS, we utilize this survey data to examine the progression of programming languages and ascertain the programming abstractions relevant to TaRDIS.

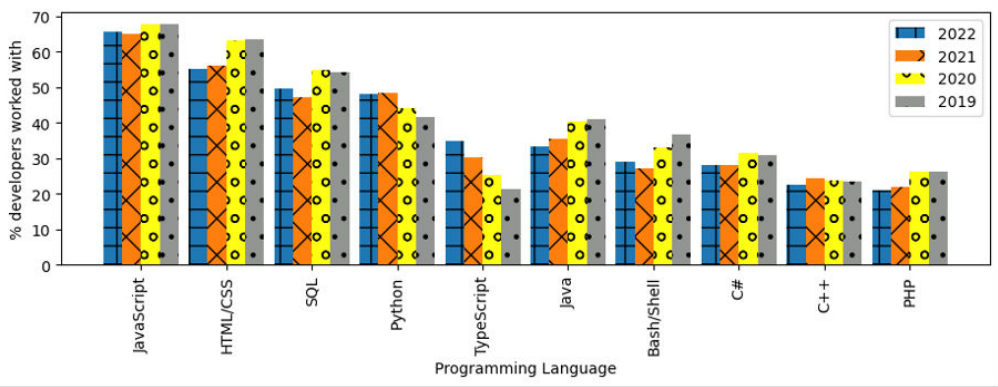

Figure 5: Trends in programming languages used by developers between 2019-2022.

Figure 5 shows the percentage of developers who have worked with different programming languages over the years. It is evident that TypeScript and Python have gained increasing adoption among developers while Java usage has declined. Another noteworthy aspect is that while TypeScript and JavaScript share a library ecosystem only TypeScript enjoys growth, hinting at an appetite for more static verification of programs. Additionally, the 2022 survey highlights that 32% of respondents express a favorable attitude towards decentralization [9].

Thus, as a part of TaRDIS, we envision an ease-of-development by developing programming abstractions for different languages.

2.2 Questionnaire Preparation and Distribution

In addition to studying the extent of different decentralised programming applications and paradigms, we aim to reach the external participants and communities outside of the core contributors within the consortium.

We design a survey questionnaire to capture developers' perspectives, preferences, and suggestions regarding various aspects related to distributed programming. This includes but is not limited to their opinions on existing techniques, challenges they have encountered, emerging trends, and potential areas for improvement or future research. The questionnaire seeks to tap into the collective knowledge and experiences of these experts to gain a comprehensive understanding of the current state of programming and identify potential directions for TaRDIS.

With the objectives of the TaRDIS project in mind, the questionnaire is designed to collect crucial information by addressing the following key questions (the questions are presented in APPENDIX A):

- Preference in terms of Programming Language: aims to understand programmers' preferences and provides insights into which languages are commonly used or preferred in the community.
- Familiarity in decentralized programming techniques: helps gauge their knowledge and experience in developing applications that operate in a decentralized or distributed manner.
- Familiarity in peer-to-peer systems: explores their understanding of the principles, architectures, and challenges associated with decentralized peer-to-peer networks.
- Familiarity with blockchain technology: helps determine their understanding of distributed ledgers, smart contracts, consensus mechanisms, and other key aspects related to blockchain-based systems.
- Web3 architectures and existing Web3 services: delves into the programmers' knowledge of Web3 architectures, which encompass decentralized applications, decentralized data storage, and other Web3-specific technologies
- Distributed machine learning and deep learning: helps assess their knowledge of distributed training, parameter synchronization, and other aspects related to scaling machine learning models across multiple nodes.
- Distributed storage: explores their familiarity with distributed file systems, object storage, and data replication techniques used in distributed environments.
- Details of networking protocols: seeks to capture programmers' knowledge and expertise in networking protocols including protocols such as TCP/IP, UDP, HTTP, and others commonly used for communication and coordination in distributed environments.

Additionally, the questionnaire captures the programmers' experience (in terms of number of years) and suggestions for building a decentralized programming toolbox. This allows them to share their practical insights, challenges faced, and recommendations for developing tools, libraries, or frameworks that facilitate decentralized application development.

2.3 IDENTIFYING PARTICIPANTS

To ensure a diverse range of responses, the survey is not only limited to the consortium members but is also shared with a broader audience through the Prolific platform [10]. This allows for input from experts outside of the immediate project consortium, bringing in fresh perspectives and insights from a wider pool of programming specialists.

2.3.1 Communities in the partners' organisations

We first identify the various groups or networks within the partners that are involved in the industrial partners of the TaRDIS project. Within these institutions, there exist diverse communities that are impacted by the challenges being addressed by TaRDIS. These communities can vary depending on the nature of the institution and the specific domains or sectors involved in the project. These communities include researchers, scientists, engineers, developers, practitioners, and domain experts play a crucial role in adopting the results from the project's developments and solutions. Further, they may belong to different departments, research groups, or professional networks within the institution, bringing their unique perspectives, skills, and experiences to the project.

The involvement of these communities is crucial for several reasons. Firstly, their domain expertise and knowledge contribute to defining relevant use cases that align with the institution's needs and priorities. They provide valuable insights into the challenges, requirements, and potential solutions within their respective domains.

Secondly, these communities participate in the implementation and deployment of the various solutions within their institution's infrastructure or systems. They provide feedback, test the technologies, and evaluate their effectiveness in solving the identified challenges. Their practical experiences and observations help refine the solutions and ensure their applicability and usefulness in real-world settings.

Thirdly, these communities, indirectly, act as ambassadors and disseminators of the project's outcomes within their institutions and broader networks. They share their experiences, lessons learned, and best practices, fostering knowledge exchange and promoting the adoption of distributed programming advancements enabled by TaRDIS.

In summary, we identify different groups in an institution as vital stakeholders of the TaRDIS project. Their involvement, expertise, and collaboration contribute to the successful evaluation, and dissemination of the project's solutions, ultimately driving advancements in distributed programming within their respective domains. Thus we target them as the participants to circulate the questionnaire.

2.3.2 In-the-wild Requirements

We proceed to collect real-world requirements and challenges faced by programmers in their day-to-day work, which are gathered through screening on the Prolific platform [10]. In the context of the TaRDIS project and the questionnaire, this process involves specifically targeting programmers to understand their needs and gather valuable insights that can inform the development of the project's solutions.

When screening for programmers on the Prolific platform, the goal is to identify individuals who have firsthand experience and expertise in programming, particularly in the context of programming languages and related areas. These programmers may come from various backgrounds, such as software development, systems engineering, or related technical fields. By reaching out to programmers, the questionnaire aims to capture their perspectives, preferences, and challenges encountered when working with distributed systems. This includes understanding the specific requirements they face when developing distributed applications, managing networked environments, or addressing scalability and performance issues.

The screening process helps ensure that the survey responses come from individuals who have practical experience and insights relevant to the TaRDIS project's objectives. By targeting programmers on the Prolific platform, the questionnaire seeks to tap into their knowledge and gather valuable inputs on the current state of distributed programming, emerging trends, and potential areas for improvement.

By collecting requirements *in-the-wild* from programmers, the TaRDIS project can gain a better understanding of the practical challenges faced by professionals in the field. This information is invaluable for shaping the project's strategies, identifying priorities, and developing solutions that directly address the needs of programmers working with distributed systems. The insights obtained from the screening process and subsequent questionnaire responses can help guide the design and development of tools, frameworks, or methodologies within TaRDIS. The goal is to create solutions that are practical, effective, and aligned with the requirements and preferences of programmers in real-world scenarios.

Overall, the screening for programmers on the Prolific platform ensures that the TaRDIS project captures the voices of those who are potential stakeholders of TaRDIS , allowing their perspectives and requirements to shape the development of innovative solutions through defining the requirements of the next generation distributed programming paradigm.

2.4 Co-Design Results

The survey received a total of 146 responses, with participants divided into two groups: external participants, representing 58.9% of the total, and internal participants from partner organizations within the TaRDIS project. This distribution demonstrates a balanced participation between the two groups.

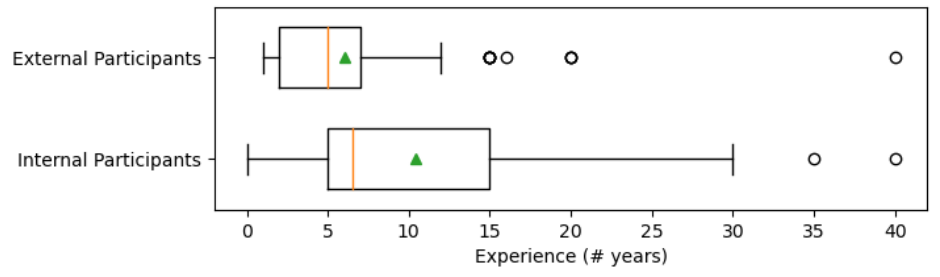

Figure 6: Programming experience of the developers participated in the survey.

To assess the experience of the programmers, Figure 6 captures the experience levels of participants from the developer community in TaRDIS partner organizations (referred to as internal participants) as well as external participants from the Prolific platform (Section 2.3). The average years of experience for external participants is 6.3 years, while internal ones have an average of 11 years of experience. This indicates that the internal participants possess relatively higher levels of experience compared to the external participants, contributing to a diverse range of insights captured by the survey.

We proceed to discuss the preferences of the participant groups regarding programming languages (Figure 7 and Figure 8), which is essential for identifying the appropriate API interfaces that TaRDIS should offer. In the survey, Python emerged as the favoured choice among both communities, with over half of the participants selecting it as their preferred language. Python is a high-level, interpreted programming language known for its simplicity and ease of use. Following Python, other popular languages among the participants include Java, Kotlin, C/C++, and Typescript. This pattern aligns with the findings of the latest Octoverse

Report [11], which highlights the most popular programming languages used across GitHub. According to the report, Python experienced significant growth, with adoption increasing by more than 22% in 2022. The high preference for Python in the survey findings indicates its widespread usage and popularity among programmers, emphasizing its importance as a supported language within TaRDIS. By providing robust support for Python, TaRDIS can ensure compatibility and ease of integration for a significant portion of its user base. Additionally, supporting other commonly preferred languages like Java, Kotlin, C/C++, and Typescript allows TaRDIS to cater to a broader range of developers and maximize its potential impact.

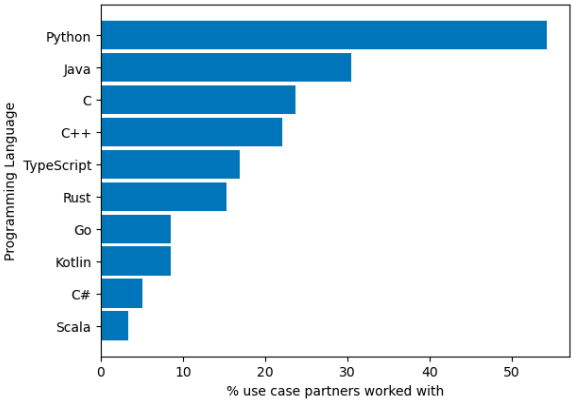

Figure 7: Programming languages used by communities in the partners' institutions.

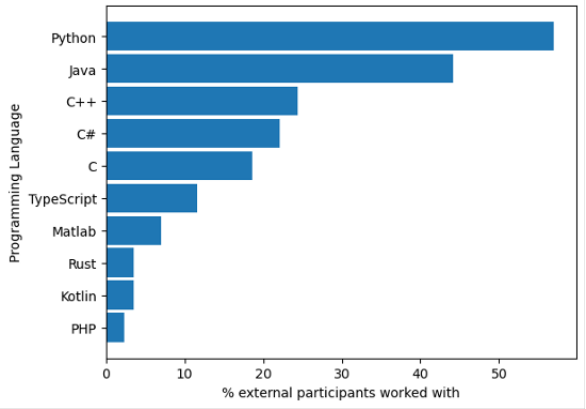

Figure 8: Programming languages used by the external participants.

Thus, through these insights, TaRDIS can effectively tailor its API interfaces and development tools. This strategic alignment contributes to the overall usability and adoption of the TaRDIS framework within the programming community.

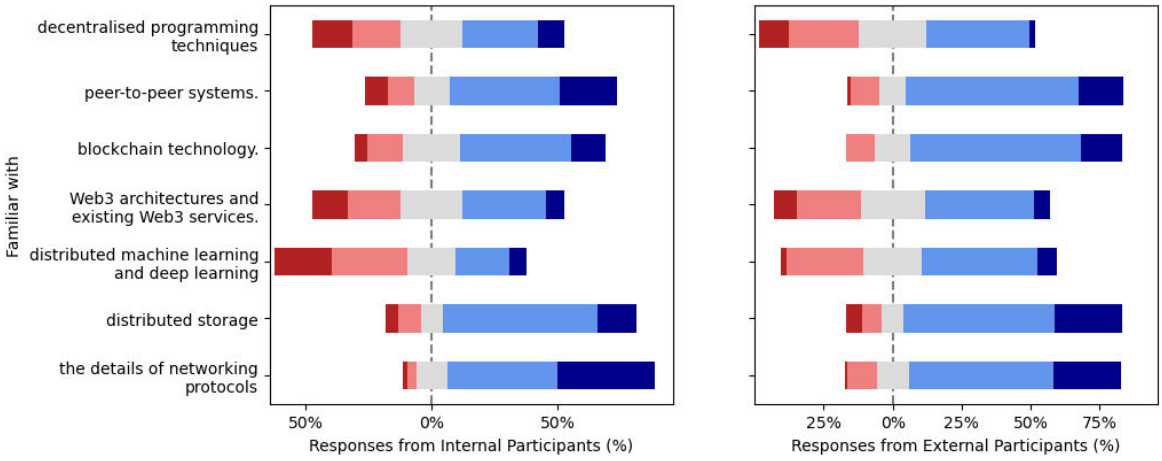

Figure 9: Internal (left) and external (right) participants familiarity with traditional and upcoming programming paradigms.

Next, we examine the participants' familiarity with various aspects such as decentralized programming, distributed machine learning, and communications. To measure this, we utilized a Likert rating scale, which ranged from "Strongly Disagree" to "Strongly Agree" on a 5-point scale. This scale allowed us to capture participants' responses with a greater degree of nuance. The results for both sets of participants, internal and external, are illustrated in Figure

9 (left) and Figure 9 (right) respectively. Overall, the findings indicate that a significant percentage of participants, ranging from 75% to 80%, possess at least a level of familiarity with traditional paradigms like networking protocols and cloud/distributed storage. This suggests that a majority of the participants have a foundational understanding of these concepts.

However, the results reveal that only approximately half of the participants are familiar with decentralized programming techniques, and a similar proportion of participants are familiar with distributed machine learning. This indicates that there is room for improvement and education in these specific areas. It is worth noting that not all participants possess expertise in every technique or technology covered in the survey. The distribution of familiarity levels among participants highlights the diverse skill sets and knowledge within the surveyed group. It suggests that there may be opportunities for TaRDIS to make the development process easier for programmers with different levels of familiarity with various techniques and technologies.

By considering these insights, *TaRDIS aims to effectively address the specific needs and skill levels of the programmers developing decentralised applications*, ultimately fostering a community of practitioners in the field of distributed programming.

Additionally, our analysis reveals that over 60% of participants have deployed Server/VM or PC as part of their infrastructure. This indicates that these traditional computing platforms remain prevalent and widely adopted among the surveyed participants. Furthermore, a significant portion of the participants, 20.5% and 7.5% respectively, have deployed solutions involving Raspberry Pi/IoT devices and embedded controllers. This diversity in deployed devices highlights the heterogeneity within the participant group in terms of the hardware and platforms they utilize for their projects. It suggests that the participants are exploring and incorporating a range of technologies and form factors to meet their specific requirements.

In terms of development approaches, an encouraging 71.8% of the respondents expressed a positive sentiment towards community-oriented development. This is reflected in their agreement or strong agreement with this development approach. Community-oriented development emphasizes collaboration, knowledge sharing, and collective problem-solving within a community of developers. The high percentage of participants who embrace this approach demonstrates a willingness to actively engage with and contribute to a larger development community, fostering an environment of cooperation and innovation.
These findings provide valuable insights into the infrastructure choices and development perspectives of the surveyed participants. In turn, this can *enhance the effectiveness and adoption of TaRDIS technologies within the broader programming community*.

Finally, we ask the participants about their expectations from TaRDIS-like decentralized programming toolbox. The primary expectation from the responses was unanimous across both participant groups, which is Usability (shared by 25.34% of the participants). This emphasized that the toolbox should be easy to use and possess a simple interface, ensuring a smooth and accessible user experience. The word cloud formed by these expectations are shown in Figure 10 and Figure 11.

Figure 10: Word cloud formed by the internal participants' expectation for a TaRDIS-like toolbox to deliver.

Figure 11: Word cloud formed by the external participants' expectation for a TaRDIS-like toolbox to deliver.

However, when we examined the preferences of the developer communities within partner institutions and external participants, some variations emerged. The participants from partner institutions prioritized scalability as a crucial expectation. This aligns with their inherent requirements, as these institutions often operate at larger scales and need solutions capable of handling increased workloads and growing demands. On the other hand, the external participants emphasized the importance of security, recognizing the need for robust protection of their data and systems. It is worth noting that external participants also valued scalability, indicating a desire for solutions that offer both security and scalability features.

In addition to usability, scalability, and security, there were other common requirements expressed by the participants. These included the need for modularity, ensuring that the

toolbox offers flexibility and extensibility to adapt to different use cases and programming paradigms. Participants also emphasized the importance of correctness, expecting the toolbox to provide reliable and bug-free functionality. Additionally, high performance was highlighted as a crucial requirement, with participants seeking efficient and optimized solutions that can deliver fast and responsive performance.

We also asked participants about preferred IDE or code editor that they regularly use in their projects. There are slight differences (**Error! Reference source not found.**) in responses from i nternal and external participants, but in both groups VS Code, Visual Studio, IntelliJ, and Eclipse are in top five, and PyCharm close contender. This information can give strong indication about integration needs for future TaRDIS toolbox.

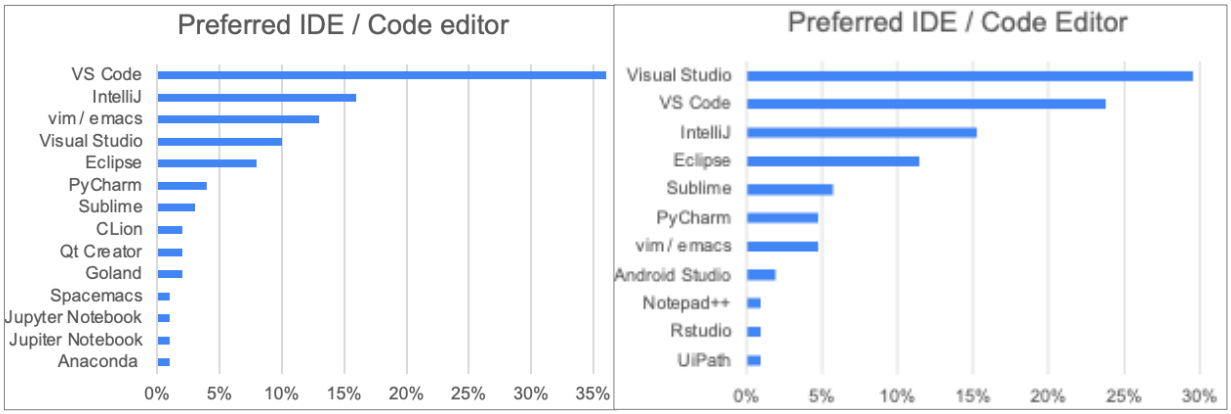

Figure 12: User preference for IDE / code editor for internal (left) and external (right) participants.

By understanding these shared expectations, we plan to shape the TaRDIS's development efforts to address the developers' needs effectively. Designing a user-friendly and intuitive interface, incorporating scalable and secure features, ensuring modularity and correctness, and optimizing performance will contribute to the creation of a comprehensive and robust decentralized programming toolbox that meets the diverse requirements of the programming community.

3 USE CASES AND REQUIREMENTS

In this section we provide an overview of the use cases and highlight the associated challenge during the implementation.

3.1 MULTI-LEVEL GRID BALANCING

3.1.1 General Introduction

Energy grids exist to connect energy generators to consumers. Balancing generation and consumption at all times is crucial for its correct (physical) operation. Nowadays, more and more energy consumers are also generating energy for its own consumption or to sell either to the grid operator or to neighbours (peers). This decentralized topology of generation and consumption of energy requires a different control approach to keep a stable energy balance in the grid. In fact, one needs to shift from the actual centralized control to a decentralized one, where assets agree in exchanging energy and grid balance is assured in this way -swarm intelligence for matching peers with energy surplus and needs.

3.1.2 Challenges

Nowadays, the long term energy grid balance is assured by requesting producers to cope with forecasted hourly consumption, one day ahead, while, when the right time comes, short term management triggers resources that can compensate for deviations in both production and consumption.

- Moving from centralized control to distributed control requires coordination between swarms of energy management devices. It is easier to handle it in groups geographically clustered. But an orchestrator is needed to forecast if the group will be self-sufficient or not within the next time range. The aim is to have the same algorithm for the energy management devices and the group orchestrator.
- Forecasting needs, broadcasting them in an atomic way and finalizing peer-to-peer (P2P) agreements in real time and in an energy efficient manner can make the business model feasible and thus attractive.
- Several parameters are involved in the process. But for the sake of efficiency only business (market) and functional parameters are described right after. Components, communication and data configurations are expected to adjust to the previous.
- Putting this new model in place will empower Energy communities and provide a smooth grid control to grid operators. The use case KPIs must be used to evaluate impact on scale-up i.e. business modelling.

TaRDIS is providing a very advanced programming model that perfectly fits into the requirements of a future decentralized control for the energy grid. Taking as reference the Smart-Grid Architectural Model (SGAM) [91] used in the sector for modelling any business development and represented below, EDP will describe a set of high-level requirements for the two higher layers represented in the SGAM (Business and Functional layers). These requirements derive directly from the use case description as a pathway to address the above-mentioned challenges. The considered domains are Distribution, DER (Distributed Energy Resources) and Customer Premises with all Zones included.

3.2 Privacy Preserving Learning Through Decentralised Training In Smart Homes

3.2.1 General Introduction

Smart homes are becoming increasingly popular as the technology continues to advance. They consist of a wide range of devices that are designed to work together as a swarm to make our lives more convenient and comfortable. Many of the devices are built to incorporate artificial intelligence algorithms to improve their functionality. However, the heterogeneity of these devices makes it difficult for the devices to share the intelligence without imparting the data with each other or to a central location, raising concerns about privacy. Further, with so many devices collecting data about us, there is a risk that our personal information may be compromised. The use-case aims to develop a privacy-preserving federated learning framework that can work in a hierarchical fashion. To build this framework, the TaRDIS toolbox will be used to abstract the infrastructure, data distribution and learning algorithms from the developer.

3.2.2 Challenges

With TaRDIS, we aim to build a practical Hierarchical Federated Learning framework for mobile environments that enables cross-device and cross-app (i.e., on-device cross silo) Federated Learning, and as-a-service to address the following challenges:

- There is a lack of libraries to support third-party app developers to train and federate models in the background. Therefore, developers either train models while users are active on their apps, directly impacting user experience, or rely on the OS scheduler, which has full control in orchestrating such heavy compute tasks in the background. To address this challenge, the use case includes a client-side middleware and library enabling third-party mobile apps to work with such OS constraints, and train FL models on-device.
- There is no support for cross-app local sharing of data or models for collaborative Machine Learning (ML) training: it requires secure and privacy preserving mechanisms for storage, communication, and permissions management across apps. With TaRDIS, we plan to build an extensible library to be used by calling a set of secure and privacy-preserving communication and storage primitives and APIs. In particular, a secure communication channel is established between apps embedding the library supporting different types of data, i.e., numerical, text and binary.
- Existing Federated Learning methods do not provide an easy to use way for app developers based on simple APIs and tools as ML-as-a-Service counterparts (eg., Google Cloud Platform, Amazon Web Services). With TaRDIS, we aim to build this interface that can communicate with diverse ML platforms.
- There is a lack of definitions to construct a hierarchy involving infrastructure and Machine Learning. We aim to overcome this with TaRDIS, which includes making modifications to the standard approaches to Federated Learning to ensure that the newly proposed Hierarchical Federated Learning approach learns the same model as in the status quo involves formalizing an algorithm (to provide Differential Privacy) to be used in an such ecosystem.

3.3 Distributed Navigation Concepts For LEO Constellations

3.3.1 General introduction

One emerging trend in the Space Industry is the provision/commercialization of products and services based on current and new swarm satellite networks. In this sense, there is a growing interest in Low Earth Orbit (LEO) constellations (i.e., swarms) of satellites. LEO communications and PNT (Position, Navigation and Timing) services are emerging (Starlink, Iridium, OneWeb). These are examples of “New Space” systems, LEO systems with tens, hundreds, or even thousands of mini satellites mainly dedicated for communication and the internet. Providing accurate position and timing services is a key factor for these services.

Orbit Determination and Time Synchronization is the process of estimating satellite position, velocity and clock parameters. ODTS is typically performed on-ground for GNSS constellation of satellites. It is performed by means of batch least squares technique in a centralized way.

The GMV use case consists of performing distributed autonomous, on-board, and real-time orbit determination and time synchronization for a large constellation of satellites in LEO.

- **Distributed** means that the processing is not performed on a single satellite but every satellite computes its ODTS solution and shares it with its surrounding visible satellites.
- **Autonomous** means that it is performed with limited ground stations support.
- **On-board** means that processing is performed on-board the satellites and not on ground.
- **Realtime** means that the ODTS solution is provided with a certain frequency (i.e. 1 Hz) and is each time related to the current epoch.

The most promising technology enabling the achievement of this use case is represented by the Inter-Satellite-Link (ISL) communication and ranging capability. This is crucial to share data between the satellites (swarm nodes).

Examples of on-board autonomous ODTS using ISL are the GPS AutoNav system which is present on the block II-R GPS satellites [12] and the third generation BeiDou navigation satellite system (BDS-3) [13]. These autonomous navigation concepts are not yet fully operational. Galileo 2nd Generation of satellites (G2G) is also planned to be equipped with ISL systems.

The initial approach towards the development of distributed ODTS is the centralized ODTS first. This is being developed as baseline. Both batch least squares and Extended Kalman filtering techniques are considered for this implementation.

3.3.2 Challenges

The biggest challenge for the ODTS use case is to reduce dependency on ground stations support. Many ISL connectivity schemes and ODTS algorithms can be explored to achieve the optimal solution. Especially with a large constellation of satellites there are many possible combinations of inter-satellite connections at each time and each combination corresponds to a specific geometry. Based on the observations geometry the measurements can have

different impacts on the performance of the navigation filter, leading to different solution accuracies. TaRDIS is intended to provide support for the above mentioned points through Machine Learning libraries and help with modeling the necessary interactions between the satellites in the distributed approach. The TaRDIS toolbox shall facilitate the design of distributed ODTS for a swarm of satellites at simulation level.

3.4 Highly Resilient Factory Shop Floor Digitalization

3.4.1 General Introduction

Actyx employs heterogeneous swarm systems to orchestrate manufacturing processes in factories. The swarm members in this case are workstations (where the actual processing takes place, e.g. by drilling/milling/turning steel parts, or assembling them), warehouses, logistics (both human workers and autonomous vehicles/robots), connectors to resource planning and business intelligence systems, and human overseers—humans use dashboards or tablet computers to interact with the swarm. Our system is used to automate the allocation and correct execution of tasks on the shop floor, depending on the low-level real-time systems to handle the details of for example robot movements. The next higher level in the management hierarchy is the ERP or MES software system, from which production orders are ingested and to which progress is reported.

In the concrete use case currently being implemented, the system controls a fleet of AGVs for transporting trays loaded with tools or parts from a commissioning area to workstations along the production lines of a plant producing machining centers. These bulky and heavy machines are moved forward to the next workstation at the end of each day, so that the next production step can be performed on the following day; this daily motion is performed using so-called *flying carpets* under human oversight and also registered in the swarm.

The main job of the system is thus to automate all intralogistics processes. When a tool or workpiece is needed at a location, a request is made and matched with either an outstanding pick-up request or a warehouse availability to form a transport order. This process is done collaboratively, without central coordination, between the AGVs and logistics workers. When a transport order is assigned, the designated worker (human or robot) will perform it and eventually signal completion. At this point, the production processes can resume; this process repeats until a product is finished and gets moved into the shipping area.

3.4.2 Challenges

The system described above needs to model the dynamic evolution of several kinds of entities (like production orders, transport requests & orders, workpieces & their carriers, tools, …) with proper restrictions on which participants can perform a given operation on them and under what preconditions. The complex task of factory intralogistics needs to be broken down into smaller pieces that the factory IT team can understand, maintain, and evolve. Furthermore, the expertise on how to best solve the logistics problem resides with individuals who are not programmers, so it is crucial that their diagrammatic process specifications are faithfully

implemented in code—most desirable by comparing the code structure to the diagrams in as direct a fashion as possible.

Once the general structure of communication and participant behavior is implemented and verified, it is imperative to validate the detailed application state computations that are too fine-grained to be represented in the overall diagrammatic view, using an appropriate environment for unit testing and deterministic process execution (i.e. an infrastructure mock).

The Actyx middleware currently only supports simplistic models for swarm membership, data placement, and task allocation. To achieve the use case goals we will need to refine the membership service with estimates or external information on whether a given device is currently disconnected, switched off, or decommissioned. The data placement will have to move away from storing all events on all nodes forever towards partial replication of events to devices on which agents will likely need to access the events in the future plus the ability to remove older events from most devices and retain them on designated archival nodes; these placement decisions significantly impact application performance and system resilience. Furthermore, while many Actyx tasks are bound to a specific edge device (like the AGV interface needs to run on the PC mounted on the AGV) there are others (like management or janitorial tasks that notice when orders take too long or requests are never picked up) that can run wherever enough CPU and storage is available. Since both data and task placement must operate opportunistically on partial views of the swarm system we foresee that machine learning is the prime candidate for tackling this problem.

Finally, large-scale manufacturing systems can in general not be understood holistically due to their overwhelming complexity. It is therefore required that we aid the production experts using TaRDIS-based software in recognising and diagnosing issues at the level of production workflows so that they can then fix them, improving energy efficiency and reducing waste of the industrial processes. Our primary tool in this regard will be anomaly detection in the execution of processes within the swarm by means of machine learning, both with the help of manual classification of problematic cases as well as labels derived from heuristics that are formulated by manufacturing experts.

Thus, in this section we gather preliminary set of challenges and requirements by the use cases. We proceed to consolidate a list of technical requirements that emerge directly from the use cases. This list also complements with the requirements provided by the questionnaire in Section 2.

Funded by the European Union

4 PRELIMINARY REQUIREMENTS

Based on the goals and challenges as presented above for each of the use cases we distill a preliminary set of requirements towards the TaRDIS toolbox. We assume that the toolbox will be used to implement large parts of the described systems and consequently translate the needs of these systems into high-level properties our toolbox will need to have. The properties listed below do not yet cover all aspects raised above, in particular high-level requirements like regulatory conformance will need to be translated to low-level requirements in the coming months; we expect the outcome of such activities to have very large overlap with the requirements named below.

In the following we use different terms to focus on aspects of swarm participant: an *agent* is a program that performs tasks in the context of the swarm (possibly on behalf of a human through a UI), a *device* is the hardware which provides runtime resources, a *peer* communicates with peers on other devices and may comprise or host multiple agents.

4.1 DESIGN REQUIREMENTS

Diagrammatic: Interaction between programmers, project managers, and domain experts is supported by a visual representation with stringent syntax and semantics, to provide a reliable link for tight collaboration between these stakeholders and support formal analyses.

System boundary: Agents perform tasks both inside and outside the TaRDIS scope, which is supported by annotating process steps with external dependencies that are themselves not modeled with TaRDIS tools (e.g., an HTTP endpoint provided by an edge or cloud service).

Available: The systems we design need to be dependable on a local scale, resilient against failures occurring elsewhere, we aim for perfect local availability of the services we provide. We are aware that this implies the possibility of conflicts and inconsistencies.

Eventual consensus: The conflicts and inconsistencies that may arise need to be recognised and reconciled, eventually leading to a system-wide agreement on all choices taken.

Opt-in strong consensus: Most interactions in the use cases can be implemented with eventual consensus, but it is not yet clear that this is true for all of them, hence we foresee that the process designer can designate process steps that require strong consensus (with details on that feature remaining to be defined).

Coordination: Not all actions are always permitted—we require prerequisites to be expressed including “X has happened” or “X has not happened”. As per *eventual consensus* these prerequisites may be temporarily violated while communication is impossible.

Modularity: Each collaborative task between agents (a.k.a. protocol session) is isolated from other tasks, interacting with the rest of the swarm system through the local inputs and outputs exchanged with the local agent. This provides encapsulation and an external interface.

Funded by the European Union

Composition: Systems are built up by composing tasks, using results from one as inputs to another.

Replication: Agents perform their tasks according to some designated role, where a role may be played concurrently by multiple agents on different devices. This supports dynamic aspects like adding participants to a power grid or machines to a factory without having to redesign the system.

4.2 Analysis Requirements

Deadlock-freedom: As long as each role is represented in the swarm, the task can be completed through permitted agent actions; we may want to add failure as a permitted outcome.

Causality: An event recorded by a local agent will be recognisable to other agents as having happened after the events that the emitting agent saw before the emission.

Eventual consensus: For each choice made in the swarm there exists a point in time after which all peers will have the same local view on its outcome; this point in time is not locally observable and before it the local views may conflict.

Information leakage: We ascertain that classified information isn't revealed to peers that have insufficient access rights.

4.3 Algorithm Requirements

Autonomy: Local agents can act independently, without requiring coordination (i.e., network communication) with other peers, based on their local information.

Peer-to-peer: Local agents can coordinate with other peers by direct means, without requiring support or permission from central services or other peers.

Task dynamicity: Agents can join and leave tasks at any time or for any reason. Note that all roles in the task still need to be represented to guarantee task completion; this requirement focuses on the perspective of the individual agent instead, assuming that sufficient other agents remain with the task.

Authenticity: An algorithm may require certain events to be strongly authenticated and react differently if this validation fails at runtime.

Authorization: An algorithm may require certain events to be not visible outside a declared set of peers or agents—the precise requirement will need to be refined as the TaRDIS toolbox matures.

Hierarchical privacy: Agents employ federated learning with differential privacy in a setting where model weights are aggregated and thus refined in a hierarchical fashion, preserving the privacy of training data.

Funded by the European Union

Dissemination: Information can be disseminated such that it is guaranteed that a given set of agents will eventually receive it.

4.4 Verification and Validation Requirements

Protocol conformance: The implemented endpoint code is verified to conform to its declared agent behaviour. This verification will likely be impractical when considering the complete feature set of a host language like TypeScript or Python; the required scope instead extends only to the correct usage of TaRDIS APIs (cf. *system boundary* above).

Behavioural verification: The modeled behaviour—matching the scope of the TaRDIS APIs—is verified for correctness using exhaustive static reasoning as far as possible, giving complete and concise feedback to the programmer regarding the discovered problems. This verification happens at the granularity and conceptual level of the *diagrammatic representation* as described above, to allow domain experts to weigh in on how to resolve issues.

Test-based validation: All logic not covered by behavioural verification need to be validated through unit and integration testing. This must be supported using suitable TaRDIS tools in conjunction with popular testing frameworks for each target language.

Bounded complexity: The execution of federated learning tasks, in particular on constrained devices like satellites, industrial machine controllers, but also mobile phones, needs to stay within complexity bounds regarding both time and storage space. These bounds can be somewhat fuzzy for a mobile phone but are strict for satellites or real-time controllers.

Swarm topology sufficiency: In order to achieve some of the algorithmic requirements and provide a system matching the assumptions underlying the analysis requirements, we need to ascertain that a designated swarm deployment will have sufficiently dense communication topology to accommodate the necessary information propagation. This is particularly pertinent in a satellite constellation but also affects the planning and validation of the communication links in the other use cases.

4.5 Runtime Requirements

ML data placement: In order to support desired event emission rates the ML inference process deciding the replication strategy will need to be very light-weight (milliseconds at most).

ML health classification: High quality swarm membership service benefits from an ML model classifying each known peer as connected, disconnected, off, or decommissioned.

ML anomaly detection: As tasks are repeatedly executed by sets of agents, the system learns the typical decision flows and timings taken for each one, which is then used to detect outliers or unintended changes. This information is intended for consumption by the domain expert operating the swarm system.

Large swarm: Agents may at times have to communicate with large numbers of peers (at least hundreds). This is one reason for designing the system using roles and allowing each role to be implemented by many agents.

Swarm dynamicity: Peers can join and leave the swarm at any time. While joining can be regulated through a mandatory procedure, leaving can occur spontaneously, e.g. when a device is destroyed or stolen.

Mobile OS friendly: Applications on mobile operating systems can typically only control their task scheduling while the app is in the foreground, but swarm systems need to operate also in the background. This suggests moving the peer behaviour into a system service that is permitted to operate while another app is in the foreground.

Local information sharing: Agents on the same device may need to securely and efficiently share data, which is not supported between apps, e.g. on mobile phones. This also suggests a background service as above.

Communication baseline: Most interactions require broadcast or multicast communication patterns while at the same time demanding reliability and auditability. This implies that the basic communication mechanism should be durable by default.

Ephemeral communication: Some interactions become irrelevant some time after they have concluded (e.g., low-level machine-to-machine handshakes). Keeping resource usage within practical bounds requires the ability to declare lifetime bounds for event types.

Funded by the European Union

5 TOWARDS BUILDING THE TOOLBOX

5.1 LOOKING BEYOND THE STATE-OF-THE-ART

5.1.1 Machine Learning

In principle, Artificial Intelligence and Machine Learning (AI/ML) algorithms aim to provide alerts, forecasts, smart recommendations on decision making problems or even detect anomalies for predictive maintenance purposes. Specifically, the functionality of an AI/ML model tries to find a relationship between input data/parameters and output (dependent) variables in order to achieve a specific goal. There are three main categories of AI/ML algorithms based on the method used to estimate this relationship, namely Supervised Learning (SL), Unsupervised Learning (USL) and Reinforcement Learning (RL). Approaches that are based on SL and USL methods share a common feature: they depend entirely on historical samples that link the input and the output data. To this end, these methods assume the existence of a huge amount of historical dataset that can be used for identifying the mathematical connection between input and output. It should be noted that both the inputs and the outputs can either be numerical (for instance the power consumption in Watts of an industrial unit) or categorical (for instance the blood type of a person can be described in 4 distinct classes) values. Depending on the nature of the output variable (i.e. the variable that is going to be predicted by the AI/ML method), the algorithms can be separated into regression algorithms (targeting to estimate numerical values) or classification algorithms (targeting to predict distinct classes).

Furthermore, the SL and USL methods differ in the data labeling. In specific, an SL method requires a labeled dataset, so as the input and output are mapped and one data sample includes the associated input/output pair. The SL algorithm then uses the labels (i.e. the expected outputs) to fit a function between features and outputs on the historical dataset. On the other hand, USL does not use labels, so it is suitable to discover hidden patterns in the historical dataset and perform clustering methods (grouping of the data based on their similarity) or dimensionality reduction (extract features that are important for the dataset). Finally, the RL methods are a beneficial fit for decision-making problems, since these ML approaches involve an agent that interacts with a well-defined environment, acting on it and affecting its parameters, while also targeting to maximize cumulative future rewards through a trial-and-error process. Figure 12 depicts the various categories of AI/ML algorithms.

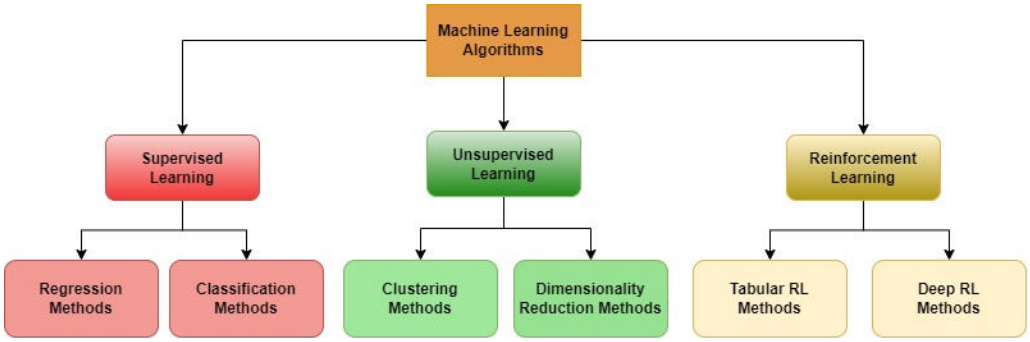

Figure 12: Branches of AI/ML algorithms.

Concerning the SL methods, regression and classification ML models can be linear or nonlinear depending on their composition. The regression models consist of linear regressors with one or multiple features, considering linear mapping between inputs and outputs, as well as negligible multicollinearity among features (features are not correlated). Instead of using linear regressors, the ML model can consist of polynomial regressors, enabling the fitting of more complex polynomial functions. In any case, linear and polynomial regressors assume a linear relationship between input and output variables, in the sense that the output can be expressed as a linear weighted combination of the inputs. Typical SL regression ML methods include Support Vector and Decision Tree Regressors, as well as Ensemble Learning techniques (e.g. Random Forest methods use ensemble outputs of Decision Trees, where multiple trees have been trained with a subset of the training data and they collectively contribute to the estimated output). Similarly, classification ML methods target to separate the output classes either through a linear line (or, in general, hyperplane) or curved lines, assuming nonlinear input/output dependencies. Linear classifiers include, amongst other, Logistic regression that aims to decide if the probability of classifying a sample is above or below 50% and Support Vector Machines (SVMs) that target to find a linear line to maximize the out-of-class separation. On the other hand, the K-Nearest Neighbors and Kernel SVMs, as well as Naïve Bayes algorithms can be utilized as nonlinear classifiers.

As aforementioned, the USL includes clustering and dimensionality reduction methods. Clustering can group together several samples of the training data based on hidden patterns in the input features. For instance the K-Means clustering involves the assignment of each sample to the nearest cluster centroid until K clusters are formed, while the agglomerative hierarchical clustering involves the continuous congregation of each pair of close samples (based on their distance in the N-dimensional feature space) until a distant merging of classes is reached. Moreover, the dimensionality reduction functionality of USL methods involves the identification of important components or features in the training data. Typical methods include the Principal Component Analysis methods that projects the feature data into two axes in order to maximize the explained variance and the Linear Discriminant Analysis that projects the feature data into two axes in order to maximize the class separation.

Finally, regarding the decision making algorithms, RL involves an agent interacting with an environment, and receiving feedback in terms of rewards or punishments depending on his/her performed actions. During the training process, the agent targets to perform a large number of possible actions on the environment (ideally all possible actions), record its past experiences and gradually learn to recommend the actions (or series of actions in order to maximize the long-term reward) that are beneficial for the environment. The traditional form of RL is the tabular Q-learning method, according to which RL agents use the so-called Q-table to estimate the value of being in a particular state of the environment and perform a particular action. Furthermore, the tabular Q-learning can be extended by the use of a neural network as a Q-function approximator, instead of using a memory-inefficient tabular format. The above methods have been successfully implemented in non-convex optimization problems, leveraging their capabilities of dealing with massive state and action spaces in order to provide sub-optimal solutions.

Funded by the European Union

An RL agent can be deployed in centralized locations and achieve global observability and action taking, monitoring a large environment. Alternatively, multiple RL agents can be deployed in a distributed manner in decentralized locations, having local observability and limited action space. In the latter scenario, the multi-agent RL framework requires inter-agent coordination to achieve global objectives, usually achieved through common reward sharing or federated learning techniques.

5.1.2 Deep Learning

Deep neural networks (DNNs) have been increasingly utilized in all the AI/ML domains described in the previous subsection, since they show remarkable capabilities in approximating both linear and nonlinear functions. Towards this direction the DNNs can be used as SL or USL regressors or classifiers, depending on the values to be predicted in their output layer. Moreover, DNNs can be also used in RL schemes, since they can approximate for instance the Q-function. In SL, DNNs show improved performance in fitting complex nonlinear mathematical relationships, since they involve multiple connected hidden layers between the input and output layers, as well as the gradient descent operation during the backpropagation procedure.

5.1.2.1 Deep Reinforcement Learning

As aforementioned, DRL models are suitable for real-time decision-making problems in very complicated environments, where the DRL agent learns through a trial-and-error procedure. The agent first monitors the environment through a state space *S* and performs an action from an available action space *A*. The implementation of the selected action on the environment leads to a new state. At the same time, the agent receives a reward from the environment depending on whether the performed action was favorable towards the optimization goal. In this context, the agent may receive a positive reward if the selected action was beneficial for the environment, a negative reward/penalty if the action leads to a new state in the opposite direction of the optimization target or a zero reward if the impact of the selected action was negligible. Once these steps have been performed, the agent records this experience (state, action, new state, reward) in its memory before proceeding to the next action, reflecting acquired knowledge. The agent continues to select random actions from possible states of the environment, targeting to test ideally all available actions to all possible environmental states and storing this knowledge in its memory as recorded experience tuples. These tuples include the quantification metric of the performed action from a given state, i.e. the received reward from the environment. The learning process involves the training of the agent to gradually perform the actions (or multiple consecutive actions) that will return the maximum reward. The Q-value can be calculated according to the Bellman equation:

$$Qt(st,at) = (1-\alpha)\bullet Qt\text{-}1(st,at) + \alpha\bullet[r(st,at) + \gamma\bullet maxa'\{Qt(st+1,a')\}]$$

where the parameter α (learning rate) is utilized as a trade-off between previous and current Q-values and γ is a factor that reflects the trade-off between immediate (received directly due to the performed action) and long-term (estimation of future rewards that will be received due to a series of actions) rewards. The main feature of DRL is that the quality of each action (the

Q-value in the Bellman equation) can be estimated with the use of a neural network, since the dimensionalities of the action and state spaces are typically very large.

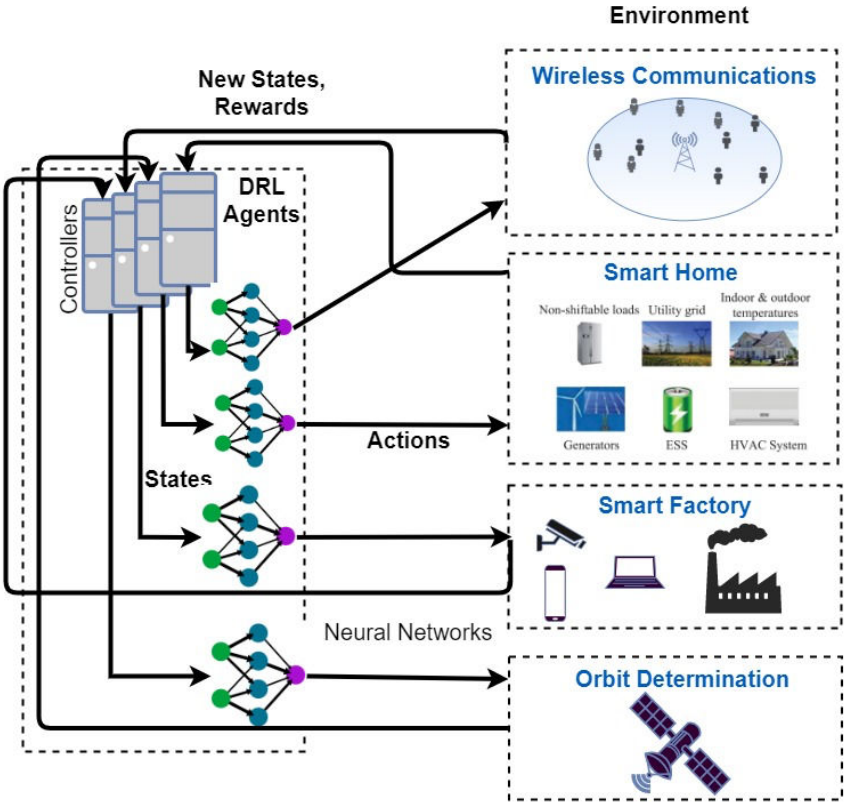

Figure 13: Training process of DRL algorithms

The training concept of the DRL agents is depicted in Figure 13, where it is assumed that the DRL agents are centralized. Evidently, the environment that the DRL agents interact with provides the state space. For instance, the environment can provide physical or network layer metrics (in case of wireless communications), battery status and inside temperature (in case of a smart home), data from IoT network or from the manufacturing processing (in case of smart factory), or satellite status (in case of orbit determination). Similarly, the performed actions and the rewards are designed based on the specifications of each environment and also the optimization goals. To this end, the actions of the DRL agents may include power control for the wireless base stations towards energy efficiency, energy management in a smart home, optimization of the industrial production chain in terms of scheduling, etc.
Method of Early Exit of Inference (EEoI), where some intermediate results with higher credibility, although only be processed on end devices can still exit from the neural network in advance, and no longer be sent to edges or the cloud. Intermediate results with lower credibility will be transmitted to the deeper network deployed on edges or the cloud for further inference.

5.1.2.2 Federated Learning

Federated Learning (FL) is a concept that allows ML models to be trained directly on edge nodes using local data and to be reused from other nodes in order to reduce unnecessary data transfer and privacy exposure. In a FL framework, each edge node (e.g., an edge server or

even a Raspberry Pi) trains its own local ML model with its own local data. A collaborative ML model with the same dimensions as the local models is also deployed on a centralized node (e.g. a cloud server). The edge nodes periodically send the parameters of their local ML models to the cloud server (for example, neuron weights in case of deep neural networks), instead of sending the local data. The ML model parameters reflect the extracted local knowledge from each edge node, without actually having data transfer to the cloud server. The collaborative knowledge of all edge nodes can be combined into the cloud ML model, using for example a weighted average value of the local ML models' parameters [14]. Then, the ML model parameters of the collaborative cloud model are sent back to the decentralized edge nodes, updating their local models with the distilled intelligence of all the edge nodes (and indirectly with all observed local data) that participate in the FL framework. The FL process between edge nodes (servers, edge or mobile devices) and a single cloud server is illustrated in Figure 14.

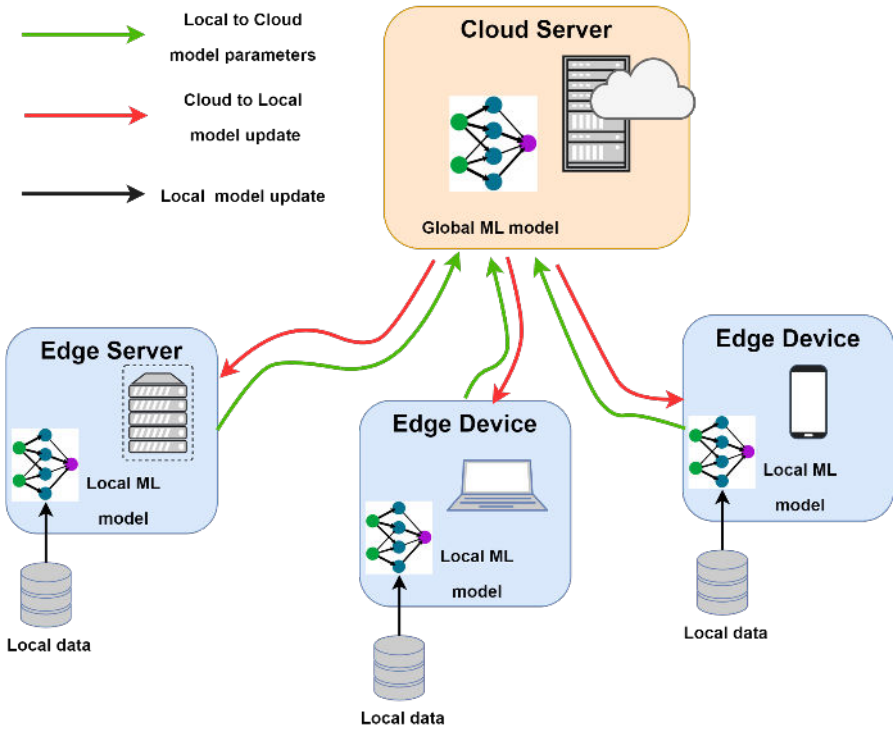

Figure 14: Federated Learning framework demonstrating how collaborative knowledge of edge nodes is combined in a single cloud ML model.

This decentralized learning scheme has several important benefits: (i) indirect knowledge transfer between the edge ML models through the cloud global model; (ii) transfer learning in cases that new nodes are connected to the network, in the sense that the ML models of the new nodes immediately gain the benefits of the distilled knowledge of the participating edge nodes in the FL framework; (iii) considerable data transfer reduction, since the exchange data amongst the edge and cloud concern only the ML model parameters and not the locally observed data; (iv) the latter also assists in the data privacy, security and data ownership concepts, since sensitive information is not transmitted through the network and the data remain at the edge.

Various FL frameworks have been proposed in the literature in order to deal with the aforementioned issues, be implemented on modern network architectures and enhance the energy efficiency of the inference process, without compromising the accuracy of the ML model estimation. According to the Method of Early Exit of Inference (EEoI), some intermediate results with high accuracy can be exited from the ML models in advance, although the data processing has only occurred on end devices. Figure 15 illustrates the EEoI method in a three-layered architecture (end devices, edge nodes and cloud). The primary target is to significantly reduce the inference time of the FL framework, but also further reduce the data transfer from the end devices to the edge nodes and the cloud. It should also be noted that the ML models that are usually deployed in end devices are not deep (for instance these ML models may consist of one or two hidden layers in case of neural networks) due to computationally complexity issues. Intermediate results with lower accuracy/credibility can be transmitted to the deeper network deployed on edges or the cloud for further inference [15].

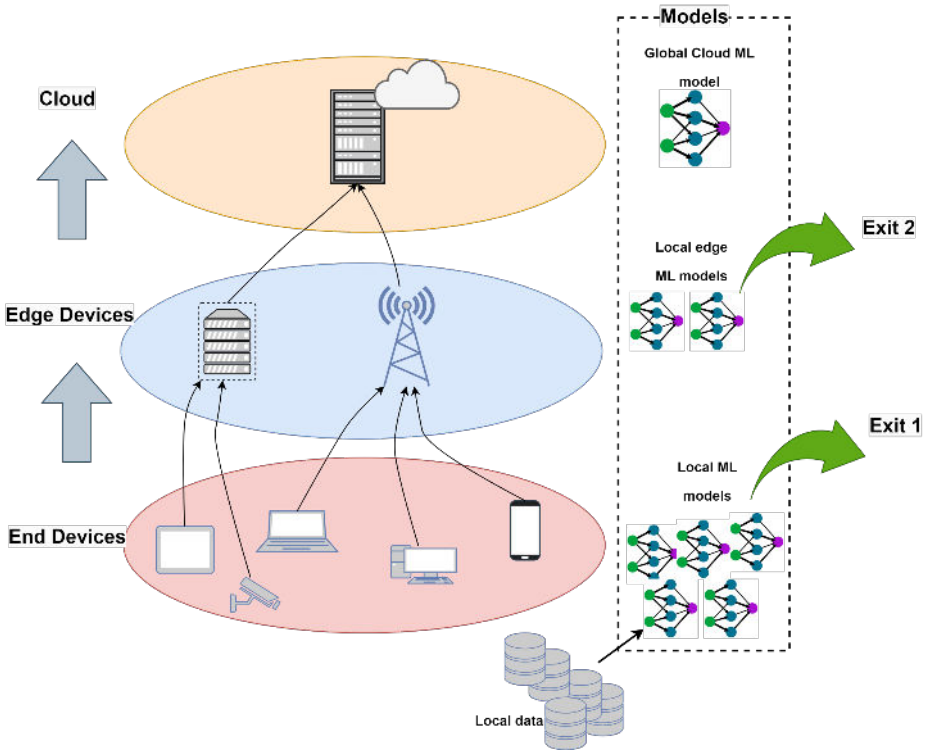

Figure 15: Method of Early Exit of Inference between end devices, edge nodes and the global ML model residing in the cloud.

In addition, the FLEE algorithm [16] proposes a FL framework for ML model training in the same layered architecture (cloud, edge and end device) that is depicted in Figure 15. However, in this case the local data encountered by the end devices and the edge nodes are used to train a single Deep Neural Network (DNN) model, whose deepness depends on the architectural layer (end devices, edge or cloud), as depicted in Figure 16. It should be noted that the exit points (or decision layers) of the DNN are specified in advance and are taken into consideration during the training process.

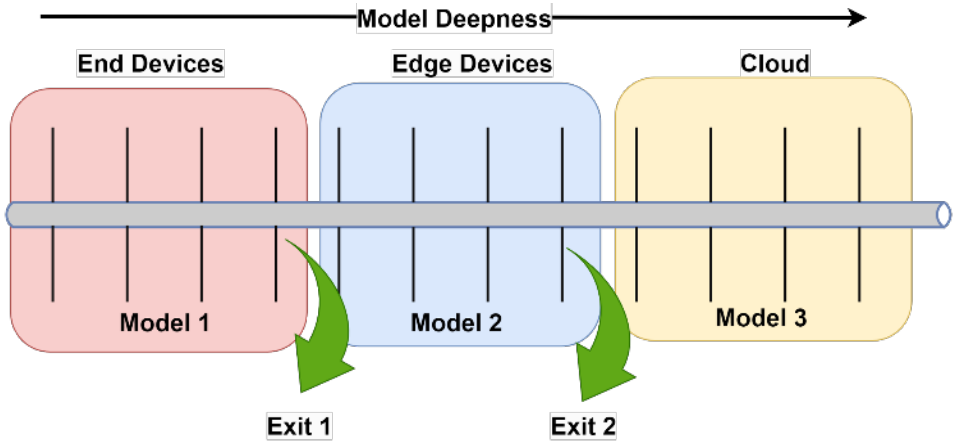

Figure 16: Training of DNN model according to the FLEE algorithm. The vertical lines represent layers of the deep neural network.

The FLEE method can be described in the following steps:

- Step 1 - Local update: Each end device uses local data to train and update the neural network composed of exit 1 and model 1.
- Step 2 - Edge aggregation: In this step, the federation process occurs at the edge-level FL framework (between the end devices and a single edge node), as depicted in Figure 15.
- Step 3 - Local and edge update: Based on the model and exit obtained in Step 2, each edge performs collaborative training on the whole model (including model 1, exit 1, model 2, and exit 2) through the weighted loss (for example weighted average on the ML models' parameters - neuron weights).
- Step 4 - Cloud aggregation: This is a cloud-level FL process (between the edge nodes and the cloud), leading to a global deep model that spans all the architectural layers.

Furthermore, according to the *Big.Little FL* architecture, light ML models can be directly deployed in the end devices, while the ML model in the cloud can be more complex, deeper and use advanced computational resources [17]. The different models that are shown in Figure 17 can be described as follows:

- *Cloud Model*: The cloud model consists of two parts, i.e., the big branch model (deeper and larger than the little branch) and the trunk model. Since the model is deployed on some cloud server with high-end computational resources (e.g. GPUs), the inference time is not expected to be a major bottleneck in the Big.Little FL method.
- *Device Model*: The device model consists of the little branch model and the trunk model. Each end device has its own copy of an ML model with the same structure but different weights (depending on the observability of the local data) in order to be more accurate during the model inference in its local scenario. Typically, the end device with the lowest computational capacity (e.g. camera or IoT device/sensor) is taken into consideration when designing the device model.
- *Global Device Model*: The global device model is located at the cloud to enable device model aggregation based on the FL framework. In specific, gradient information from device models is periodically acknowledged to the cloud, the global device model is updated and aggregated with the cloud model. At the end of round in the FL scheme, the end devices receive the updated global device model parameters that are used to update their own local ML models.

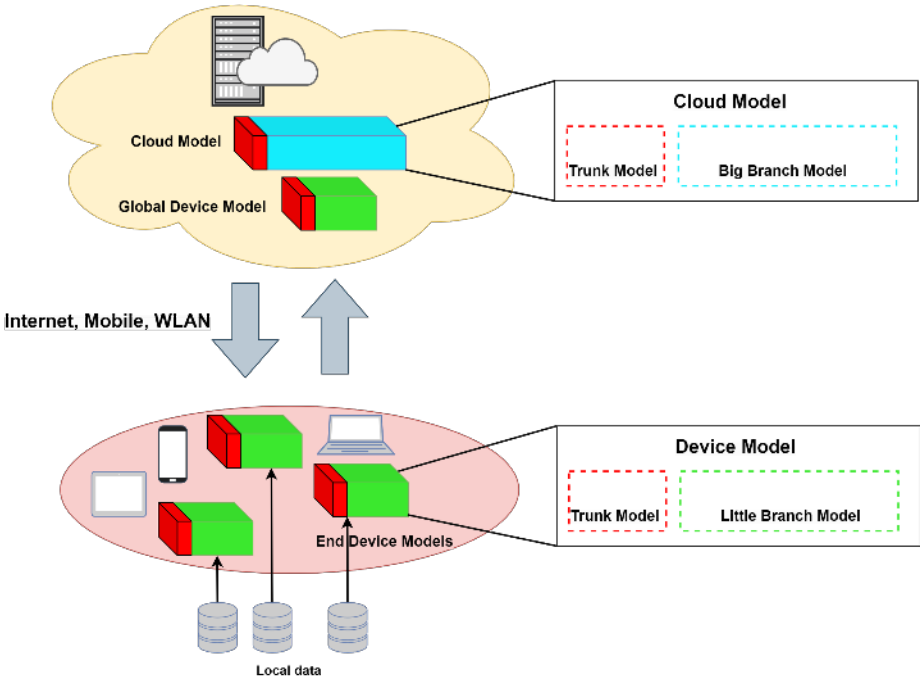

Figure 17: Big.Little FL architecture involves cloud (trunk and big branch), as well as device and global device (trunk and little branch) models.

Finally, the FL framework can be also combined with multiple access techniques for sharing the spectrum. This is extremely important in recent and upcoming wireless environments, since according to beyond 5G cellular standards, multiple transmitters/receivers that share the spectrum at the same time will be located in the same wireless area, causing co-channel interference and effectively reducing the spectrum efficiency. A promising technique that has been developed in non-orthogonal multiple access schemes for integrating heterogeneous IoT ecosystems in beyond 5G network is Over the Air computation (AirComp).

AirComp is based on the temporal superposition of multiple signals from different sources and is designed for communication efficiency purposes, as well as enhancing the privacy of the data transmitted and the energy of the transmitters/sources. As shown in Figure 18, the receiver/fusion center (based station in this case) collects the time-domain signals of all transmitters and post-process them, fusing them together. An AirComp system has a better spectrum utilization than normal multiple access frameworks, while also enabling the data privacy and being energy-efficient, since the power levels of the transmitters are adjusted for optimized post-processing in the fusion center.

Funded by the European Union

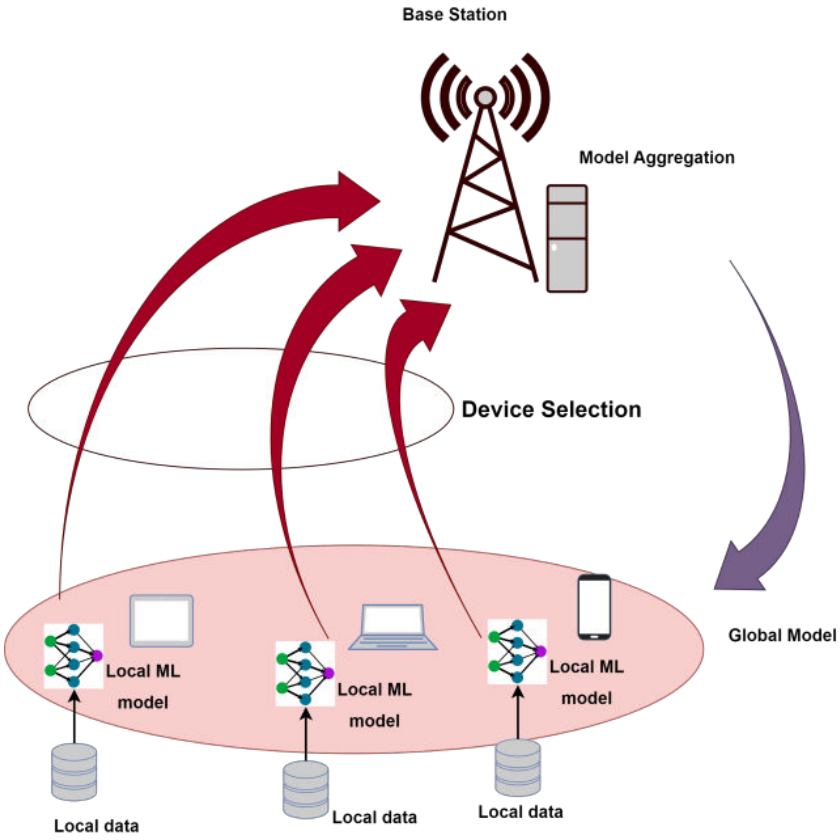

Figure 18: FL framework in an AirComp system.

Furthermore, ML model training is enabled in the FL framework through AirComp communications. In specific, the FL framework can also exhibit device selection, as shown in the FedAvg framework for AirComp [18]:

- Step 1: The Base Station (BS) selects a subset of mobile devices to participate in the particular round of the FL scheme.
- Step 2: The BS sends the updated global model to the selected devices
- Step 3: Each selected device runs a local update algorithm (e.g., stochastic gradient algorithm) based on its local dataset and global model, updating the local model
- Step 4: The BS aggregates all the local updated models, computing their weighted average as the updated global model

5.1.3 Programming paradigms

State-of-the-art: Programming distributed applications spanning across the cloud-edge continuum is a difficult and error-prone endeavour, requiring developers with a high level of expertise. The main challenge is that the correctness and reliability of such applications hinges upon complex aspects – e.g., conformance to desired communication protocols between distributed components, fault-tolerance, adaptation, data placement and consistency - that have little or no support in typical programming languages and frameworks. The state-of-the-art in programming distributed systems is chiefly focused on client-server and cloud applications, with mature approaches and tools for centralised service-oriented systems (e.g., Enterprise Service Bus), scaling endpoints (e.g., elastic load balancers), data processing (e.g.,

Apache Spark, Storm, Flink, Hadoop), and communication (e.g., Kafka, RabbitMQ). Correspondingly, several approaches in academic literature focus on facilitating applications in specific domains (e.g., map-reduce computations [19], or data stream processing [20], or web applications [21]). Several modern languages and frameworks (e.g., Erlang [22], Akka, Microsoft Orleans [23]) are based on the actor model [24], with active research on new implementations [25]. Although actors conceptually allow for decentralised applications, they are rather low-level and imperative in their programming approach, hence using them as building blocks for large systems (without further abstraction layers) is challenging. Recent language proposals [26-29] provide specialised distribution abstractions for expressing mixed consistency models for data, node topology, and data placement. Unfortunately, none of these approaches addresses the dynamicity and heterogeneity of decentralised swarm systems, where computation may take place across different devices on the cloud-edge continuum, having varying capabilities and supporting different implementation technologies.

Beyond state-of-the-art: TaRDIS will develop a novel paradigm to the development of distributed applications built on heterogeneous swarms. The TaRDIS approach is based on a declarative model, allowing programmers to write applications where the system behaviour is chiefly expressed in terms of events (capturing e.g., communication, devices joining or leaving, data updates) and system components are described with properties and capabilities (e.g., where a device is located, what programming languages it supports, what events it can handle, what security and privacy guarantees it can provide). This declarative view abstracts the implementation details behind APIs to access the TaRDIS runtime services pertaining application deployment, system communication, data access, AI/ML computation) while the TaRDIS IDE assists programmers in using these APIs correctly. A key benefit of this approach is that the declarative model and device capabilities will allow for the automated verification of TaRDIS applications (statically and at run-time), thus reducing the need for extensive (and costly) handwritten test suites.

5.1.4 Data placement in peer-to-peer systems

State-of-the-art: Two building blocks in the design of large-scale systems are the protocols for managing the system's membership and providing efficient communication, and the protocols for data management. As systems scale to a large number of nodes and different networking conditions, it becomes increasingly infeasible to use protocols that rely on global knowledge and decentralised protocols need to be used. Overlay networks have been proposed to connect the nodes of large-scale systems in a decentralised way, managing the node membership and providing efficient communication primitives, with many protocols being proposed for settings ranging from large scale distributed and peer-to-peer systems [30–32], container networks in cloud settings [33–34] and edge systems [35]. Managing data in these settings is also challenging, as solutions proposed for cloud settings, either adopting strong [36], weak [37] [38], or hybrid [39] consistency models are not designed to cope with the scale, heterogeneity, and security requirements of these settings. Several systems have been addressing these challenges, by proposing solutions specific for edge nodes [40], hybrid cloud-edge settings that include peer-to-peer synchronisation among client nodes [41], or addressing security challenges [42]. Although providing strong consistency is infeasible in most large-scale decentralised settings, it is still important to provide enough guarantees for correct application execution – an extensive number of works have addressed this problem [43] [44].

Beyond the state-of-the-art: In the context of TaRDIS, we intend to design overlay networks that span from the cloud to the edge and swarms of nodes. Doing that will require the combination of a hierarchical design with localised solutions for managing swarms of nodes requiring peer-to-peer communication and coordination. As for data management, we plan to design solutions that provide enough guarantees for application execution while coping with the limited connectivity expected in some of our use cases. While most solutions in cloud setting build on a combination of sharding inside a data centre with full replication across data centres, our setting will require support for partial replication in edge and client nodes. Finally, the decentralised nature of the setting being addressed in some of TaRDIS use cases will require addressing the potential incorrect behaviour due to malfunction of system nodes or malicious nodes.

5.1.5 Behavioural Types for communication analysis

State-of-the-art: Ensuring safety and reliability of communicating systems is the subject of a significant body of work. A key approach is that of behavioural types [45] (BTs), which specify the intended interaction patterns of systems, such that well-typed systems adhere to the prescribed interactions. BTs can be incorporated into existing languages[46] and describe both internal and external system behaviour. The main approaches for the former are typestates[47]; for the latter, contracts, and session types [48]. Multiparty session types [49] (MPST) support specifying the interactive behaviour of protocols with many participants and guarantee deadlock-freedom by construction. Extensions to core theories include features to support heterogeneous swarms: Failure handling extensions include affine sessions, permitting processes to fail, and coordinator-based failure handling techniques [50]; other extensions enable dynamically evolving connections between protocol participants[51] and dynamically sized participant pools [52]. Recent works preliminarily address compositional verification [53] of open systems [54].

Beyond state-of-the-art: In TaRDIS, we will build upon the above approaches, enabling the application of MPST to heterogeneous swarms. This includes generalising failure handling to support crash-recovery and relaxing network reliability assumptions, so that systems account for network and hardware failures. We will also extend MPST to allow for dynamic joining/leaving of protocol participants, leveraging work on Conversation Types, and explore the integration of model checking techniques to augment BT and MPST applicability. Moreover, we will study MPST composition techniques to enable compositional and more general forms of open system reasoning.

5.1.6 Security

State-of-the-art: There is much work on the symbolic verification of security protocols in finite state systems [55-58] and infinite state systems, but without mutable long-term state [34]. StatVerif addresses the mutable state limitation [59] [60]. Compositional reasoning for security protocols includes parallel [61] and sequential [62] composition and vertical composition [63] (e.g., TLS provides a secure channel for a variety of applications). Protecting information that is not a (cryptographic) secret is called a privacy-type goal and is significantly harder to verify automatically. The best techniques are observational equivalence relations, i.e., where an intruder cannot distinguish processes by the steps they make. Much work has focused on how the verification can be automated in finite-state systems [64], and otherwise [65], albeit with severe restrictions [66]. Alpha-beta privacy [67] specifies such goals positively, i.e., by what

information is deliberately released through the course of the system; it is a violation if the intruder can deduce anything that is not entailed by the goals.

Beyond state-of-the-art: heterogeneous swarms pose several challenges to these verification methods. We will develop novel abstractions to deal with the special needs of such systems, in particular their APIs. We will also combine symbolic model-checking with abstract interpretation to yield methods that can be used for both effective attack searching and proving the absence of attacks in a system. Compositionality methods must be extended to be compatible with the TaRDIS APIs. Alpha-beta privacy permits a simple specification language that can be analysed with existing techniques and enables direct implementation of this concept using simple reachability problems, improving performance, and requiring fewer restrictions on the type of systems that can be considered.

5.1.7 Data Integrity

State-of-the-art: In global applications, replicating data closer to the users has become a popular technique to reduce user latencies. In these settings, data convergence and integrity can be preserved with strong consistency, but with a significant performance and availability cost. An alternative is to adopt weaker consistency guarantees, which may introduce (temporary) state divergence and violate application invariants. To ease the development of geo-distributed applications, replicated data types (RDTs) have been proposed. An RDT exposes the same interface as its sequential counterpart and embeds in its implementation mechanisms to enforce correctness in a geo-replicated setting. Unfortunately, designing a new RDT that guarantees state convergence and preserves data integrity is difficult and reserved to experts. Several verification techniques have been proposed [68–70] to check RDT properties. There have been many works on static analysis tools to check invariant preservation [43] [71] [72] and detect operations that violate application-level invariants. However, they assume a cloud-centric perspective, with data fully replicated.

Beyond the state-of-the-art: In TaRDIS, we will advance the state-of-the-art as current techniques assume full data replication across data centres and a degree of system homogeneity that is incompatible with the decentralised swarms targeted by TaRDIS. The initial goal is to develop verification techniques that can check consistency and data integrity with partial replication of data. For this result, we will use the well-known technique of simulation between the partial replication schema and a global one. The heterogeneity of some of TaRDIS use cases will require that we also address partial replication when data is scattered across nodes that can have different consistency models.

5.1.8 Formal Verification of distributed AI

State-of-the-art: Only recently, frameworks such as Flower [73] (for federated learning) and BlueFog [74] (for fully distributed learning) have been introduced, promising scalable decentralised ML workloads on heterogeneous edge devices. However, the communication protocols used in these frameworks are lacking trustworthy methodologies that can provide safe and reliable systems.

Beyond state-of-the-art: We will design a framework for the safe orchestration of decentralised swarm ML. Further developments of TaRDIS will enable the coordination of the ML primitives ensuring that each primitive has access to the data that it requires. The framework will ensure the safe execution of machine learning actions at collaborative smart

edge-nodes. Finally, we will integrate the verification analyses with the AI-based optimisation. Resource orchestration will be made transparent and hence more trustworthy by exploiting transparent and secure data management, including swarm ML/DL models and models for reinforcement learning [75]. We will investigate local model explainability based on LIME [76] and local surrogate decision trees to be linked with allocated resource schemes to increase explainability of decisions on resource allocations to humans.

5.1.9 Interoperable Execution Environments

State-of-the-art: When new technologies are proposed, many organisations and industry resist their adoption [77] due to several aspects such as the cost of the change, lack of human resources, or due to the need to interact with legacy systems. One way to mitigate this aspect is to develop mechanisms that allow the seamless integration with legacy systems, middleware, or other tools. In the past this has been achieved many times using adaptors. Examples of this includes the use of adapters in Legion [41], a peer-to-peer framework that allows to (transparently) enrich web applications with direct client-to-client data replication and off-line operation. Legion features adapters that allow the easy integration of legacy server-side storage services to provide data durability. Adapters have also been employed to enrich consistency guarantees on legacy cloud-based storage systems [78], enabling client applications to fine-tune which session guarantees they want to expose to applications despite the ability of the central service to provide such guarantees. While these examples show the potential of this approach, both rely on a significant effort to develop an adapter, which must be custom made for a particular system.

Beyond the state-of-the-art: In TaRDIS we plan to leverage the approach of using adapters to allow seamless integration and inter-operation between solutions developed by TaRDIS and existing software solutions, commonly used in industry, to develop and monitor distributed systems (Prometheus, Nagios) and to support inter-operation between processes (e.g., libP2P, Kafka) or storage (e.g., Cassandra, Redis). However, to simplify the task of generating such adapters, we plan to explore mechanisms to automatically generate such adapters based on a specification, by taking advantage of previous contributions of the consortium in this context [79].

5.2 Next Steps

The preliminary information collected during activities in this phase of the project has been made available to all TaRDIS partners since the early stages of the project. This provided initial inputs to developing programming abstractions for the Cloud-Edge Continuum, more specifically looking at the models and abstractions for decentralised applications. This early exchange of information between the industrial and academic partners has allowed the latter to provide feedback to the former, and request specific details about the use case. We intend to keep this communication channel throughout the project duration.

Further, a tentative definition of Heterogeneous Swarm for the TaRDIS project has been proposed: *"a distributed system whose nodes (differing in hardware and/or software) jointly execute some decentralised algorithm (including AI and ML, e.g., for orchestration and optimisation), via structured communication between a dynamic set of known parties that may join or leave the swarm application."*

The following next steps are planned to refine the definition of the TaRDIS use case requirements.

- To produce an informal description of some sample executions of each use case, via message sequence charts or state machines (or similar representations).
- To define a notion of boundary for TaRDIS applications, separating TaRDIS application components (i.e. devices and programs using the TaRDIS APIs and framework) from external components (i.e. devices and programs that cannot run TaRDIS APIs and framework, but are expected to interact with TaRDIS applications).
- To identify the set of programming languages that are needed by the TaRDIS use cases, and will be supported by the TaRDIS API, framework and tools. The vision is that TaRDIS will provide an event-driven, reactive and callback-based API for a selected number of programming languages (needed by the use cases), with a design amenable to supporting more languages in the future.
- To identify the communication topologies of the use cases, distinguishing e.g., between static hierarchical scenarios, dynamic peer-to-peer scenarios, and combinations of the two. The objective is to support the most dynamic scenarios, and address the more static scenarios as special cases, without introducing excessive burden or difficulties for the programmers.
- To identify the reliability assumptions about the communication in the various use cases and outline how communication failures are expected to be handled by application programmers (e.g., with explicit failure-handling code, or with forms of automatic failure handling in the “let it crash” style of Erlang).
- To identify, for each use case, some examples of desirable application properties that the TaRDIS APIs, framework and tools will help programmers verify, by suitably combining static and run-time verification. Such example properties may be classified as:
    - safety properties (“an undesired event never happens”)
    - liveness properties (“a desired event eventually happens”)
    - security and privacy properties (authentication, agreement, secrecy)
    - data properties (consistency, availability, integrity)

Funded by the European Union

6 CONCLUSION

Thus, the recent years have witnessed a remarkable surge in decentralized application development, driven by the advantages they offer in terms of eliminating intermediaries, leveraging decentralized networks and infrastructures, and enhancing data privacy and user empowerment. However, this growth has brought forth challenges stemming from the heterogeneity within decentralized networks, including diverse devices, protocols, and technologies.

As a first step to address these challenges and simplify the development process of decentralized applications, we adopt a top-down co-design approach to define the requirements for TaRDIS toolbox. This approach involves gathering insights from various sources and engaging with a diverse range of programmers. The co-design process begins with analyzing large-scale data collected from external decentralized platforms, which provides valuable insights into trends, patterns, and challenges within the domain.

The key contributions of this report include exploring the growth and challenges of decentralized applications, gathering insights from developers operating at scale, conducting questionnaires to understand programming experts' preferences and expectations, examining use cases and addressing their challenges, and deriving a comprehensive set of initial technical requirements. We consolidate these insights to ensure that the TaRDIS toolbox effectively meets the diverse needs of users in the decentralized programming landscape, thereby to overcome the challenges posed by heterogeneous settings.

Funded by the European Union

REFERENCES

[1] "What is IPFS?," IPFS Documentation, [Online]. Available: https://docs.ipfs.tech/concepts/what-is-ipfs/.
[2] "IPFS: Content-Addressed, Versioned, P2P File System," IPFS Documentation, [Online]. Available: https://docs.ipfs.tech/concepts/further-reading/academic-papers/#ipfs-content-addressed-versioned-p2p-file-system
[3] IPFS, GitHub. [Online]. Available: https://github.com/ipfs/ipfs.
[4] Trautwein, D., Raman, A., Tyson, G., Castro, I., Scott, W., Schubotz, M., Gipp, B. and Psaras, Y., 2022, August. Design and evaluation of IPFS: a storage layer for the decentralized web. In Proceedings of the ACM SIGCOMM 2022 Conference (pp. 739-752).
[5] Zia, H.B., He, J., Raman, A., Castro, I., Sastry, N. and Tyson, G., 2023. Flocking to mastodon: Tracking the great twitter migration. arXiv preprint arXiv:2302.14294.
[6] Anaobi, I.H., Raman, A., Castro, I., Zia, H.B., Ibosiola, D. and Tyson, G., 2023, April. Will Admins Cope? Decentralized Moderation in the Fediverse. In Proceedings of the ACM Web Conference 2023 (pp. 3109-3120).
[7] Zuo, W., Raman, A., Mondragón, R.J. and Tyson, G., 2023, April. Set in Stone: Analysis of an Immutable Web3 Social Media Platform. In Proceedings of the ACM Web Conference 2023 (pp. 1865-1874).
[8] "Stack Overflow Developer Survey," Stack Overflow, [Online]. Available: https://survey.stackoverflow.co/.
[9] "Stack Overflow Developer Survey 2022 - Technology: Web 3.0," Stack Overflow, [Online]. Available: https://survey.stackoverflow.co/2022#technology-web-3.
[10] Prolific, [Online]. Available: https://www.prolific.co/.
[11] Octoverse 2022. The state of open source software. [Online]Available: https://octoverse.github.com/.
[12] Rajan, John A., "Highlights of GPS II-R Autonomous Navigation," Proceedings of the 58th Annual Meeting of The Institute of Navigation and CIGTF 21st Guidance Test Symposium (2002), Albuquerque, NM, June 2002, pp. 354-363.
[13] Lv, Y.; Geng, T.; Zhao, Q.; Xie, X.; Zhang, F.; Wang, X. Evaluation of BDS-3 Orbit Determination Strategies Using Ground-Tracking and Inter-Satellite Link Observation. Remote Sens. 2020, 12, 2647. https://doi.org/10.3390/rs12162647
[14] H. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized Data,” Feb. 2016.
[15] Teerapittayanon, S., McDanel, B. and Kung, H.T., 2017, June. Distributed deep neural networks over the cloud, the edge and end devices. In 2017 IEEE 37th international conference on distributed computing systems (ICDCS) (pp. 328-339). IEEE.
[16] Zhong, Z., Bao, W., Wang, J., Zhu, X. and Zhang, X., 2022. FLEE: A Hierarchical Federated Learning Framework for Distributed Deep Neural Network over Cloud, Edge and End Device. ACM Transactions on Intelligent Systems and Technology (TIST).
[17] Zhang, X., Hu, M., Xia, J., Wei, T., Chen, M. and Hu, S., 2020. Efficient federated learning for cloud-based AIoT applications. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 40(11), pp.2211-2223.
[18] Yang, K., Jiang, T., Shi, Y. and Ding, Z., 2020. Federated learning via over-the-air computation. IEEE Transactions on Wireless Communications, 19(3), pp.2022-2035.
[19] J. Dean and S. Ghemawat, “MapReduce,” *Commun ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[20] M. Armbrust *et al.*, “Spark SQL: Relational Data Processing in Spark,” in *SIGMOD 2015*.
[21] G. Boudol, Z. Luo, T. Rezk, and M. Serrano, “Reasoning about Web Applications: An Operational Semantics for HOP,” *ACM Trans. Program. Lang. Syst.*, vol. 34, no. 2, pp. 10:1–10:40, 2012.
[22] J. Armstrong, “Erlang,” *Commun ACM*, vol. 53, no. 9, pp. 68–75, Sep. 2010.
[23] S. Bykov, A. Geller, G. Kliot, J. Larus, R. Pandya, and J. Thelin, “Orleans: Cloud computing for everyone,” *SOCC 2011*.
[24] G. A. Agha, “Actors: A Model of Concurrent Computation in Distributed Systems.,” 1986.
[25] Christopher Meiklejohn, Heather Miller, and Peter Alvaro, “PARTISAN: Scaling the Distributed Actor Runtime,” in *USENIX* 2019.
[[26] P. Weisenburger, M. Köhler, and G. Salvaneschi, “Distributed system development with ScalaLoci,” in OOPSLA 2018.
[27] N. Eskandani, M. Köhler, A. Margara, and G. Salvaneschi, “Distributed object-oriented programming with multiple consistency levels in ConSysT,” *in SPLASH Companion 2019*.
[28] M. Milano and A. C. Myers, “MixT: a language for mixing consistency in geodistributed transactions,” *ACM SIGPLAN Notices*, vol. 53, no. 4, pp. 226–241, Dec. 2018.
[29] K. de Porre, F. Myter, C. Scholliers, and E. Gonzalez Boix, “CScript: A distributed programming language for building mixed-consistency applications,” *Journal of Parallel and Distributed Computing*, vol. 144, pp. 109–123, Oct. 2020.
[30] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” SIGCOMM Comput. Commun. Rev., vol. 31, no. 4, pp. 149–160, Aug. 2001.

[31] J. Leitão, J. Pereira, and L. E. T. Rodrigues, “HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast,” in DSN 2007.
[32] M. F. S. Ferreira, J. Leitão, and L. E. T. Rodrigues, “Thicket: A Protocol for Building and Maintaining Multiple Trees in a P2P Overlay,” SRDS 2010.
[33] “Flannel is a network fabric for containers, designed for Kubernetes.” https://github.com/flannel-io/flannel (accessed Apr. 02, 2022).
[34] “Weave: Simple, resilient multi-host containers networking and more.” https://github.com/weaveworks/weave (accessed Apr. 02, 2022).
[35] P. Á. Costa, P. Fouto, and J. Leitão, “Overlay Networks for Edge Management,” in NCA 2020.
[36] J. C. Corbett et al., “Spanner: Google’s Globally Distributed Database,” ACM Trans. Comput. Syst, vol. 31, no. 8, 2013.
[37] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, “Don’t settle for eventual: Scalable causal consistency for wide-area storage with COPS,” SOSP 2011.
[38] S. Almeida, J. Leitão, and L. Rodrigues, “ChainReaction: A causal+ consistent datastore based on chain replication,” EuroSys 2013.
[39] C. Li, D. Porto, A. Clement, J. Gehrke, N. M. Preguiça, and R. Rodrigues, “Making Geo-Replicated Systems Fast as Possible, Consistent when Necessary,” in OSDI 2012.
[40] H. Gupta and U. Ramachandran, “FogStore: A Geo-Distributed Key-Value Store Guaranteeing Low Latency for Strongly Consistent Access,” in DEBS 2018.
[41] A. van der Linde, P. Fouto, J. Leitão, N. M. Preguiça, S. J. Castiñeira, and A. Bieniusa, “Legion: Enriching Internet Services with Peer-to-Peer Interactions,” in WWW 2017.
[42] A. van der Linde, J. Leitão, and N. M. Preguiça, “Practical Client-side Replication: Weak Consistency Semantics for Insecure Settings,” Proc. VLDB Endow., vol. 13, no. 11, pp. 2590–2605, 2020.
[43] V. Balegas, S. Duarte, C. Ferreira, R. Rodrigues, and N. Preguiça, “IPA: Invariant-Preserving Applications for weakly consistent replicated databases,” *VLDB Endowment*, vol. 12, no. 4, pp. 404–418, 2018.
[44] K. de Porre, C. Ferreira, N. M. Preguiça, and E. G. Boix, “ECROs: building global scale systems from sequential code,” in OOPSLA 2021.
[45] H. Hüttel *et al.*, “Foundations of session types and behavioural contracts,” *ACM Computing Surveys*, vol. 49, no. 1, Apr. 2016.
[46] D. Ancona *et al.*, “Behavioral types in programming languages,” *Foundations and Trends in Programming Languages*, vol. 3, no. 2–3, pp. 95–230, 2016.
[47] R. E. Strom and S. Yemini, “Typestate: A Programming Language Concept for Enhancing Software Reliability,” *IEEE Transactions on Software Engineering*, vol. SE-12, no. 1, pp. 157–171, 1986.
[48] K. Honda, V. T. Vasconcelos, and M. Kubo, “Language primitives and type discipline for structured communication-based programming,” *in ESOP 1998.*
[49] N. Yoshida and L. Gheri, “A Very Gentle Introduction to Multiparty Session Types,” in *ICDCIT 2020.*
[50] M. Viering, R. Hu, P. Eugster, and L. Ziarek, “A multiparty session typing discipline for fault-tolerant event-driven distributed programming,” *in OOPSLA 2021*.
[51] R. Hu and N. Yoshida, “Explicit Connection Actions in Multiparty Session Types,” in *FASE 2017*.
[52] R. Demangeon and K. Honda, “Nested Protocols in Session Types,” in *CONCUR 2012*.
[53] F. Barbanera, M. Dezani-Ciancaglini, I. Lanese, and E. Tuosto, “Composition and decomposition of multiparty sessions,” *Journal of Logical and Algebraic Methods in Programming*, vol. 119, Feb. 2021.
[54] R. Horne, “Session subtyping and multiparty compatibility using circular sequents,” *in CONCUR 2020*.
[55] G. Lowe, “Casper: A compiler for the analysis of security protocols,” in CSFW 1997.
[56] D. Basin, S. Mödersheim, and L. Viganò, “OFMC: A symbolic model checker for security protocols,” *International Journal of Information Security 2004 4:3*, vol. 4, no. 3, pp. 181–208, Jun. 2005.
[57] A. Armando *et al.*, “The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications,” in *CAV 2005*.
[58] A. Armando *et al.*, “The AVANTSSAR Platform for the Automated Validation of Trust and Security of Service-Oriented Architectures,” *in TACAS* 2012.
[59] B. Blanchet, “Security protocols,” *Information Processing Letters*, vol. 95, no. 5, pp. 473–479, Sep. 2005.
[60] M. Arapinis, J. Phillips, E. Ritter, and M. D. Ryan, “StatVerif: Verification of stateful processes,” *Journal of Computer Security*, vol. 22, no. 5, pp. 743–821, Jan. 2014.
[61] M. Arapinis, V. Cheval, and S. Delaune, “Composing Security Protocols: From Confidentiality to Privacy,” in *POST 2015*.
[62] V. Cheval, V. Cortier, and B. Warinschi, “Secure Composition of PKIs with Public Key Protocols,” in CSF 2017.
[63] S. Gondron and S. Mödersheim, “Vertical Composition and Sound Payload Abstraction for Stateful Protocols,” in 34th IEEE Computer Security Foundations Symposium, CSF 2021.
[64] V. Cheval, S. Kremer, and I. Rakotonirina, “The DEEPSEC Prover,” in CAV 2018.
[65] B. Blanchet and B. Smyth, “Automated reasoning for equivalences in the applied pi calculus with barriers,” J. Comput. Secur., vol. 26, no. 3, pp. 367–422, 2018.
[66] V. Cheval and B. Blanchet, “Proving More Observational Equivalences with ProVerif,” in POST 2013.
[67] S. Mödersheim and L. Viganò, “Alpha-Beta Privacy,” ACM Trans. Priv. Secur., vol. 22, no. 1, pp. 7:1–7:35, 2019.

[68] G. Kaki, S. Priya, K. C. Sivaramakrishnan, and S. Jagannathan, “Mergeable replicated data types,” *in OOPSLA 2019*.
[69] M. Weidner, H. Miller, and C. Meiklejohn, “Composing and decomposing op-based CRDTs with semidirect products,” *in ICFP 2020*.
[70] V. Balegas *et al.*, “Putting consistency back into eventual consistency,” *in EuroSys 2015.*
[71] A. Gotsman, H. Yang, C. Ferreira, M. Najafzadeh, and M. Shapiro, “‘Cause I’m strong enough: Reasoning about consistency choices in distributed systems,” *in POPL 2016*.
[72] C. Li, D. Porto, A. Clement, J. Gehrke, N. M. Preguiça, and R. Rodrigues, “Making Geo-Replicated Systems Fast as Possible, Consistent when Necessary,” in *OSDI 2012.*
[73] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, and N. D. Lane, “Flower: A Friendly Federated Learning Research Framework,” CoRR, vol. abs/2007.14390, 2020.
[74] B. Ying, K. Yuan, H. Hu, Y. Chen, and W. Yin, “BlueFog: Make Decentralized Algorithms Practical for Optimization and Deep Learning,” CoRR, vol. abs/2111.04287, 2021.
[75] S. Zillner, D. Bisset, M. Milano, E. Curry, C. Södergård, and T. Tuikka, “Strategic Research, Innovation and Deployment Agenda: AI, Data and Robotics Partnership,” 2020.
[76] M. T. Ribeiro, S. Singh, and C. Guestrin, “‘Why Should I Trust You?’: Explaining the Predictions of Any Classifier,” in SIGKDD 2016.
[77] A. Alexandrova, L. Rapanotti, and I. Horrocks, “The legacy problem in government agencies: an exploratory study,” in DGO 2015.
[78] F. Freitas, J. Leitão, N. M. Preguiça, and R. Rodrigues, “Fine-Grained Consistency Upgrades for Online Services,” in SRDS 2017.
[79] J. C. Seco, P. Ferreira, H. Lourenço, C. Ferreira, and L. Ferrão, “Robust Contract Evolution in a TypeSafe MicroServices Architecture,” Art Sci. Eng. Program., vol. 4, no. 3, p. 10, 2020.

APPENDIX A

Questionnaire:

1. What is your preference in programming languages?
   [Multiple choice: Rust, TypeScript, Python, Java, Kotlin, C++, C#, MatLab, other (text field)]

2. What is your preference in IDE / code editor?
   [Options: IntelliJ, Eclipse, VS Code, Visual Studio, Sublime, vim/emacs]

3. Which frameworks (if any) do you prefer in your deployments?
   [text field]

4. For how many years have you been a programmer (paid or unpaid)?
   [number field]

5. Is some of your code public?
   [checkbox]

6. You are familiar with decentralised programming techniques (e.g. dApps).
   [Options Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree].

7. You are familiar with peer-to-peer systems. (e.g. Bittorrent, Kademlia, Chord, Gnutella)
   [options Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree].

8. You are familiar with blockchain technology. (e.g. Bitcoin, Ethereum, smart contracts)
   [options Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree].

9. You are familiar with Web3 architectures and existing Web3 services. (e.g. IPFS, libp2p)
   [options Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree].

10. You are familiar with distributed machine learning and deep learning (e.g. Federated learning).
    [Options Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree].

11. You are familiar with distributed storage (e.g. Azure Cloud, Amazon S3, Google Cloud Storage, Scality, IPFS).
    [Options Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree].

12. You are familiar with the details of networking protocols (e.g. IP, TCP, UDP).
    [Options Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree].

13. In case you have experience with decentralised architecture, what were your most important reasons for choosing it?
    (keywords are sufficient) [text field]

Funded by the European Union

14. Do you use different kinds of devices within your deployment?
    [multiple choice: server/VM, PC, notebook, smartphone, tablet, RaspberryPi/IoT, embedded controllers, other (textbox)]

15. Do you use any of the following networking paradigms within your deployments?
    [Multiple choice: Microservices, OnPremise/Edge, Fog, IoT, Serverless, Cloud, other (textbox)]

16. Pick one of your latest projects to gauge: latency was more important than bandwidth.
    [five choices]

17. Pick one of your latest projects to gauge: availability was more important than consistency.
    [Options: Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree].

18. Pick one of your latest projects to gauge: resilience was more important than performance. [Options: Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree].

19. Pick one of your latest projects to gauge: scalability was more important than simplicity.
    [Options: Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree].

20. If we are to build a decentralised programming toolbox, what are the top three features you are looking for?
    (keywords are sufficient) [text field]

21. It is important for you to have open-source development for a decentralised programming toolkit.
    [Options: Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree]

22. It is important for you to have community-driven development for a decentralised programming toolkit.
    [Options: Strongly Disagree, Disagree, Neither Agree nor Disagree, Agree, Strongly Agree]

23. Please leave here any other comments that you would like to share with us regarding the development of a decentralised programming toolkit.
    [text field]

Funded by the European Union