Validating Federated Learning Performance in Practice: An Agricultural Edge Hardware Testbed Analysis

A thesis

submitted partial fulfilment

of the requirements for the degree

of

Master of Engineering

at

The University of Waikato

by

Mahnoor Akhund

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2025

DECLARATION

I declare that the this thesis is the result of my own original research and work.

To the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due acknowledgement has been made in the text.

I further declare that no generative artificial intelligence (AI) tools were used for the conceptual development, analysis, or generation of the substantive content presented in this thesis. All core text, research findings, analysis, and conceptual development are solely my own work.

Use of assistance tools:

1. **Editing and proofreading:** For the purpose of improving grammatical correctness, spelling, punctuation, and clarity, I utilised the software tool Grammarly. The use of this tool was limited to editing and refining the text originally written by me.

2. **LaTeX formatting assistance:** The generative AI tool Google Gemini 2.0 Flash was used for technical assistance with LaTeX formatting. This assistance was limited to tasks such as correcting or suggesting LaTeX code for the presentation of tables and charts, and ensuring the correct usage of typographical elements (such as backticks “, quotation marks ""). This tool was not used to generate or alter the content, data, structure, or meaning within these elements or any other part of the thesis.

I also declare that all figures presented within this thesis were originally created by me using Microsoft PowerPoint. All charts presented within this thesis were generated by me using Microsoft Power BI, based on the original results obtained during my research activities for this work.

The original design files for figures and charts, as well as the underlying data used to generate the charts, can be made available upon request.

Abstract

Smart agriculture, driven by the Internet of Things (IoT) and artificial intelligence, generates vast amounts of data from distributed sensors and devices on farms. Traditional centralised machine learning approaches struggle in rural settings due to intermittent connectivity, limited bandwidth, and data privacy concerns. Federated learning (FL) has emerged as a promising approach to address these challenges by collaboratively training models directly on edge devices, for example farm sensors and edge computers, without sending raw data off-site. This thesis presents the design, development, and evaluation of a standards-driven, hardware-based federated learning testbed for smart agriculture. The testbed consists of six NVIDIA Jetson Nano edge computing nodes. A primary objective was to evaluate and compare modern, open-source FL frameworks in a realistic edge environment. Therefore, we specifically tested the FLIGHT framework. FLIGHT was selected for its notable features, including a Function-as-a-Service (FaaS) architecture facilitating serverless deployment and native support for hierarchical topologies relevant to distributed farm networks. We developed a reproducible methodology for deploying and benchmarking FLIGHT on these resource-constrained devices, using the Fashion-MNIST image classification dataset as a proxy for agricultural sensor data to compare performance against known benchmarks and simulation expectations. Key contributions include: 1) the physical testbed itself; 2) an open and repeatable experimental framework centered around FLIGHT; and 3) a comprehensive performance analysis under realistic network conditions and device constraints, providing data to bridge the gap between simulation results and practical deployment challenges. The results demonstrate that the federated model achieves competitive accuracy while significantly reducing raw data transfer. Findings indicate that careful configuration of FL can mitigate the impact of limited connectivity, and that even low-power devices can collaboratively train useful models within reasonable time-frames. These insights validate the viability of FL in smart farming scenarios. The developed testbed and its accompanying benchmarking methodology lay a foundation for future research and deployment of FL in agriculture, bridging the gap between theoretical simulations and real-world farm deployments. This work's significance lies in providing both a practical tool for researchers to rigorously evaluate FL strategies in edge environments and guidance for stakeholders aiming to deploy privacy-preserving AI in agriculture at scale.

Acknowledgements

I'm deeply grateful for all the support and guidance that made finishing this thesis possible.

First and foremost, I wish to express my heartfelt appreciation to my supervisors, Associate Professor Judy Bowen and Associate Professor Melanie Ooi. Their insightful feedback, guidance, and unwavering commitment have profoundly shaped my research journey. I have been incredibly fortunate to learn from their unique perspectives and diverse approaches to supervision - each contributing significantly to both the depth and breadth of my academic development.

This research journey was not undertaken in isolation, and I am grateful to have been part of a wider research community. I thank the "Rural AI"[1] team for welcoming me to their biweekly discussions with team members and affiliated researchers from the University of Chicago, University of Newcastle, University of Wisconsin, and Cardiff University. This provided invaluable insights and broadened my understanding of contemporary research in this field.

A special note of thanks goes to Dr. Nathaniel Hudson from University of Chicago. His generosity in helping me understand the intricacies of the FLIGHT framework, which is central to this thesis, was essential. His explanations of the software implementation significantly aided my own work on the hardware testbed.

Lastly, none of this would have been possible without the support of my family. Thank you for your belief in me and your patience throughout this rewarding process.

Each contribution has been instrumental to the completion of this thesis, and I am profoundly thankful for the collective support that made this achievement possible.

[1] `https://sites.google.com/view/rural-ai/partners`

Table of Contents

Table of Figures

Table of Tables

Chapter 1

Introduction

In the fields of tomorrow, sensors may outnumber seeds. This statement captures the essence of a transformation underway in global agriculture. Feeding a growing population, expected to reach nearly 10 billion people by 2050, presents one of humanity's most significant challenges [1]. Agriculture, the foundation of our food supply, must become substantially more productive and efficient, all while facing increasing pressures from climate change and resource limitations. In response to this critical need, farming is undergoing a technological revolution, often referred to as "Agriculture 4.0" [2, 3]. Much like how manufacturing has been transformed by digital technology (Industry 4.0), Agriculture 4.0 involves bringing advanced tools like sensors, connectivity, and data analysis into farm operations.

This can be imagined as fields equipped with numerous small sensors acting like digital eyes and ears, constantly monitoring conditions such as the moisture in the soil, local weather patterns, or even the health of individual plants using specialised cameras [4, 5]. This network of connected devices forms the Internet of Things (IoT) within agriculture. The data collected by these IoT sensors provides an incredibly detailed, real-time picture of what's happening on the farm. This wealth of information allows farmers to move towards precision agriculture, making highly targeted decisions instead of applying resources uniformly. For example, instead of watering an entire field the same amount, data can show exactly which sections need more or less water, allowing for precise application that saves water and potentially boosts crop yields [6]. Making sense of this vast amount of data often requires sophisticated computer analysis techniques, falling under the umbrella of Artificial Intelligence (AI) and Machine Learning (ML), where computer systems learn patterns from data to make predictions or recommendations.

However, implementing this vision of a data-driven "smart farm" faces significant practical hurdles, especially concerning the data itself. Firstly, many farms are located in rural areas where internet access can be slow, unreliable, or ex-

pensive [7–9]. The sheer volume of data generated by potentially hundreds or thousands of sensors can be enormous. Sending all this raw data continuously to a distant, powerful computer (a central server) for analysis, which is how the typical ML models are trained, might be technically difficult or financially prohibitive [10]. Secondly, the data generated on a farm is often highly sensitive. Information about crop yields, soil quality, disease outbreaks, or specific farming practices can be considered valuable business intelligence or private information that farmers may not wish to share openly [11–13]. Concerns about data privacy, security, and who ultimately controls the data create reluctance towards systems requiring data to be pooled in a central location.

To overcome these critical barriers of connectivity and data privacy, a different approach to ML called federated learning (FL) has emerged [14, 15]. Instead of collecting all the data in one place to train a single model, FL works in reverse. It sends a copy of the ML model out to the individual devices where the data is stored. These individual devices can be computers or gateways on the farm, also referred to as edge devices because they operate at the "edge" of the network, close to where the data is generated by sources like sensors. Each device then uses its own local data to train and improve its copy of the model. This means that the raw, sensitive data never leaves the local device. Only the learning updates, which are essentially summaries of what the model learned locally and therefore much smaller than the raw data, are sent back, typically to a central coordinator. The coordinator then combines these updates from many devices to create an improved, shared "global" model [16]. This process repeats, allowing collaborative learning across many locations without centralising the private data and significantly reducing the amount of information that needs to be transmitted over the network [10, 17].

While FL offers an elegant solution on paper, making it work effectively in the challenging environment of a real farm brings its own set of difficulties. The edge devices used on farms might have limited processing power, memory, and battery life compared to powerful servers [18]. Network connections, even if only needed for sending small updates, might still be intermittent or slow [7, 9]. Furthermore, the physical hardware must endure harsh conditions like extreme temperatures, dust, and moisture [7]. From a data perspective, information gathered from different farms or fields can be highly varied due to different soil types, weather, crops and sensor types. This is known technically as non-IID or non-identically and independently distributed data which can make it harder for the combined FL model to learn effectively [19, 20]. Security is also a concern; even though raw data isn't shared, measures might be needed to protect the model updates themselves from tampering or misuse [21, 22]. Moreover, simply managing the whole process, telling potentially hundreds of different farm devices when to train, collecting their results, and combining them reliably, is complex. This coordination problem becomes easier with modern computing approaches often called 'serverless'. These approaches allow tasks to run across many distributed devices automatically, without needing dedicated, constantly

running management servers, saving resources and simplifying the setup [23–25].

Recognising these practical complexities, research exploring FL for agriculture has increased. However, to understand how ready these solutions truly are for real-world use, we conducted our own Systematic Literature Review (SLR) (detailed in Section 2.4). Our analysis revealed that the vast majority (around 70%) of studies claiming to deploy FL in agriculture are still primarily tested using simulations or in controlled laboratory settings. These studies often operate at what are called Technology Readiness Levels (TRL) 4 to 6 which means the technology is proven in a lab or limited environment but not yet fully validated in its final, operational setting. Such controlled tests are useful starting points but often fail to account for the unpredictable nature of real farm conditions: the unstable internet, the power limits of actual devices, the wide variety of hardware used, and the day-to-day operational issues. Consequently, there is a significant lack of reliable, empirical data on how FL systems perform end-to-end when running on the type of physical hardware actually used in agricultural IoT.

This gap is critical because existing test environments used for FL research often don't mirror agricultural reality. They might use powerful computers acting as "clients", assume perfect network connections, or simulate many devices on one machine [9, 18]. These setups don't adequately test performance under the specific resource constraints (limited computing power, memory, energy) and communication challenges (low bandwidth, latency) found in agriculture. To truly assess FL's viability, we need to test it on hardware and networks that closely resemble those deployed on farms, measuring real-world performance metrics like processing load (CPU/GPU usage), memory requirements, power consumption, and the actual time it takes to communicate updates over realistic connections.

Therefore, this thesis identifies and aims to fill this gap: the need for a reproducible, physically-instantiated federated learning testbed specifically designed for evaluating edge computing performance in the context of smart agriculture. By "reproducible", we mean the testbed is built using commonly available hardware and open-source software, with its setup and experiments documented so that other researchers can easily replicate our work, verify the results, and build upon them. By "physically-instantiated", we emphasise that this testbed uses actual physical edge devices and network connections, not just simulations, allowing us to capture genuine hardware performance and limitations. The contribution of this work is the creation of such a platform: a serverless FL testbed designed to evaluate how FL performs under realistic agricultural edge conditions.

1.1 Scope and Objectives of the Thesis

The scope of this thesis encompasses the design, implementation, and empirical evaluation of the FL testbed. We concentrate on a representative use case of FL relevant to smart agriculture, namely, an image-based classification task that simulates a scenario like disease detection or crop type classification, using a proxy dataset (FashionMNIST) partitioned across edge nodes. By doing so, we exercise the full pipeline of an FL system, which includes data partitioning, local training, model aggregation etc., without the need for an active farm deployment at this initial stage.

It is important to delineate what is inside and outside the scope of our work. We focus on system-oriented questions: Can a network of low-power edge devices collaboratively train a model to high accuracy within a reasonable timeframe? What are the performance trade-offs in terms of communication overhead and computation load? How can we instrument and measure the system to evaluate these trade-offs? Consequently, the thesis is not about developing new FL algorithms or novel ML models for agriculture. We employ an established framework, FLIGHT, so that we can concentrate on the system performance and engineering aspects. Similarly, while the ultimate aim is to handle real sensor data, in this thesis we do not integrate live farm sensors into the loop; instead, we emulate edge data by storing dataset subsets on each device. This controlled setup allows us to test the system under known conditions.

The specific objectives of the study are as follows:

- **Objective 1: Testbed Development** :- Design and build a physical testbed for federated learning using edge devices. This includes configuring the hardware (edge nodes and server), setting up the networking, and deploying an FL software framework. The testbed should support instrumentation for collecting metrics like timing, bandwidth usage, and resource utilisation.
- **Objective 2: Experimental Evaluation** :- Execute a series of experiments on the testbed to evaluate key performance indicators such as model accuracy, communication overhead and system usage. We also aim to test different conditions, such as varying the number of participating clients and the proportion of the dataset used.
- **Objective 4: Analysis of Results** :- Analyse and interpret the experimental data to draw insights about the feasibility and efficiency of federated learning in an edge setting. Identify any bottlenecks and evaluate how well the system can meet the requirements for smart agriculture scenarios.
- **Objective 5: Recommendations and Future Roadmap** :- Based on the findings, formulate recommendations for deploying FL in real agricul-

tural environments and outline the next steps for research to improve the testbed.

By keeping the scope targeted to these objectives, the thesis ensures a deep focus on the federated learning testbed itself as a research contribution, rather than branching into all aspects of agricultural AI. The deliverables from this work include the physical testbed setup, the software artifacts (configuration scripts, code for running FL experiments), and a detailed evaluation that will be useful for others interested in edge-based federated learning.

1.2 Contributions

This research yields several key contributions to the domains of federated learning, edge computing, and smart agriculture.

1. **Development of a Completely Reproducible FL Testbed:** We present the architecture and design of a federated learning testbed composed of real hardware devices representative of farm IoT nodes. To the best of our knowledge, this is one of the first reproducible testbed specifically targeting FL in a smart agriculture context. The design accounts for the constraints of rural deployment and provides a template that others can replicate or adapt for similar use cases.

2. **Comprehensive Empirical Evaluation & Recommendations for Deployment in Agriculture:** We conduct a thorough experimental evaluation of the testbed's performance. These results provide quantitative evidence of the feasibility and constraints of FL on edge hardware. Drawing from the testbed results and experience, we formulate concrete recommendations for practitioners and stakeholders in agriculture technology. These recommendations aim to bridge the gap between experimental research and real-world deployment.

3. **Foundation for Future Research:** We conduct the first comprehensive systematic review that evaluates existing deployable FL solutions, establishing a definitive baseline of current capabilities and limitations.

Overall, these contributions advance the practical infrastructure for FL research. The work not only demonstrates a working solution but also provides insights and tools that others can use to accelerate innovation in privacy-preserving, distributed AI for resource-constrained environments.

1.3 Thesis Structure

The structure of this thesis is as follows:

In **Chapter 2**, the background and related work establishes the foundational context for this investigation by tracing Agriculture 4.0's evolution and key technologies. It introduces federated learning (FL), covering its principles and challenges in agriculture. A systematic literature review (SLR) of FL deployments in smart farming follows. This synthesis uncovers the research gap that drives this thesis, laying the groundwork for designing and evaluating the proposed FL testbed.

In **Chapter 3**, the design of the federated learning testbed covers the decision of the FL architecture, the specific hardware and software components selected, and elaborates on the core design principles of reproducibility and scalability that guided its development.

In **Chapter 4**, the implementation and deployment describes the simulation and practical realisation of the designed testbed covering the systematic setup and configuration of the virtual machines and Jetson Nano.

In **Chapter 5**, the benchmarking methodologies defines the evaluation framework, specifying key performance metrics, justifying their selection over alternatives, and highlighting the advantages of hardware-based evaluation.

In **Chapter 6**, the results, analysis, and discussion presents the empirical findings from the simulation and testbed experiments, discussing the practical implications for smart agriculture deployments.

In **Chapter 7**, the conclusion and future work summarises the thesis contributions, discusses broader implications and recommendations, acknowledges limitations, and proposes avenues for future research to further advance FL deployment in edge environments.

Chapter 2

Background and Related Works

This chapter lays the groundwork for the thesis by exploring the context of smart agriculture, the rise of relevant technologies, the potential and challenges of federated learning (FL) within this domain, and the current state of research, ultimately identifying the specific gaps this work aims to address.

2.1 The Smart Agriculture Landscape

The agricultural sector is undergoing a profound transformation driven by digital technologies, often referred to as Agriculture 4.0. This section explores the foundations of this shift, its evolution, and the key technologies enabling it.

2.1.1 The Industry 4.0 Influence and Agriculture 4.0 Concept

Industry 4.0, the fourth industrial revolution, is characterised by the convergence of cyber-physical systems (CPS), the Internet of Things (IoT), advanced data analytics, and artificial intelligence (AI) within industrial processes [26,27]. In manufacturing, this model underpins "smart factories", facilitating real-time data exchange, optimised decision-making, and increased automation [28]. Beyond efficiency gains, Industry 4.0 offers potential for broad economic, environmental, and social impacts by significantly improving automation and operational intelligence across various sectors [29].

These foundational concepts, represented in Figure 2.1, are increasingly being translated into the agricultural domain, resulting in the Agriculture 4.0 framework [2, 3]. This framework applies core Industry 4.0 technologies to farming practices with the primary objective of creating "smart farms", where technology is strategically deployed to improve both productivity and environmental

performance [30]. Specifically, IoT facilitates ubiquitous sensing, gathering extensive real-time data from diverse sources like soil sensors, weather stations, drones, and GPS-enabled machinery, while also providing the essential connectivity for data transmission. Concurrently, AI processes this data, often leveraging big data analytics, to drive intelligent analytics for more informed, predictive, and optimised decision-making. Together, these interconnected elements provide farmers with significantly improved capabilities for real-time monitoring and control to advanced automation of farming tasks—ranging from precision planting and irrigation to autonomous vehicle operation [31–33]. A key practical outcome is optimised resource allocation; for instance, combining sensor data with AI algorithms allows targeted application of inputs like water and fertiliser and improving resource-use efficiency [6].

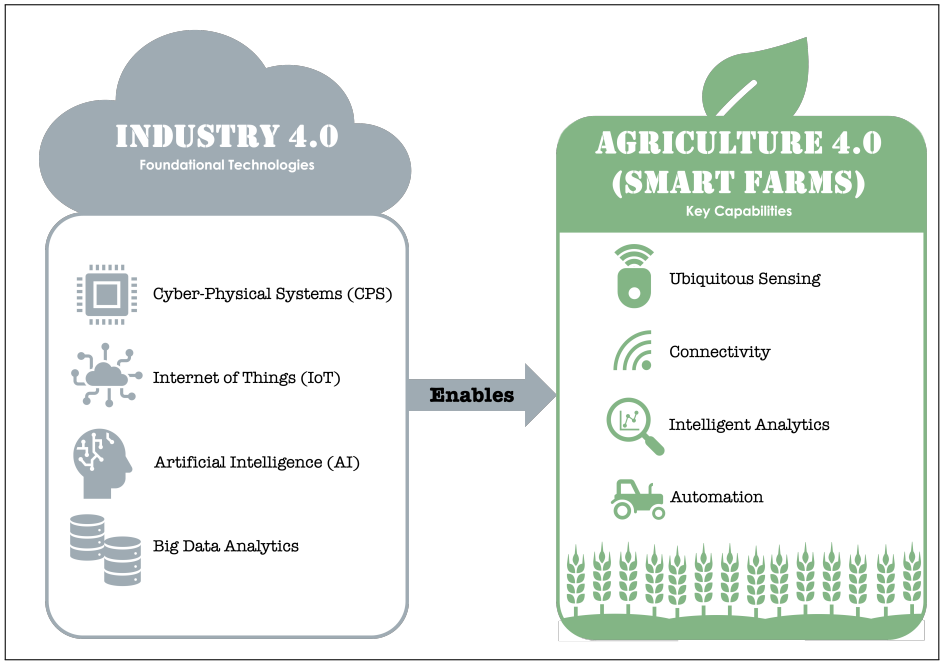


Figure 2.1: Conceptual diagram illustrating key foundational technologies of Industry 4.0 (left) and how they translate into the components of Agriculture 4.0 (right), leading to the development of "Smart Farms".

Such advancements are pertinent given significant global pressures, including the projected need to increase 70% of food production by 2050 to meet growing population demands [34]. Consequently, Industry 4.0's application in agriculture offers a technological framework to address challenges such as food security, resource scarcity, and climate change adaptation.

Nevertheless, adopting Agriculture 4.0 entails considerable socio-economic chal-

lenges and implications [35, 36]. It requires substantial capital investment and necessitates workforce adaptation, demanding skills in data literacy, digital technologies, and technical management. Supporting the existing agricultural workforce, often characterised by a higher average age [37], in acquiring these skills is a key consideration. Conversely, Agriculture 4.0 may attract new talent to the sector and stimulate rural economic development. Recognising these factors, governments and industry organisations are implementing initiatives, including national strategies and funding programs, to support digital innovation in agriculture [38, 39].

In essence, the application of Industry 4.0 principles is fostering Agriculture 4.0, an era of farming characterised by data-centricity, intelligence, and connectivity. Although this transformation presents significant potential for improved productivity and sustainability, its successful realisation depends on the effective management of the associated technical, economic, and human resource challenges.

2.1.2 Evolution and Key Digital Trends in Farming

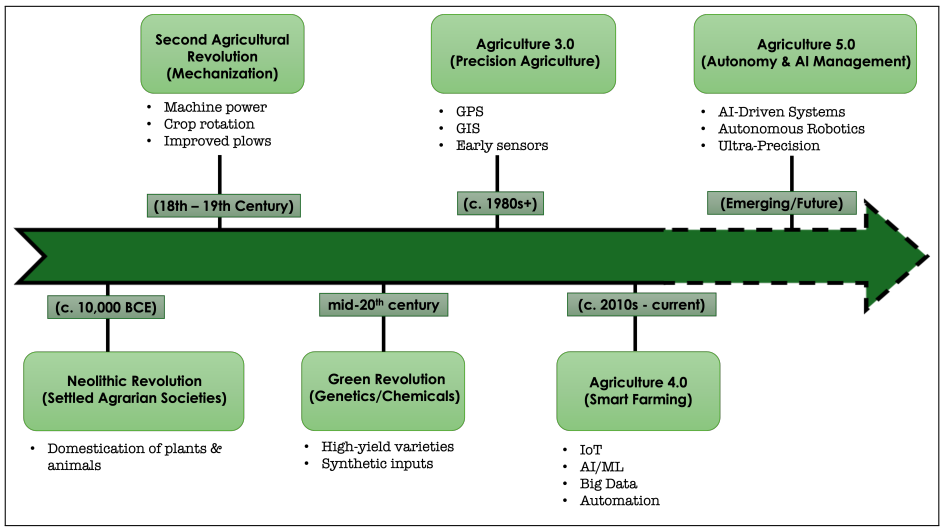


Figure 2.2: Timeline of major agricultural revolutions

Historically, agriculture has undergone transformative revolutions, each improving productivity and reshaping societal structures. The Neolithic Revolution initiated settled agrarian societies [40]. The Second Agricultural Revolution, starting in the 18th century, introduced mechanisation, amplifying output via machine power [41, 42]. The mid-20th century Green Revolution deployed high-yield varieties, synthetic inputs, and advanced breeding techniques, dramatically

increasing global crop production [43]. This historical progression through distinct agricultural eras is visually summarised in Figure 2.2, which outlines the major revolutions alongside their approximate timelines and defining characteristics.

Currently, agriculture is experiencing the Fourth Agricultural Revolution (Agriculture 4.0) [44]. This phase represents a significant shift, transforming agriculture into a technology-intensive sector leveraging computational power and connectivity. This digital transition evolved progressively. The 1980s-1990s saw the rise of precision agriculture (sometimes termed Agriculture 3.0), facilitated by personal computers, Geographic Information Systems (GIS), and Global Positioning Systems (GPS) [45]. These tools allowed spatially variable input application and yield mapping, marking the initial integration of sensor data [46]. The 2000s brought improved remote sensing through satellites and UAVs [47, 48] and improved telecommunications infrastructure, allowing more granular monitoring and increasing farmer access to real-time information and management software [49]. By the 2010s, convergence with cloud computing and the IoT established the basis for Agriculture 4.0. Digital farming platforms emerged, offering integrated data management solutions from sensor acquisition to cloud storage and analytics, often employing machine learning for decision support [50]. Examples include distributed sensor networks informing automated irrigation optimisation [51, 52]. Factors accelerating this phase include declining technology costs and increasing emphasis on sustainable production efficiency [53].

Concurrently, the role of data in agriculture has evolved from sparse collection points to continuous Big Data streams [54]. Data is increasingly recognised as a critical operational asset, which simultaneously raises significant concerns regarding ownership, privacy, and security [11]. Furthermore, concepts such as Agriculture 5.0 are emerging, envisioning future highly autonomous, AI-driven farming systems [55]. This next phase anticipates AI managing operations, potentially using robotic fleets for tasks like weeding or harvesting to address labor shortages and drive ultra-precision agriculture, with preliminary examples in controlled environment agriculture (CEA) and autonomous farm research [56, 57].

In brevity, the trajectory of agricultural innovation shows a shift from primarily mechanical and chemical advancements towards information-centric technologies. Agriculture 4.0 represents the current integration of physical and digital realms in farming. The consistent objective across these revolutions is producing more food with greater efficiency and sustainability. Agriculture 4.0, leveraging IoT and AI, offers significant potential towards this goal, but also generates vast amounts of distributed data requiring new management and analysis paradigms.

2.1.3 Enabling Technologies: IoT, Edge, and Serverless Systems

The realisation of smart agriculture is predicated on a suite of enabling technologies that facilitate data acquisition, communication, processing, and actuation within the farm environment. These technologies generate the distributed data streams characteristic of Agriculture 4.0 and provide mechanisms for managing the associated computational workloads.

The Internet of Things (IoT) and Edge Devices: The Sensory Nervous System

The Internet of Things (IoT) constitutes a cornerstone technology for smart agriculture. It encompasses networks of interconnected physical devices embedded with sensors, actuators, and communication interfaces, transforming diverse farming operations into quantifiable data streams and controllable processes [58]. These devices span a wide spectrum, including environmental sensors measuring parameters like soil moisture, temperature, humidity, and photosynthetically active radiation (PAR), alongside more complex systems such as GPS-enabled livestock tracking collars or machine vision systems for detecting crop pests and diseases [4, 5]. The strategic deployment of such distributed IoT devices grants farmers unprecedented, fine-grained, real-time visibility into environmental and operational conditions, overcoming previous limitations in continuous monitoring. This adoption trend is projected to continue, driven by decreasing technology costs and perceived benefits [59].

A prevalent architectural pattern for agricultural IoT involves multiple tiers. At the base, sensor nodes are distributed throughout fields or facilities to acquire data. These nodes are often battery-powered and utilise low-power wide-area network (LPWAN) technologies, for example LoRaWAN, NB-IoT, Sigfox [60], or shorter-range protocols, like Zigbee and Bluetooth Low-Energy (BLE) [61], for wireless communication to an edge device or gateway. This gateway typically has greater computational resources and serves critical functions: aggregating data from numerous sensor nodes, performing preliminary local processing via edge computing, and relaying processed or summarised data to remote cloud platforms [62]. This layered topology improves efficiency by obviating the need for direct internet connectivity for every sensor and allowing optimised data routing.

Complementing these fixed installations, mobile platforms like Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs) have also become integral components of the agricultural IoT ecosystem [63, 64]. UAVs equipped with diverse imaging payloads, like RGB, multispectral and thermal cameras, can rapidly survey large areas, providing valuable spatial insights into crop health variability, stress detection (water or nutrient deficiencies), disease outbreaks, plant counts, and biomass estimation. Integrating these aerial obser-

vations with data from ground-based IoT sensors assists powerful, multi-scale monitoring. For instance, thermal UAV imagery might identify localised potential water stress, prompting queries to proximal soil moisture sensors for confirmation before triggering targeted automated irrigation.

Within this framework, edge computing, processing data close to its source rather than relying solely on centralised cloud infrastructure [65], is critical in rural agricultural settings, where connectivity is often unreliable and bandwidth limited [8]. Whether deployed on gateways, on-farm servers, or embedded microcontrollers, edge systems allow real-time analytics and decision-making without cloud-induced delays. For example, an edge controller can instantly open an irrigation valve when soil moisture falls below a threshold, bypassing the latency of a cloud-dependent loop. This approach also conserves network resources by transmitting only summarised alerts rather than full-resolution data. Similarly, lightweight convolutional neural networks can run directly on edge sensors to identify pests in camera imagery and send only counts or event notifications upstream [66].

The integration of distributed IoT sensing with edge computing yields agricultural systems that are both more resilient and more responsive. By treating each edge device as an autonomous intelligent node, farmers gain holistic control over operations without constant cloud dependence. For instance, a greenhouse's edge computer can autonomously regulate temperature, humidity, and ventilation based solely on local sensor data, continuing to operate even during extended connectivity outages. In such remote or intermittently connected environments, edge nodes can buffer new measurements, perform local model inference, and then synchronise buffered data or fetch updated models once network access is restored. This model of local autonomy aligns perfectly with federated learning's distributed computation approach (discussed in Section 2.2), as it leverages existing edge infrastructure to train and update models directly on-device.

Beyond technological enablement, the application of IoT and edge devices yields demonstrable practical benefits. Farmers employing these techniques report improvements in crop yields, often estimated between 5-20% [67], alongside significant reductions in input costs. For instance, IoT-guided variable-rate irrigation can cut water consumption substantially, with reported savings around 30% [68], also reducing associated energy costs. Aggregated across large scales, these efficiencies contribute markedly to farm profitability and resource sustainability. Moreover, the rich data streams generated by IoT networks are crucial inputs for developing predictive models, such as forecasting pest outbreaks, disease risks, or optimal harvest windows, enabling proactive management strategies that mitigate potential losses.

In short, IoT deployments and edge computing capabilities constitute the sensory and peripheral processing infrastructure of Agriculture 4.0. Sensors act

as the distributed "eyes and ears", edge devices provide local intelligence and rapid "reflexes," and connectivity integrates these components into a comprehensive "farm brain" capable of optimising operations. Figure 2.3 provides a conceptual illustration of this 'farm nervous system'. As this technological nervous system becomes increasingly sophisticated, incorporating mobile sensing platforms (drones, robots) and advanced edge AI capabilities, the potential for greater autonomy and precision in farming expands significantly. IoT is widely recognised as a primary driver of the vast agricultural datasets that underpin advanced analytical techniques, like machine learning and, pertinent to this thesis, federated learning.

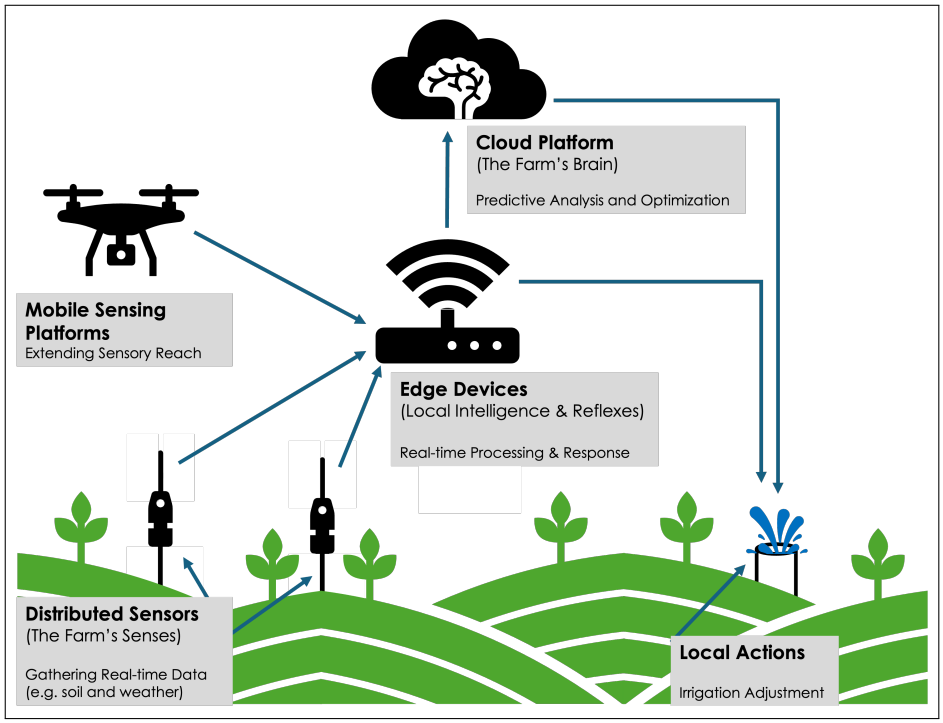


Figure 2.3: The farm's "Nervous System" analogy: 'Senses' inform edge 'reflexes' for local processing/action, and the cloud 'brain' for strategic analysis and learning.

Serverless Computing for Distributed Agricultural Environments

The proliferation of distributed sensors and edge nodes in smart agriculture adds significant complexity to managing computation across geographically dispersed, heterogeneous resources. Serverless computing, especially the Function-as-a-Service (FaaS) model, offers a compelling paradigm for orchestrating these distributed tasks [23]. In FaaS, developers package their logic as discrete, stateless functions that the platform invokes on-demand, triggered by events such as

incoming sensor data, scheduled intervals, or API calls, while transparently handling resource provisioning, dynamic scaling, and infrastructure maintenance [24]. This event-driven, scale-to-zero capability aligns perfectly with the intermittent, bursty workloads typical of agricultural IoT to allow efficient resource utilisation and simplified orchestration.

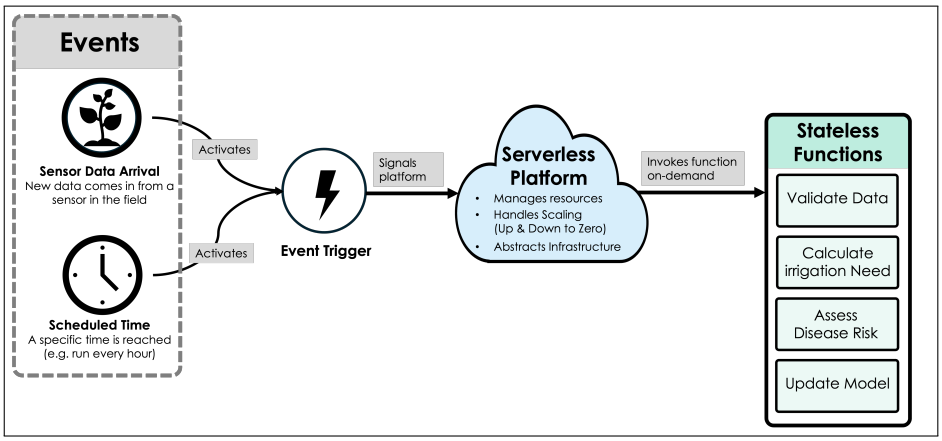


Figure 2.4: A typical event-driven serverless (FaaS) workflow

Translating this model to the smart agriculture context offers practical benefits for streamlining data processing and analytics. For example, the arrival of aggregated sensor data at a gateway can act as an event automatically triggering a specific serverless function. This function might then perform necessary tasks like data cleaning, validation, updating a local predictive model, or issuing alerts based on predefined conditions. Such event-driven execution eliminates the need for continuously running server processes on potentially resource-limited edge devices. This results in conserving power and bandwidth as functions consume resources only during active execution. Furthermore, this approach naturally promotes modular software design as distinct agricultural tasks, for example `calculate_irrigation_need` or `assess_disease_risk`, can be developed as independent functions, simplifying development and maintenance while the framework manages their execution. Figure 2.4 provides a conceptual overview of this event-driven serverless workflow.

While major cloud providers offer mature serverless platforms, like AWS Lambda, Azure Functions, Google Cloud Functions [24], their typical reliance on stable, low-latency connections to central data centres can be problematic for many agricultural settings. The dispersed geography and potential network limitations often necessitate frameworks that effectively extend serverless principles to the edge. Globus Compute (formerly funcX) provides a relevant example, engineered specifically for distributed scientific computing, including edge scenarios [25]. It operates using a hybrid architecture: a central cloud service

coordinates task dispatch, while lightweight software endpoints, deployed by users onto their distributed resources (such as farm gateways or local servers), execute the functions locally. Critically for agricultural deployments often facing network restrictions like network address translation (NAT), which lets multiple private-network hosts share a single public IP address, and firewalls, Globus Compute endpoints establish outgoing connections to the cloud service. This design allows the central service to route tasks to the endpoints without requiring direct inbound access, overcoming common connectivity hurdles.

Adopting edge-native serverless computing, facilitated by platforms like Globus Compute, yields several advantages. It decouples application logic from the specifics of underlying hardware, allowing the same function, such as sensor data analysis, to potentially run across diverse edge devices. The framework also inherently improves reliability; tasks intended for a temporarily offline endpoint can be queued and executed upon reconnection, or potentially rerouted if redundancy is configured. Moreover, it provides crucial elasticity. During peak activity, such as harvest season generating high data volumes, the platform can automatically scale function instances across available resources, then scale down during quieter periods. This optimises computational resource use, a vital consideration in energy-conscious agricultural operations.

The practical efficacy of applying serverless concepts to enable AI in remote agriculture is exemplified by the "Rural AI" work of Patros et al. [69]. They successfully employed serverless functions (FaaS) to orchestrate a federated learning pipeline for pasture weed detection across distributed farm nodes. This demonstrated how the serverless model can manage complex, distributed workflows like FL without requiring persistent server processes at each edge location. Thereby, they managed to efficiently utilise network and compute resources only when actively needed, even in environments with constrained connectivity. However, while serverless platforms adeptly manage execution orchestration, distributed learning paradigms like federated learning inherently involve substantial data movement, such as transferring model parameters or gradients, which can strain limited rural bandwidth. Addressing this specific challenge requires complementary tooling. ProxyStore [70] represents such a solution, designed explicitly to mitigate data transfer bottlenecks in distributed environments. It achieves this by providing transparent object proxies, allowing large data objects to be passed by reference instead of by value between functions, even those executing on different physical nodes. Significantly, ProxyStore facilitates direct peer-to-peer data transfer between endpoints where possible, bypassing the need to route large data payloads through a central intermediary. This mechanism drastically reduces communication overhead and end-to-end latency, making it a considerable advantage in bandwidth-limited agricultural settings. For instance, feature vectors extracted from drone imagery by a function on one edge device could be transferred directly via ProxyStore to a training function running on a separate, perhaps more powerful, node.

Beyond computation and data logistics, serverless computing at the edge also inherently improves system extensibility and maintainability in agricultural deployments. Introducing new analytical capabilities, such as a frost prediction algorithm triggered by specific sensor readings, can often be accomplished by deploying a new, self-contained function without requiring disruptive changes to the core system architecture. This modularity has potential to provide service providers or cooperatives the capacity to offer specialised "function services", for example a pest monitoring service, deployable on-demand to subscribed farms' edge devices, imposing resource costs only when active. This on-demand deployment and execution model is a hallmark advantage of serverless computing.

The FLIGHT framework offers a compelling synthesis of these technologies for scalable federated learning [71]. It strategically leverages Globus Compute for its robust distributed function execution capabilities and integrates ProxyStore to manage inter-node data logistics efficiently. Within FLIGHT, distinct federated learning steps, such as local model training and subsequent aggregation, are mapped onto serverless functions. Globus Compute dispatches these functions to designated client endpoints. ProxyStore then facilitates the efficient collection of resulting model updates by reference, allowing direct transfer to the aggregation point. This modular, event-driven design inherently supports scalability; expanding the system involves executing the same functions across more endpoints, with the underlying platform managing concurrency and resource allocation. Validating this approach, experimental evaluations have shown FLIGHT effectively supports large-scale federated learning, achieving significant performance gains through parallelisation and optimised communication [71].

To summarise, serverless computing offers an operational model highly compatible with the distributed, dynamic, and often resource-constrained nature characteristic of smart agriculture systems. It effectively abstracts the complexity of coordinating computation across diverse nodes, optimises resource utilisation through on-demand execution, and provides inherent scalability. When combined with specialised platforms like Globus Compute for edge execution and tools like ProxyStore for efficient data handling, it allows the construction and management of robust distributed analytics and advanced federated learning systems more effectively. As agriculture continues to integrate sophisticated IoT and AI capabilities, serverless approaches are positioned to become crucial components of the software infrastructure, supporting complex computations to be deployed and executed seamlessly within the farm environment itself.

2.2 Federated Learning: A Potential Solution

The distributed nature of data generation in smart agriculture, coupled with privacy concerns and communication limitations, necessitates machine learning approaches that differ from traditional centralised methods. Federated Learning (FL) emerges as an alternative approach designed specifically for such scenarios.

2.2.1 Fundamental Concepts and Core Architecture

In contrast to traditional centralised machine learning, which requires consolidating datasets from diverse origins onto a central server for model training, FL inverts this flow by bringing the model training process to the data's location [14, 15]. A typical FL system architecture involves a central coordinating server, often termed an orchestrator, and multiple distributed clients, such as edge devices, sensors, or distinct organisational entities like farms, each possessing its own local dataset. The canonical FL process, frequently implemented synchronously, unfolds iteratively through several key stages.

Initially, the central coordinator defines and initialises a global machine learning model, establishing its architecture and initial parameters, which might be randomly set or derived from pre-training. Subsequently, the coordinator distributes these current global model parameters to a selected subset of participating clients. The strategy for selecting clients in each round can vary, often considering factors like device availability, resource constraints (computation, power, bandwidth), and specific sampling criteria designed to ensure representative participation and system scalability [72].

Once selected, each client receives the global model parameters and proceeds with local training, utilising only its private dataset. This local training phase typically involves executing a predefined number of optimisation steps, such as several epochs of stochastic gradient descent, to update the model parameters based on the unique characteristics reflected in the local data. Critically, throughout this entire local training process, the raw training data remains securely confined to the client device. Following the completion of local training, each client computes a model update. This update might consist of the fully updated local model parameters, the difference (delta) between the updated and initial parameters, or the gradients calculated during local optimisation. Only this computed update, representing the learning derived from the local data, is transmitted back to the central coordinator; the raw data itself is never shared. Finally, the coordinator performs the crucial aggregation step. It gathers the model updates received from the participating clients within the round and combines them to produce an improved iteration of the global model. The most widely adopted aggregation algorithm is Federated Averaging (FedAvg) [16]. In FedAvg, the server computes a weighted average of the client model parameters (or updates), typically weighting each client's contribution proportionally to the size of its local dataset. This aggregation effectively synthesises the collective knowledge gained across all participating clients during that round.

This cycle of distribution, local training, update communication, and aggregation constitutes one round of federated learning. The process repeats iteratively, with the newly aggregated global model serving as the starting point for the next

round, continuing until predefined stopping criteria are satisfied. These criteria might include reaching a target number of communication rounds, observing convergence in the global model's performance on a held-out validation set, or achieving a desired level of accuracy.

This standard synchronous, client-server architecture, often depicted as a star topology [73], fundamentally ensures data locality. By keeping training data resident on client devices, FL inherently mitigates significant data privacy risks associated with central data collection. Furthermore, it drastically reduces communication overhead compared to centralised approaches, as only model parameters or updates, typically much smaller in size than raw datasets, are exchanged over the network. These inherent characteristics make FL particularly appealing for sectors like agriculture, where farm data may be considered sensitive or proprietary, and where network bandwidth in rural areas can often be constrained.

Several architectural variations extend the basic FL model. Asynchronous Federated Learning [74], for instance, relaxes the strict synchronisation requirement. In this variant, clients can submit their updates to the server as soon as local training is complete, rather than waiting for all others in the round. The server then integrates these updates immediately, potentially employing techniques to account for model staleness (using older versions of the global model). While asynchronous methods can improve efficiency and mitigate the "straggler problem", where the round duration is dictated by the slowest client, especially in heterogeneous environments with varying client speeds and availability, they often introduce greater complexity into the aggregation logic.

Further distinctions arise based on the nature of the participating clients and how data is distributed among them [75]. Cross-device FL typically involves a very large number of potentially resource-constrained devices, like mobile phones or IoT sensors, where each device holds a relatively small dataset and might participate sporadically. This scenario places a strong emphasis on massive scalability, robustness to client dropouts, and effectively handling highly non-Identical and Independently Distributed (non-IID) data distributions across devices. Conversely, cross-silo FL involves a smaller cohort of more reliable clients, often representing organisations or institutions such as different farms or research centres, each possessing a substantial dataset. Cross-silo settings might involve navigating stronger institutional trust boundaries but often benefit from more structured data and reliable participation.

Complementing these categories, FL methodologies are also classified by how data features and samples are partitioned across clients [76]. Horizontal Federated Learning (HFL) applies when clients share the same feature space but hold different data samples or records. For instance, multiple farms collecting the same types of sensor measurements for their unique fields. This HFL scenario is the most common context and the primary focus assumed within this the-

sis. Vertical Federated Learning (VFL) addresses situations where clients hold data pertaining to the same set of samples but possess different sets of features. For example, one entity holding soil composition data while another holds corresponding yield data for the exact same fields. Federated Transfer Learning (FTL) tackles more complex scenarios involving only partial overlaps in both data samples and feature sets, often employing knowledge transfer techniques to bridge the gaps.

Addressing the nuances of distributed training, the choice of aggregation algorithm represents another critical architectural decision point beyond the foundational FedAvg [16, 77]. Numerous alternatives have been proposed to tackle specific challenges inherent in FL. FedProx, for example, modifies the local client objective function by adding a proximal term, which penalises large deviations from the current global model during local training [78]. This regularisation helps improve stability and convergence, particularly when dealing with significant statistical heterogeneity (non-IID data) across clients. Other sophisticated algorithms like FedNova, FedOpt, and various robust aggregation techniques aim to address issues such as client drift (where local models diverge significantly), statistical and system heterogeneity, communication bottlenecks, or potential security vulnerabilities like data poisoning attacks [16, 78]. Additionally, Secure Aggregation protocols represent a crucial enhancement, employing cryptographic techniques to allow the server to compute the sum or average of client updates without being able to decrypt or inspect any individual client's contribution, further strengthening privacy guarantees [72].

Moreover, to better align with real-world network structures or scale to vast numbers of clients, the basic star topology may be insufficient. Hierarchical Federated Learning (often also abbreviated HFL, requiring context to distinguish from Horizontal FL, sometimes termed tiered FL) addresses this by introducing intermediate levels of aggregation [79]. In this architecture, devices within a specific geographic region, organisational unit, or edge network might first aggregate their updates at a local or edge server. These intermediate aggregators then act as clients in a higher-level aggregation process coordinated by a central server. Such hierarchical structures can better mirror physical network layouts (e.g., sensors on a farm aggregating at a farm gateway, which then participates in aggregation with other farms at a regional server), potentially reducing communication latency and bandwidth usage over long-haul networks while enabling faster convergence of models tailored to specific regions or sub-groups. This approach represents one key FL topology, visually compared alongside centralised learning and other FL architectures like client-server and decentralised networks in figure 2.5. While centralised learning sends raw data to a central server, FL topologies send model updates upwards: directly in a Client-Server model, through intermediate stages in a Hierarchical setup, or exchanged between peers in a Decentralised model.

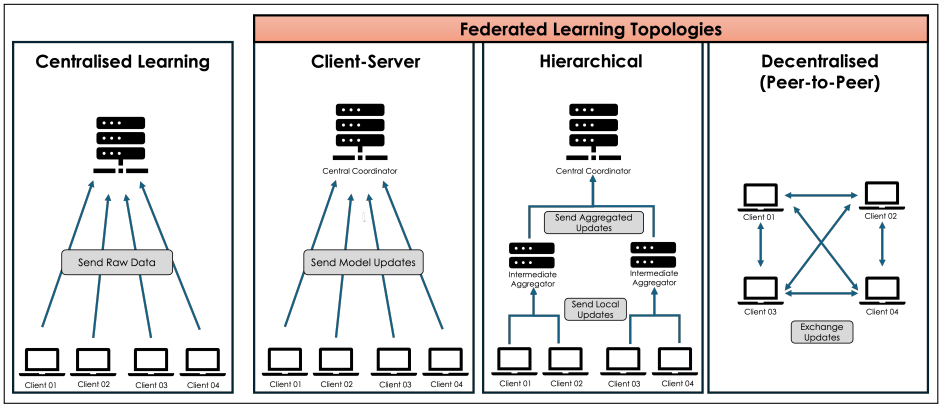


Figure 2.5: Upward information transfer in learning architectures.

In essence, the fundamental architecture of Federated Learning, whether standard client-server or hierarchical, facilitates collaborative model development across distributed data repositories under central coordination, without centralising the raw data itself. By shifting the approach from "data-to-code" (moving all data to a central algorithm) to "code-to-data" (sending the algorithm/model out to the data's location for local training) and restricting data movement primarily to model updates, FL offers inherent advantages in privacy preservation and communication efficiency.

2.2.2 Applications and Privacy Benefits in Agriculture

Initially popularised through applications like Google's training of predictive keyboard models directly on user smartphones without uploading sensitive typing data [80], FL exemplifies a powerful approach to privacy-preserving machine learning (PPML). The impetus behind its growing relevance stems from several compelling advantages, particularly relevant to the agricultural sector.

The most significant driver for FL adoption is the inherent improvements of data privacy and security. By design, FL ensures that raw data remains localised on client devices throughout the training process. This fundamental characteristic directly addresses pressing privacy concerns and aids organisations in complying with stringent data protection regulations, such as the GDPR in Europe or HIPAA in healthcare contexts [17]. While the nature of agricultural data differs from personal health information, data pertaining to farm operations, such as yields, soil conditions and pest prevalence, is often considered highly sensitive business intelligence or may be subject to data sovereignty rules [12]. Federated learning provides a mechanism for collaborative analytics, even among competing farms or within agricultural cooperatives, while allowing individual data owners to retain full control over their proprietary information. This fosters a greater willingness among stakeholders to participate in valuable collective data

initiatives compared to approaches requiring centralised data pooling [81].

Complementing its privacy advantages, FL offers benefits in communication efficiency. Transmitting model updates, which typically consist of parameters or gradients, generally requires significantly less network bandwidth than transferring entire raw datasets [10]. This reduction is particularly impactful when dealing with large data modalities common in modern agriculture, such as high-resolution aerial or satellite imagery, or extensive time-series logs from dense sensor networks. The communication efficiency of FL is especially critical in agricultural settings where rural internet connectivity can be limited, unreliable, or costly. Furthermore, techniques like model compression or update sparsification can be employed to further minimise the communication overhead associated with exchanging model updates [82, 83].

FL also aligns naturally with the accelerating trend towards edge computing. By distributing the computationally intensive task of model training across participating edge devices (the clients), FL leverages latent compute capacity available at the network edge [10]. This distribution can alleviate potential bottlenecks at central servers and effectively transforms the challenge of managing vast amounts of edge-generated data into an opportunity for localised, distributed processing. Moreover, the distributed nature of FL inherently contributes to improved system robustness and fault tolerance [84]. The failure or temporary unavailability of individual clients, a common occurrence in real-world deployments with potentially unreliable devices or network connections, typically does not halt the entire training process. The system can dynamically continue learning from the remaining pool of active clients. This inherent resilience is particularly advantageous in agricultural environments. Local training might even proceed during periods of network disconnection, with clients synchronising their computed updates once connectivity is restored.

Beyond technical efficiencies, FL acts as a enabler for collaboration across organisational or competitive boundaries. It provides a framework for entities that are unable or unwilling to share raw data, due to competitive sensitivities, intellectual property concerns, or regulatory restrictions, to nonetheless collaborate on building superior machine learning models. Well-documented examples include hospitals jointly training diagnostic models without sharing patient records [85, 86]. In agriculture, this translates to facilitating research institutions, farming cooperatives, or even individual farms to build shared predictive models, such as for forecasting pest outbreaks across diverse regions or optimising resource use, without exposing their sensitive operational datasets [13].

Finally, there is potential for FL models to achieve better generalisation compared to models trained solely on data from a single source [87]. By learning from diverse datasets residing in varied environments, representing different soil types, microclimates, and management strategies, the aggregated global model is implicitly exposed to a broader range of data distributions. While careful

algorithm design is needed to handle heterogeneity effectively, FL can act as a form of implicit regularisation, potentially preventing overfitting to specific local conditions. Some studies report FL achieving accuracy comparable to centralised training [88, 89].

These compelling benefits translate into a growing number of promising applications for federated learning within the domain of smart agriculture. FL is being explored for training robust image recognition models for crop disease and pest detection using distributed image datasets collected from various farms, without requiring the sharing of potentially sensitive imagery of afflicted crops [81, 90]. Similarly, it allows the development of improved yield prediction models by leveraging sensitive historical yield data combined with real-time sensor readings from multiple participating farms. Other potential applications include building shared models for soil health analytics based on distributed sensor data, training models for early detection of health issues in livestock using sensor data from different herds, and creating collectively optimised models for resource management, such as precision irrigation or fertilisation strategies informed by data from diverse field conditions [91] [92]. Extending beyond the farm gate, FL could also facilitate collaboration across the agricultural supply chain, allowing stakeholders like farmers, processors, and distributors to train models for demand forecasting or logistics optimisation without revealing proprietary operational details.

It is crucial, however, to maintain a balanced perspective. While FL offers a significant improvement in privacy compared to conventional data centralisation, it is not an infallible solution. Potential vulnerabilities remain, which are discussed further in Section 2.3.

2.3 Hurdles to Implementation: Challenges in Practice

While FL offers compelling theoretical advantages, its practical implementation, particularly in the complex and variable domain of smart agriculture, faces significant challenges. These hurdles stem from the distributed nature of the approach, the heterogeneity of data and systems involved, and the intricacies of ensuring robust and secure operation. Successfully navigating these challenges is critical for realising the potential of FL in agriculture.

2.3.1 Federated Learning Implementation Challenges

These challenges are inherent to the FL process itself when applied to real-world, decentralised environments. Figure 2.6 provides a visual overview of these primary challenge categories, which are discussed in detail below.

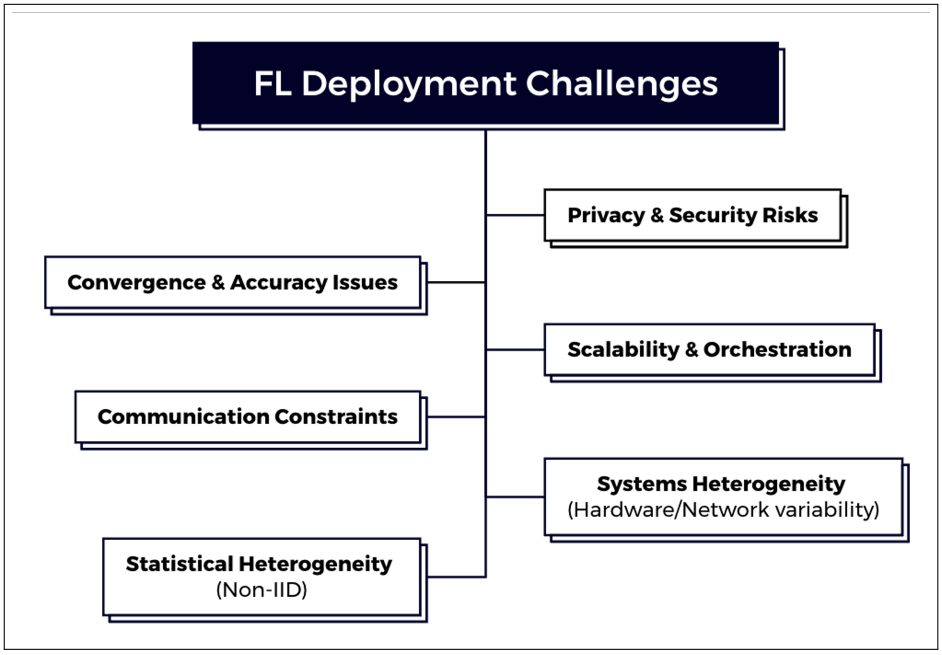


Figure 2.6: Overview of primary challenges in federated learning deployment.

Statistical Heterogeneity (Non-IID Data)

One of the most prominent challenges is statistical heterogeneity, commonly referred to as the non-IID (non-Independent and Identically Distributed) data problem [19]. In federated learning, client datasets rarely conform to the IID assumption common in traditional machine learning. This issue is particularly pronounced in agriculture, where individual farms exhibit unique and substantial variations in soil types, microclimates, chosen crop varieties, management practices, sensor calibrations, and overall data quality [20]. Consequently, the local data distributions across participating farms can differ significantly. This statistical heterogeneity poses a serious impediment to the convergence of standard FL algorithms like FedAvg. When clients possess divergent data distributions, their local model updates can pull the shared global model in conflicting directions, potentially slowing down convergence, causing unstable oscillations during training, or resulting in a final global model that performs sub-optimally for many, if not all, participating clients. Addressing this requires sophisticated mitigation strategies. Algorithmic modifications are a key area of research; for instance, algorithms like FedProx introduce regularisation terms into the local training objective to penalise large deviations from the global model, thereby promoting stability [77]. Other approaches involve adapting aggregation weights or client learning rates based on data characteristics. Personalisation techniques represent another avenue, aiming to train a generalised base model while allowing individual clients to fine-tune specific components to better suit their local

data nuances [93]. Methodologies like client clustering, which groups clients with similar data patterns, or even limited sharing of anonymised data summaries or synthetically generated data (though potentially counteracting FL's core privacy premise), are also explored. Effectively handling non-IID data remains an active and critical research frontier for deploying FL successfully in inherently diverse agricultural ecosystems [94].

System Heterogeneity and Resource Constraints

Beyond data variations, systems heterogeneity presents another significant hurdle [18]. FL clients often display considerable diversity in their computational capabilities (CPU power, available memory, presence of accelerators), network bandwidth, power availability (ranging from mains-powered servers to battery-operated sensors), and storage capacity. In agriculture, this spectrum is wide, encompassing everything from powerful on-farm servers to resource-constrained IoT devices. This variability directly leads to the "straggler problem" in synchronous FL, where the entire training round is delayed by the slowest participating devices [95]. Furthermore, some low-resource devices may lack the capacity to execute complex model training computations or store large model architectures. Tackling systems heterogeneity involves several complementary strategies. Adopting asynchronous FL protocols allows faster clients to proceed without being held back by stragglers, although this introduces complexity in managing potentially stale updates. Strategic client selection, choosing only capable and available clients for each round, can mitigate the impact of resource limitations. Adaptive workload assignment, adjusting the amount of local computation based on device capabilities, offers another way to balance participation. Additionally, model optimisation techniques, such as model pruning, quantisation (reducing numerical precision), or knowledge distillation, are employed to create lightweight models suitable for deployment on resource-constrained edge devices [96]. Hierarchical architectures can also play a role by offloading intensive computations from weaker end-devices to more powerful local aggregators, like farm gateways.

Communication Constraints and Network Reliability

Communication constraints and network reliability pose persistent challenges [9]. Although FL significantly reduces the volume of data transferred compared to centralising raw datasets, it still relies on potentially frequent communication rounds for exchanging model updates. Rural agricultural environments are often characterised by low-bandwidth, high-latency, or intermittent network connectivity. This reality necessitates the development and use of communication-efficient FL strategies. Techniques focused on update compression, such as quantisation, sparsification (transmitting only the most significant update values), or sketching, aim to reduce the size of the messages communicated in each round [97–99]. Another approach involves reducing communication frequency by performing more local computation (e.g., increasing the number of

local training epochs) between communication rounds; however, this must be balanced against the potential risk of increased "client drift," where local models diverge too far from the global objective [100]. Designing communication protocols robust to message loss or unexpected client dropouts, perhaps using asynchronous updates or adaptive timeouts, is also essential [9]. Hierarchical aggregation structures can further alleviate wide-area network strain by confining frequent, smaller updates to more reliable local networks (e.g., within a single farm) and performing less frequent aggregation across farms over potentially less reliable external networks [101].

Privacy and Security Vulnerabilities

While FL enhances privacy by keeping raw data localised, it is not inherently immune to all privacy and security vulnerabilities. Malicious actors, potentially including the central server or other participating clients, might attempt inference attacks to deduce sensitive information about a client's private data by carefully analysing the sequence of model updates they transmit (e.g., membership inference attacks, property inference, or even partial data reconstruction) [21,102,103]. The feasibility of such attacks depends heavily on the model architecture, the specific update mechanism used, and any auxiliary information available to the adversary. Another major concern is model poisoning, where malicious clients intentionally submit corrupted or manipulated model updates designed either to degrade the overall performance of the global model (an untargeted attack) or, more subtly, to cause it to misclassify specific inputs targeted by the attacker [22]. Non-malicious Byzantine failures, where clients send faulty updates due to software bugs or hardware errors, can also disrupt the learning process. Mitigating these threats requires a multi-layered defense strategy combining cryptographic methods, robust statistical techniques, and careful system design. Secure Aggregation protocols, often based on homomorphic encryption or secure multi-party computation, allow the server to compute the aggregate update without decrypting individual contributions, providing strong privacy against server-side adversaries but typically incurring additional computational and communication overhead [22]. Differential Privacy involves injecting mathematically calibrated noise into updates or the final model to provide formal privacy guarantees against inference attacks, although usually at the cost of some reduction in model accuracy. Robust aggregation rules, such as using median or trimmed mean instead of simple averaging, aim to reduce the influence of outlier updates submitted by malicious or faulty clients [104]. System-level measures like robust client authentication and potentially auditing mechanisms (balanced with privacy considerations) are also important. Selecting the appropriate combination of these defenses involves careful consideration of the specific threat model, the sensitivity of the agricultural data, and acceptable trade-offs between privacy/security, model utility, and system efficiency.

Convergence, Accuracy, and Scalability Issues

Achieving robust convergence behaviour and ensuring high model accuracy can also be more challenging in FL compared to centralised training, particularly under the non-IID and communication-constrained conditions prevalent in agriculture. Federated models might converge more slowly or plateau at a slightly lower accuracy level. Numerous factors influence convergence dynamics, including the choice of aggregation algorithm, learning rates, the number of local epochs performed per round, the client sampling strategy employed, and, critically, the degree of statistical heterogeneity across clients. Effectively tuning these hyperparameters without direct access to the distributed datasets requires sophisticated validation strategies, such as relying on local validation sets held by clients, designating specific clients purely for validation purposes, or developing adaptive algorithms that adjust parameters during training.

Finally, managing scalability and orchestration for large-scale FL deployments, potentially involving thousands or even tens of thousands of diverse agricultural devices (sensors, gateways, tractors, drones), presents significant systems engineering challenges. Robust frameworks are essential for handling client discovery and registration, participant selection, task scheduling and dispatch, communication management across varied networks, efficient failure detection and handling, and comprehensive monitoring – all at scale. While foundational open-source frameworks like TensorFlow Federated, PySyft, and Flower provide valuable tools, practical deployment in agriculture often requires substantial integration efforts with existing IoT management platforms and considerable customisation to align with specific agricultural workflows and device ecosystems [105–107].

Addressing these multifaceted challenges requires a holistic approach, combining algorithmic innovation, robust system engineering, and careful consideration of the specific agricultural context. The development and evaluation of FL systems must therefore move beyond idealised simulations to rigorously assess performance under realistic conditions of heterogeneity, unreliability, and potential adversarial pressures. Recognising the importance of grounding theoretical advancements in practical evidence, and acknowledging that much existing work focuses on simulations rather than operational deployment, it becomes essential to systematically survey the landscape of actual FL implementations in agriculture. Therefore, section 2.4 details a Systematic Literature Review (SLR) conducted to understand the extent to which FL has been practically implemented and empirically validated within such real-world agricultural IoT environments.

2.3.2 System Complexity: Software Engineering Demands

Beyond the challenges inherent to the FL algorithmics, the underlying complexity of building, deploying, and maintaining the distributed systems required for

smart agriculture, including those supporting FL, introduces significant software engineering demands.

The successful realisation of Agriculture 4.0 extends beyond hardware deployment, demanding robust software architectures and rigorous software engineering (SE) practices. These are essential to ensure the reliable, integrated operation of diverse components within challenging agricultural environments. A smart farm functions as a complex cyber-physical system (CPS), integrating physical elements like sensors and machinery with software layers governing data handling, analysis, and control. Consequently, established SE principles become critical for the effective design, development, and ongoing maintenance of these intricate systems.

Structurally, smart agriculture systems typically adhere to a multi-layered architectural pattern [108, 109]. This often includes: (1) an edge layer with embedded sensors and controllers deployed in the field; (2) a communication layer facilitating data transmission; and (3) a cloud or server layer for data aggregation, storage, and advanced analytics. Engineering robust software for each layer is paramount, encompassing firmware for resource-constrained edge devices, reliable network protocols, scalable cloud services, and intuitive user interfaces [110].

Given the operational context, system reliability and resilience emerge as primary SE considerations. Agricultural settings inherently pose challenges, including harsh environmental conditions and variable power availability, while rural network connectivity can be intermittent or limited [7]. Software architectures must therefore be engineered for resilience against such disruptions, often employing mechanisms like dynamic service discovery, data synchronisation, and fault-tolerant designs [111]. Ensuring that single-component failures do not cascade into systemic issues is crucial, making SE practices such as modularity, redundancy, and comprehensive error handling indispensable [112, 113].

Another critical dimension, arising from the typical multi-vendor farm environment, is interoperability and adherence to standards. Farms frequently integrate heterogeneous equipment, often utilising proprietary software and data formats [114]. A key SE task is, therefore, the integration of these disparate systems into a cohesive whole. This need has spurred initiatives towards standardising agricultural data formats (e.g. ISO 11783 ISOBUS [115]) and promoting open Application Programming Interfaces (APIs) [116, 117]. Well-architected platforms must accommodate diverse devices and communication protocols like MQTT and HTTP/REST [118], often using middleware to abstract complexity and unify data models [119]. Effectively engineered systems are vital to prevent fragmented data silos and allow seamless data flow from field to decision-maker [120].

Furthermore, the practical deployment and long-term operation of these com-

plex systems necessitate careful consideration of scalability and maintainability [121]. Systems must be designed to scale from initial pilots to potentially large, multi-site deployments without fundamental redesign. Architectural patterns like cloud-native design, microservices, and serverless computing offer strategies for achieving this (as discussed in Section 2.1.3). Equally important is maintainability, especially given limited on-site IT support in typical farm settings. Software should facilitate remote monitoring, diagnostics, and updates [122], supported by SE best practices like version control, CI/CD pipelines, and rigorous testing to ensure system integrity during evolution [123].

Finally, the application of SE in agriculture continues to evolve with emerging paradigms like digital twins and DevOps for IoT. Digital twins offer virtual representations for simulation and predictive maintenance [124], while DevOps principles can accelerate innovation cycles for field-deployed systems. These advanced techniques further underscore the sophisticated SE required.

Beyond these technical requirements, the ultimate adoption and efficacy of Agriculture 4.0 technologies hinge significantly on user-centric design. Primary users, such as farmers and agronomists, typically require intuitive tools that solve operational problems without imposing excessive complexity [125]. Historically, poor alignment with farm workflows has hindered technology adoption. Therefore, Agriculture 4.0 software must deliver tangible value. Integrating end-user feedback throughout the design process, using participatory methodologies, is crucial for developing practical, usable solutions.

Ultimately, software engineering provides the crucial underpinning for Agriculture 4.0. Without well-designed, reliable, interoperable, scalable, maintainable, and usable software, the potential benefits of advanced hardware and data analytics cannot be fully realised. As the sector progresses towards more complex solutions, including the federated learning approaches discussed later, rigorous SE remains determinant in translating technological potential into impactful agricultural tools.

2.4 Assessing Current Progress: Systematic Literature Review

Given the potential of FL and the significant practical challenges, understanding the extent to which FL has been successfully implemented and evaluated in real-world agricultural IoT settings is crucial. To this end, a Systematic Literature Review (SLR) was conducted.

2.4.1 Scope of this Review

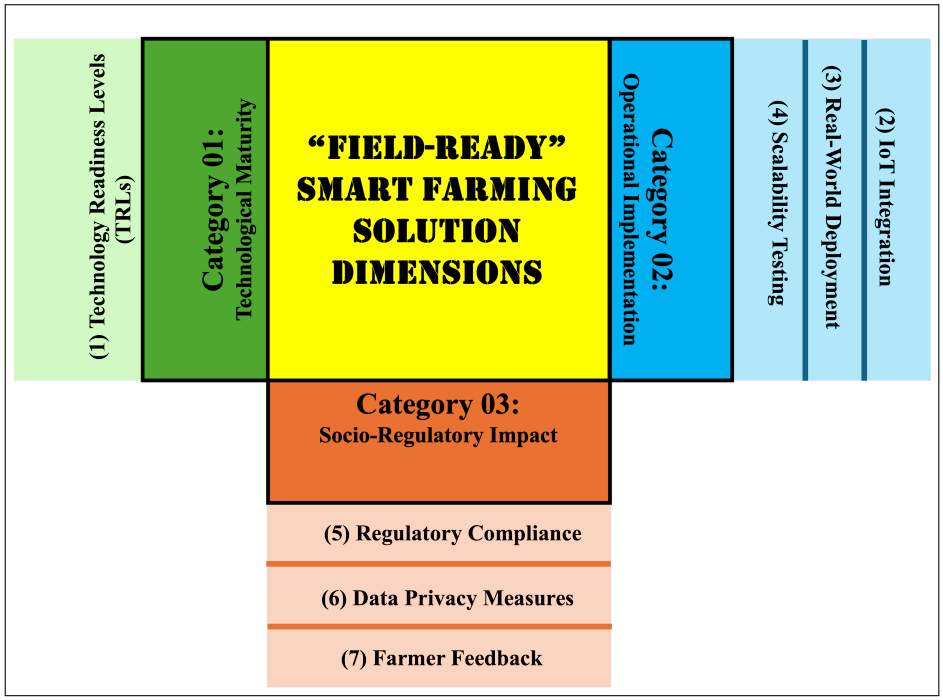


Figure 2.7: Seven key dimensions, grouped into three main categories, for evaluating field-ready federated learning in agricultural IoT systems.

To objectively assess the state of FL deployment in smart farming, we followed the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) methodology [126] to guide our search and selection process. This means that our review is systematic, transparent and reproducible. Therefore, our review is designed so that anyone following the same steps would reach a similar body of literature providing this domain of research a solid reference point on the current state of knowledge. This transparency is important for the emerging field of Agriculture 4.0, which benefits from clear baselines of what has been tried and tested.

Given that FL in agriculture is still a developing interdisciplinary topic, the pool of relevant literature is relatively small. After screening for relevance, a total of 13 studies were included in this review. We acknowledge that 13 is a modest number of studies; however, this number reflects the early stage of the field and the stringent focus on "field-ready" deployments. Rather than catalogue every conceptual paper on FL, we concentrated on those backed by some implementation in a farming context, to truly gauge the progress toward real-world use.

Each selected study was read in depth.

To make sense of these diverse studies and answer our core question about field readiness (highlighted in Table 2.1), we developed an evaluation framework as highlighted in Figure 2.7. This framework is informed by prior literature on technology adoption and deployment in agriculture. Seven key dimensions were identified that together paint a picture of how "ready" a given smart farming innovation is for real-world farming. Below, we introduce each dimension and explain its importance for evaluating field-readiness in the agricultural domain.

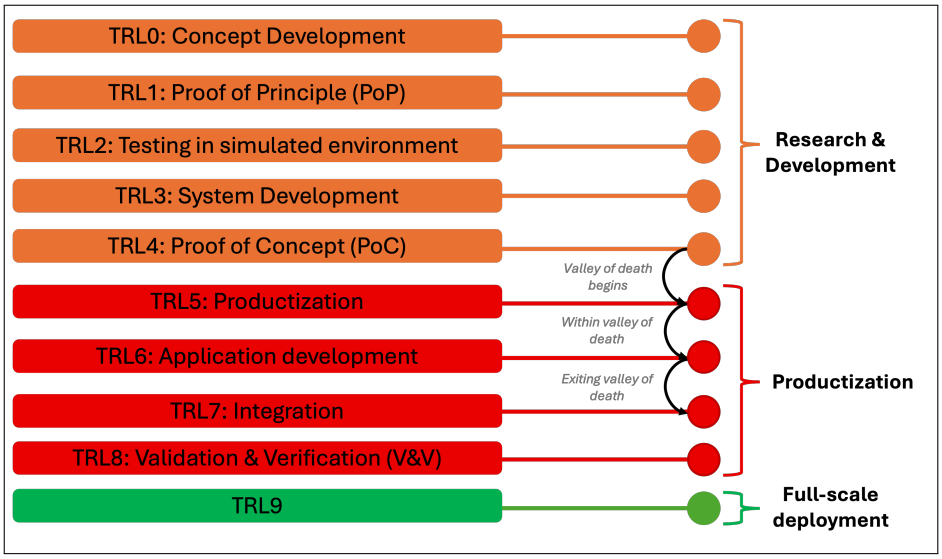


Figure 2.8: Technology Readiness Level (TRL) framework with key development stages and 'valley of death' zones

Dimension 01 - Technology Readiness Level (TRL)

TRL denotes the maturity of a technology, typically on a scale from 1 to 9 as highlighted in figure 2.8, where TRL1 means basic principles observed and TRL9 means a system has been proven in an operational environment [127,128]. We include TRL to gauge how far along each study's solution is in the journey from concept to practical deployment. A low TRL, for example something only tested in simulation or lab conditions, suggests the idea is still largely theoretical. A high TRL which mean tested on real farms or even commercially available indicates the solution is closer to being field-ready. Using TRL provides a quantitative sense of maturity and lets us compare studies on a roughly objective scale of development progress. In our evaluation, we rely on the authors' statements or infer the TRL based on experiment descriptions. Including TRL ensures we acknowledge the difference between an algorithm that works on paper versus one that works in practice.

Dimension 02 - IoT Integration

Smart farming solutions do not exist in isolation, instead they must integrate with IoT hardware and farm information systems to be truly deployable. By IoT integration, we mean the degree to which the FL solution connects with real farming devices such as sensors, tractors, drones, edge devices and data streams. This dimension captures interoperability and technical compatibility. This criterion is derived from the frequent observation that interoperability is a major hurdle in practice [129]. If a federated learning approach only lives in the cloud or in a simulated dataset, its field-readiness is limited. We look for evidence that the studies tied their algorithms into real sensor networks. Strong IoT integration shows that the researchers considered the systems engineering aspect, not just the analytics. It also often surfaces practical issues, like network bandwidth limits and device power constraints, that are invisible in purely theoretical work.

Dimension 03 - Real-World Deployment

Complementing TRL and IoT integration, we explicitly check whether the solution has been deployed or tested in a real-world farm environment. Many promising technologies perform well in controlled experiments but fall apart in uncontrolled settings due to factors like weather, variable user behaviour, or unforeseen bugs. This dimension records whether each study moved beyond simulation or laboratory testing. We consider contextual realism: for instance, if a study claims to optimise irrigation via FL, did it test with real soil moisture sensors under field conditions with all the noise and interruptions that entails? Real-world deployment is vital because it signals the solution has overcome initial adoption barriers, at least to get deployed, and it provides a chance to observe other factors like reliability, ease of use, and performance under true farming conditions. In evaluating this dimension, we classify the extent of deployment (Yes, No, Partial) based on each paper's account.

Dimension 04 - Scalability

A field-ready smart farming system must be able to scale up, in terms of number of devices, volume of data, and number of participating users or farms, without degrading performance or manageability. We include scalability as a dimension because agriculture IoT systems can start small but, if successful, may need to cover whole regions or multiple stakeholder networks. Our evaluation, therefore, focuses specifically on how studies address scalability. We first determine if a study claims its proposed system is scalable. If such a claim is made, we then assess whether the extent of this scalability is clearly defined and validated through empirical testing. A claim without such definition or evidence is considered purely conceptual scalability. Only when both the claim and a clear definition or evidence are present do we consider scalability to have been adequately addressed in a way that informs deployment potential. Literature

consistently highlights communication bottlenecks and the limited computational power of edge/IoT nodes in rural settings as major technical barriers for FL in agriculture [130]. Therefore, we view the demonstration of scalability, not just its conceptual mention, as a critical litmus test for a prototype's potential to graduate from small trials to larger farm-wide or multi-farm systems. If a study only proves functionality, for example, 5 devices on a lab network without addressing how it performs with specific number of dozens or hundreds of sensors under real-world farm conditions, its confidence for practical scalability is an open technical question.

Dimension 05 - Regulatory Compliance

Any agricultural technology must navigate the web of regulations and standards that govern data, safety, and operations on farms. In recent years, data privacy laws, such as GDPR in Europe, have begun to cover farm data, and there are also regulations around the use of drones, autonomous machines, or even AI-driven advisory services. We include regulatory compliance as a dimension to see if the studies accounted for these legal and ethical frameworks. Unclear data governance and lack of regulatory clarity can undermine farmer trust and slow adoption [131]. Conversely, clear assurances of compliance has a greater potential for adoption. In evaluating this dimension, we note whether the 13 studies mention adherence to any standards or laws. While academic papers don't always discuss legal aspects, we view any such discussion as a positive sign that the solution is being conceived with real-world deployment in mind. A field-ready solution should not only work technically but also be deployable legally and ethically.

Dimension 06 - Data Privacy Measures

Although Federated Learning keeps raw data local, it is not a silver bullet for all privacy concerns. This separate criterion for data privacy is to note how each study handles sensitive information. This includes any additional privacy-preserving techniques used, as well as how well the approach aligns with farmers' privacy expectations. Privacy remains one of the top-cited concerns among farmers regarding digital farming. Therefore, we examine if the studies went the extra mile to safeguard data for example by formally verifying that no farm's data can be reconstructed from the shared model. We also consider whether the studies discuss data ownership and consent such as can each farm fully control its contribution and can opt out.

Dimension 07 - Farmer Feedback

Finally, and perhaps most importantly from an adoption standpoint, we look at any feedback from farmers or end-users that the studies collected. Technology that looks great on paper can fail if it doesn't align with the people intended to use it. This dimension captures any user-centric evaluation reported such

as through user interviews, surveys, participatory design sessions, or feedback from pilot participants. The inclusion of farmer feedback is grounded in the principle of user-centered design, which is strongly advocated for overcoming adoption barriers [129]. A lack of any farmer input or evaluation suggests the solution may be out of touch with user needs which is a warning sign for field readiness. This criterion is qualitative by nature: we qualitatively summarise what (if anything) farmers had to say about the technology, and whether their feedback was positive or led to changes. Ultimately, a technology will only be adopted if it resonates with farmers' values and day-to-day workflows, so we consider this ground truth check with end-users indispensable.

These seven dimensions span both quantitative and qualitative aspects of field readiness. Some criteria can be scored or measured, for instance, we can assign a numeric TRL level to each study. Others are more descriptive and evaluated through narrative evidence. This mix of hard metrics and soft insights is deliberate. A purely technical assessment would overlook human factors, while a purely social assessment would ignore whether the technology actually works at scale. By combining both, our framework provides a holistic view of "field-readiness." For example, a given solution might score high on TRL (quantitative readiness) but could still fail if it neglects farmer input. Likewise, strong farmer endorsement might not mean much if the system only works for few sensors in a lab. The seven dimensions together allow us to systematically dissect each of the 13 studies along the various threads that determine success in real-world agriculture. This structured evaluation is, to our knowledge, the first attempt to systematically benchmark deployable federated learning for smart farming. In doing so, we aim to establish a foundation for future research and development to build upon. If one knows the baseline, what has been tried and where the gaps lie, then new efforts can target those gaps more effectively and evaluate the extent of progress made after couple of years.

Ultimately, our findings aren't just academic observations; they issue a critical call to action with real-world implications. The promise of Agriculture 4.0 cannot be fulfilled for farming through isolated efforts. Real progress requires a concerted push: combining expertise, sharing insights openly, and designing systems built for practical collaboration across farms, regions, and organisations. Federated learning, by its nature, aligns with this collaborative spirit. However, translating its potential into field-ready reality demands that all key players, including researchers, agribusiness firms, and farmers themselves, actively coordinate to tackle the deployment challenges head-on. Making technologies like FL truly field-ready should not only be considered as a technical pursuit but an urgent one for humanity to achieve Zero Hunger (UN SDG 2).

2.4.2 Review Methodology

To investigate the deployment and testing of federated learning in smart farming Internet of Things (IoT) systems, we conducted a systematic literature review.

Adhering to the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) 2020 guidelines [126] ensured that our research process was transparent, exhaustive, and replicable. Our primary objective was to evaluate the Technology Readiness Levels (TRLs) of existing solutions, thereby assessing their maturity and identifying the necessary steps to facilitate their progression from academic research to industrial application. By focusing on TRLs, we aimed to address the critical gap, often referred to as the "valley of death", where innovative technologies struggle to transition into commercial viability due to a multitude of challenges. To the best of the authors' knowledge, this is the first systematic review to comprehensively evaluate the TRLs of federated learning applications in smart farming IoT systems, providing critical insights into bridging this gap.

Search Strategy

The literature search was performed on October 2nd, 2024, utilising the Scopus database[1]. Scopus was chosen for its extensive indexing of journals and conference proceedings across disciplines integral to this study, including computer science, engineering, and agricultural sciences. Its comprehensive coverage allowed for a broad capture of relevant literature, thereby improving the reliability and validity of our review.

We crafted the search query with precision to encompass key concepts central to our investigation: federated learning, smart farming, IoT, and practical implementation aspects critical for assessing TRLs. The specific search string employed was:

```
"Federated Learning" OR "Distributed Learning" AND "Smart Farming"
OR "Agriculture" OR "Horticulture" AND "Internet of Things" OR "IoT"
AND "Deploy*"[2] OR "Test*" OR "Implement*"
```

[1] `http://www.scopus.com/`

[2] The inclusion of truncation symbols (e.g., "Deploy*") was used to capture the various word forms such deploy**ed**, deploy**ing** and deploy**ment**, ensuring the comprehensiveness of the search.

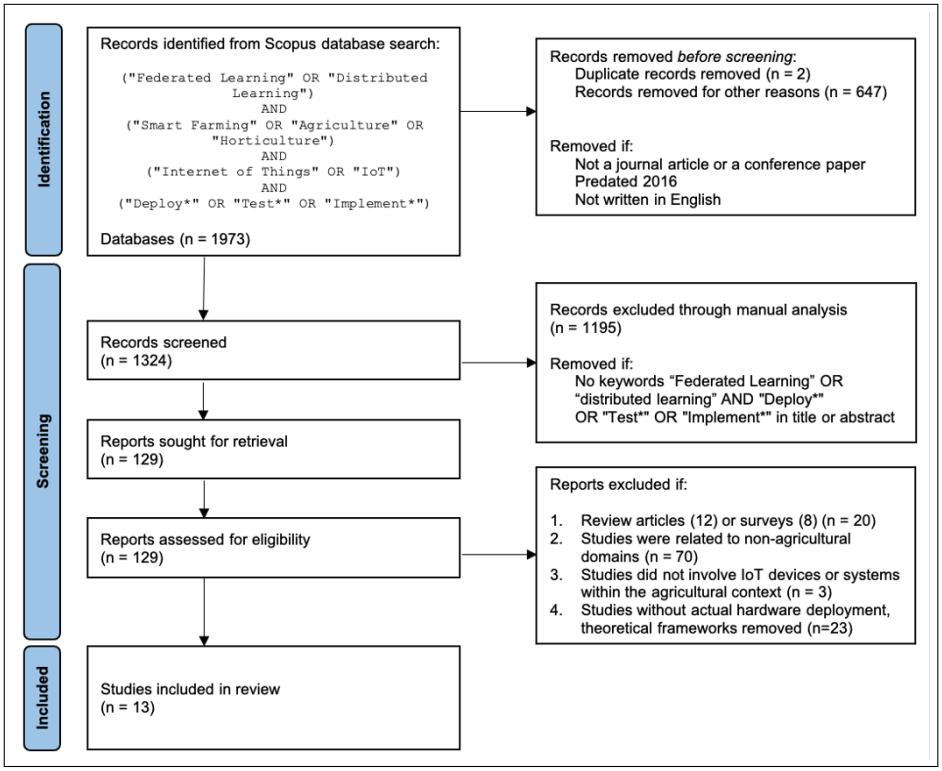


Figure 2.9: Selection criteria for the literature review using the PRISMA review process

The initial search yielded 1,973 records, indicative of the burgeoning interest and research activity at the intersection of federated learning, smart farming, and IoT technologies. The term "federated learning" was introduced by Google in 2016, marking the inception of this research domain. To illustrate the temporal escalation in research activity, we analysed the distribution of these publications over the years from 2017 to 2024 (Fig. 2.10). This exponential growth underscores the timeliness and relevance of our review, reflecting the rapidly intensifying academic interest in applying federated learning to smart farming IoT systems.

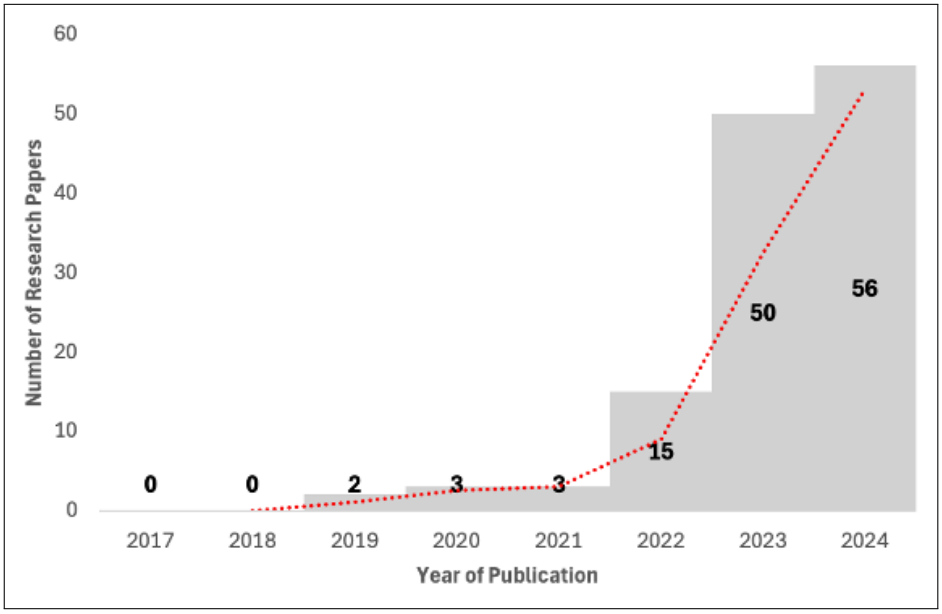


Figure 2.10: Yearly distribution of the 129 screened research papers assessed for deployment and testing of federated learning in smart farming IoT systems from 2017 to 2024

Inclusion and Exclusion Criteria

To refine the dataset and ensure the inclusion of relevant studies, a set of inclusion and exclusion criteria was systematically applied. Only journal articles and conference papers published in English from 2017 onwards were considered. Studies had to involve the deployment, testing, or implementation of federated learning in smart farming IoT systems to be deemed relevant.

Duplicate records were identified and removed, resulting in 1,324 records after the initial screening. Non-peer-reviewed articles, non-English publications, and studies not containing the necessary keywords in the title or abstract were excluded to maintain focus on the research objectives. A manual screening of titles and abstracts was conducted on the 1,324 records to exclude studies that did not emphasise practical aspects of federated learning deployment in agriculture. This process led to the exclusion of 1,195 studies, ensuring that only those with a clear focus on the research topic were retained.

The remaining 129 papers underwent a full-text eligibility assessment. Studies were excluded if they were review articles or surveys (n = 20), unrelated or only remotely related to agriculture (n = 70), lacked IoT devices or systems within the agricultural context (n = 3), or focused solely on theoretical frameworks without any forms of deployment (n = 23). Following this selection process, 13

studies were included in the final review, representing the most relevant research available on the deployment and testing of federated learning in smart farming IoT systems.

Data Extraction and Categorisation

This systematic review addresses seven key research questions, summarised in Table 2.1. For each dimension, we examined the final set of 13 papers to document specific implementations, limitations, and notable innovations.

By systematically applying these seven dimensions to each study, we aimed to identify the primary factors shaping technological maturity and adoption. Our goal is to provide a roadmap for bridging the "valley of death" between promising research prototypes and commercially viable agricultural solutions.

Table 2.1: Research Questions Used for Categorisation

Question No.	**Research Question**	**Context**
R1: Technology Readiness Level (TRL) Assessment	What is the technology readiness level (TRL) of current federated learning deployments in smart farming?	Assessing TRLs helps distinguish theoretical concepts from laboratory prototypes, field-tested systems, and operational deployments. This clarity is crucial for determining whether FL solutions can transition from academic research to practical use in agriculture. Without a systematic TRL analysis, stakeholders lack insights into the feasibility, robustness, and scalability of FL implementations in real farming contexts.
R2: IoT Device Integration	Are IoT devices explicitly used in these studies?	Effective IoT integration is vital for bridging advanced ML algorithms with on-farm sensing and actuation. Without explicit hardware deployment, it is unclear how FL solutions interface with physical environments or handle constraints like limited power and connectivity.
R3: Real-World Deployment	Is the FL solution deployed in a real-world agricultural setting?	FL solutions that remain confined to simulations or lab-based experiments may not translate well to the variability of agricultural environments.
R4: Data Privacy Measures	What data privacy measures are implemented?	FL's distributed architecture is often promoted as inherently privacy-preserving; however, additional measures, such as differential privacy, secure aggregation, or encryption, may be necessary to prevent inference attacks and maintain regulatory compliance.
R5: Scalability Testing	Is scalability adequately tested?	Agriculture often involves large, geographically dispersed operations with heterogeneous devices. FL systems that perform well in small-scale or controlled scenarios may fail when scaled to hundreds of sensors across multiple fields.
R6: Regulatory Compliance	Does the study address regulatory compliance?	As agricultural data becomes increasingly sensitive and regulated, compliance with data protection, environmental, and agricultural policies is essential for real-world adoption.
R7: Farmer Feedback Incorporation	Is farmer feedback incorporated in the study?	Farmers are the primary end-users of smart farming technologies. Systems designed without direct input from farmers risk low adoption rates due to usability barriers, mismatched priorities, or insufficient trust.

2.4.3 SLR Findings: Results and Discussion

Technology Readiness Level (TRL) Assessment

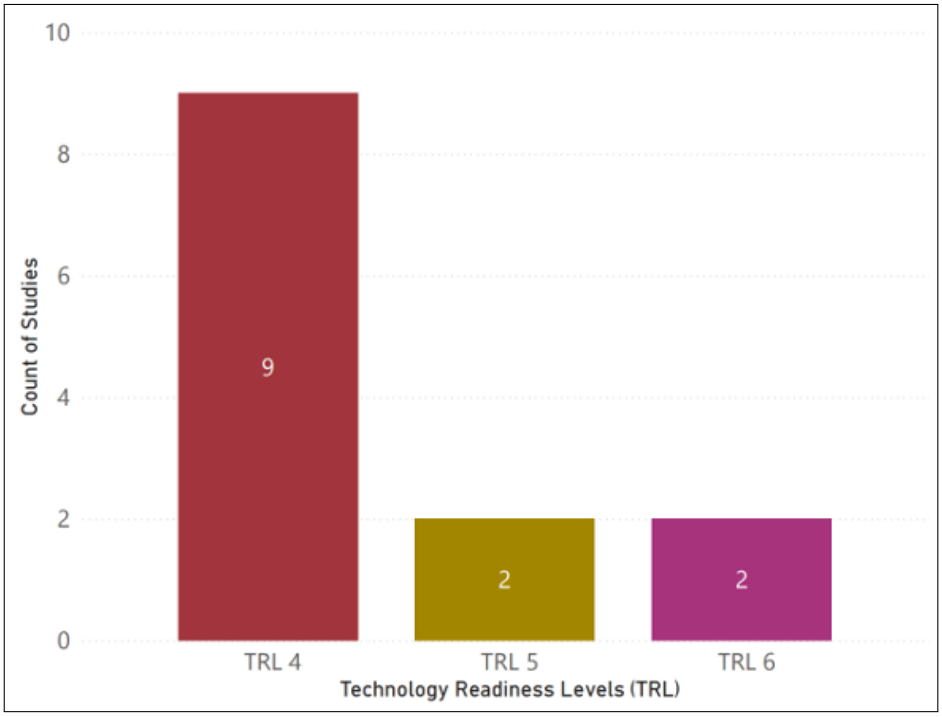


Figure 2.11: Distribution of studies by TRL

A comprehensive evaluation of the 13 reviewed studies reveals a diverse yet predominantly early-stage landscape for federated learning (FL) in smart-farming IoT. Although none of these works explicitly declares a specific NASA TRL designation, the methods, deployment contexts, and validation details they report allow us to infer approximate readiness levels. Collectively, these publications clarify the transformative promise of FL, improving data-driven agriculture while preserving privacy, and equally highlight a pronounced gap between proof-of-concept demonstrations and robust, commercial-scale implementations.

Figure 2.11 highlights that a notable majority of the studies (nine of the thirteen) appear to align most closely with TRL 4, indicating that the systems have graduated beyond theoretical formulations and have undergone controlled or partially simulated testing. For instance, Aggarwal et al. [132] emphasise resource-efficient FL to diagnose crop leaf diseases but base their experiments on curated datasets or lab simulations rather than genuine on-field sensor networks. Likewise, Choubey and Divya [133] investigate how to reduce computational

overhead via transfer learning but confine their results to dataset-based validations. These efforts illuminate the growing sophistication of FL architectures, ranging from lightweight model pruning to advanced neural-network training strategies - yet none of them presents a convincing demonstration of stable operation in real farm environments. Fahim-Ul-Islam et al. [134] offer another TRL 4 illustration: they showcase advanced classification algorithms (including Transformers) trained on existing image data, underscoring the power of deep learning for disease detection but omitting a thorough sensor-based infrastructure or a multi-season field deployment.

Several further examples reinforce this TRL 4 categorisation. Thonglek and Maipradit [135] use hierarchical FL with real sensor data collected at water-level stations, but their scope is relatively narrow and does not span entire farmland or long-term agronomic trials. Mamba Kabala et al. [136] apply CNNs and Vision Transformers to the widely used PlantVillage dataset, convincingly demonstrating that decentralised models can equal or surpass centralised approaches in classification accuracy. However, like the others, it remains a dataset-driven experiment with no mention of actual in-field testing. Aggarwal et al. [137] reference memory usage, CPU constraints, and partial cross-silo data; while they seek to optimise for edge-based conditions, the operational environment is still best described as controlled. Similarly, Praharaj et al. [138] situate collaborative anomaly detection within a partial cloud environment but do not detail a large-scale farm pilot or ongoing sensor data streams. Sharma et al. [139] and Thonglek and Rattanatamrong [140] further illustrate TRL 4–type settings. Sharma et al. focus on generating synthetic weed images via GAN-based data augmentation rather than implementing full IoT-FL integration.

Thonglek and Rattanatamrong [140] explore a neighbor-based FL approach on small edge boards, underscoring a creative technique to reduce communication overhead, yet not scaling to multi-farm testbeds or season-long reliability. This cluster of TRL 4 papers [132–140] collectively portrays a dynamic field full of algorithmic innovation, particularly aimed at addressing privacy, communication efficiency, or classification accuracy. Authors typically emphasise either the architectural novelty, such as advanced network designs or specialised data augmentation techniques, or the synergy of FL with IoT-inspired constraints (e.g., limited bandwidth, device heterogeneity). Despite their promise, these works remain lab- or dataset-oriented, typically validated through short-term simulations or offline data sets and rarely bridging into the messy, variable realities of farmland. As a result, they exemplify "proven under controlled conditions" but do not indicate that these systems are anywhere near mass operationalisation.

In contrast, two other studies appear to advance to TRL 5, where solutions have begun transitioning from the lab to partially relevant testbeds. Kaushal et al. [141] demonstrate a real test scenario for weed detection in rural environments, featuring an edge-based FL approach that extends beyond pure simulation. Although the system is tested with authentic farmland constraints,

like limited connectivity or resource-scarce hardware, it does not present large-scale replication across multiple locations or repeated harvest cycles, thus falling short of a full operational demonstration. Pincheira et al. [142] likewise edge closer to real deployment by equipping a mobile robotic platform with modular sensors for orchard monitoring. Their orchard-based experiments reveal the feasibility of bridging static IoT's shortcomings, yet these remain narrower pilot demonstrations rather than validated, consistent usage across entire growing seasons. In both cases, the introduction of genuine hardware integration or partial field activities marks a noteworthy step beyond purely synthetic data or single-lab conditions, but the absence of multi-region or multi-season results reflects a transitional maturity - a hallmark of TRL 5 rather than 6 or above.

Finally, TRL 6 emerges in two studies, Devaraj et al. [79] and Singh and Adhikari [91], each showcasing sustained field-level experiments in real agricultural contexts. Devaraj et al. [79] employ hierarchical FL with an actual sensor network measuring soil attributes and weather conditions for tomato crops. Their partial horticultural trials confirm that FL can surmount some data heterogeneity and communication barriers in the field. However, the scale remains relatively constrained: the horticultural environment is real, but not extensive enough to confirm robust reliability across multiple farms or longer timescales. Similarly, Singh and Adhikari [91] center on a prototype called "AgriFed," demonstrating irrigation management across a sensor-array in real paddy fields. This trial addresses outdoor constraints like variable connectivity and the unpredictability of field-based hardware, signifying an advance over the smaller, controlled prototypes at TRL 4 or 5. However, the number of sensors, the range of coverage, and the longevity of testing remain modest. As such, neither study claims the repeatable, industrial-level functionality that TRL 7 or above would entail.

Taken together, these thirteen papers collectively depict FL in smart farming as an emerging discipline with high innovation potential but few outcomes that have graduated to an advanced state of real-world readiness. The largest cohort (nine papers) remains anchored at TRL 4, showcasing proof-of-concept achievements without crossing the threshold into multi-farm or multi-month reliability tests. Two works raise the bar to TRL 5 by conducting partial real-environment demonstrations, and another two, by testing FL solutions in more extended or elaborate contexts, arrive at TRL 6. Crucially, there is no evidence that any solution has advanced to TRL 7–9, where operational usage, standardisation, or large-scale adoption across multiple growers would be expected. This discrepancy echoes the well-known "valley of death" in translational research, where algorithmically impressive prototypes struggle to incorporate essential components such as robust communication protocols, diverse sensor arrays, farmer training, user-friendly interfaces, and economic feasibility analyses.

Nonetheless, the fact that multiple authors have advanced from hypothetical lab scenarios to partial on-ground experiments is itself evidence of momentum.

FL's central advantages, such as preserving farmer data privacy, reducing dependence on a single central server, and fostering collaborative knowledge among distributed nodes, resonate particularly strongly with farming communities wary of data-sharing. Also, the impetus toward green technology fosters interest in optimising inputs like water, fertiliser, and pesticides, and FL-based solutions could help deliver real-time, data-driven recommendations at the local scale. Still, these technological achievements require more stakeholder-driven research to ensure that solutions not only function under "best-case" pilot conditions but also adapt smoothly to the day-to-day unpredictability of agricultural operations. Collaboration among domain specialists, agronomists, AI researchers, telecommunication providers, and policy experts is needed to standardise usage guidelines, define data governance rules, and refine hardware–software synergy.

IoT Device Integration

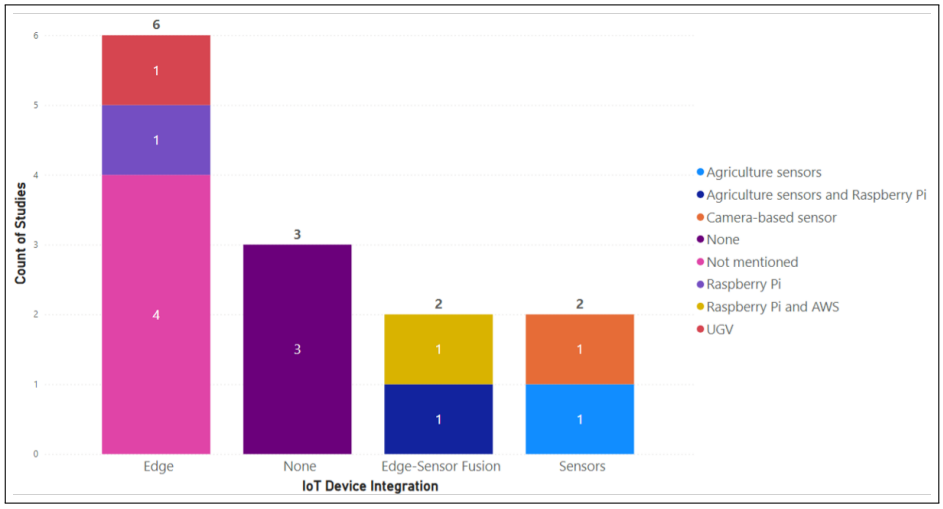


Figure 2.12: IoT device integration

A defining characteristic of FL in smart-farming contexts is the reliance on distributed IoT devices to gather, process, and share data in real time. Although the 13 reviewed studies vary greatly in their scope, Figure 2.12 shows some focusing on single-sensor deployments, others integrating unmanned ground vehicles (UGVs), they collectively underscore how FL's success in agriculture hinges on well-chosen, robust, and scalable IoT infrastructures.

Most research endeavours emphasise sensor-based data capture as the fundamental backbone. For instance, Aggarwal et al. [132] harness an "Internet of Agriculture Things" setup where devices measure leaf images or basic field parameters (e.g., humidity, temperature) to power FL training for disease detec-

tion. Devaraj et al. [79] similarly employ a comprehensive sensor system, spanning soil moisture, temperature, and weather inputs, to update hierarchical FL models that track crop health. In Thonglek and Maipradit's work [135], IoT stations measure hydrological variables, albeit within a narrower domain, while Singh and Adhikari [91] integrate sensors for soil moisture and temperature directly in a real paddy field, thereby enabling immediate irrigation decisions. In each of these cases, the sensors themselves form the cornerstone of localised data collection before models or updates are transmitted for aggregation.

Beyond conventional static sensors, a growing number of studies incorporate edge devices with local compute. Thonglek and Rattanatamrong [140] stand out by situating FL directly on small ARM-based boards, each simulating a miniaturised sensor hub. Instead of offloading raw data, these boards train partial models locally and exchange only model parameters, reducing bandwidth overhead and reinforcing data privacy. Kaushal et al. [141] likewise adopt an edge-centric approach, balancing tasks across resource-constrained nodes in rural fields to handle weed-detection workloads. Praharaj et al. [138] propose a layered cooperative environment for anomaly detection, in which edge devices filter or preprocess sensor data before updating a global model in the cloud. These edge-based tactics are especially crucial where continuous network connectivity is unreliable, forcing each node to handle bursts of local activity until sporadic synchronisation is feasible.

Two papers bring mobile platforms into focus as IoT data-collection units, adding another layer of dynamism to FL in agriculture. In Pincheira et al.'s work [142], an unmanned ground vehicle navigates an orchard, equipped with sensors to build a digital twin in real time. This approach offloads data-gathering tasks from static sensors, expands coverage to remote areas of the farm, and delivers fresh training data to refine FL models. Though more limited in scale, it demonstrates how a single mobile agent can address coverage gaps common in conventional IoT deployments. Devaraj et al. [79], while not a robotics study per se, similarly illustrate how localised sensor networks, coordinated via an FL gateway, can adapt to varied terrain or crop layouts.

The heterogeneity of IoT hardware emerges as both a challenge and an opportunity. Fahim-Ul-Islam et al. [134] reference camera-based acquisitions for disease detection, focusing on leaf imagery rather than environmental sensors, whereas Sharma et al. [139] involve large image sets for weed identification but no explicit mention of sensor-based IoT frameworks. Mamba Kabala et al. [136] straddle a middle ground, discussing possible edge-based camera solutions but not detailing on-device constraints. These differences hint at the complexities of bridging distinct data streams, temperature logs, multispectral images, and even external feeds like weather forecasts, into a unified FL pipeline. Most studies rely on FL's built-in capacity to handle non-IID (non-independent and identically distributed) data, acknowledging that farmland conditions, sensor placement, and device calibration can vary greatly across fields.

Despite successes, each study highlights resource constraints typical of IoT devices. Thonglek and Rattanatamrong [140] and Praharaj et al. [138] emphasise that computational overhead must be kept in check so devices can carry out partial training or pre-processing without depleting batteries or overwhelming limited CPUs. Some propose hardware-aware solutions, such as reducing batch sise, pruning neural networks, or limiting the frequency of updates. The results are often promising, but few test truly large networks of hundreds or thousands of nodes - the scale real-world agriculture may demand. Thus, while these local processing methods foster autonomy and privacy, they also complicate model coordination in high-latency or low-power conditions.

Overall, the integration of IoT devices across these thirteen studies demonstrates a consistent objective: to harness real-time data in a manner that not only fuels FL model performance but also respects each farm's resource limitations. Sensors remain the primary data source, whether for monitoring microclimates, soils, or disease patterns, but advanced edge devices, drones, and UGVs increasingly supplement or replace static installations. Across this spectrum, each research group underscores the synergy between FL's decentralised data handling and the IoT's highly distributed sensor architecture.

In conclusion, the second question "Are IoT devices explicitly used in the study?" can be answered affirmatively for most if not all of the thirteen papers. Indeed, sensors, edge devices, or robotic platforms consistently appear as integral components. Studies vary in the sophistication of their IoT architectures, from straightforward sensor arrays that feed an FL algorithm to more advanced setups involving UGVs or complex multi-tier networks. Although the practical complexities of network protocols, bandwidth constraints, and hardware reliability are not always equally emphasised, the cumulative evidence shows that physical IoT deployment underpins much of the progress toward real-time, privacy-preserving intelligence in agricultural domains. The next challenge lies in scaling these solutions beyond small demonstration plots, integrating multiple sensor types seamlessly, and navigating the real operational unpredictability that farms present.

Real-World Deployment

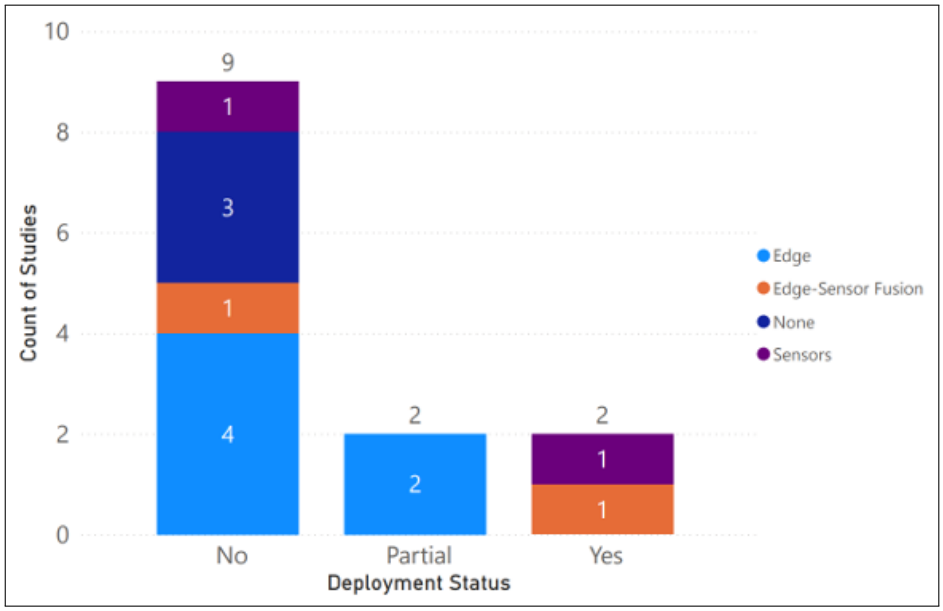


Figure 2.13: Field deployment status of FL solutions

Whether a FL solution is actually deployed in real-world agricultural settings is a decisive factor in gauging its practical value and long-term viability. Among the 13 reviewed papers, Figure 2.13 demonstrates that most remain at the stage of lab-based or small-scale tests, but a handful do venture into genuine field conditions, albeit typically on a limited scope or duration. This section summarises how each study approaches real-world validation, underscoring both achievements and lingering challenges that hamper broader adoption.

A few works move beyond mere simulations or offline datasets and deploy FL in actual farms or related outdoor environments. Devaraj et al. [79] integrate hierarchical FL with soil and environmental sensors in a functional tomato-farming context, using a horticultural deployment to demonstrate FL's capacity for handling heterogeneous data and intermittent connectivity. Although its geographic scope is relatively contained, focusing on tomato crops in a defined area, the study confirms that cooperative training can persist under authentic farm conditions. Singh and Adhikari [91] describe an "AgriFed" system installed in paddy fields, where soil moisture and temperature data feed local FL nodes for real-time water allocation decisions. These in-field trials illustrate that privacy-preserving machine learning can help farmers optimise inputs, though the experiment does not extend across multiple farms or multiple harvest seasons.

Two other papers exhibit partial real-world deployments but do not span large-scale operational usage. Kaushal et al. [141] focus on weed detection in rural settings, using an edge-based FL environment tested with actual farmland constraints such as limited device resources and variable connectivity but do not detail large, multi-region expansions or sustained multi-season results. Pincheira et al. [142] likewise situate their proof-of-concept in an orchard, where an unmanned ground vehicle (UGV) collects sensor data to create a digital twin and runs rudimentary FL tasks. Though genuinely field-based, it remains more of a specialised pilot than a fully operational system across diverse agricultural conditions. These efforts serve as stepping stones toward more robust and sustained deployments but are better classified as smaller pilot demonstrations rather than large-scale real-world rollouts.

By contrast, the majority of the papers [132–140], rely predominantly on curated datasets, lab simulations, or constrained test networks. In each of these cases, the technological novelty, ranging from resource-efficient FL algorithms to advanced neural architectures, lays essential groundwork, but the validation rarely extends into an ongoing, fully operational farm environment.

Such limited real-world engagement indicates that most FL research in smart farming is still transitioning from conceptual or lab-based tests (often with TRLs around 4–5, as discussed earlier) to partial in situ demonstrations (TRL 6). Even the more field-oriented studies, despite providing compelling evidence that FL can function outdoors, remain mostly constrained in scope, location, or time. Challenges like communications overhead, device heterogeneity, sensor reliability, and user acceptance become far more acute in actual farm conditions than they are in stable laboratory or dataset-based trials. Connectivity disruptions, unpredictable climate variations, and the complexity of multi-partner data agreements all compound to make truly robust FL deployments difficult.

Nonetheless, these smaller real-world pilots do reveal a strong potential to address pressing agricultural issues, especially around irrigation optimisation and crop health monitoring. Singh and Adhikari [91], for example, highlight how local computations can reduce reliance on stable networks while personalising water management for each paddy field node. Meanwhile, Devaraj et al. [79] demonstrate that FL-based analytics can continuously adapt to localised microclimate changes, bridging soil sensor data, weather patterns, and hierarchical model aggregation. Such successes underscore how FL, if systematically scaled, could deliver near-real-time agronomic intelligence while safeguarding farmer data confidentiality.

However, none of the thirteen studies shows a large-scale, multi-farm system operating over multiple planting cycles or across significantly varied geographic regions. The fundamental question, “Is the FL solution deployed in a real-world agricultural setting?”, receives only a partial “yes” for a minority of the

works. Despite valuable efforts by Kaushal et al. [141], Devaraj et al. [79], Singh and Adhikari [91], and Pincheira et al. [142], the pilot nature of these deployments indicates that FL in farming is only beginning to be field-tested. Broad, commercial-level usage or multi-year validation is not yet in evidence, reflecting the research community's ongoing struggle to transition from demonstration projects to fully fledged industrial adoption.

Data Privacy Measures

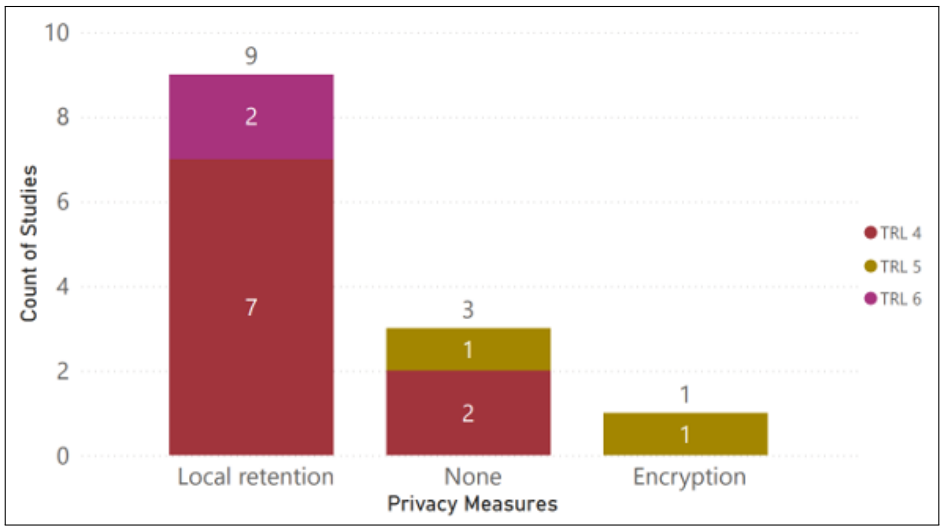


Figure 2.14: Data privacy strategies in FL studies

Data privacy is often cited as a core justification for FL in agricultural IoT contexts, because farmers are typically wary of surrendering sensitive operational data to central servers. Although all thirteen studies [79, 91, 132–142] commend FL's "data-never-leaves-the-device" paradigm, the actual depth of privacy engineering and scrutiny varies widely. Most works conflate FL's distributed structure with inherent data protection but seldom explore more advanced cryptographic or differential-privacy solutions in a detailed, production-oriented manner. The following deeper examination underscores both the successes and limitations evident across these studies.

A recurring thread as shown in figure 2.14 is the assumption that local data retention alone is sufficient to ensure privacy. Aggarwal et al. [132], Kabala et al. [136], and Aggarwal and colleagues [137] illustrate how leaf images or sensor readings never physically leave each farm node, thereby precluding centralised data gathering. This design choice undoubtedly lowers the risk of direct data exfiltration. However, few authors go further to address how adversarial participants might reverse-engineer local data distributions from aggregated model

updates - an issue known from existing FL research in other domains. For example, some works in medical FL investigate secure aggregation or gradient encryption to mitigate potential model-inversion attacks, yet these advanced techniques barely appear in the agricultural literature. The field would benefit from systematic testing of these methods under farmland conditions, especially given that distributed IoT nodes can be physically compromised.

Kaushal and colleagues [141] provide one of the few works that explicitly references encryption to protect weed-related image data and secure heuristic load distribution. Although they do not detail the exact cryptographic protocols or overhead analyses, the mention reflects a growing recognition of potential data breaches even within FL's decentralised setting.

Devaraj et al. [79], operating in a real horticultural environment, do use hierarchical FL to minimise data centralisation but refrain from a deep dive into secure multi-party computation or encryption overhead on edge devices. This underscores a broader shortcoming: while local training might protect farm data from direct exposure, it can also hamper model accuracy if the environment's device constraints prevent frequent updates or adoption of advanced privacy layers.

Singh and Adhikari [91] frame data protection as a key advantage of doing computations on the spot. However, like many others, they limit the discussion to the principle that "raw data stays on farm premises," not addressing malicious participants or gradient leakage. By contrast, Praharaj et al. [138] and Choubey and Divya [133] champion local training as essential for protecting farmers' sensitive conditions and disease data, but they focus more on algorithmic performance and do not delve into sophisticated cryptographic protocols beyond local data retention. Equally important is the real-world acceptability of these privacy approaches. Few authors measure how farmers perceive their privacy or whether they demand more assurances. For instance, Kaushal and colleagues [141] mention secure heuristics but offer no user feedback on whether these measures were deemed trustworthy by farm operators. Similarly, Thonglek and Rattanatamrong [140] discuss a neighbor-based FL strategy that reduces reliance on a central aggregator, but do not incorporate interviews with farmers or agribusiness stakeholders about data ownership, compliance obligations, or potential liability in the event of a breach.

The technical overhead of advanced privacy solutions is another pressing issue. Techniques such as differential privacy, secure aggregation, homomorphic encryption, or multi-party computation can tax limited sensor hardware. Some mention arises in Aggarwal et al.'s research on rice-leaf disease [137], which tries to compress updates for resource efficiency, but not specifically for stronger privacy. Similar compression or pruning might be adapted to reduce the cost of cryptographic techniques, but no study provides a robust evaluation of that trade-off in field conditions. Given that farmland nodes often rely on intermit-

tent power or mobile data links, slow or frequent transmissions could degrade real-time analytics. Without systematically measuring overhead, it is unclear how "privacy-heavy" techniques might scale to hundreds or thousands of devices across multiple fields and growing seasons.

In sum, although the impetus for FL in agriculture often stems from privacy concerns, the thirteen reviewed works [79, 91, 132–142] treat advanced privacy measures unevenly. Most uphold local data storage as the primary shield; only a handful mention explicit cryptographic measures, and they do so without rigorous overhead or user-acceptance evaluations. However, this review's emphasis solely on deployment-focused research (n=13) could lead to a neglect of significant data privacy aspects discussed in non-deployment literature. This constraint may omit crucial theoretical underpinnings and interdisciplinary perspectives, including advancements in cryptographic protocols and socio-legal safeguards relevant to data protection in agricultural federated learning. However, in our literature, it suggests a research gap at the intersection of strong privacy tools (such as secure aggregation or differential privacy) and the unique resource constraints in farmland IoT. Further, while many papers highlight that privacy fosters trust, none systematically investigates how farmers, co-ops, or supply-chain partners respond to the FL solutions deployed.

Scalability Testing

While FL has demonstrated promise in small pilots and lab-based proofs of concept, its capacity to function effectively across sprawling farmland or extensive sensor networks remains underexplored in the thirteen reviewed studies [79, 91, 132–142] . Most papers limit themselves to either dataset-driven experiments or modest real-world demonstrations involving relatively few devices.

A common thread involves reducing the bandwidth impact of frequent model updates. Multiple studies highlight that naive FL protocols generate considerable network traffic in proportion to the number of participating sensors, thus impeding multi-site scalability. To address this, some propose compressing or pruning model parameters before transmitting updates, while others transmit only sparse gradients. However these techniques, featured most explicitly in works by Choubey and Divy [133] and Aggarwal et al. [137] focusing on resource efficiency, tend to remain confined to simulations or small testbeds rather than rigorous multi-farm field campaigns. As a result, it is unclear whether compressed or sparse updates fully alleviate congestion when potentially hundreds of farmland sensor nodes are online simultaneously. Even the neighbor-based approach by Thonglek and Rattanatamrong [140], which avoids a single central node by structuring local devices in a neighbor-based topology, relies on a limited-scale prototype and does not measure how well the approach endures escalating node counts, intermittent connectivity, or hardware failures.

A second aspect involves managing the computational load that arises when each sensor or edge device must train a local model. In relatively controlled setups, such as those presented by Aggarwal et al. [132] or Fahim-Ul-Islam et al. [134], the authors emphasise the feasibility of running FL on low-powered devices, but do not provide multi-node performance metrics under real deployment conditions. Meanwhile, Praharaj et al. [138] discuss cooperative analytics for anomaly detection across a handful of IoT gateways, again missing the step of testing hundreds of widely dispersed farm devices. The same gap appears even in orchard or paddy-field contexts. Singh and Adhikari [91] and Devaraj et al. [79] do deploy real sensors outdoors, confirming FL viability in a small region, yet they do not investigate scaling beyond a limited cluster of sensor nodes. While these results underscore FL's potential to coordinate local computations at the farm edge, they do not confirm that the approach can handle dozens of fields or thousands of sensors.

Data heterogeneity also factors into scalability concerns. Because farmland soils, climate conditions, and sensor calibrations can differ radically between locations, FL solutions must cope with non-identical data distributions (non-IID). Although the inherent design of FL lends itself to handling non-IID data, none of the works details a large deployment with drastically diverse conditions that might degrade model convergence or hamper training reliability. Even in approaches that incorporate more sophisticated hardware or alternative data-collection strategies, multi-node scalability is not subjected to thorough stress tests. Pincheira et al.'s UGV study [142] stands out by replacing numerous static sensors with a single unmanned ground vehicle that roams orchard rows, thus elegantly sidestepping the need to manage crowds of devices. While effective at bridging coverage gaps, this strategy does not confirm how well the solution handles expansions if multiple UGVs or a large orchard area are deployed. Kaushal and colleagues [141] explore security-aware load balancing for weed detection, referencing a distributed environment in rural settings. However, they supply no extensive multi-site or multi-season experiment that would validate how well the architecture scales beyond a pilot site.

Overall, the studies recognise that FL's iterative communication and local training demands become major hurdles as the number of devices or geographical spread increases. Communication overhead, device heterogeneity, and limited on-device computation remain the principal stumbling blocks; yet the authors often address them with partial solutions, such as compressive updates or hierarchical topologies, without robust field validations. Hence, while these papers articulate many of the correct strategies for managing scale (e.g., model compression, neighbor-based FL, edge-centric computing), they only provide small-scale or lab-based evidence to substantiate those claims. None examines in detail how solutions behave with dozens or hundreds of IoT nodes across large tracts of farmland or multiple harvest cycles.

Consequently, the answer to whether scalability is adequately tested is largely

negative for most of the thirteen studies [79, 91, 132–142] . A few demonstrate modest expansions, verifying feasibility in orchard or single-farm contexts, but none achieve the breadth, node count, or temporal longevity needed to confirm truly scalable performance. Addressing scalability will require more ambitious multi-location field trials, robust fallback mechanisms for unstable connectivity, and integrated resource analyses that track battery usage, communication failures, and training latencies at different node densities. Only through these large-scale evaluations can researchers confirm FL's capacity to coordinate thousands of devices effectively, maintain reliable operation over entire growing seasons, and thereby justify broader adoption in the agricultural sector.

Regulatory Compliance

The question of regulatory compliance, spanning data protection laws, environmental regulations, liability, and other legal frameworks, looms large for FL in smart-farming IoT, yet the thirteen papers [79, 91, 132–142] rarely explore it in any depth. While many authors emphasise privacy or security from an engineering standpoint, explicit references to GDPR, industry standards, or potential liability regimes are largely absent. This gap underscores a persistent disconnect between FL's promising technical advances and the often stringent legal requirements that accompany data-driven agricultural systems.

Across these works, regulatory compliance emerges most frequently as an implied motivation for local data processing. The notion is that retaining data on-farm sidesteps certain privacy obligations mandated by national or international laws. For example, Devaraj et al. [79] and Singh and Adhikari [91] each tout data minimisation by ensuring that sensitive sensor readings remain at the local node rather than a centralised server. However, neither explicitly maps these design choices to compliance with, say, the GDPR's principles of purpose limitation or data minimisation. Instead, the discussion remains generic, assuming that local data retention alone suffices to mitigate legal exposure - an assumption that may not hold if regulators consider model gradients or updates as personal or farm-proprietary data. In fact, several papers that mention security or encryption [138, 141, 142] stop short of specifying how these measures fulfill or interface with formal legal obligations. A handful of the studies do hint at compliance challenges or obligations, but usually in very broad terms. Thonglek and Maipradit [135] focus on environmental monitoring in Thailand, where water-resource management can be subject to government oversight, but the authors do not delve into specific Thai data or environmental laws. Fahim-Ul-Islam et al. [134] and Sharma et al. [139] discuss advanced ML techniques for disease detection and data augmentation, yet do not address potential legal ramifications if inaccurate predictions lead to overuse of pesticides or other regulatory missteps. In short, these discussions remain on the margins, overshadowed by the core technical agendas of FL algorithms, model accuracy, and device-level constraints. Similarly, none of the references systematically explores how to assign liability in FL-based decision-making. If a cooperative of farm-

ers or an agribusiness collectively trains a model that yields erroneous advice, such as an incorrect irrigation schedule or an underestimation of disease severity, there is no consensus on who might bear legal responsibility. Kaushal et al. [141] and Praharaj et al. [138] highlight cooperative data sharing but do not elaborate on whether formal contracts or disclaimers exist to manage risk in the event of financial or environmental harm. This omission suggests that, despite FL's potential to unify distributed data, the question of accountability in AI-driven agriculture remains largely uncharted in academic discussions.

The absence of regulatory references in the majority of these papers, then, is stark. Authors concentrate heavily on system efficiency, data privacy, or edge-based resource usage but rarely incorporate compliance frameworks from relevant jurisdictions - be they the European Union's GDPR, the U.S. Farm Bill's data guidelines, or local agricultural extension regulations. The sole exception might be tangential references that local data retention "helps with compliance," but they do not assess whether gradient exchanges or aggregated model parameters might still inadvertently reveal personal or farm-specific proprietary data. Nor do they explore disclaimers or certifications that might be essential for large-scale commercial adoption.

Looking ahead, bridging this legal–technical gap will be essential if FL is to move from pilot projects to robust, real-world adoption in agriculture. Legal experts typically emphasise "privacy by design," auditing and logging mechanisms, and clear assignment of liability - all of which go beyond the narrower question of how to keep raw data local. Cooperative ventures among agritech companies, farm co-ops, or government agencies may push for standard data-sharing contracts that reflect FL's decentralised architecture while aligning with codes of practice or environment-related policies. Researchers could then adopt frameworks akin to those proposed in other sectors, such as secure record-keeping, verifiable claims, or tamper-proof audit trails, to demonstrate that FL solutions are not merely technologically sound but also lawfully compliant and responsibly governed. Therefore, while FL might inherently address some privacy obligations by limiting data movement, none of the thirteen works [79, 91, 132–142] undertakes a serious evaluation of how well their solutions comply with existing regulations, nor do they propose new legal guidelines tailored to distributed AI in agriculture. Until these issues receive deeper attention, real-world deployments, especially at scale may stall, hindered by concerns about liability, data ownership, cross-jurisdiction data flows, and overall trust. Further collaboration between legal scholars, policymakers, and AI researchers will be crucial in developing consistent standards that can pave the way for broad adoption and regulatory acceptance of FL-based smart-farming solutions.

Farmer Feedback

Although farmers are ultimately the end users of smart-farming IoT and FL systems, the thirteen reviewed studies devote relatively little direct attention to

capturing farmers' perspectives. Most works emphasise technical innovations, without systematically engaging those who will operate these solutions daily. This gap becomes evident when we examine whether and how authors report farmer insights, adoption barriers, or usability testing in real-world agricultural settings.

Across all thirteen papers, none provides robust, structured evaluations of farmer satisfaction or acceptance. Instead, references to end-user perspectives typically appear as brief mentions of user-friendliness or potential training needs. For instance, Devaraj et al. [79] address the importance of a user-friendly interface for sensor systems, hinting that farmers require straightforward controls and displays to benefit from hierarchical FL in tomato fields. Thonglek and Maipradit [135] similarly note that participants in Thailand's water-level monitoring project would benefit from intuitive dashboards or alerts; however, the paper does not include direct interviews or feedback sessions with local growers.

In the research by Singh and Adhikari [91] and Pincheira et al. [142], the authors underscore reliability and scalability in real-world farm environments, implicitly recognising that farmers value dependable, low-maintenance systems. However, they stop short of detailing actual user acceptance testing or including farmers' opinions on system deployment. Some works, such as those by Kaushal et al. [141] or Aggarwal et al. [132], briefly mention rural connectivity challenges that indirectly affect farmers, but never reference whether producers are satisfied with or able to operate the technology after deployment. Even papers focusing on advanced sensor networks, like the one by Thonglek and Rattanatamrong [140], remain primarily algorithmic, with no reported stakeholder consultations.

The lack of direct farmer input in the reviewed works indicates that these solutions are, by and large, still in prototype or early pilot stages, where the urgency to prove technical feasibility overshadows the equally important need to ensure user acceptance. Farmer-centric design typically demands usability research, field demonstrations, and repeated feedback loops - factors that none of the thirteen papers fully incorporate. Consequently, while the question "Is farmer feedback incorporated in the study?" yields an entirely negative response for the reviewed literature, the reasons vary. Some authors appear to assume that local, on-device data processing will automatically align with farmer preferences for privacy or autonomy, while others allude to eventual training sessions without describing how or when feedback will be collected. This discrepancy highlights a fundamental gap: technology that is scientifically sophisticated will face adoption hurdles if it does not meaningfully address farmers' practical concerns and constraints, such as installation costs, time burdens, interface complexity, or trust in machine-generated advisories.

Moving forward, integrating farmer feedback more thoroughly could involve conducting structured surveys, co-design workshops, or pilot programs that ex-

tend beyond single-season tests. Such collaborations would help refine both the user interface and the system's technical underpinnings, ensuring that FL solutions tackle real agronomic pain points rather than hypothetical ones. Furthermore, deeper involvement of farmer cooperatives or agricultural extension services might expedite user adoption by clarifying technology benefits and offering hands-on training. Therefore, while the reviewed works [79, 91, 132–142] demonstrate forward momentum in FL-based IoT for agriculture, most remain at a developmental stage without substantial farmer participation. Achieving broader impact will likely require more intentional, farmer-centric research to confirm that these emerging solutions genuinely solve producers' problems and do so in a user-friendly, locally appropriate manner.

Table 2.2: SLR Summary table

Paper	TLR	IoT Device Integration	Real-World Deployment	Data Privacy Measures	Scalability Tested	Regulatory Compliance	Farmer Feedback
[141]	5	Edge	Partial	Encryption	Conceptual	No	No
[132]	4	Sensors	No	Local retention	No	No	No
[79]	6	Sensors	Yes	Local retention	Conceptual	No	No
[133]	4	Edge	No	Local retention	No	No	No
[135]	4	Edge	No	None	Conceptual	No	No
[134]	4	None	No	Local retention	Conceptual	No	No
[91]	6	Edge-Sensor Fusion	Yes	Local retention	No	No	No
[139]	4	None	No	None	No	No	No
[136]	4	None	No	Local retention	Conceptual	No	No
[137]	4	Edge	No	Local retention	No	No	No
[138]	4	Edge-Sensor Fusion	No	Local retention	No	No	No
[140]	4	Edge	No	Local retention	No	No	No
[142]	5	Edge	Partial	None	Conceptual	No	No

2.4.4 SLR Conclusion

This systematic review reveals a fundamental gap between the theoretical promise and real-world applicability of FL in smart farming. While FL has demonstrated strong potential for privacy-preserving AI, decentralised computation, and efficient resource management, its deployment remains constrained by technological, operational, and socio-regulatory limitations. The majority of studies reviewed exist within TRL 4–6, focusing on proof-of-concept experiments, controlled lab settings, and small-scale pilots, but falling short of large-scale, field-ready implementations. Although some research ventures into partial real-world deployments, none exhibit the necessary robustness, scalability, or longevity required for sustained agricultural adoption. This disparity underscores the well-documented "valley of death" in translational AI research, where promising technologies struggle to transition from prototype to practical deployment.

Scalability remains one of the most significant barriers to FL adoption in smart farming. While many studies propose model compression, hierarchical aggregation, or edge-based processing, none provide compelling evidence that these solutions can sustain performance across thousands of IoT devices, diverse ge-

ographic locations, or multiple growing seasons. Smart farming environments demand resilient, distributed, and low-latency AI models that can function under unpredictable connectivity, sensor failures, and extreme weather conditions - challenges that existing studies fail to address comprehensively. Without rigorous multi-farm, multi-year stress testing, the transition from research to industry-ready solutions remains an open question.

Beyond technical feasibility, regulatory compliance and user acceptance represent major hurdles. Despite FL's emphasis on data sovereignty and decentralised learning, none of the reviewed studies substantiate how their frameworks align with regional and international privacy laws, such as GDPR or national agricultural policies. Legal ambiguities surrounding liability, data ownership, and model governance remain unresolved, raising concerns about accountability when AI-driven recommendations lead to unintended consequences, such as yield losses or resource mismanagement. Furthermore, the lack of farmer engagement and co-design efforts in existing research is particularly concerning. Adoption of AI in agriculture hinges not just on technical efficiency, but also on whether farmers perceive it as trustworthy, usable, and beneficial to their operations. Without systematic end-user testing, iterative design feedback, and economic feasibility analyses, FL risks becoming another technologically sophisticated but impractical solution.

It is important to highlight the limitation of this review. While we prioritised deployment-oriented studies (n=13) to evaluate FL in smart farming, this narrow scope risks overlooking insights from non-deployment focused literature, such as theoretical frameworks, policy analyses, or socio-economic studies that may address gaps in scalability, regulation, or farmer engagement. By excluding papers without explicit deployment testing, the review necessarily omits broader interdisciplinary discussions that could contextualise FL within agricultural systems. However, the 13 included studies, operating at TRL 4-6, represent the most advanced efforts to date in field-oriented FL, emphasising a real-world validation over purely algorithmic innovation. This focus ensures assessment of deployment challenges but may underrepresent emerging trends in early-stage research, such as, novel cryptographic protocols and policy frameworks. For FL in smart farming to move beyond theoretical experimentation and into industry-wide deployment, future research must prioritise large-scale, cross-season evaluations, ensuring that systems can handle heterogeneous data, environmental uncertainties, and real-time agronomic decision-making. Also, interdisciplinary collaboration is critical; AI researchers must work alongside agronomists, policymakers, and rural technology providers to standardise data governance, establish compliance frameworks, and develop practical deployment strategies. Equally important is the need for farmer-centric innovation: without user-friendly interfaces, clear incentives, and transparent decision-making models, even the most advanced FL solutions will struggle to gain traction. Ultimately, the goal for researchers in this space isn't just theory; it's about creating truly enabling technologies capable of helping solve critical problem,

from climate adaption to feeding a growing world, where current approaches fall short. Progress toward these goals depends on building a foundation from reproducible findings. Our systematic review contributes to this by establishing, for the first time, a multi-dimensional benchmark for the deployment readiness of FL in smart farming. It's difficult to gauge the pace of progress without prior reference points. Therefore, our baseline provides this starting point, allowing future work to target the identified gaps more effectively and allowing future research efforts to measure actual progress, that can be reproduced objectively, in the coming years. Knowing this initial state helps gauge whether the current pace of development holds realistic potential for realising the advanced, interconnected systems often associated with Agriculture 5.0, moving beyond the foundations laid by Agriculture 4.0.

This background chapter, combined with the SLR findings, sets the stage for our thesis to precisely target those needs: building and evaluating a federated learning testbed that brings theoretical advances into a practical, scalable, and evaluated-in-context solution for smart agriculture.

2.5 Identifying the Research Gap

The review of enabling technologies and FL principles highlights the theoretical suitability of FL for addressing privacy and data distribution challenges in smart agriculture. However, the discussion of practical hurdles (Section 2.3) underscores the significant difficulties in implementing FL effectively, particularly concerning statistical and system heterogeneity, communication limits, and security. The SLR (Section 2.4) aimed to assess the extent to which these challenges are being overcome in practice. The SLR findings indicate a notable gap between theoretical FL research and validated implementations in operational agricultural environments. Current studies frequently rely on simulations or limited trials, which may not fully capture the complexities of physical hardware, network dynamics, and the specific requirements of agricultural IoT systems. This limits the understanding of FL's practical performance and deployability in this domain.

Based on this synthesis, the following specific research gaps are identified:

Lack of Representative, Reproducible Physical Testbeds

There is a notable scarcity of documented, reproducible physical testbeds utilising hardware characteristic of agricultural edge deployments. This limits empirical validation of FL performance under realistic hardware and network conditions specific to agriculture.

Limited Empirical Data on End-to-End System Performance

Consequently, insufficient empirical data exists quantifying the holistic performance of FL systems on resource-constrained agricultural edge devices. Key metrics are predominantly estimated via simulation rather than measured on relevant physical infrastructure.

Insufficient Understanding of Simulation vs. Reality Discrepancies

The prevalent reliance on simulation necessitates a critical examination of how well simulations predict real-world FL performance on edge hardware. There is a gap in research directly comparing key performance metrics obtained from simulation versus those measured on a physical testbed using identical configurations, particularly regarding factors like training time and potential bottlenecks.

Need for Practical Deployment Insights Derived from Empirical Evidence

While theoretical challenges of FL are well-discussed, there is a need for practical insights and recommendations regarding FL deployment in agriculture that are directly derived from the experience and quantitative data obtained through operating and benchmarking a physical edge testbed. Translating empirical findings into actionable guidance for stakeholders remains an underexplored area.

2.6 Chapter Summary

This chapter provided the essential background for the thesis, beginning with an overview of the digital transformation reshaping agriculture into the data-intensive paradigm of Agriculture 4.0. The role of enabling technologies, particularly the Internet of Things and edge computing infrastructure, was examined, establishing how modern farming practices generate vast, decentralised datasets often subject to privacy or proprietary concerns (Section 2.1). This context sets the stage for exploring alternative machine learning approaches. Federated Learning was presented as a compelling solution, offering a framework for collaborative intelligence that inherently respects data locality and privacy constraints (Section 2.2). Despite its theoretical appeal for smart agriculture, the discussion then pivoted to the significant practical obstacles hindering its widespread deployment (Section 2.3). Key challenges elucidated include managing the inherent heterogeneity of agricultural data (non-IID) and compute resources, navigating unreliable or limited network connectivity, ensuring robustness against security threats, and addressing the considerable software engineering demands of these complex systems. To gauge the current state of practical application, a Systematic Literature Review was conducted (Section 2.4). The findings from this review were crucial, underscoring that while FL research in agriculture is active, it largely remains confined to theoretical studies

or simulations, with limited evidence of robust, end-to-end validation on physical systems representative of farm environments. Taken together, the identified practical challenges and the current assessment reveal critical deficiencies (Section 2.5), primarily the lack of dedicated, reproducible testbeds for empirical evaluation, a resulting scarcity of real-world performance data, and uncertainty regarding the applicability of simulation results. This thesis directly addresses these gaps by proposing, implementing, and evaluating a physical Federated Learning testbed tailored for smart agriculture, thereby aiming to provide crucial empirical insights to advance the field.

Chapter 3

Design of the Federated Learning Testbed

3.1 Conceptualising the Testbed

Designing a serverless federated learning (FL) testbed for smart agriculture requires balancing realism with experimental control. Our objective was to create a realistic FL testbed specifically for smart agriculture, aiming to emulate a network of edge devices, like farm sensors or smart cameras, collaboratively training a model without centralising their data, leveraging a serverless execution model. We conceptualised a small-scale but representative testbed composed of six Jetson Nano devices acting as federated nodes. In our design, reflecting a hierarchical architecture often suitable for serverless FL, one node assumes a coordination role responsible for orchestrating the process, one acts as the global aggregator of models, and the others function as worker devices that perform local training. This setup mirrors typical FL scenarios but relies on a specfic software stack designed for distributed, serverless-style execution rather than a traditional central server managing all client interactions directly.

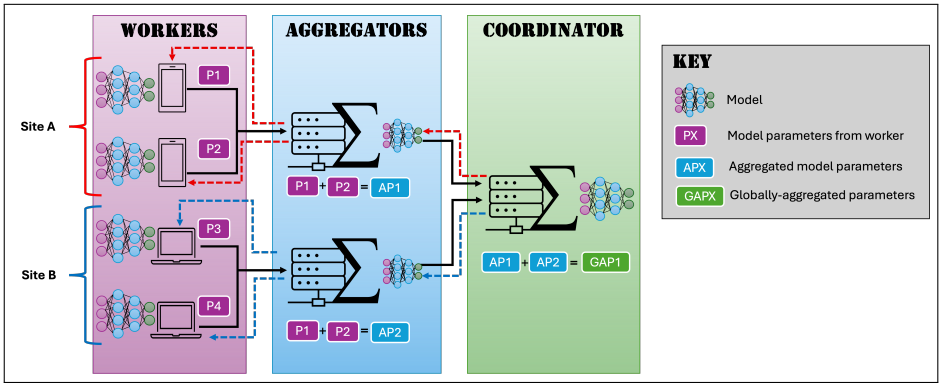
To achieve this serverless operation and manage the FL process effectively across these distributed nodes, we employed a specific software stack consisting of FLIGHT, Globus Compute, and ProxyStore, elaborated further in Section 3.4. This stack is important because it provides the necessary infrastructure for running distributed computations without dedicated server management.

By using physical edge hardware like the Jetson Nanos and establishing realistic networking conditions over standard Wi-Fi, the testbed provides valuable insights into real-world performance bottlenecks and behaviours, while remaining manageable in size for thorough analysis. Importantly, we prioritised reproducibility and scalability from the outset. Reproducibility is done through documented setup scripts (detailed in the Appendices), allowing others to recreate

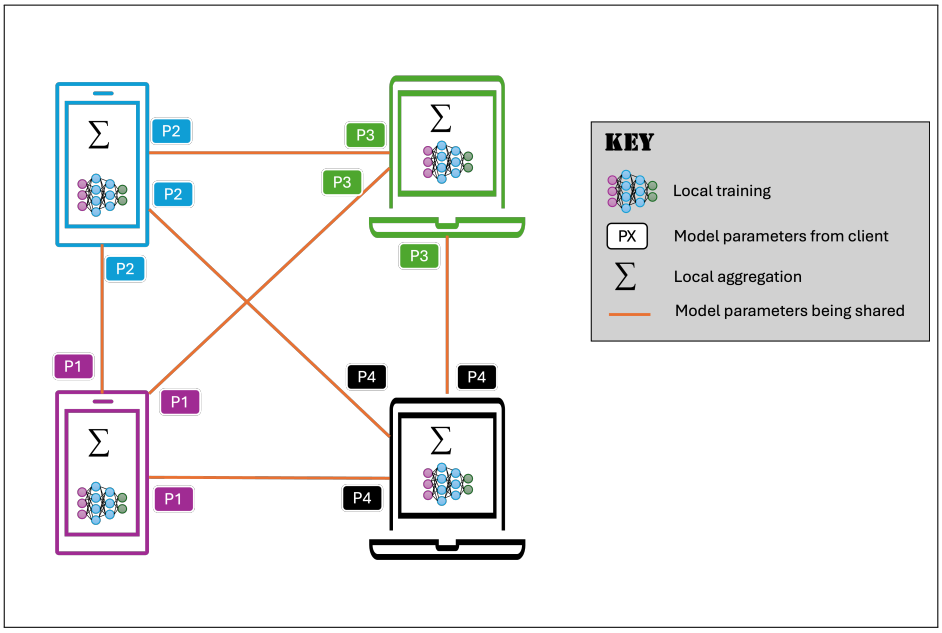
the environment precisely. Scalability is built-in via a modular design, making the testbed easily extendable to include more nodes or potentially additional architectural tiers. This means our results can be confidently generalised to larger, more complex scenarios.

3.2 FL Architecture: Hierarchical vs. Peer-to-Peer

A critical design decision was the choice of FL network architecture. We evaluated two primary approaches: a hierarchical (multi-tier) topology and a peer-to-peer (fully decentralised) topology, as illustrated in Figure 3.1. Understanding the trade-offs between these is important for designing an effective FL system for agriculture. In a hierarchical FL (HFL) architecture, depicted conceptually in Figure 3.1a, nodes are organised in tiers. Clients, which can be sensors or edge devices, perform local training and send their model updates upwards to designated intermediate aggregator nodes, for example a farm gateway. These intermediate aggregators combine the updates received from their assigned clients and, in turn, send a single, aggregated update further up the hierarchy, potentially to a top-level global aggregator such as a central cloud server. The key characteristic is this structured, typically vertical, flow of information with aggregation occurring at specific points within the hierarchy. Conversely, a peer-to-peer (P2P) FL architecture, shown in Figure 3.1b, operates without any designated central or intermediate aggregators. As the connecting lines in Figure 3.1b illustrate, clients communicate directly with each other (their "peers" or neighbors) to exchange model updates using various protocols. Convergence towards a global model happens emergently through these repeated peer interactions.



(a) HFL architecture



(b) P2P architecture

Figure 3.1: Illustration of a hierarchical federated learning architecture (a) vs. a fully peer-to-peer architecture (b).

Comparing these architectures for agricultural settings, the hierarchical approach (Figure 3.1a) often presents advantages in managing communication overhead. By performing aggregation at intermediate nodes, such as a field gateway combining updates from numerous local sensors before sending a single aggregated update to a central farm server, HFL significantly reduces traffic over potentially constrained wide-area or backbone network links. This contrasts

with a P2P setup (Figure 3.1b) where updates might need to propagate widely across many peer-to-peer links to ensure global convergence, potentially leading to a higher total number of message transmissions and greater cumulative bandwidth usage across the entire system, especially if peers need to communicate frequently across different fields or networks. While the total data aggregated might be similar, HFL's ability to concentrate traffic over fewer, potentially higher-capacity links is often more efficient, with studies indicating significant reductions in communication overhead compared to flatter topologies [71].

In terms of complexity and fault tolerance, HFL centralises the primary algorithmic complexity at the aggregator and coordinator nodes, keeping the client logic relatively simple (i.e. train and send). While the aggregator represents a point of potential failure, the failure of an intermediate aggregator typically only impacts its specific branch, allowing other parts of the hierarchy to continue functioning, therefore offering good fault tolerance. The P2P architecture (Figure 3.1b), while offering excellent fault tolerance due to the absence of any single point of failure, distributes significant complexity across all participating nodes. Each peer must potentially handle neighbor discovery, peer selection, decentralised averaging or consensus protocols, and message management, increasing the computational and algorithmic burden on potentially resource-constrained edge devices.

Both architectures can achieve high scalability in principle. HFL scales naturally by adding more clients under existing aggregators or adding new tiers to the hierarchy. P2P can also scale to large numbers, but managing the network topology, ensuring efficient information propagation, and guaranteeing convergence become increasingly complex algorithmic and networking challenges as the number of nodes grows. Furthermore, the hierarchical model often aligns more readily with existing FL frameworks and deployment tools, including FLIGHT used in our testbed, which natively support centralised or tiered aggregation, simplifying implementation compared to building robust P2P protocols from scratch.

Considering the specific context of smart agriculture, where devices may be naturally clustered such as per field, where minimising traffic over strained wide-area links is important, and where manageable complexity is preferred for deployment - the Hierarchical FL architecture (Figure 3.1a) emerged as the more compelling choice for our testbed. It offers a pragmatic balance, prioritising communication efficiency and manageable implementation complexity while providing good fault tolerance and clear pathways for scaling, making it well-suited for developing and evaluating FL solutions in this domain. The comparative summary reflecting these points is provided in Table 3.1.

Feature	Hierarchical FL	Peer-to-Peer FL
Aggregation	Centralised	Decentralised
Complexity	Moderate	High
Communication Overhead	Lower	Higher
Fault Tolerance	Good	Excellent
Scalability	High	High (complex scaling)

Table 3.1: FL architecture comparison

3.3 Edge Hardware: NVIDIA Jetson Nano

Choosing appropriate hardware for the FL nodes was another vital design decision, requiring a platform capable of on-device training while representing realistic edge constraints found in agriculture. We looked at common types of small, self-contained computers often used for these kinds of projects, known as Single-Board Computers (SBCs). While other options exist, like very simple microcontrollers (MCUs) or highly specialised chips (FPGAs), SBCs offered the best balance [143]. MCUs are generally not powerful enough to handle the task of training an AI model, and FPGAs are complex and expensive. Among the popular SBCs suitable for AI tasks, we considered options like the well-known Raspberry Pi, Google's Coral boards, and NVIDIA's Jetson series.

After comparing these, we chose the NVIDIA Jetson Nano developer kit to serve as all the nodes in our testbed (the workers, the aggregator, and the coordinator). The Jetson Nano is specifically designed as a small, relatively low-power computer for running edge AI applications. It includes a standard main processor (CPU) but it also has a built-in Graphics Processing Unit (GPU) [144]. While GPUs were originally designed for handling graphics in video games, they turned out to be extremely good at the type of parallel mathematics involved in training AI models. Having this built-in GPU means the Jetson Nano can perform the necessary AI training tasks much more efficiently and faster than a device relying only on its main CPU.

This combination of features made the Jetson Nano particularly suitable compared to the alternatives for our project's needs. For example, while a Raspberry Pi is very affordable and popular [1], it lacks a powerful built-in GPU designed for AI. To get similar AI performance, we would likely need to attach an external AI "accelerator" chip, adding extra cost, complexity, and potential compatibility issues. The Jetson Nano includes this AI-boosting capability right on the board. Compared to Google's Coral boards [2] [3], which have a specialised Google chip (TPU) that is very fast for using pre-trained AI models (inference) with specific software [143], the Jetson Nano's GPU is generally more flexible. This

[1] `https://www.raspberrypi.com/products/raspberry-pi-4-model-b/`
[2] `https://coral.ai/products/dev-board//`
[3] `https://coral.ai/products/dev-board-mini/`

flexibility is important for research like ours where we need to experiment with the training process using standard AI development tools, like PyTorch, which are well-supported by NVIDIA's GPU technology (CUDA). Furthermore, the Jetson Nano provides a realistic amount of processing power and memory (4 GB RAM) which is enough to handle moderately complex AI models suitable for agricultural tasks, like analysing images or sensor data, but still limited enough to genuinely represent the constraints of an edge device, unlike a powerful desktop computer.

The choice of the Jetson Nano also brings practical benefits supporting our design principles. It's popularity in edge AI research also supports reproducibility and community knowledge transfer. Other researchers or practitioners can replicate our testbed setup using the same readily available hardware. Moreover, the Jetson's relatively low cost and small form-factor make it feasible to deploy multiple units across a farm or research lab. The homogeneous hardware setup (all nodes having identical specification) was chosen initially to eliminate performance variability due to hardware differences, allowing us to isolate the effects of the FL process itself and establish a clear baseline. While real-world agricultural deployments will undoubtedly involve heterogeneous devices (mixing different sensors, gateways, etc.), starting with a homogeneous foundation is a standard practice for controlled experimentation. Furthermore, the Jetson Nano includes various connectivity options, such as Ethernet and Wi-Fi via adapter, which can be utilised to network the devices. This choice of hardware has precedent: recent studies have used Jetson Nanos to evaluate resource consumption during on-device federated learning [145], confirming their suitability for handling the computational demands in FL contexts.

3.4 Software Stack: FLIGHT, Globus Compute and ProxyStore

With the hardware in place, the next design layer is the software stack required to realise our serverless FL architecture. We built our testbed on top of FLIGHT (Federated Learning in General Hierarchical Topologies) [71], a Function-as-a-Service (FaaS) based federated learning framework. FLIGHT was chosen because it is a modern framework explicitly designed to support hierarchical FL processes and asynchronous execution. Unlike traditional FL frameworks that assume a simple server-client topology, FLIGHT allows arbitrary multi-tier networks of aggregators and workers, aligning with our hierarchical design. It also separates the control plane (orchestration logic) from the data plane (model and data transfers), which is essential for flexible deployment. In FLIGHT's architecture, a Coordinator entity manages the overall FL workflow (control commands such as instructing nodes to train or aggregate), while separate Aggregator and Worker entities handle the data-intensive tasks such as receiving models, training on data, computing updates. This separation is realised by

leveraging a serverless computing approach: FLIGHT utilises Globus Compute [4] (previously known as funcX) as the FaaS platform to execute training and aggregation tasks on distributed nodes, and ProxyStore [5] to handle data movement behind the scenes.

Globus Compute is a cloud-based platform that allows us to register each Jetson Nano as an endpoint capable of running remote functions. Instead of writing our own networking or Remote Procedure Call (RPC) code, we can simply define the training procedure as a Python function and ask Globus Compute to run that function on a specific Nano endpoint. This abstraction, where the underlying server infrastructure for messaging, task queuing, and managing endpoint connectivity is handled by the platform, is a key characteristic of the serverless model we aimed for. The coordinator (which can run on a separate machine or one of the Jetsons) invokes these remote functions via the Globus Compute service. This approach greatly simplifies orchestration: the coordinator does not need direct network connectivity to each worker which can be problematic in NAT or firewall-limited environments. The Globus Compute service handles the messaging and job distribution via its cloud message queue, meaning that as long as each Nano can reach the Globus service (outbound internet access), it can receive tasks and return results. This design epitomises the serverless advantage of scalability: adding more workers only requires registering new endpoints and does not change the core logic. This design was aligned with the scalability principle which means adding more workers only requires registering new endpoints and does not change the core logic. This design also aligns with the reproducibility principle, since the same code can run in a simulated environment by registering virtual endpoints. Furthermore, Globus Compute has built-in reliability (queuing, retries) and has been used in large-scale deployments (thousands of endpoints), giving us confidence that the testbed could be extended in scale.

However, using a cloud FaaS service for FL also introduces challenges, particularly with data and model sizes. By default, Globus Compute tasks have limits on the size of input and output data. In FL, model weight objects can be large (potentially exceeding those limits), and repeatedly sending these over the control channel could become a bottleneck. To address this, FLIGHT integrates ProxyStore, which we included in our stack. ProxyStore is a data management library that enables out-of-band data transfer between endpoints. We can trace how this facilitates a typical FL round as illustrated in Figure 3.2. The process begins with the coordinator entity (green section) initiating the training round (Step 1), which involves distributing the current global model to the selected worker nodes (w1-w4, purple section). This distribution is handled by having the coordinator place the model into ProxyStore (via its orange coord endpoint), making it accessible to workers. The coordinator then uses Globus

[4] https://globus-compute.readthedocs.io/en/latest/index.html
[5] https://docs.proxystore.dev/latest/

Compute (via its blue coord endpoint) to trigger the local training function on the workers' Jetson Nanos (via their blue Globus Compute endpoints). Each worker function fetches the global model using its orange ProxyStore endpoint, trains using local data, and produces updated model parameters (grey squares). When a worker (say, W1) finishes local training, it uses its ProxyStore endpoint (orange W1) to store the potentially large updated parameters. Instead of returning the large model directly through the Globus Compute function result, which could hit size limits or cause delays, the worker returns only a small proxy (a lightweight reference) back to the system. The dotted orange lines in the figure conceptually show these parameters, referenced by their proxies, becoming accessible to the Aggregator via its own orange agg ProxyStore endpoint. Subsequently, FLIGHT, through the Coordinator, instructs the Aggregator's blue agg Globus Compute endpoint to perform the aggregation task. The aggregation function, running on the Aggregator node, receives the proxies from the workers, uses its orange agg endpoint to resolve them and efficiently fetch the actual model parameter objects via ProxyStore, and then performs the mathematical aggregation (represented by the Sigma symbol). This newly aggregated model can then be passed back, via ProxyStore, to the Coordinator to become the updated "Globally-Aggregated Model" for the next cycle. This decoupling of control (Globus Compute function calls and small proxies) and data (ProxyStore handling bulk transfers) makes sure that our large model updates do not clog the control plane. This design significantly improves efficiency and scalability, making the serverless execution of FL with potentially large models practical. It is both more bandwidth-efficient and allows us to potentially train larger models than would be possible with the FaaS service alone. The entire process orchestrated by FLIGHT, leveraging Globus Compute for execution and ProxyStore for data logistics as depicted in Figure 3.2, allows for a more robust and efficient federated learning on our edge testbed.

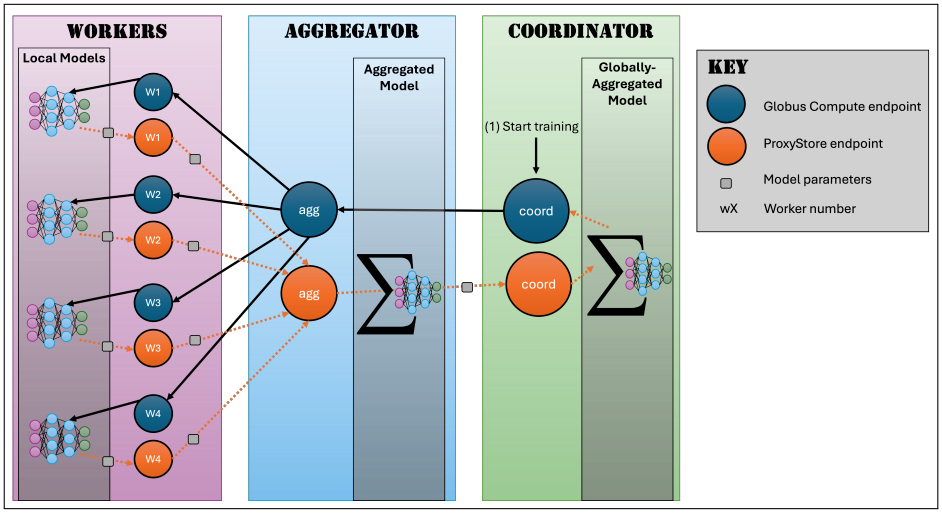


Figure 3.2: Federated learning software architecture using FLIGHT, globus compute, and proxystore

The overall software architecture can thus be summarised as follows: The FLIGHT framework provides the high-level FL logic (managing rounds, participant selection, etc.) and abstracts the use of Globus Compute and ProxyStore. Globus Compute serves as the execution layer that dispatches training tasks to Jetson Nano workers and an aggregation task to the Jetson Nano aggregator. ProxyStore operates alongside to move the model parameters between those tasks in an optimised manner. Together, these components effectively realise the serverless FL architecture envisioned for this testbed. All these components are open-source and cross-platform, which reinforces reproducibility. We configured our testbed using Python-based scripts provided by FLIGHT, customising them for our topology and adding logging (discussed later). The design ensures that whether we run on physical Jetsons Nanos or a simulated environment, the same software stack orchestrates the federated learning process in a consistent way. Any technical setup details (such as installation of the Globus Compute endpoint software on the Jetson Nanos, or the configuration of ProxyStore) are documented in Appendices B - I for completeness. By using this robust stack, we adhere to the design principles: reproducibility (others can set up the same environment using the documented tools) and scalability (the framework has been shown to scale to thousands of nodes, far beyond our current testbed, indicating room for growth [71]).

3.5 Design Principles: Reproducibility & Scalability

From the outset, we emphasised two overarching design principles: reproducibility and scalability.

Reproducibility: The testbed should yield results that can be independently reproduced and verified. To this end, we chose well-documented hardware and software. Using identical Jetson Nano units for all nodes ensures that each participant in the FL process starts from the same baseline capabilities. We also developed the testbed in a way that it can be deployed in a simulated environment (using virtual machines) in addition to the real hardware deployment. FLIGHT inherently supports both simulation and real deployment; we leveraged this by mirroring the setup on Virtual Machines (VMs). All configuration files, environment setup scripts, and the exact versions of frameworks used are recorded (see Appendices. B - I). This level of detail allows anyone reading this thesis to reconstruct the environment. Moreover, to reduce nondeterminism, we fixed random seeds in the training processes where applicable, and we ran multiple trials to ensure consistent behaviour. Reproducibility is crucial not only for scientific validity but also to facilitate the transfer of this testbed to actual agricultural field trials in the future with minimal modification.

Scalability: The testbed is designed such that it can scale in the number of devices and complexity of topology. Our choice of a hierarchical architecture is inherently scalable this means additional worker nodes can be added under the same aggregator, or further tiers of aggregators can be introduced if the network grows, without fundamental changes to the algorithm. The use of Globus Compute as an orchestration layer means that increasing the number of endpoints (devices) is straightforward: each new device just registers with the Globus service and can then participate in the federated training. FLIGHT has demonstrated scalability to large numbers of clients in research settings, which provides confidence that our testbed design will hold as we scale up. We also considered scalability in terms of data and model sizes. By incorporating ProxyStore to handle large model transfers, the testbed can accommodate more complex models that might be needed for agriculture applications (e.g., high-resolution image analysis) without redesigning the communication mechanism. Scalability in the context of smart agriculture could also mean the ability to integrate with cloud back-ends or other edge clusters; our design using modular services (FaaS, data proxy) makes such integration feasible down the line.

To summarise, the design phase produced a federated learning testbed blueprint that uses a hierarchical FL architecture deployed on Jetson Nano edge devices, coordinated via the FLIGHT framework with Globus Compute and ProxyStore. This design is justified by the need for efficient communication, realistic edge computation, and the goal of creating a testbed that is both reproducible

and scalable. The following chapter will describe how we implemented and deployed this design in practice, both on physical hardware and in a simulated environment.

Chapter 4

Implementation and Deployment

4.1 Overview of Implementation

Following the design outlined in Chapter 3, the deployment was implemented across two parallel environments: (1) a physical deployment on NVIDIA Jetson Nano devices in Section 4.2, and (2) a virtualised simulation using VMware Fusion in Section 4.3. This chapter outlines the detailed steps taken to set up the testbed and the simulation environment. It explains how each Jetson Nano was configured to act as a FL participant, details the installation and integration of key software components across all nodes and describes how the entire FL cycle was verified end-to-end. Also, modifications made to the main orchestration logic within run.py are explained. These adjustments allowed the systematic collection of operational metrics that were not readily available from the default configuration of the existing framework. For technical details, precise installation commands, example configuration files (like flock.yaml), and the complete, updated orchestration source code can be found in Appendices B - I.

4.2 Deployment on Jetson Nano Nodes

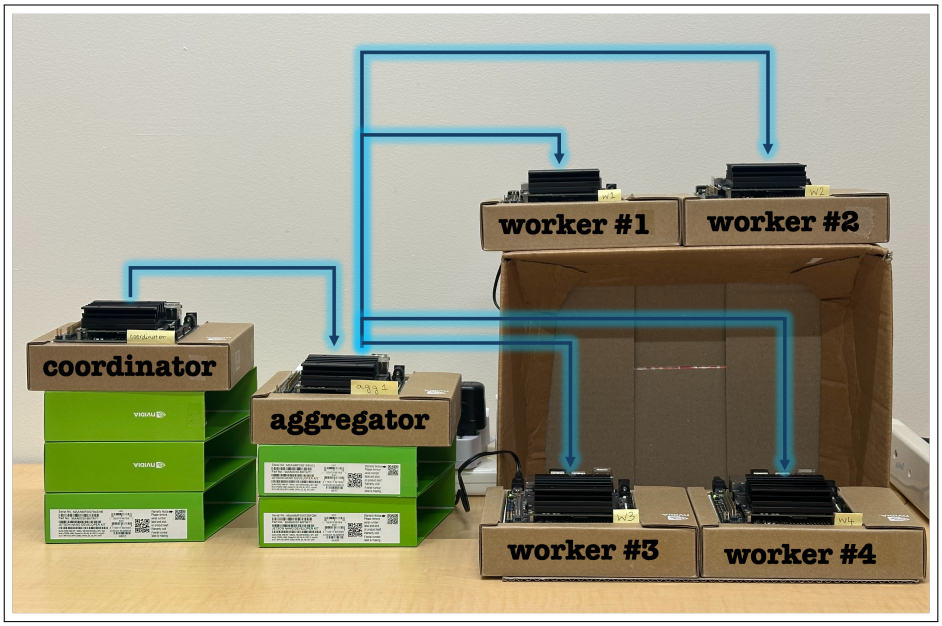


Figure 4.1: The physical FL testbed setup. Overlaid arrows illustrate the primary flow of instructions and data (e.g. model updates) between these roles during the FL process.

The physical testbed was implemented using six NVIDIA Jetson Nano Developer Kits, each equipped with 4GB of RAM. Each device was assigned a specific role within the federated learning topology: one acted as the Coordinator, another as the Aggregator, and the remaining four as Worker nodes responsible for performing local training tasks as depicted in figure. 4.1.

4.2.1 Node Configuration and Software Installation

Initially, each Jetson Nano was set up by flashing it with NVIDIA's JetPack SDK following instructions from its official documentation [1]. This provided an Ubuntu-based operating system already equipped with necessary CUDA libraries and GPU drivers optimised for the onboard 128-core Maxwell GPU. As these devices lack built-in Wi-Fi capability, we equipped each with a TP-Link TL-WN725N (N150) WiFi 4 Nano USB Wireless Adapter [2] to establish wireless network connectivity, making it easier to manage software installation and

[1] https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit\#intro

[2] https://www.pbtech.co.nz/product/NETTPL0725/TP-Link-TL-WN725N-N150-WiFi-4-Nano-USB-Wireless-Ad

device operations remotely. Detailed steps are described in Appendix. B.

A notable constraint of the physical Jetson Nano devices is their limited 4GB of RAM. To mitigate potential memory bottlenecks and ensure smoother operation, swap memory was configured on each Jetson Nano. This provided additional virtual memory, crucial for handling memory pressure not encountered to the same extent in the virtual simulation environment where RAM allocation was less constrained (4GB allocated per VM, but backed by the host's potentially larger physical RAM and more efficient OS-level swapping if needed). A 16GB swap file was created and enabled using standard Linux utilities (fallocate, chmod, mkswap, swapon) and configured to be persistent across reboots by adding an entry to /etc/fstab. Detailed commands for configuring the swap file can be found in Appendix C.1.

Once basic system access and swap configuration were confirmed, a consistent Python environment (version 3.8) was installed on each device, including critical dependencies such as pytorch, the Flox federated learning framework, Globus Compute SDK for orchestrating tasks, endpoint software for task execution, and ProxyStore for data transfers.

Each Jetson Nano was registered as a Globus Compute endpoint using command-line utilities (globus-compute-endpoint configure and globus-compute-endpoint start), assigning a unique UUID to each device. This allowed the coordinator to dispatch tasks precisely to each endpoint. Concurrently, ProxyStore endpoint services were initiated on each node, creating listening sockets for direct peer-to-peer data transfers, primarily involving the exchange of model parameters and updates. Completing these steps allowed each Jetson Nano to actively participate in the federated learning workflow, capable of receiving remote commands via Globus Compute and exchanging serialised data objects, such as pytorch model state dictionaries, through ProxyStore. Specific differences in the setup for coordinator, aggregator, and worker nodes are detailed further in Appendix C and Appendix. D.

4.2.2 Federated Learning Workflow Execution on Jetson Nano Testbed

The practical execution of the FL workflow relied on the Flox framework and the primary orchestration script, run.py, executed on the coordinator node. The process began with the coordinator reading the configuration details from the flock.yaml file (sample script provided in Appendix. I). This file defined the hierarchical structure of the network (Coordinator –> Aggregator –> Workers) and linked specific roles to their corresponding Globus Compute and ProxyStore endpoint UUIDs. These details were used to build an internal representation of the network for clear communication pathways and clearly defined responsibilities for each node. At the same time, the coordinator node also established secure connections to the Globus Compute service and initialised its ProxyStore

client, setting up connections with the ProxyStore endpoints running on both aggregator and worker nodes. This initialisation closely followed the original design intent, optimised for the limited resources available on Jetson Nano devices. It utilised hierarchical delegation and prepared efficient communication channels through ProxyStore and Globus Compute.

For data preparation, the Fashion-MNIST[3] dataset was downloaded through torchvision. This is a widely recognised benchmark in machine learning consisting of 70,000 28x28 grayscale images across 10 distinct clothing categories. Although it is not agricultural data, its selection for this system-focused study was in its several key advantages. Firstly, its frequent use in published FL and distributed ML research provides a baseline for comparability [146] [147]. This allows us to evaluate our testbed by allowing direct comparisons within this thesis. In Chapter 6, we use Fashion-MNIST as the consistent computational task to compare the performance of our simulation environment against the physical Jetson Nano testbed. For example, we compare the completion times between simulation and the physical hardware under varying numbers of workers and communication rounds (Section 6.3 & 6.4), and analyse differences in system resource utilisation like CPU load under the same task conditions (Section 6.5). While we don't perform direct numerical comparisons to specific external studies in this work (due to variations in hardware, network, and precise FL configurations), using this standard dataset ensures our internal findings on the simulation-vs-reality gap and the impact of system parameters are grounded in a well-understood task, making them more interpretable and valuable to the research community.

Secondly, Fashion-MNIST presents a classification task complex enough to impose a realistic computational workload on the resource-constrained Jetson Nano edge devices [148]. This means we are genuinely stress-testing the performance of the FL system components under a meaningful computational load, not just trivial processing. Simpler datasets, like the original MNIST digits dataset [4], while very common, often represent a computational task that is too lightweight for devices like the Jetson Nano; training might complete so quickly that it wouldn't sufficiently exercise the system or reveal potential bottlenecks in the FL orchestration or data transfer mechanisms. More complex datasets were also considered, such as CIFAR-10 or CIFAR-100 [5] (standard benchmarks involving color images, typically requiring larger models) or potential large-scale, high-resolution agricultural image datasets such as PlantVillage [6]. However, these more complex tasks risk overwhelming the Jetson Nano's specific resource constraints, particularly its 4GB RAM). Attempting FL training with such demanding datasets could lead to impractically long experiment times, memory exhaustion errors, or situations where the observed performance is almost en-

[3] https://www.kaggle.com/datasets/zalando-research/fashionmnist
[4] https://www.kaggle.com/datasets/hojjatk/mnist-dataset
[5] https://www.cs.toronto.edu/~kriz/cifar.html
[6] https://paperswithcode.com/dataset/plantvillage?utm_source=chatgpt.com

tirely dictated by the hardware's raw compute limits, thereby obscuring the performance characteristics of the federated learning software stack and protocols which were the primary focus of this investigation.

Most importantly for evaluating FL systems, Fashion-MNIST's multi-class structure facilitates the simulation of statistically heterogeneous (non-IID) data distributions. Real-world FL deployments almost invariably encounter non-IID data, where devices hold skewed or unique data subsets. Using Fashion-MNIST, we can programmatically create such distributions across our worker nodes (which we did using methods like the Dirichlet distribution), allowing us to assess how effectively our testbed architecture handle this fundamental and pervasive challenge in practical federated learning. Therefore, it must be emphasised that our primary objective was not to solve an image classification problem related to clothing, nor to directly simulate an agricultural task with this dataset. Instead, we leveraged Fashion-MNIST as a standardised, controllable, and well-understood workload specifically to perform a thorough evaluation of our FL testbed implementation under realistic conditions of data variability and distribution at the edge.

As mentioned previously, to replicate the kind of data variability typically found in real-world edge scenarios the dataset was distributed unevenly across worker nodes. This distribution was done using Flox's federated_split utility, which applied a Dirichlet distribution (with $\alpha = 3.0$, as specified in the script) to create unique, skewed subsets of data for each worker. This meant that each worker held locally distinct data. During training, ProxyStore had efficient access to these data shards through lightweight references, preventing costly network transfers of the full raw datasets.

At the start of each round, manually defined in the run.py script, the aggregator node, which maintained the global model state (implemented as SmallConvModel, a convolutional neural network defined in the script with two convolutional/pooling layers and three fully connected layers), serialised the current global model parameters and published them through its local ProxyStore endpoint. Globus Compute tasks then dispatched references (proxies) to these parameters to each worker node.

Upon receiving a training task, each worker node used the ProxyStore proxy to efficiently access the global model parameters directly from the aggregator's endpoint, benefiting from ProxyStore optimisations such as zero-copy serialisation to minimise memory use and transfer latency. The workers then performed local training.

After completing local training, each worker calculated the difference (delta) between their locally updated model weights and the global model weights they initially received. These weight deltas, being significantly smaller than full model parameters, were then uploaded to each worker's ProxyStore end-

point. ProxyStore further optimised these deltas through serialisation and optional compression. Finally, the workers returned proxies for these deltas to the aggregator through Globus Compute task results, concluding each global round.

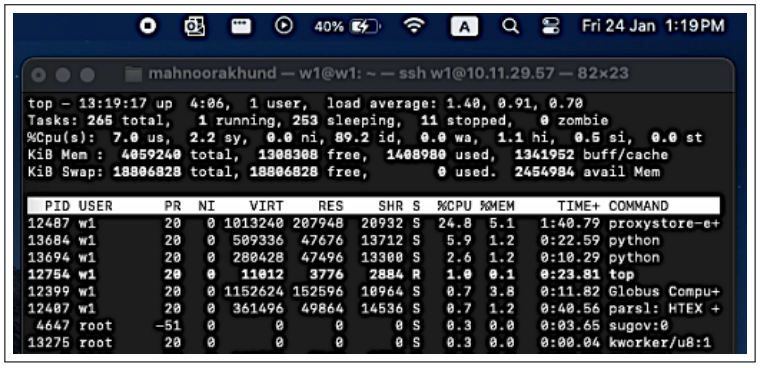
The aggregation phase took place on the designated aggregator node after all worker nodes completed their training tasks for the current round. At this stage, the aggregator received references (proxies) to the weight deltas computed by each worker. Using ProxyStore, it retrieved these deltas, reconstructed the weight updates, and carried out the Federated Averaging (FedAvg) process, specified by the strategy="fedavg" parameter in the federated_fit call. FedAvg involved computing a weighted average of these deltas, typically based on the amount of data each worker processed during local training, and updating the global model accordingly. This aggregation step mainly relied on pytorch tensor operations executed on the aggregator's ARM CPU.

Once aggregated, the updated global model parameters were published back to the aggregator's ProxyStore endpoint, making them accessible to workers for the next training round. The coordination of tasks across diverse device speeds and varying network conditions was managed through Flox's synchronisation protocol (kind="sync-v2"), leveraging Globus Compute's task synchronisation mechanisms. This approach made sure that each global round only advanced once results from all Workers were received or properly handled in cases of delays or timeouts.

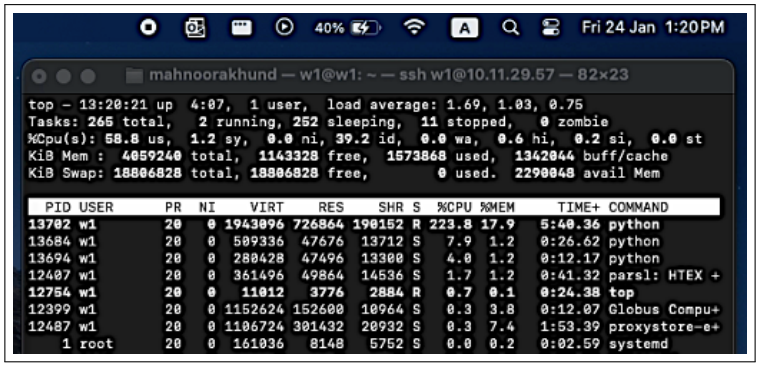
After completing all predefined global training rounds, also manually configured in the run.py script, the coordinator retrieved the final global model state from the aggregator using ProxyStore. It then conducted a final evaluation centrally, employing the standard FashionMNIST test dataset loaded through a pytorch DataLoader, measuring the model's accuracy as specified in the concluding sections of the run.py script. Finally, the system explicitly freed resources by clearing ProxyStore caches and terminating Globus Compute tasks to prevent resource leaks and ensure stable long-term operation of the edge devices.

The successful completion of this workflow on physical Jetson hardware demonstrated the integrated system's functionality and provided a realistic framework for performance assessment under typical edge computing constraints.

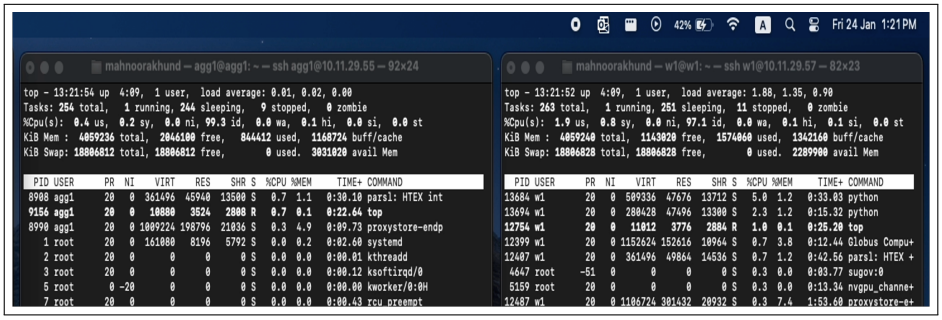
4.2.3 Monitoring Node Activity During Execution



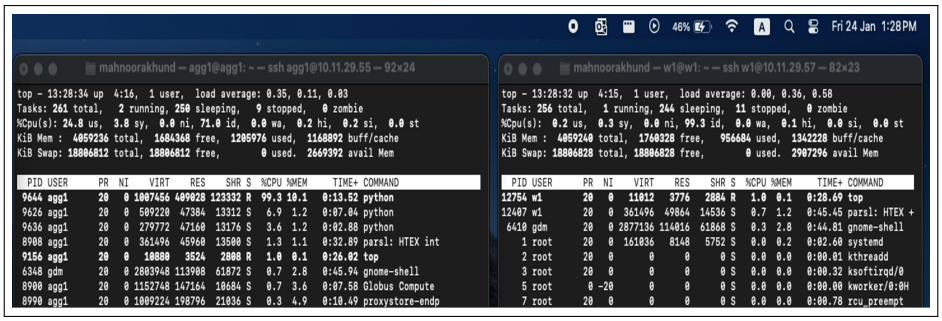
(a) Worker 1 at 1:19:17 PM, showing initial process activity and moderate load before peak training.



(b) Worker 1 at 1:20:21 PM, showing high CPU usage (223%) by the main training python process (PID 13702).



(c) Simultaneous view around 1:21:5x PM (Transition Phase). **Left:** Aggregator showing low load, waiting for updates. **Right:** Worker 1 finishing update transfer.



(d) Simultaneous view around 1:28:3x PM (Aggregation Phase). **Left:** Aggregator showing high CPU usage (99.3%) by python process (PID 9644) performing aggregation. **Right:** Worker 1 showing very low load, idle.

Figure 4.2: Snapshots of `'top'` command output on Jetson Nano nodes between worker 1 and aggregator during federated learning.

To observe the system's behaviour during the FL process executed via the run.py script, the `top` command was utilised on individual nodes accessed via ssh. This allowed for real-time monitoring of process activity and resource utilisation. Figure. 4.2 provides snapshots illustrating typical activity patterns on the Aggregator (agg1) and one of the Worker nodes (w1) during different phases of a training run.

(A) Worker Node (w1) During Active Training: The sequence of snapshots taken on Worker 1 (w1@w1, Figure 4.2a and 4.2b) illustrates the progression of activity during local training. After initial setup, at 1:19:17 PM, the system shows moderate load utlising 24.8% of CPU with active helper processes. Subsequently, after more than a minute later, at 1:20:21 PM, a specific python process (PID 13702) becomes dominant, exhibiting very high CPU utilisation of 223.8%, indicative of the core local model training computations.

(B) Transition Near Round Completion: Figure 4.2c, presents a simul-

taneous view of the Aggregator and Worker 1 captured around 1:21:5x PM, as the worker concludes its training round. On the Worker node (right), the high-CPU training process (PID 13702 from figure 4.2b) is no longer visible or is idle, suggesting local computation is complete. Communication and task management processes (proxystore-ep PID 12487, Globus Compute+ PID 12399, parsl: HTEX +) remain active, handling the finalisation and transfer of model updates. At this moment, the Aggregator node (left) shows very low load, indicating it is idle while waiting to receive updates from all workers.

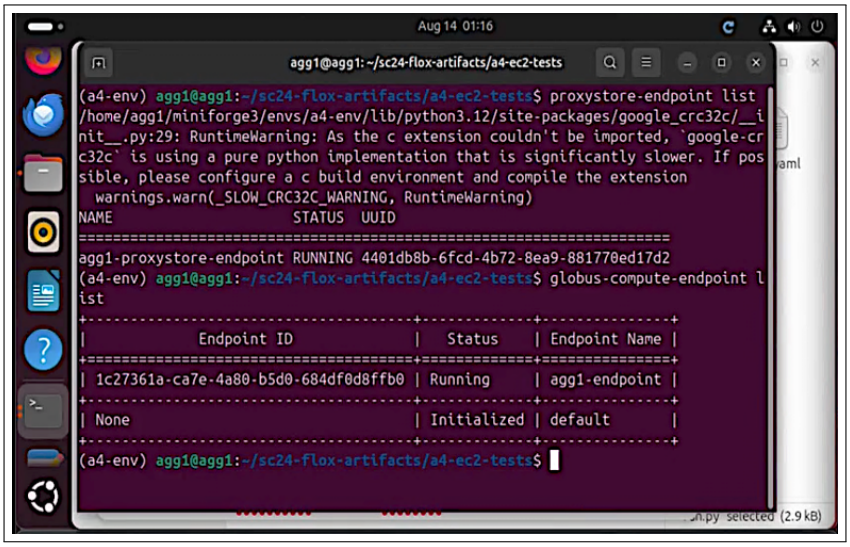
(C) Aggregator Node (agg1) During Aggregation Phase: Figure 4.2d, captured later at 1:28:3x PM, shows the resource usage during the aggregation phase. The Aggregator node now exhibits significant activity. A primary python process (PID 9644) consumes nearly 100% CPU, strongly suggesting it is performing the Federated Averaging computation by retrieving weight deltas (via proxystore-endp PID 8990) and averaging them. Other related python processes (PIDs 9626, 9636) and the Parsl/HTEX interface (PID 8908) show supporting activity. Simultaneously, the Worker node (w1@w1, right) shows very low load and minimal CPU usage, indicating it has completed its tasks for the round and is waiting for the next instruction or the experiment's conclusion.

These snapshots collectively illustrate the dynamic resource utilisation patterns: intense, multi-core computation on workers during local training; communication overhead involving ProxyStore and Globus Compute helper processes; transitional idle periods on the aggregator while waiting for workers; and a distinct computational load on the aggregator during the model averaging phase, followed by worker idle time. Therefore, monitoring process activity using tools like top was used to provide a practical means to observe and verify that the distinct phases of the federated learning cycle, local training on workers, data transfer via ProxyStore/Globus Compute, and model aggregation on the aggregator, are executing as intended within the implemented testbed.

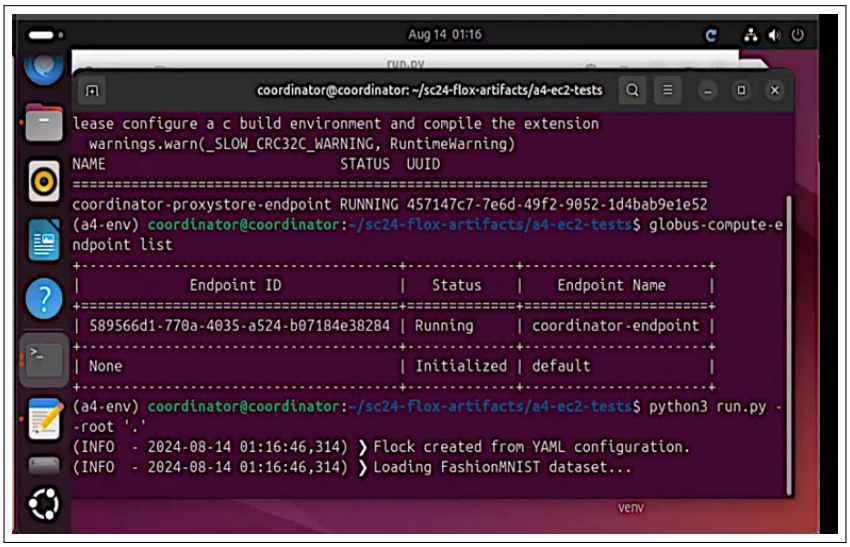
4.3 Parallel Simulation with VMware Fusion

Concurrently with the physical hardware deployment, a virtualised simulation environment was constructed utilising VMware Fusion on a host machine: a MacBook Air (Apple M3 processor, 16GB RAM) running macOS. This parallel setup served several strategic objectives: it provided a controlled environment for rapid prototyping and debugging of the federated learning logic. Six Ubuntu 20.04 LTS virtual machines (VMs) were provisioned, each allocated computational resources intended to approximate the general capabilities of the Jetson Nano devices. Each VM underwent an analogous software installation procedure, equipping it with python 3.8, the standard x86 distribution of pytorch, the Flox framework, Globus Compute endpoint software, and ProxyStore. Distinct Globus Compute endpoint registrations were created for each VM to allow the coordinator VM to address them individually, albeit all linked to a single

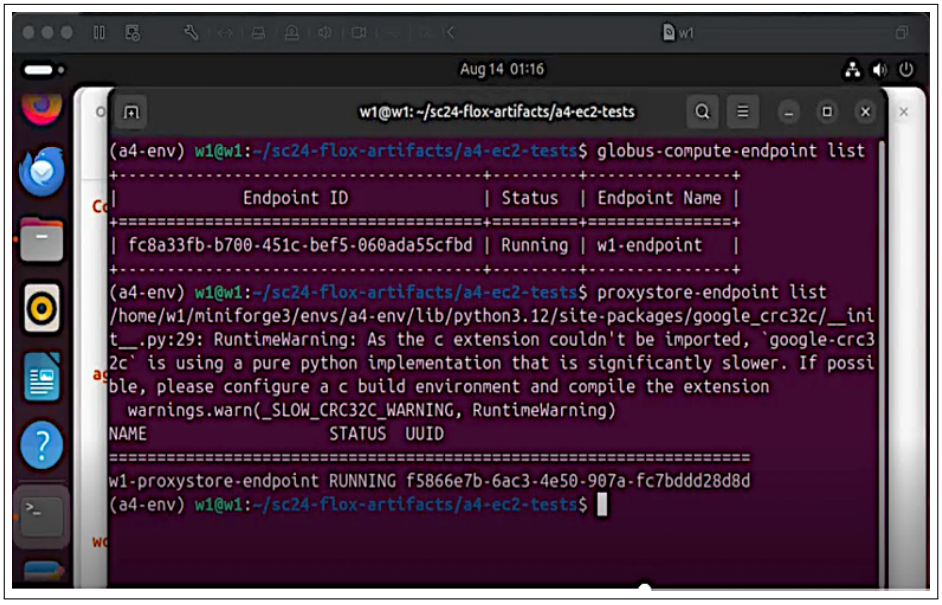
globus user account for administrative simplicity. Network configuration within VMware utilised the default NAT mode, granting VMs shared access to the host's internet connection while facilitating inter-VM communication over the host's virtual network fabric.



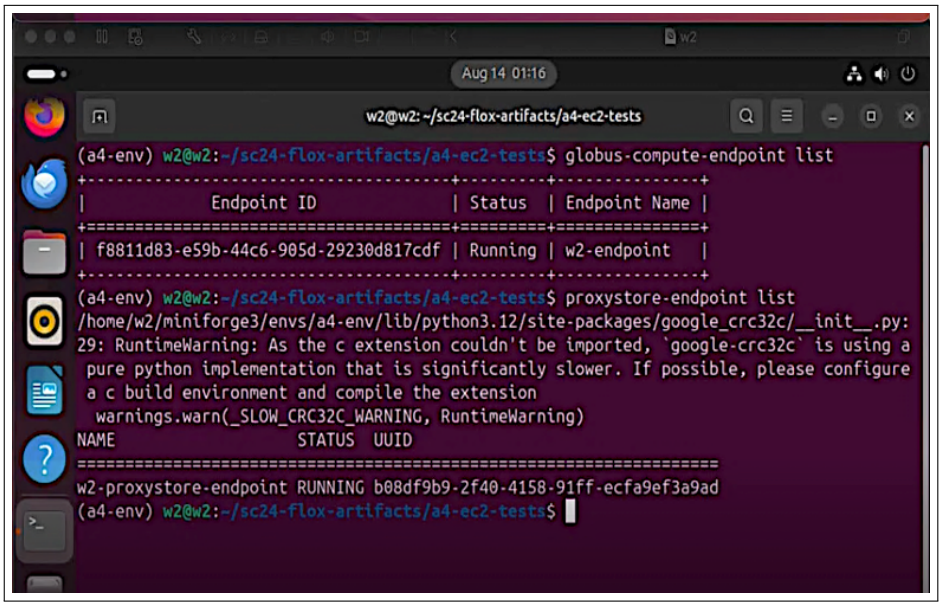
(a) Aggregator Node



(b) Coordinator Node



(c) Worker 01 Node



(d) Worker 02 Node

Figure 4.3: Verification of the VMware simulation environment via concurrent terminal sessions showing active ProxyStore and Globus Compute endpoints on distinct nodes (Coordinator, Aggregator, Workers) and the initiation of `run.py`.

Figure 4.3 provides an example of a visual confirmation of this parallel setup and the readiness of the components before execution. It shows multiple terminal windows connected simultaneously to distinct VMs: Coordinator, Aggregator, Worker 1, and Worker 2. Within each terminal, the successful execution of proxystore-endpoint list and globus-compute-endpoint list confirms that both the ProxyStore and Globus Compute endpoint services are actively running, each identified by unique UUIDs/Endpoint IDs and names. This demonstrates that each VM is correctly configured and independently addressable within the federated system. The Coordinator's terminal, Figure 4.3b, also shows the invocation of the main script (python3 run.py) and the initial log output, signifying the start of the federated workflow.

Within this virtualised setting, the identical orchestration script (run.py) used in the physical deployment were executed. The VM designated as the coordinator initiated remote function calls via Globus Compute to the worker and aggregator VMs, replicating the exact federated learning protocol involving model distribution, local training, delta update transfer via ProxyStore, and FedAvg aggregation as detailed for the Jetson implementation. Inter-VM communication, benefiting from the high bandwidth and low latency of the host machine's internal network pathways, naturally exhibited superior performance compared to the physical testbed's Wi-Fi network. This difference was intentionally permitted, as the simulation was partly intended to establish a performance baseline under near-ideal communication conditions, thereby helping to isolate and quantify the impact of hardware limitations and network characteristics observed in the physical Jetson deployment. Therefore, this parallel simulation provided a counterpart to the physical testbed, validating the software stack's portability and offering a controlled environment for comparative performance analysis.

4.4 Modifications to the Orchestration Script

To establish the empirical basis for the performance evaluation presented in Chapter 6, the primary orchestration script, `run.py`, was modified to capture key performance metrics. These improvements focused primarily on capturing detailed system-level resource usage and performance metrics during the federated learning execution.

Regarding system performance, the total execution time of the experiment was recorded by capturing timestamps (time.time()) immediately before initiating the federated_fit function and immediately after its completion, providing a clear measure of total runtime (completion_time). Also, the script used the psutil library to measure instantaneous system-wide CPU usage (psutil.cpu_percent) just before and after the federated training process. It's important to note that the reported CPU usage (%) metric reflects only the final snapshot at the experiment's conclusion (end_cpu) and not representing average or peak usage during the entire process.

For data transfer analysis, we introduced estimate_data_transferred function, which provides a simplified estimation of the total data volume transferred between nodes for model updates. The calculation assumed that each model parameter, represented as a 32-bit float (4 bytes), was transferred from the Aggregator to each Worker and back again each round. The estimated size was computed based on the total number of parameters in the SmallConvModel, multiplied by the number of Workers (num_workers), the number of rounds (num_rounds), and factored by 2 for the round-trip transfer. This estimation offers a theoretical upper bound, as actual data transfers using ProxyStore and compression mechanisms would typically result in significantly lower volumes. However, the script initially sets num_workers to 1, which needs updating to accurately reflect the actual deployment detailed in flock.yaml.

Finally, after completing the federated training, the script performed a centralised evaluation by calculating the final test accuracy on the standard FashionMNIST test dataset. Also, it evaluated accuracy on the full FashionMNIST training dataset, providing insight into how well the model fits the data it encountered during training. Although detailed round-by-round metrics were captured by the history object returned from federated_fit, the script provided a concise summary at the end, explicitly reporting Completion Time (s), Test Accuracy, Train Accuracy, final snapshot CPU Usage (%), and approximate Data Transferred (MB).

These collected metrics, generated by the changes we made to the run.py script (provided in Appendix G), are foundational to the performance analysis detailed in Chapter 6.

4.5 Reproducibility Assets

In the interest of reproducibility, the following resources accompany this work:

1. Snapshots of the all six configured Virtual Machines used for the simulation environment are available at: `https://tinyurl.com/4a7jvptm`
2. A complete SD card image capturing the configured operating system, installed software stack, and code for all the six Jetson Nano nodes is available at: `https://tinyurl.com/c3vxvsxx`
3. All steps, configuration files and modified source code are included in Appendices B - I.

4.6 Chapter Summary

This chapter detailed the practical implementation and deployment of the federated learning testbed described in Chapter 3. Two parallel environments

were established: a physical testbed using six NVIDIA Jetson Nano devices (Coordinator, Aggregator, four Workers) and, a virtual simulation using six Ubuntu VMs in VMware Fusion. For the Jetson deployment, the chapter outlined the setup process, including flashing JetPack SDK, installing the required software stack and registering each device as a Globus Compute and ProxyStore endpoint. It then described the execution of the FL workflow for training a SmallConvModel on a non-IID distribution of the FashionMNIST dataset. The parallel VMware simulation mirrored this setup and workflow, providing a controlled environment for baseline performance comparison. Modifications to the `run.py` orchestration script were implemented to capture key performance metrics thereby establishing the foundation for the benchmarking methodologies explored in Chapter 5 and the subsequent analysis and discussion of findings in Chapter 6. Finally, the chapter underscored the commitment to reproducibility by detailing the available resources, including VM snapshots, Jetson SD card images, and complete source code with configuration files, referenced in the appendices.

Chapter 5

Benchmarking methodologies

5.1 Introduction: The Role of Benchmarking in Federated Learning

Benchmarking is a critical component in the development and deployment of federated learning (FL) systems, especially as FL transitions from theory to real-world applications. A well-designed benchmarking framework allows researchers and practitioners to evaluate how an FL approach performs under such realistic conditions before deployment. In particular, comprehensive benchmarking is essential for real-world FL deployments because it reveals performance bottlenecks and trade-offs that may be overlooked in idealised simulations [149]. For example, practical FL deployments often involve heterogeneous data distributions, varying device capabilities, intermittent connectivity, and varying client availability, all of which can significantly affect learning outcomes. Overlooking any one of these aspects can lead to misleading conclusions in evaluation. Therefore, establishing robust benchmarks helps ensure that an FL system is truly ready for deployment in environments like smart agriculture, where conditions are far from the controlled settings of a lab.

Benchmarking in FL serves two major purposes. First, it provides quantitative metrics to compare different FL algorithms and configurations on common ground. Given the multitude of FL strategies proposed, such as different optimisation algorithms, compression techniques, privacy mechanisms, a standardised set of metrics and scenarios is needed to assess which approaches perform better under certain conditions. Second, benchmarking helps identify the practical feasibility of FL solutions. A method that achieves high accuracy in theory might be impractical if it requires excessive communication or computation on edge devices. By measuring factors like training time and resource usage, bench-

marks bridge the gap between simulation and deployment [150, 151]. This is particularly important in domains such as smart agriculture, where devices are resource-constrained and networks may be slow or unreliable [152]. In such contexts, a benchmark must tell us not only how accurate a model is, but also at what cost (in time, bandwidth, energy, etc.) that accuracy was achieved.

This chapter discusses the evaluation metrics chosen for our FL testbed in the smart agriculture context, justifying each metric's importance with respect to existing FL literature and the particular constraints of agricultural IoT deployments. We will compare our chosen metrics with alternative evaluation approaches used in related research. By doing so, we demonstrate how our benchmarking framework helps bridge the gap between simulation and deployment, ensuring that FL methods are viable for smart farming applications that have stringent requirements on power, bandwidth, and device heterogeneity.

5.2 Key Evaluation Metrics for FL Performance in Smart Agriculture

In designing the FL testbed for smart agriculture, we identified five key metrics to evaluate performance: **(i) Completion Time, (ii) Training Accuracy, (iii) Test Accuracy, (iv) CPU Utilisation, and (v) Data Transferred**. Each metric addresses a particular dimension of performance, balancing learning efficacy (accuracy metrics) with efficiency and feasibility (time, resource, and communication metrics). Table 5.1 provides an overview of these metrics and their significance. In the following subsections, we justify why each metric was chosen, drawing on federated learning literature and the context of smart agriculture deployments.

Table 5.1: Summary of chosen evaluation metrics and their significance for federated learning in a smart agriculture testbed

Metric	**Explanation**	**Significance in FL**	**Significance in Smart Agriculture Context**
Completion Time (Wall-clock Time)	Measures how long the federated training takes to finish.	This reflects system efficiency and is crucial for real-world deployments. In production FL, counting communication rounds alone is inadequate since different rounds can have variable durations; instead, the actual elapsed time to converge is the key performance indicator.	Lower completion time means models can be updated more frequently (e.g. to adapt to new crop disease data) and decision-making (such as detecting pest outbreaks) can happen in a timely manner.
Training Accuracy (On Local/Training Data)	Indicates how well the model fits the training data distributed across clients.	Monitoring training accuracy helps to assess the convergence behaviour of the FL algorithm and detect issues like overfitting or stagnation early. In FL contexts, training accuracy per round can reveal if the model is learning effectively on the aggregated data and can highlight differences due to data heterogeneity (e.g. some clients' data might be easier or harder to fit).	Tracking training accuracy ensures that the model is learning the patterns present in farmers' local datasets, such as leaf images from their fields, which is important for debugging and understanding model progress.

Continued on next page

Table 5.1 – Continued from previous page

Metric	Explanation	Significance in FL	Significance in Smart Agriculture Context
Test Accuracy (Global/Validation Accuracy)	Evaluates the model's generalisation performance on a held-out dataset or global validation set.	This is typically the primary metric for model quality in FL, as it reflects how well the collaboratively trained model will perform on new, unseen data. High test accuracy is essential for the model to be practically useful.	We include test accuracy as a core metric because, ultimately, the success of FL in smart agriculture is judged by the accuracy of predictions, such as disease detection or yield prediction, on real-world data not seen during training.
CPU Utilisation (Device Computation Load)	Monitors the percentage of CPU resources used during training on the devices (and/or the server).	This reflects the computational burden of the FL workload on edge devices. High CPU utilisation can indicate that the training is computationally intensive, which on resource-limited agricultural IoT devices may lead to slower training, thermal throttling, or excessive battery drain. By tracking CPU utilisation, we gauge the practicality of the FL workload on devices like farm sensors or edge processors.	For smart agriculture, where many devices are battery-powered and have modest CPUs, it is crucial to ensure the FL process does not overwhelm the device as continuous high CPU load could interfere with other sensor tasks or reduce device lifespan.

Continued on next page

Table 5.1 – Continued from previous page

Metric	Explanation	Significance in FL	Significance in Smart Agriculture Context
Data Transferred (Communication Volume)	Measures the amount of data exchanged between clients and the server.	This metric captures the communication cost of FL. In federated training, each round requires transmitting model updates; over many rounds these transfers accumulate and can be significant, especially in low-bandwidth or metered network environments. Minimising data transferred is often crucial in FL because communication is a known bottleneck.	In the context of smart agriculture, devices may connect over cellular, satellite, or low-power wide area networks (which have limited bandwidth and high latency). Measuring data transferred allows us to ensure the FL approach remains feasible under such network constraints and helps evaluate the impact of model size or update frequency on bandwidth usage.

5.2.1 Completion Time (Wall-clock Training Time)

Completion Time refers to the total wall-clock time, the actual time elapsed as measured by a clock, taken to train the model in the federated setting (for a given number of rounds or until a target accuracy is reached). We chose completion time as a metric because in real-world deployments time is of the essence as models need to be trained (or updated) within practical timeframes. Unlike in a simulated or theoretical analysis where one might count the number of communication rounds, completion time measures the actual latency of the entire federated learning process. This distinction is important: as noted by recent FL studies [151]. For instance, one round involving slow devices or poor connectivity might take much longer than another round with fast, well-connected clients. Therefore, wall-clock time to convergence is a more meaningful indicator of performance. Li et al. [151] emphasise that the time elapsed before reaching a target accuracy is the key metric to evaluate when comparing FL mechanisms. In our testbed, completion time directly captures the impact of network delays, device speeds, and any system overhead.

In FL literature, system efficiency optimisations often aim to reduce completion time. Techniques like model compression, client selection, or asynchronous update schemes are frequently evaluated by how much they speed up training

to a certain accuracy. For example, frameworks such as FedScale [149] explicitly measure wall-clock training time to compare FL algorithms under realistic conditions. If an FL method achieves slightly higher accuracy but takes double the time, this may be unacceptable for time-sensitive applications. Smart agriculture deployments can be time-sensitive in scenarios like detecting a fast-spreading crop disease or pest infestation, in this case, the model must be trained and deployed quickly enough for farmers to take action. By benchmarking completion time, we ensure the FL system can deliver updated models within a timeframe that is practical for agricultural decision cycles such as within a day or a growing season, depending on the use case).

Our methodology records the total training time from the start of federated training until completion. This includes all overhead: communication delays, local computation on clients, and any aggregation or idle times. By doing so on both a simulated environment and the real hardware testbed, we capture how factors like wireless communication and weaker processors can significantly lengthen training time compared to ideal simulations. We will later show that the same number of rounds that complete in a few minutes in simulation can take over an hour on real devices (see Chapter 6). This dramatic difference reinforces why measuring actual completion time is essential to bridge the gap between simulation and deployment.

5.2.2 Model Accuracy (Training and Test Accuracy)

Accuracy is a fundamental metric for any machine learning model, and in FL we monitor it on two levels: the accuracy on the training data (aggregated across clients) and the accuracy on an independent test dataset. We include Training Accuracy mainly to observe the overall learning progress and the effectiveness of the training process on the federated data. In a federated scenario, training accuracy can be interpreted carefully since data is partitioned among clients, we will compute a global training accuracy by aggregating all clients' training data or averaging their local accuracies weighted by data size. This gives insight into whether the model is fitting the data it is trained on. A continuously increasing training accuracy indicates the model is learning the patterns in the possibly non-IID training data. If training accuracy plateaus or diverges, it could signal convergence problems or severe heterogeneity issues. Monitoring training accuracy is also useful for diagnosing problems like overfitting: if training accuracy continues to improve while test accuracy stagnates or drops, the model may be overfitting to the local data peculiarities. In FL literature, authors often track training loss or training accuracy per round to study convergence behaviour [153], for example, to analyse how increasing local epochs affects training dynamics. In smart agriculture context, a high training accuracy would mean the model has fit well to the collective data of those farms.

Test Accuracy (or validation accuracy) is the ultimate gauge of the model's generalisation performance. We chose test accuracy as a primary metric because it

reflects how well the federated model will perform on new data, which is the end goal of model training. Nearly all federated learning studies report some form of test set performance, as this is needed to compare algorithms' effectiveness. For instance, FedAvg and other baseline algorithms are typically evaluated by their final test accuracy on benchmark datasets [149].

Our testbed uses a held-out dataset, FashionMNIST images not included in any client's training, to evaluate the global model upon completion. Although FashionMNIST involves classifying clothing items, the underlying evaluation methodology is applicable and critical in smart agriculture contexts. Specifically, achieving high accuracy in this testbed indicates that the federated learning approach can reliably generalise across diverse conditions. This mirrors scenarios in agriculture where models must accurately distinguish between healthy and diseased crops across varied environmental and farm management conditions. Conversely, insufficient test accuracy on FashionMNIST suggests potential limitations in the federated learning method, indicating that it may similarly struggle to achieve practical accuracy in agricultural applications, reducing its adoption by end-users despite other technical merits.

It is worth noting why we track both training and test accuracy rather than just one. In a perfectly balanced and IID data scenario, training accuracy and test accuracy would usually move in tandem with test accuracy slightly lower. However, in FL with non-IID data, there can be a gap between them, and observing both can provide insights. For example, in one experiment on our testbed, we observed the training accuracy continuing to improve while the test accuracy plateaued; this pointed to the model overfitting to the union of client data and not generalising beyond it. In another case, training accuracy remained lower than test accuracy which can happen if clients have noisy or very hard training data but the global model generalises well to the simpler test data. Tracking these metrics allowed us to ensure that the model not only converges on the training data but also generalises. Prior work has also considered how to weight accuracy across clients (to ensure fairness) [153], but in our context of agriculture (where clients are fields or farms), we primarily focus on overall accuracy, assuming each farm's data is equally important to overall model quality.

5.2.3 CPU Utilisation (Computational Resource Usage)

Federated learning shifts the computational burden of model training from a central server to numerous client devices. In smart agriculture, these client devices could be edge computers at farms, such as IoT gateways, drones, or small embedded systems, such as NVIDIA Jetson Nano devices or Raspberry Pis, that collect and process data. We include CPU Utilisation as a metric to quantify the computational load imposed by FL training on these devices. High CPU utilisation during training indicates that the device is heavily taxed by the FL workload. This is a concern for three reasons. Firstly, **Resource Con-**

tention: On an agricultural sensor hub, FL training might be a background task, while the device also handles real-time sensor data logging or control commands. If the FL task consumes too much CPU, it could interfere with these critical functions. By measuring CPU utilisation, we ensure that the training task leaves enough headroom for the device's primary functions. Secondly, **Energy Consumption:** CPU usage is directly linked to energy consumption on battery-powered or solar-powered devices. Smart farming sensors are often energy-restricted and rely on batteries, making it vital to keep computationally intensive tasks efficient. A near-100% CPU utilisation over extended periods can quickly drain batteries or necessitate frequent recharging, which is impractical in remote fields. Although we did not measure battery drain directly, CPU utilisation serves as a proxy because if an algorithm yields significantly higher CPU usage, one can infer higher energy use. Finally, **Thermal and Performance Constraints:** Devices operating outdoors in potentially warm environments could overheat if stressed. Embedded CPUs may throttle when at high utilisation for long durations, which would effectively slow down the training. Monitoring CPU utilisation helps detect such scenarios and guides us to possibly simplify the model or reduce the frequency of training to avoid straining devices.

In the literature on federated learning and edge computing, computational load is recognised as a limiting factor for deploying deep learning on the edge [149]. Strategies like using smaller neural network models or quantised models are often employed to reduce the computation requirement on clients. By including CPU utilisation as an evaluation metric, we align with these concerns: our benchmark will highlight if an FL approach demands too much computation for a typical agricultural device. For instance, if training a complex CNN causes an edge device's CPU to run at 100% for an hour, we might conclude this approach is not suitable for real deployment, and consider alternatives like a smaller model or offloading some computation.

By quantitatively tracking CPU utilisation, our benchmarking framework ensures that computational feasibility is evaluated alongside accuracy. This metric is particularly significant for smart agriculture FL, because farms may not have the luxury of powerful GPUs or unlimited energy and instead, they have cheap, low-power devices that must perform reliably under constrained conditions.

5.2.4 Data Transferred (Communication Overhead)

Communication efficiency is a well-known challenge in federated learning. In each round of federated training, model parameters or gradient updates must be sent from the server to clients and/or from clients back to the server. Especially in deep learning models, these updates can be large (millions of parameters), and when multiplied by many rounds and many clients, the total data transferred can become a bottleneck. We chose to explicitly measure the total volume of data transferred during training (typically measured in megabytes, summing

across all communication in all rounds). This metric directly addresses the network overhead of FL.

The importance of communication metrics is widely acknowledged. Caldas et al. in the LEAF benchmark introduced metrics accounting for the number of bytes uploaded and downloaded by devices as a way to quantify the resources required in federated scenarios [153]. By doing so, they acknowledged that rigorous evaluation metrics in FL should include communication cost, since transmitting updates can incur significant delays and energy use, especially on wireless networks. In our testbed's context (smart agriculture), clients may be connected via rural broadband, cellular networks, or even satellite or low-power networks like LoRaWAN. These connections often have limited bandwidth and high latency. In IoT settings relevant to agriculture, bandwidth constraints and intermittent connectivity are paramount concerns [152]. Transmitting even a few megabytes can be slow or expensive. By measuring data transferred, we ensure that we quantify how much communication our FL approach requires, and we can compare it against what networks in the target deployment can handle.

For example, if our FL training of a model requires 100 MB of traffic per device per day, but a farm's IoT gateway only has a 3G cellular link with a data cap, this approach might be impractical. Conversely, a method that only transfers 5 MB might be acceptable. Our metric captures both uplink and downlink data: i.e. we include model weights sent to clients and model updates sent back to the server. Some studies report these separately (upload vs download) since the cost can differ, but we sum them for simplicity. We can also consider the effect of compression techniques in this metric: if we use compressed model updates, the data transferred metric should reflect the reduced size, thus directly showing the benefit of such techniques on communication load.

In sum, the Data Transferred metric in our benchmark ensures that the FL solution is evaluated for network efficiency. It is a first-class metric because, in remote agricultural deployments, communication can often be the slowest and most fragile part of the system as they may have patchy Wi-Fi or cellular coverage in a farm. An FL system that is communication-heavy might perform well in a research lab, where data transfer is via LAN, but could fail or stall in real deployment. Therefore, by measuring and striving to minimise data transfer, we make our FL approach more viable for the intended smart agriculture environment.

5.3 Comparison with Alternative Metrics and Evaluation Approaches

The chosen metrics above are not the only ones available for evaluating federated learning. In related research, various other metrics and approaches have been used depending on the focus of the study. In this section, we compare our metrics to some alternative evaluation approaches and explain why we prioritised our chosen metrics over others. Specifically, we discuss energy consumption as an alternative to CPU utilisation, Precision/Recall/F1 score as alternatives or complements to accuracy, and the use of communication rounds or rounds-to-accuracy as an evaluation method. We also describe why we did not include metrics like memory usage or specific fairness metrics in this section, focusing instead on the core efficiency and accuracy aspects.

5.3.1 Energy Consumption vs CPU Utilisation

One could argue that measuring the actual energy consumed by devices during training would be a more direct metric for efficiency than CPU utilisation. Energy consumption is critical in IoT settings; some FL studies have targeted reducing energy usage and even measured it. For example, Woisetschläger et al. [150] in FLEdge systematically study the energy efficiency of training on different hardware, reporting that certain embedded GPUs can be 3× more energy-efficient than server GPUs for FL workloads. However, measuring energy consumption accurately requires specialised instrumentation or hardware support. In FLEdge, they achieved this by monitoring power draw via Power-over-Ethernet supplies and network switches. This is a setup that may not be available in all testbeds. Moreover, comparing energy results across different setups can be difficult; authors have noted that current evaluations often rely on estimated energy from computational operations or FLOPs, which are not comparable across devices. In our testbed, we decided not to include direct energy consumption measurements and instead use CPU utilisation as a proxy for a few reasons. First, our focus was on relative comparisons of different FL configurations on the same hardware; for this purpose, CPU utilisation serves well, for instance, if one approach uses 30% CPU and another uses 80% on identical devices, the latter clearly consumes more energy, even if we don't have the exact Joule values. Second, integrating power measurement for every device would have added complexity to the experimental setup, and as a university testbed, we faced some practical constraints in hardware instrumentation. By contrast, CPU utilisation can be obtained via software on each device easily by using system monitors or logs. We acknowledge that energy consumption is an important metric especially for battery-operated sensors in agriculture, and future extensions of our work could incorporate it. In fact, one can derive rough energy estimates if the power draw of devices at certain CPU utilisations is known. But given that our chosen CPU metric already reflects the cause of high energy use which is excessive computation, we deemed it sufficient for our

comparative analysis. Also, focusing on CPU utilisation aligns with the optimisation objectives found in FL literature, where methods like model offloading or quantisation aim to reduce computation, hence CPU, to indirectly save energy.

5.3.2 Precision, Recall, and F1 Score vs Accuracy

In classification tasks, especially those with class imbalance, metrics like precision, recall, and F1-score can be more informative than raw accuracy. F1-score which is the harmonic mean of precision and recall is often used in scenarios where one wants to balance false positives and false negatives. The decision to focus on accuracy and not to report F1-score was guided by both our specific problem characteristics and common practice in FL evaluations. First, the dataset we used for benchmarking (FashionMNIST) was relatively balanced among classes, and we were primarily interested in overall accuracy of identification. In such cases, accuracy and F1-score tend to correlate closely. In one of the studies we reviewed on FL for plant disease, the authors reported accuracy, precision, recall, and F1 for various models [154]; the values were all high and quite consistent (with F1 around the same percentage as accuracy for each model, given balanced data). Their analysis concluded that models with the highest accuracy also had the highest F1, so little additional insight was gained from reporting both. Second, from the FL system perspective, our thesis is more concerned with the system performance differences (time, data, etc.) and ensuring the model's general utility (accuracy). If this were a machine learning thesis, we would certainly go deeper into precision/recall trade-offs. However, since our emphasis is on benchmarking the FL process, we used accuracy as a representative metric of model quality as it sufficed to capture model performance for the balanced classification task at hand.

5.3.3 Counting Communication Rounds vs Wall-clock Time

Another alternative approach used in many FL research papers is to report the number of communication rounds to reach a certain accuracy (or to converge). This is a natural metric in simulation studies because each round is assumed to take a uniform amount of time, so fewer rounds implies faster training. We explicitly chose Completion Time over Number of Rounds for our benchmark, due to the realities of heterogeneous deployments. As discussed, in real FL systems, different rounds can have different durations [151]. For example, if a straggler client with poor connectivity participates in a round, it could delay that round significantly compared to others. Moreover, the notion of a "round" becomes blurred if we consider asynchronous FL or client dropouts, where not all clients go through the same number of rounds. Our testbed findings strongly support this view. In a controlled simulation (with all clients identical and no network latency), one might see, say, 5 rounds needed for 85% accuracy. But on our real testbed, running those 5 rounds took much longer, and some rounds were slower than others due to variations in client processing times. If we had only counted rounds, we might wrongly assume the method is fast,

whereas the actual clock time tells a different story. Therefore, we did not adopt "communication rounds" as a primary reported metric. We do, however, keep track of it internally or report it secondarily since it's essentially the loop index of the training process. It's useful when normalising some other metrics, for instance, data transferred per round. By doing so, our benchmarking aligns with production considerations. For example, if one wanted to schedule model retraining on a farm's edge server nightly, we care whether the training finishes in 6 hours or 12 hours, not just that it took 50 rounds.

5.3.4 Other Potential Metrics

A few other metrics were considered but not included in this chapter's scope. One is memory usage, RAM consumption, on the client devices during training. This can be important because edge devices have limited memory and training a large model might not even be feasible if it exhausts RAM. We did encounter memory limitations on some devices, for example, the Jetson Nano has only 4 GB of RAM, which can be a bottleneck with larger models. However, we treated this issue as a feasibility constraint (i.e. if a model doesn't fit in memory, we simply cannot run it) rather than a metric to benchmark, since once a model does fit in memory, its memory footprint doesn't fluctuate much during training. Another metric is privacy or security metrics such as measuring the success of an attacker or the degree of differential privacy achieved. These are highly relevant in federated learning at large [149], but in our smart agriculture testbed, we did not implement any specific privacy-preserving technique that would warrant such evaluation. We assume the data is sensitive, a farm's data stays on the farm, which is why FL is used, but analysing privacy leakage or defense is beyond our current scope. Lastly, fairness metrics (how even performance is across clients) could be considered, for example, the lowest accuracy achieved on any client's local data. We did not explicitly measure this, but we note that in agriculture, certain farms might have more data or more difficult tasks as perhaps one farm has a rare disease that the model doesn't learn well. While important socially, we treated that as part of the data heterogeneity challenge rather than including a separate fairness metric.

To summarise, our chosen metrics (time, accuracies, CPU, data) cover the pillars of performance that we deemed most critical to evaluate for a federated learning system's viability in a real-world smart farming scenario. Alternative metrics like energy and F1-score are valuable, but either overlap with the information provided by our chosen metrics or pose measurement difficulties that outweigh their benefit in our context. We followed the guidance from recent works to prioritise wall-clock time and accuracy [151] , and we ensured our metrics align with the practical constraints of bandwidth, power, and heterogeneity that define smart agriculture IoT environments [152].

5.4 Summary

In this chapter, we presented a comprehensive evaluation of a federated learning testbed tailored for smart agriculture, focusing on a set of core benchmarking metrics. We introduced five key metrics, completion time, training accuracy, test accuracy, CPU utilisation, and data transferred, and justified each in the context of FL literature and agricultural deployments. We compared these metrics to alternatives and clarified our choices. We conclude by reflecting on how this benchmarking framework bridges the gap between simulation and real-world deployment, and why it is especially suited for the smart agriculture domain.

Bridging the Gap: A recurring theme has been that solely relying on simulations or a single metric can be misleading for real-world performance. Our benchmarking framework addresses this by integrating system-level metrics (time, CPU, data) with model-centric metrics (accuracy) and by validating results on actual hardware. In effect, it creates a bridge between algorithmic research and practical implementation. For example, an FL algorithm that looks excellent in simulation (fast convergence in terms of rounds, high accuracy) might turn out to be impractical when tested on field devices due to unacceptably long wall-clock time or huge communication demands. By catching these issues in our testbed stage, we ensure that only those FL strategies that meet both accuracy and efficiency criteria are considered viable. This approach echoes the goals of recent FL benchmark efforts like FedScale [149] and FLEdge [150], which emphasise evaluating FL under realistic conditions to uncover hidden costs. In our case, by actually deploying on Jetson Nanos over Wi-Fi, we revealed the true training times and resource usage, which a pure simulation could not show. Thus, when we move towards deploying FL in a smart farm, we have confidence that our measured X minutes of training time and X MB of data transfer (for a certain configuration) are realistic figures that the deployment must handle. If those were unacceptable, we would know to adjust the approach, for example, by using a smaller model or fewer rounds, before deployment rather than discovering the issue in the field.

Another aspect of bridging the gap is informed decision-making for deployment. Our benchmark provides a quantitative basis for decisions like: Should we upgrade hardware? (If CPU utilisation is consistently 100%, maybe a device with a better CPU or a GPU is needed.) Do we need network upgrades or alternate connectivity? (If data transfer is high, perhaps a farm should invest in better internet or use an edge aggregator to reduce cloud communication.) Or even: Is federated learning providing enough benefit to justify its overhead compared to centralising the data? For instance, if test accuracy is only marginally higher in FL than a model trained on smaller local data, but FL incurs a lot of cost, one might question the approach. The benchmark numbers help quantify that gain vs. cost.

Suitability for Smart Agriculture: The metrics and framework we used are particularly pertinent to smart agriculture for several reasons. Firstly, **power constraints:** Farms often deploy sensors or devices running on solar or battery, which cannot support heavy computation or communication continuously. By measuring CPU and inferring energy, we directly address this constraint ensuring the FL process could run, for example, overnight when solar is charging or intermittently without draining batteries. Agriculture applications might accept a slower model update if it means lower power usage, and our metrics allow analysing that trade-off. Secondly, **bandwidth and connectivity:** Rural areas may have spotty connectivity. Our focus on data transferred and completion time under realistic network conditions means the resulting FL system is designed with bandwidth limitations in mind. For example, if a farm only has a narrowband connection, our benchmark might suggest using very infrequent communication, in other words, more local training, or smaller models. We can also identify if certain clients, such as a sensor in a distant field with weak signal, will be a bottleneck; those might be handled differently like only participate occasionally. The smart agriculture context often involves hardware heterogeneity as well, a farm might have a mix of devices: some powerful machines in a control center, some low-power sensors at the edge. Our testbed used uniform devices for simplicity, but the framework could easily test mixed setups by including one very slow device in the mix and observing impact on metrics.

By benchmarking thoroughly, we also contribute a sort of reference performance for anyone looking to deploy FL in similar contexts. For example, a cooperative of vineyards might see from our work that training a disease detection model across 5 vineyards took about X hours and Y MB on devices similar to Raspberry Pi, achieving Z% accuracy. They can use those numbers to plan their own deployment or to decide if they need better equipment. In academic terms, this chapter's results add to the small but growing body of literature on FL in agriculture. Our specific contribution is providing empirical evidence of performance on real edge hardware for this domain.

To sum up, our benchmarking framework, with its carefully chosen metrics and mixed evaluation modes, serves as a bridge from simulation to deployment. It takes federated learning algorithms out of the theoretical realm and tests them against the harsh realities of limited power, limited bandwidth, and hardware variability that characterise smart agriculture settings. This ensures that the solutions we propose are not just accurate in principle, but also deployable in practice. The framework is particularly well-suited to smart agriculture because it accounts for the exact constraints that farmers and agri-tech deployers care about: Will the model be accurate enough? How long will it take to train? Can my devices handle it without running out of battery or data? By answering these questions with data and analysis, Chapter 5 lays the groundwork for deploying federated learning to allow privacy-preserving, collaborative AI in agriculture, bringing the theoretical benefits of FL into real-world use on the farm.

Chapter 6

Results, Analysis, and Discussion

6.1 Overview of Benchmarking Experiments

This chapter presents the results of our benchmarking experiments and provides an in-depth analysis of the findings. The goal of these experiments was to compare a simulated federated learning (FL) environment against a real FL testbed deployment for a smart agriculture use-case. We evaluate both settings across several key metrics: Completion Time, Model Accuracy (on training and test sets), CPU Utilisation, and Data Transferred per round. By examining these metrics under varying conditions (number of FL workers, fraction of dataset used, and number of training rounds), we can assess the performance trade-offs and realism of the testbed relative to simulation. The discussion integrates these results with existing literature on FL performance, edge computing constraints, and smart agriculture requirements.

In the experiments, we considered 1 to 4 FL workers (clients) participating in training, using either 50% or 100% of the available dataset, and running for 1, 3, 5, or 10 global rounds of federated training. The same federated learning algorithm (Federated Averaging, FedAvg) and model architecture were used in both the simulation and the real testbed to ensure comparability. The simulation environment executed the FL process in software (on a single machine, mimicking multiple clients), whereas the testbed consisted of physical edge devices (NVIDIA Jetson Nano boards) connected via a wireless network. Each client device in the testbed trains on its local data and communicates model updates to the aggregator, similar to the simulation's logical clients. The full results of these experiments are recorded in Appendix A, and key trends are highlighted here. In sum, the simulation provides an optimistic baseline for performance, while the real testbed reflects actual deployment constraints, such as limited computing power and network overhead. This gap between simulation and real-

ity is important to quantify: federated learning simulations often fail to capture the true runtime and system overheads of real deployments. By analysing both environments side by side, we can better understand how an FL system would perform in a smart agriculture scenario and discuss practical implications for deployment.

6.2 Completion Time: Simulation vs. Real Testbed Performance

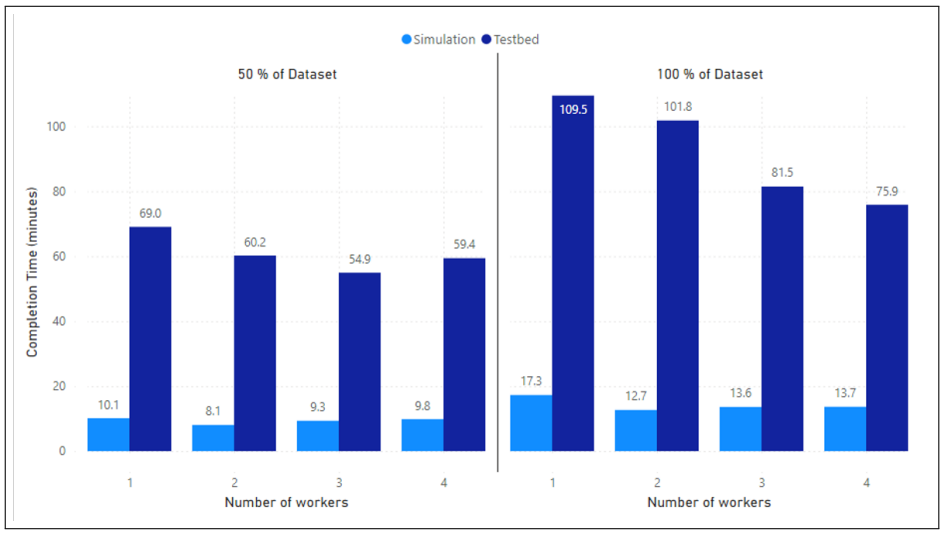


Figure 6.1: Completion time by number of workers (1,2,3,4) for 10 rounds

One of the most striking differences between the simulation and the physical testbed is the completion time required to finish a given number of FL training rounds. Across all configurations, the real FL testbed was significantly higher than the simulation environment. For example, with a single worker utilising the full dataset for 10 rounds of training, the simulation completed in about 17.3 minutes, whereas the testbed required 109.5 minutes to finish the same workload (Figure. 6.1). This is roughly 6 times longer. A similar factor of 5-8× slower performance was consistently observed for the testbed in other configurations. These results confirm that simulation experiments can dramatically underestimate the time needed for FL training in real-world settings. The discrepancy is attributable to several real-world factors present in the testbed but abstracted or idealised in simulation: the limited processing power of edge devices, the latency and bandwidth constraints of wireless networking, and the overhead of coordinating physically separate clients.

Hardware and computation: In the simulation, all client computations were effectively performed on a single high-performance machine, which has abundant CPU/GPU resources and fast memory. In contrast, the testbed's clients were Jetson Nano devices which are low-power embedded systems with significantly less processing capability. Training a deep learning model on such hardware is inherently slower. The Jetson Nano's GPU, while capable of accelerating inference, cannot match the throughput of desktop or server GPUs for training tasks; the CPU and memory limitations further bottleneck training speed. Our findings align with this expectation: the simulator (running on a standard PC with ample resources) completed local training tasks much faster than the Jetson devices could.

Networking and communication: In addition to slower computation, the testbed introduces network communication delays for exchanging model updates between the server and clients. In the simulation, communication was simulated via in-memory or localhost data transfer, effectively with negligible latency and very high bandwidth. The real testbed, however, relied on Wi-Fi communication. Each training round involves broadcasting the global model to all workers and then sending back the updated model weights from each worker to the aggregator. Even though the absolute amount of data transferred each round is modest (0.34 MB per client as observed in Figure 6.7), the physical act of transmitting this data over Wi-Fi incurs unavoidable "normal network delay". This delay comprises both latency (the time for signals to travel) and transfer time (dependent on actual bandwidth). While we didn't isolate communication time in a separate plot, its contribution is implicitly demonstrated by the substantial overall time increase in the testbed compared to the simulation baseline where communication cost was effectively zero. Potential network contention, protocol overheads, and minor fluctuations in connectivity, even in our controlled lab setting, add to this real-world communication cost which is absent in the simulation.

It is important to note that the completion times in the testbed include not only active computation but also idle waiting times. Federated learning in our setup is synchronous: the server must wait for all client updates in a round before aggregating and moving to the next round. Therefore, the slowest client in each round, a straggler, can determine the round duration. With homogeneous hardware (all Jetson Nanos), one might expect similar speed per client, but minor differences, such as thermal throttling or background processes, which could introduce straggling. In our results, we saw relatively consistent training times across clients, but as we scale to more workers, synchronisation overhead grows. In Section 6.4 we analyse how adding more workers affects training time. Here we note that in real networks, synchronisation can be a major bottleneck. Other researchers have highlighted this issue: when client computational capabilities vary or network conditions differ, FedAvg can underperform and take much longer to converge [155]. Our testbed avoided extreme heterogeneity as our devices and data were similar), but the principle remains that a real system

must manage synchronisation delays.

To put our completion time results into perspective, consider a concrete scenario: training for 5 rounds on the full dataset with 4 workers. In simulation, this finished in 7.5 minutes, whereas on the testbed it took 39.8 minutes. If the FL process were to be deployed on a farm with similar hardware, an update that might be expected from simulation to complete during a short window would actually take well over half an hour in practice. This could impact how frequently model updates can be performed in the field. If near real-time model updates were required, for example for updating a pest detection model hourly, the simulation might misleadingly suggest feasibility, whereas the testbed shows it might not be achievable without hardware upgrades or algorithmic optimisations. Therefore, the real-world time cost of FL on edge devices is high, underscoring the need for techniques to improve efficiency if such systems are to be used in daily agricultural operations. Techniques such as model compression, fewer rounds, or asynchronous training could be considered to mitigate these delays [76], but each comes with trade-offs in accuracy or complexity as discussed later.

Despite the slower speeds, an encouraging finding is that adding more workers in the testbed did reduce the overall training time up to a point, this is due to parallelisation which is discussed next in Section 6.3. The key takeaway here is that the absolute training times in a real FL deployment can be an order of magnitude greater than in a controlled simulation. Any proposed FL system for smart agriculture must account for this gap during planning and not rely solely on simulated benchmarks. Our testbed provided a reality check for performance, revealing bottlenecks that would be invisible in simulation.

6.3 Impact of Number of Workers on Training Time and Speedup

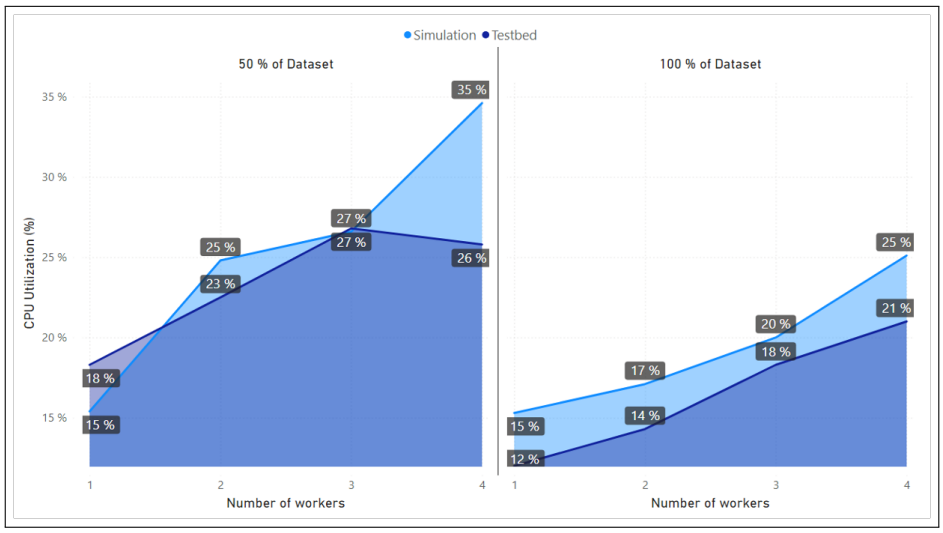


Figure 6.2: CPU utilisation (%) for 10 rounds

The experiments were designed to evaluate how scaling the number of participating devices (workers) influences training time. Intuitively, distributing the training task among more workers can provide parallel speedups, each device handles a portion of the data, but it also incurs greater communication and coordination overhead. Our results reflect this trade-off between parallelism and overhead. We observed a non-linear relationship between the number of workers and the total completion time, with an optimal number of workers that minimises training time for a given data size, beyond which additional workers yield diminishing returns.

In the simulation environment, using 2 workers consistently achieved the fastest completion times for the configurations tested. For instance, with 10 rounds on the 50% dataset, 2 workers finished in 8.06 minutes, compared to 10.13 minutes for 1 worker and 9.84 minutes for 4 workers. Similarly, on the 100% dataset which is larger, hence would require more to train per round, 2 workers (12.67 min) outperformed 1 worker (17.28 min) and was slightly faster than 3 or 4 workers (13.6 min each) for 10 rounds. This suggests that splitting the workload between two clients provided near-linear speedup, almost halving the time of 1 worker, without yet incurring heavy overhead. However, going to 4 workers did not further improve the simulation time. In fact, the 4-worker case was marginally slower than the 2-worker case in simulation. This is likely because with 4 workers, although each round's local training computation is

even smaller per device, the overhead of managing 4 updates and perhaps the increased aggregate CPU usage on the simulation host offset the benefits. In simulation, all client processes contend for the same host resources, so beyond a point, adding more parallel clients leads to resource contention (CPU, memory or I/O) on the single machine. We saw the simulator's CPU utilisation rise from 15% with 1 worker to 35% with 4 workers (for 10 rounds), as observed in Figure. 6.2, indicating the host was working harder to coordinate more clients.

In the physical testbed, the pattern was similar but with some differences. Because each additional worker in the testbed is an actual device, adding workers can more directly increase parallelism as each Jetson Nano trains its portion concurrently. At the same time, more devices mean more network messages and synchronisation. For the 50% dataset, we found that 3 workers achieved the shortest training time (54.9 min for 10 rounds), slightly faster than 2 workers (60.2 min) or 4 workers (59.4 min). In this case, going from 3 to 4 workers actually increased the time by about 8%, indicating the overhead outweighed the small reduction in per-device workload. With only 50% of the data distributed among 4 clients, each client had relatively little data to train on, so the system quickly became communication-bound which means spending proportionally more time exchanging weights than doing local computation. On the other hand, for the 100% dataset (larger training data), the 4-worker testbed run was the fastest (75.9 min for 10 rounds), beating 3 workers (81.5 min), 2 workers (101.8 min), and of course 1 worker (109.5 min). Here, because each round involved more data, the benefit of splitting that data among more workers was more pronounced; the communication overhead, while present, did not negate the speedup from parallel training on four devices. We essentially see that when the local workload is larger, adding workers helps (up to 4 in our test), but when the local workload is small, too many workers can hurt performance due to overhead.

These observations highlight the trade-off between computation and communication in federated learning. With few workers, computation dominates the round time as each device has a lot to train, so parallelising computation yields big gains. With many workers, each does little computation, and therefore the fixed costs of synchronisation and communication start to dominate the total time. Our results are in line with general FL system design insights: there is often a sweet spot in the number of clients to use simultaneously. If we had scaled to even more workers (beyond 4), we would expect eventually that the overhead would far outweigh parallel benefits, especially given the limited network capacity since all devices are sharing a wireless router. Also, FL literature often points out that involving too many clients per round can lead to diminishing returns or even slowdowns unless communication is very efficient [76]. Techniques like client selection, using only a subset of clients each round, and communication compression are sometimes employed to address this, but in our controlled experiment we systematically varied client count to see these effects directly.

From a smart agriculture deployment perspective, this means that there is an optimal scale for FL training given a fixed infrastructure. For example, if a farm has 10 IoT devices with data, it might not be best to use all 10 in every round; using a smaller subset in each round and rotating through them could actually complete training faster if network bandwidth is a bottleneck. Our testbed results with 3 vs 4 workers hint at this: using all available devices is not always the fastest solution. This is a counterintuitive but important insight for practitioners. To add on to this, the results suggest that federating across a moderate number of edge devices can indeed reduce the wall-clock training time compared to one device doing all the work, as long as the coordination overhead is managed.

It is worth noting that in the simulation, the fastest time was actually with 2 workers, not 4, whereas in the testbed, the fastest was with 4 (for full data). This difference underscores that simulation may mis-estimate the optimal number of workers because of how resources are modeled. In simulation, 4 workers on one machine contend with each other, whereas in reality 4 physical devices can truly work in parallel. Consequently, the testbed was able to leverage the full parallelism of 4 separate CPUs/GPUs, albeit slow ones, which the single-machine simulation could not perfectly emulate. This is another facet of the simulation-vs-reality gap: a simulator might incorrectly suggest that increasing client count yields no benefit (or a negative benefit) if it's limited by host resources, whereas a real deployment might still see improvements up to some higher threshold of clients. Our real-world results therefore provide a more reliable guide for scaling FL in practice.

To summarise this section, adding workers (clients) in FL can shorten training time by parallelising local model training, but it also introduces more communication. Both our simulation and testbed indicate an optimal range of clients for efficiency. Practitioners should be cautious in assuming linear speedups with more devices as in reality there is a balance, and beyond a certain number of clients, the overhead can slow down the overall process. For the given hardware and data, 2–4 clients provided the best performance, and more might have caused slowdowns. This insight can inform how to design FL systems in agriculture, for example, how many edge sensors to concurrently involve in a training round. We next examine how these different configurations affected the model's accuracy and what trade-offs exist between training time and model performance.

6.4 Model Accuracy and Convergence Behaviour

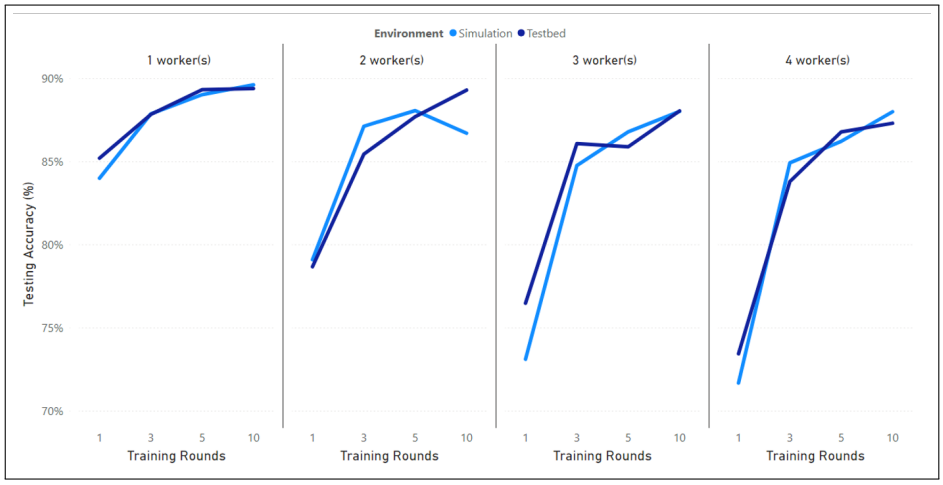


Figure 6.3: Testbed testing accuracy (%) vs training rounds (1,3,5,10)

While training time is critical, it must be balanced against the model accuracy achieved. In federated learning, many factors like the number of rounds, the data distribution across clients, and local training epochs, influence the model's accuracy on a validation or test set. Our experiments tracked the test accuracy after each scenario, as well as the training accuracy on the combined training data. Here we discuss how accuracy varied between the simulation and testbed, how it was impacted by the number of rounds and workers, and the general trade-offs observed between training time and accuracy.

Simulation vs. testbed accuracy: Despite the large differences in completion time, it is noteworthy that the final model accuracies achieved in the simulation and testbed were very similar for equivalent scenarios. The simulation and testbed runs essentially followed the same learning trajectory as they used the same data and hyperparameters, and any differences in accuracy between them were small with a difference between 0.1% and 1%. For example, using 1 worker on the full dataset for 10 rounds yielded a test accuracy of 89.6% in simulation vs. 89.4% on the testbed, virtually identical. Minor discrepancies can be attributed to nondeterminism in training, such as differences in floating-point computations on PC vs ARM or timing of random seed initialisation, rather than any systemic issue. This parity in accuracy is an important validation of our testbed: it shows that, given enough rounds, the real system can reach the same accuracy as the simulated ideal case. In other words, the testbed's slower training did not harm the final model quality; it just took longer to get there. This also indicates that our simulation correctly models the learning algorithm, FedAvg, without any simplifications that would affect

convergence results. Therefore, for model accuracy metrics, we can discuss the results generally without needing to separate simulation vs testbed.

Effect of number of rounds: As expected, and as illustrated in Figure 6.3, increasing the number of federated training rounds improved the model accuracy, but with diminishing returns. Each additional round allows the model to incorporate more updates from the distributed data, consequently usually reducing the training loss and improving validation metrics, up to a point of convergence. For instance, in the 4-worker, 100% dataset condition on the testbed, the test accuracy climbed from 71.7% after 1 round, to 84.9% after 3 rounds, 86.8% after 5 rounds, and 87.3% after 10 rounds. The biggest jump here was from 1 to 3 rounds with 13% increase, while the gain from 5 to 10 rounds was only about +0.5%. A similar pattern was seen in all setups: the first few rounds yield substantial accuracy gains, but by round 10 the model is nearing convergence with marginal improvements. One can infer that beyond 10 rounds, the accuracy might plateau around 88–90% for this task. This is consistent with typical learning curves in FL and centralised training, early epochs/rounds contribute the most learning, and later ones fine-tune smaller details.

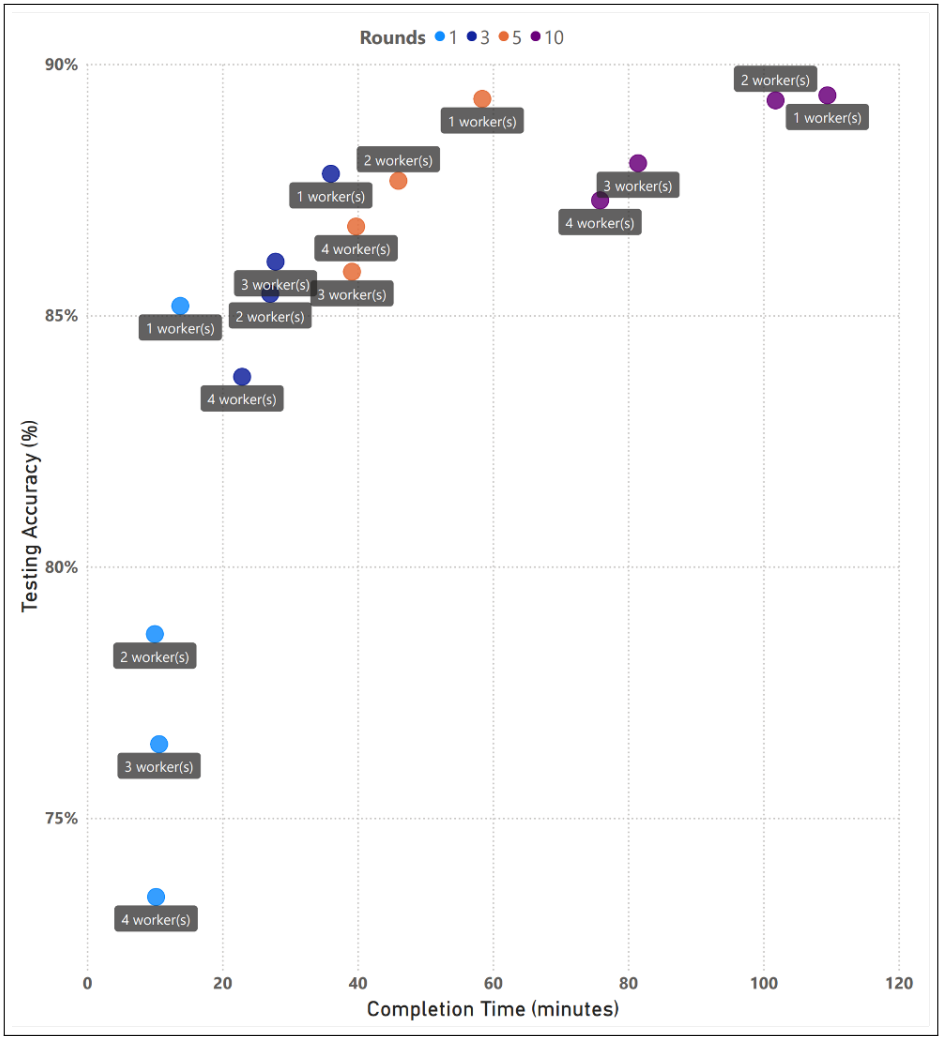


Figure 6.4: Testbed trade-off between completion time and testing accuracy (%)

The trade-off between training time and accuracy becomes clear here. Figure 6.4 shows more rounds improve accuracy but at the cost of more communication and time. For example, moving from 5 to 10 rounds on the testbed roughly doubled the training time (e.g. from 58 min to 109 min for 1 worker on full data) but only increased test accuracy from 89.3% to 89.4%, which is essentially a negligible gain in that case. In contrast, the jump from 1 to 3 rounds increased accuracy by several percentage points (e.g. 85.2% to 87.8% in that same 1-worker full-data case) for a reasonable increase in time (13.8 min to 36.0 min). Therefore, the optimal number of rounds depends on the acceptable accuracy-time trade-off in the application. In a real deployment, one might decide that

5 rounds is sufficient if 89% accuracy meets the agricultural task requirements, thereby saving considerable time and energy compared to doing 10 rounds for 0.5–1% extra accuracy. This mirrors the notion of "early stopping" in machine learning to avoid over-exerting resources for minimal gains.

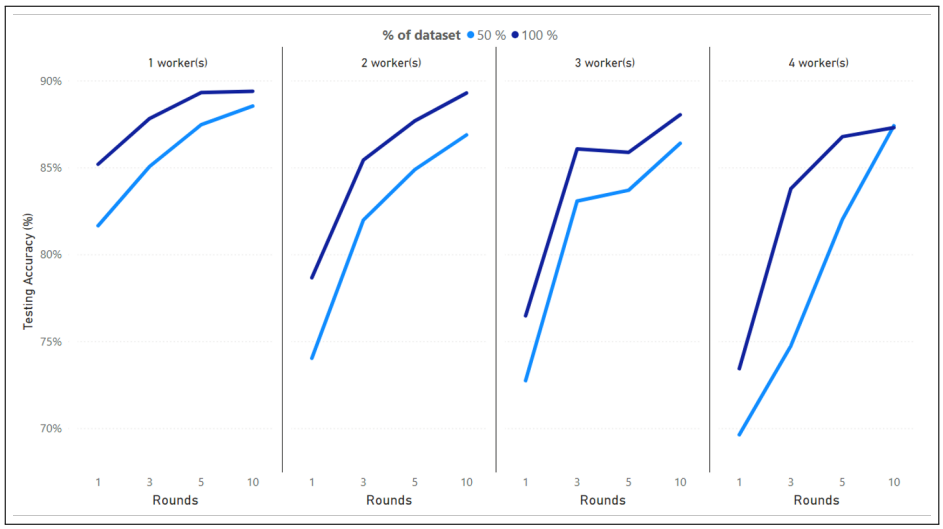


Figure 6.5: Testbed testing accuracy (%) vs training rounds (1,3,5,10)

Effect of number of workers on accuracy: Figure 6.5 highlights an interesting trend, for a fixed number of rounds, using more workers tended to yield slightly lower final accuracy than using fewer workers (or a single worker). In other words, the model trained with 4 distributed clients in 5 rounds did not perform quite as well as the model trained with 1 client in 5 rounds, which in the 1-client case essentially means all data was centralised on that one client. For example, in simulation with 5 rounds on the full dataset, the test accuracy was 89.0% with 1 worker, versus 86.2% with 4 workers. A similar gap was seen on the testbed (89.3% vs 86.7% for that scenario). This can be attributed to two factors: **(1) Data fragmentation:-** with multiple workers, each only sees a portion of the data. For example, 4 workers might each see roughly 25% of the training data. If the data is not perfectly IID, each model update is based on a narrower view of the problem, which can slow down global convergence or converge to a slightly inferior model if the ensemble of updates isn't as comprehensive as a single batch on all data. **(2) Fewer local epochs per data point :-** in our FL setup, each round every client likely performed one epoch (or a fixed number of epochs) on its local data. With one worker, one round = one epoch on the whole dataset. But with four workers, one round = one epoch on each quarter of data, which is effectively only one quarter of an epoch on the full data per round since weights are averaged and not sequentially trained through all data in one go. Therefore, after equal numbers of rounds, the multi-worker

case has effectively seen the full dataset fewer times. It needs more rounds to catch up in training progress.

This explains why at 10 rounds the gap in accuracy between 1 worker and 4 workers narrows significantly. Given enough rounds, the multi-worker federated model can approach the accuracy of the single-worker (centralised) model, but it requires more communication rounds to compensate for the dispersed learning. This phenomenon is well-documented: the original FedAvg paper noted that federated training might underperform compared to centralised training, especially under non-uniform data distributions or varying client compute, unless sufficient rounds or modifications are used [76]. In practice, one often finds that federated models allows for a minor loss of accuracy in order to protect data privacy [156]. This means there is a small accuracy penalty to doing distributed learning, which is the price of keeping data decentralised. Our results quantify that penalty in a controlled way: a difference of 1-3 percentage points in accuracy for the same number of training rounds when training is split among 4 devices versus centralised. The reward, of course, is that no single device ever had to send or receive raw data from the others which is a critical advantage for privacy in smart agriculture scenarios where raw data, perhaps farmers' field images or sensor readings, should not be pooled centrally.

Another way to look at this is through convergence speed: How many rounds does it take to reach a certain accuracy with different numbers of workers? In our experiments, to reach 85% test accuracy on the 100% dataset, 1 worker needed only 1-2 rounds, whereas 4 workers needed 3 rounds to get from 71.7% (at round 1) up to 84.9% (by round 3). To reach 89% accuracy, 1 worker took 5 rounds, whereas 4 workers did not quite reach 89% even by 10 rounds. Extrapolating, 4 workers might need perhaps 15 rounds to hit 89%, hence requiring more communication cycles. This reinforces that distributed FL may require more iterations to reach the same accuracy as a centrally trained model. The simulation vs testbed did not affect this intrinsic property, but the testbed would incur a much larger time cost for those extra rounds, as detailed earlier. Therefore, a practitioner must decide: Is a small accuracy hit acceptable in exchange for significantly less training time and less communication? Or is maximum accuracy paramount, justifying the longer training? These are the time-accuracy trade-offs at the heart of FL system design.

Importantly, accuracy did benefit from using the full dataset (100%) vs only 50% of the data. Across all configurations, the final model accuracy with 100% data was 2-4% higher than using 50% of the data. This is expected since more training data provides better generalisation. For example, with 10 rounds and 1 worker, test accuracy was 88.5% on 50% data vs 89.4% on 100% data (testbed results). With 4 workers and 10 rounds, 50% data yielded 87.4% vs 87.2% for 100% data. The gap isn't huge, which suggests the 50% subset was still representative enough to achieve around 98% of the full accuracy, but in a couple of cases, especially at low rounds, insufficient data clearly limited the

accuracy. This underlines that data volume matters for model quality, and one should aim to involve as much useful data as possible in federated training. In smart agriculture contexts, this could mean involving data from multiple farms or seasons. Our testbed results confirm that even when network and device constraints double the training time for using the full dataset, the accuracy payoff can justify it, especially for critical applications like crop disease detection where every percentage point of accuracy might count in identifying issues.

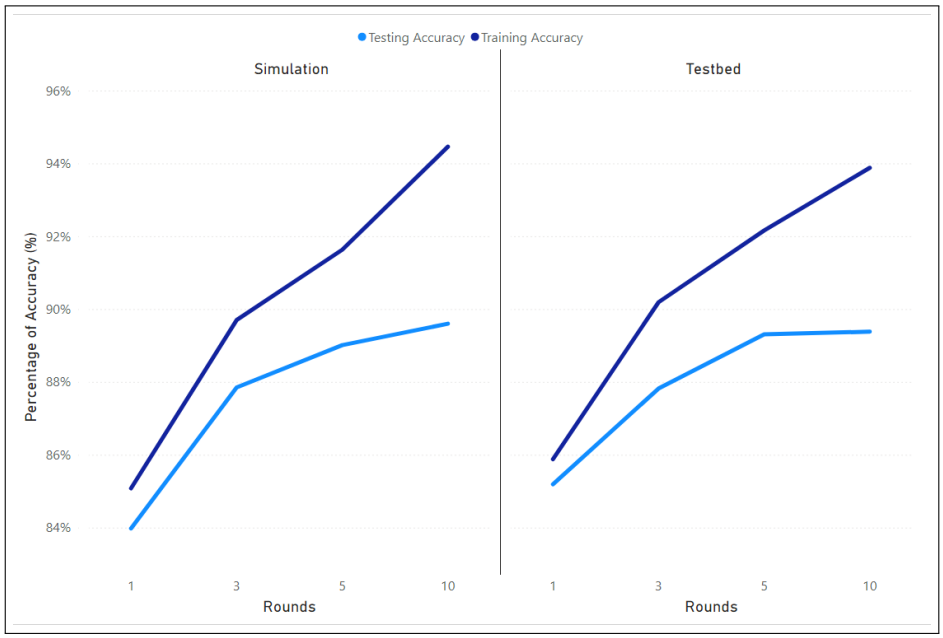


Figure 6.6: Overfitting test per round (1,3,5,10) for 1 worker with 100% Dataset

Training vs test accuracy (overfitting considerations): We also monitored training accuracy, which in all cases was slightly higher than test accuracy, as expected. With more rounds, the training accuracy continued to increase even after test accuracy plateaued, leading to a growing gap. This is a sign of an overfitting that is starting to occur, as illustrated by the widening gap in Figure 6.6. For instance, in simulation with 1 worker on full data, training accuracy reached 94.5% vs test 89.6% at 10 rounds. This is about a 5% gap, whereas at 1 round they were 85.1% vs 84.0% which is only 1% gap. In federated settings, overfitting can occur on the clients' local data if training goes on for too long without new data or regularisation. Our use of a moderate number of rounds kept overfitting in check as the gaps remained reasonably small, but this observation suggests that there is an optimal stopping point for training to maximise generalisation. In practice, techniques like early stopping on a validation set or regularising local training by limiting local epochs or using dropout could be used in FL to ensure the global model doesn't overfit to the union

of client data. For our experiments, it appears 5-10 rounds was near optimal for test accuracy, as additional training might only boost the training accuracy indicating memorisation.

To close this section, our analysis of accuracy shows that the federated learning approach achieved strong model performance, approaching what one might expect from centralised training on this dataset. The simulation and testbed achieved comparable accuracies, proving the soundness of the FL algorithm implementation in both. However, there are nuanced trade-offs: distributing the learning across more clients can slow down the convergence in terms of rounds required and can slightly lower the accuracy achieved in a fixed number of rounds, compared to a fewer-client approach. This reflects known challenges in FL such as non-IID data and gradient variance across clients [155]. Still, given sufficient rounds, which is a question of time and resources, the federated model can reach nearly the same accuracy as centralised. The advantage is that it does so without centralising data, a crucial benefit for privacy and feasibility in many smart agriculture scenarios where data is generated in distributed sites such as farms and greenhouses.

Having examined performance and accuracy, we now interpret what these findings mean for the deployment of federated learning in smart agriculture, and how our testbed experience informs considerations like system scalability, robustness, and reproducibility.

6.5 System Utilisation and Resource Analysis

We recorded CPU utilisation during the experiments to gauge how intensively the system resources were used in both environments. This provides insight into whether the bottlenecks were computational (high CPU usage) or elsewhere (e.g. I/O or waiting on network). In the simulation runs, CPU usage on the host machine increased with the number of workers, as it had to handle more parallel training threads. For example, with 4 simulated workers, the CPU utilisation averaged around 30%, compared to 15% with a single worker. This indicates the simulation was effectively multi-threaded and the workload scaled out on the CPU. The fact that the CPU did not reach near 100% suggests that either the workload was not extremely heavy likely due to a relatively small model and dataset, or that some bottlenecks like waiting for data or synchronisation points kept the CPU idle at times even as it orchestrated multiple clients.

On the testbed, we measured CPU usage and observed values in the range of 15–25% on average, with a slight uptick as the number of workers increased as there is more aggregation work being done and therefore handling more network I/O. For instance, with 4 workers the server CPU was around 25% utilised during training, compared to 18% with 1 worker. The client devices (Jetson Nanos) each have their own CPU. The modest CPU usage implies that the ag-

gregator was not a bottleneck as it had plenty of headroom (75% idle) even at the highest load. The heavy lifting was done on the clients.

The role of GPUs in our experiments deserves clarification. The simulation ran on a standard CPU. The Jetson Nano has a 128-core Maxwell GPU which can accelerate neural network training, but it is memory-bandwidth-limited. We did not capture GPU metrics, but if GPU was used, one would expect lower training times on Jetson than purely CPU training. Comparing CPU vs GPU performance in this context, we rely on known characteristics: A desktop-grade GPU can be an order of magnitude faster than the Jetson's GPU. If our simulation had been run on a powerful GPU, the time gap between simulation and testbed might have been even larger. Conversely, if the Jetson clients were switched to a more powerful device with a better GPU, for example a Jetson Xavier NX or an edge server, the testbed times would decrease. Essentially, the hardware differences (CPU/GPU speed, memory) drive the performance gap.

One specific observation in the results is that for 1-worker scenarios on the testbed, the CPU utilisation percentage actually decreased as the number of rounds increased. For example, 16.6% at 1 round vs 12.0% at 10 rounds, for 1 worker 100% dataset. This counter-intuitive drop is likely because the measurement was averaged over the entire training session: with more rounds, a larger fraction of time is spent communicating or waiting especially if that 1 worker finished local computation quickly and the rest was overhead. Therefore, the average CPU usage over time can go down if the process spends more time idle or waiting for network as rounds increase. In multi-worker cases, CPU usage stayed higher, possibly because there was always some activity as with multiple clients, the server was almost continuously receiving or processing one of the updates. The key point is that the system utilisation numbers suggest that computation on the edge devices was a heavy task but not beyond their capability as they could handle it, just slowly, and without maxing out CPU constantly due to interspersed communication, and the aggregation was never a performance bottleneck in terms of raw compute.

To summarise this section, our analysis of system resource usage highlights that the compute bottleneck lay in the client devices (Jetson Nanos), consistent with the understanding that edge devices have constrained processing power. We also see that the communication overhead did not fully saturate the server. This is good as it means that the server can handle more clients or more frequent updates if needed, but communication did affect overall timings as discussed. The robustness of the testbed was demonstrated by its ability to handle multiple rounds with multiple devices without failure. The entire process was repeatable and stable over numerous runs, indicating that the system can be reliably used for experimentation or even pilot deployments.

6.6 Communication Overhead and Data Transfer

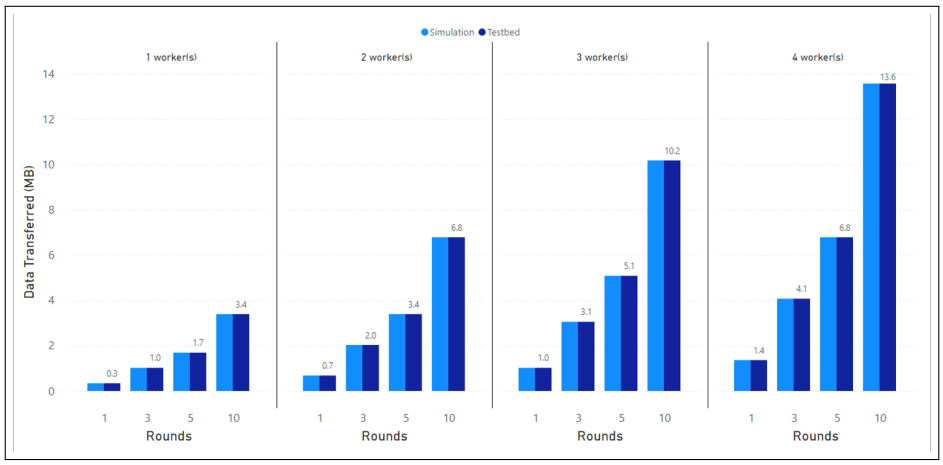


Figure 6.7: Data transferred (MB) by rounds (1,3,5,10)

Communication is a crucial aspect of federated learning performance. In our experiments, we measured the total data transferred over the course of training. This includes sending the global model to each worker at the start of each round and receiving the updated model from each worker at the end of the round. The size of each model, and therefore each update, in our setup was approximately 0.34 MB, as evidenced by the data transferred for 1 worker in 1 round. With this figure, we can verify that the data transfer scales linearly with number of workers and number of rounds: for example, in 4 workers $\times$ 10 rounds, the table shows 13.6 MB transferred which is 4 workers * 10 rounds * 0.34 MB, counting both the uplink or downlink.

In both simulation and testbed, the amount of data transferred is identical for equivalent runs, since it purely depends on the protocol (how many bytes are sent). Therefore, there was no discrepancy in data volume, only in how long it took to transfer. In the testbed, transferring 0.34 MB to a worker and back might take on the order of a few hundred milliseconds over Wi-Fi (assuming 5-10 Mbps effective throughput when accounting for overhead and contention). This is normally negligible compared to a training computation that might take many seconds on a Jetson as, in early rounds with heavy training, communication was a small fraction of the round time. However, as noted earlier, when clients have very little to compute, for example, in later rounds or in the many-clients scenario, the communication overhead becomes more visible.

From the results, we can infer that communication was not the primary bot-

tleneck in our testbed given the model size; the bulk of the time was spent on local training. For instance, 4 workers transferring 13.56 MB over 10 rounds: even if our Wi-Fi was modest, 13.56 MB spread over 10 rounds (1.356 MB per round total across all clients) is trivial for a modern wireless network (which can handle dozens of MB per second). Therefore, latency, which would be the fixed round-trip time, is more of a factor than raw throughput. Each round has at least one round-trip of latency. If our latency per round was 0.1 seconds, then over 10 rounds that adds 1 second which is completely overshadowed by minutes of compute. If latency was larger, perhaps our Wi-Fi introduces 0.5s each way, that could add a few seconds per round. In either case, for our scale, network latency overhead might be on the order of a few seconds total, which is <5% of the total time in most runs. Therefore, while communication was certainly present, it wasn't choking the process.

However, this picture can change in a larger scale deployment. If the model were much larger, or if the network was slower, communication could easily become the dominant factor. The literature on FL frequently cites communication as a major obstacle when scaling to many devices [76]. In smart agriculture, devices might be connected over low-bandwidth links or intermittent connectivity. Our testbed's use of a relatively fast Wi-Fi in a lab is an optimistic scenario; in the field, one might have to rely on, for example, a low-power wide-area network (LPWAN) which could make even 0.34 MB a significant transfer.

Another aspect is that we used a synchronous training approach where all clients need to communicate each round. This means all data transfers happen in a burst each round. In asynchronous FL, communications could be staggered, which might improve throughput usage but introduces other complexities such as stale updates. There is active research on how to make FL communication more efficient, for instance, by sending only model differences or using update sparsification [76]. In our testbed, we did not employ such methods; every round, full model weights were exchanged. This is the simplest approach and was sufficient for the moderate model size we had.

Finally, our results confirm that the data transferred grows linearly with both the number of rounds and number of clients, which is expected from the FL protocol. If we were to scale to100 rounds or 10 clients, we'd see proportionally more data moved. The implication for scalability is that a central server in a real deployment should be provisioned with network capacity roughly proportional to the desired update frequency times the update size times number of clients. The volume in our case is small, a few megabytes, but at larger scale it could become gigabytes of traffic in aggregate.

One might question: is the communication pattern realistic for smart agriculture? In many agriculture IoT settings, devices might not be online continuously or might have to send data over long distances. FL does allow a degree of flexibility where clients could train offline and send updates when connected. Our

testbed assumed continuous connectivity for the duration of training. We did not simulate dropped connections or clients that fail to send updates, which can happen in real life [84]. A robust deployment would need to handle such cases. From a robustness standpoint, our testbed ran smoothly likely because conditions were stable; but it's important to acknowledge that in production, communication failures are a real issue to design for.

To sum up this section, network communication in our FL testbed introduced modest overhead but did not bottleneck the training, given the experimental parameters. The results demonstrate that our implementation's communication was efficient and the measured data transfer aligned with theoretical expectations. This validates the correctness of our networking aspect which is that the testbed truly sent the amount of data we anticipated with no hidden inefficiencies. For smart agriculture deployments, careful consideration must be given to network reliability and capacity. If anything, our results are on the optimistic side due to good network conditions; actual farms may need to leverage edge computing (local aggregation nodes, etc.) to mitigate weaker connectivity. For example, an edge cluster approach could aggregate updates within a farm locally over a local network and only send a summary to the cloud, reducing long-haul communication. Such hierarchical FL setups are proposed to address communication bottlenecks in FL at scale. In our single-hop testbed, that wasn't necessary but remains an avenue for future scaling.

6.7 Implications for Smart Agriculture and Deployment Considerations

The primary motivation for developing this FL testbed was to explore its viability and performance for smart agriculture applications. Having analysed the results, we now discuss what they mean in the context of agriculture and how a practitioner might use these insights when deploying FL in the field. We also address the robustness, reproducibility, and scalability of our testbed approach, which are important considerations for moving from research to real-world use.

6.7.1 Privacy and Data Locality

Smart farming involves distributed data sources, for example, sensors in different locations, machines on various fields, or multiple farms collaborating. Federated learning aligns well with this scenario by allowing a global model to be trained without aggregating all data to a central server. Our successful training of an FL model in the testbed demonstrates that this concept is feasible: even with modest hardware, the models reached high accuracy. This suggests that agriculture stakeholders can benefit from collective models, for tasks like crop disease detection, yield prediction, or livestock monitoring, without having to pool sensitive raw data which could include images of crops or farm management data. This is a significant advantage since data privacy is crucial for farmers

and organisations as they often hesitate to share raw data due to privacy or competitive concerns. By keeping data on devices such as their tractors, drones or weather stations, and only sharing models, FL could enable collaborative analytics in agriculture that were not previously possible. Our experiments reinforce this by showing the trade-off: a small hit in accuracy for the benefit of data never leaving the device. Therefore, our work provides empirical evidence of how an FL system might perform in practice on farm-like edge devices.

6.7.2 Real-time Decision Making

Agriculture often requires real-time or near-real-time monitoring for example in cases of detecting an irrigation leak or a pest outbreak as soon as possible. Edge computing, which includes FL on edge devices, is touted to allow such real-time decision-making on the farm. However, our results temper expectations about real-time training. Training an FL model on edge devices as we did can take long for even relatively small tasks. This is not an issue if model training is done as a background process, for example, updating a model once per day with new data. In many agriculture scenarios, that is acceptable where the model can be retrained overnight when devices are idle, and then the updated model is used for inference during the day. FL fits into this by periodically exchanging model updates, perhaps daily or weekly. It would be unrealistic with current edge hardware to expect that an FL model could be trained from scratch in minutes on-site. Instead, careful scheduling is needed. On the other hand, the inference using the trained model is typically fast, usually in milliseconds, and can happen in real-time on the same edge devices. Therefore, a plausible deployment pattern is: continuously use the current model for inference on the farm for real-time detection, and occasionally run FL training rounds to improve the model. Our testbed's performance data helps in planning such schedules, for example, by knowing that an update of 5 rounds will take about an hour with 4 devices lets the engineer decide how often to trigger it.

6.7.3 Energy and Battery Considerations

Many agriculture devices are battery or solar-powered. Training a model is computationally intensive and will draw significant power. This could drain batteries or interfere with the primary sensing tasks. Our measured CPU utilisations and times can be used to estimate energy consumption on the Jetson (the Jetson Nano can draw up to 5-10W under load). For a 1-hour training session, that's 5-10 Wh per device, which might be acceptable daily for a solar setup, but it's not negligible. In some cases, it might be better to offload training to a plugged-in device like a central farm computer or only involve devices that have enough power. The fact that our testbed functioned uninterrupted indicates robustness in a controlled environment with constant power and connectivity, but in the field one must consider power management. Techniques like reducing the frequency of communication or using only a subset of clients each round, so some can sleep, can conserve energy. Also, the use of faster

convergence algorithms to reduce rounds needed directly translates to energy savings.

6.7.4 Scalability to Larger Networks

Our testbed was limited to 4 workers due to hardware availability. Real deployments could involve dozens of nodes. Scalability in terms of number of clients is a challenge. The results we have give clues: as we projected, more clients will add more overhead per round for communication and coordination. If we scaled to 8 clients on the same Wi-Fi, the completion time might start to increase if bandwidth becomes an issue or if the server CPU has to handle more sockets. One solution is to introduce hierarchical FL by grouping clients under intermediate aggregators, effectively implementing a two-tier federated learning. That approach can reduce the strain on any single server and is highly relevant to agriculture, where you might have per-field aggregators sending to a farm-level aggregator, and then to a global model. While our current testbed didn't completely test hierarchical FL in its fullest form by adding another intermediate aggregator for the coordinator to aggregator, it's an obvious next step for scaling. The data transferred and time per round we observed would scale roughly linearly with clients until hitting some network limit. If each Jetson had identical data size, doubling clients roughly doubles the data sent per round. So, to maintain round time, one would need either more bandwidth or to tolerate longer rounds. Another factor is client heterogeneity where not all devices may be as capable as each other. In farms, one node might be a powerful computer in a tractor, while another is a tiny sensor node. FL algorithms need to account for stragglers so that one slow node doesn't hold up every round. Our homogeneous testbed didn't face this, but a robust system should perhaps set a cutoff time for each round to proceed with aggregation once a certain time passes, to avoid indefinite waiting. This is part of making the system robust to real-world conditions.

6.7.5 Reproducibility and Testbed Validity

A noteworthy outcome of this project is that we now have a working FL testbed that closely mirrors the simulation. This means that researchers or practitioners can reproduce our experiments on similar hardware and expect similar results, which is valuable for validating FL algorithms under realistic conditions. Often, FL research is done solely in simulation, and as others have pointed out, there is a need for more real-world testing [155]. Our testbed contributes to filling that gap by providing a template for how to set up FL on actual devices. We paid attention to instrumentation (logging time, accuracy, resource usage) so that behaviour is observable as this is crucial for debugging and improving FL protocols. The fact that our testbed results aligned well with known theoretical expectations such as linear scaling of communication and convergence trends gives confidence in its correctness and reproducibility. We could run the same experiments multiple times and get consistent accuracy outcomes as the vari-

ability was low. This consistency is important for scientific rigor. Robustness was also demonstrated in that the system did not crash or diverge even when training increased the number of rounds and clients; the FL algorithm remained stable with no exploding gradients or model divergence, which can sometimes happen in distributed training if learning rates are mis-tuned. We attribute this stability to the FLIGHT framework which made a careful selection of hyperparameters and the inherent simplicity of FedAvg which is known to be robust in IID settings.

6.7.6 Integration with Smart Agriculture Systems

Finally, we consider how an FL testbed like ours could be integrated into a larger smart agriculture system. Typically, a precision agriculture system might have an IoT platform where data is collected, some edge computing platform for analytics, and a user interface for farmers. FL would slot into the analytics part where instead of raw data being sent to a cloud service for model training, the model training happens in the background on the edge. One practical deployment could be: each farm has a central gateway like a Jetson or a small server that acts as the FL coordinator, and devices in the farm like tractors, cameras and sensors act as workers. Periodically, gateways from multiple farms coordinate. This way, a model can be learned from data across farms bringing in diversity and volume but each farm's data stays on the farm which is a highly attractive scenario for cooperatives or agricultural research collaborations. Our experiments with up to 4 clients could be seen as a proxy for 4 farms or 4 field sections collaborating. The high accuracy achieved indicates that if each client's data had unique insights where one farm has slightly different soil conditions and another has different crop variety, the combined model can be superior to any single farm's model. FL's promise is partly that the whole is greater than the sum of parts because models benefit from broader data. We did not explicitly measure a case of non-IID data where each client's data distribution differs significantly, which is common in agriculture due to regional differences. Those are beyond our current scope but are relevant next steps for making FL truly effective in agriculture, where heterogeneity is the norm.

6.8 Summary

The experiments demonstrate that a federated learning testbed can successfully be built and used to benchmark performance for smart agriculture use-cases. Completion times on real edge hardware are substantially higher than in simulation, but still within a tolerable range for periodic model updates on the order of tens of minutes to hours. Model accuracy remains high, validating the efficacy of FL even with fragmented data and limited devices. There are clear trade-offs between the number of devices, rounds, and accuracy: more devices can speed up training but may require more rounds to reach the same accuracy; more rounds improve accuracy but incur time and energy costs. Networking

and system factors (CPU speed, Wi-Fi bandwidth) significantly affect performance, underlining the need to test on real hardware as we did. In the context of smart agriculture, these results imply that federated learning is a promising approach to enable collaborative, privacy-preserving machine learning. However, one must carefully plan deployments to account for longer training times and the optimal settings that balance model improvement with resource usage.

Our FL testbed proved to be robust by handling continuous operation and multiple clients, reproducible by yielding verifiable results), and provides a foundation for scaling up. By incorporating insights from existing literature and our empirical data, future improvements can target the main pain points identified such as incorporating client drop-out tolerance to handle flaky farm connectivity, using model compression to alleviate network usage, or leveraging on-device accelerators to reduce training time. The next chapter will build upon these findings to formulate guidelines and potential enhancements for federated learning in smart agriculture, as well as concluding remarks on the impact of this work.

Chapter 7

Conclusion and Future Work

7.1 Summary of the Thesis

This thesis set out to investigate the feasibility of Federated Learning (FL) in resource-constrained smart agriculture settings by designing, implementing, and evaluating a standards-based hardware testbed. The core contribution is a physical FL testbed composed of real edge devices (NVIDIA Jetson Nano boards) configured in a hierarchical FL architecture with a coordinator, a global aggregator, and multiple worker nodes. Leveraging the open-source FLIGHT framework and using a representative image classification task (the Fashion-MNIST dataset as a proxy for agricultural data), we demonstrated how these devices can collaboratively train a model without centralising data. We developed a reproducible deployment methodology specifically for FLIGHT on edge hardware and measured key performance metrics. The experimental results confirmed that even low-power edge devices can train useful models within reasonable timeframes, although training on physical hardware is significantly slower than in simulation. Notably, the federated approach achieved competitive accuracy while dramatically reducing raw data transfers, indicating that FL can address connectivity limitations and privacy requirements common in rural farms. Overall, the thesis bridges the gap between simulation and real-world deployment of FL in agriculture: it provides a working testbed and empirical insights that validate FL's viability in smart farming scenarios. These contributions, which includes a reproducible FL testbed with an open-source experimental framework using FLIGHT and a comprehensive evaluation, lay a foundation for future research and guide practitioners in deploying privacy-preserving machine learning on farms.

7.2 Implications for Research and Deployment

The development and evaluation of this federated learning testbed carry several important implications for both the research community and those looking to deploy FL in operational agricultural systems.

Implications for research: This work demonstrates the feasibility of moving beyond simulation and into physical experimentation for FL. By validating that a network of small edge devices can collaboratively train a model with acceptable performance, we provide a real-world evidence point that can encourage researchers to test their novel FL algorithms under more realistic conditions. This is critical for identifying algorithmic issues that only manifest in distributed settings. Such insights can guide researchers to develop more robust FL algorithms, for example, by accounting for asynchronous updates or variable compute capabilities, that are truly deployment-ready. Moreover, our testbed can serve as a benchmarking platform: researchers can use it to directly compare the performance of different FL strategies, such as FedAvg vs. FedProx, or various compression techniques, under identical hardware/network conditions, something that was previously hard to do due to difficulties in reproducing replicable testbeds. This could accelerate innovation by quickly eliminating out approaches that don't translate well to practice and highlighting those that offer improvements not just in theory but also in system efficiency.

Implications for real-world deployment: The successful implementation of FL on a hardware testbed builds confidence that similar setups can be created on actual farms or agricultural facilities. Stakeholders in industry and agriculture technology can take our architecture and results as a reference model. One implication is that farms or agricultural research stations could start with small-scale pilots of FL using hardware like we employed, to evaluate benefits for their specific applications, for example, an agricultural cooperative training a shared pest detection model across several orchards. Our results imply that if the computing task is modest as many agricultural ML tasks are, focusing on specific classifications or regressions, even low-cost devices can handle the training workload. This lowers the barrier to adoption because it suggests that integrating FL doesn't necessarily require investment in high-end hardware; rather, existing farm computers or affordable devices like the Jetson Nano or Raspberry Pi could suffice for initial deployments. Another implication is related to data privacy and governance: by showing that useful models can be trained without aggregating raw data centrally, this work provides a blueprint for compliance with data sovereignty requirements. Agricultural agencies or companies that collect data from farmers could use FL techniques to extract insights like regional yield predictions without ever taking possession of the raw farm data, thereby respecting privacy and potentially easing regulatory constraints. This is particularly significant in regions where data protection laws or farmer unions mandate strict control over data sharing.

In deploying FL for agriculture, our findings also highlight several practical considerations. We observed that network unreliability can slow training; therefore, for real deployments, an implication is that network infrastructure improvements even something as simple as establishing a local Wi-Fi mesh on a farm to connect devices could directly improve the efficacy of FL. Conversely, if improving connectivity is not possible, one might design the FL process to be more resilient, for example, allowing more local training between communications, or using relay nodes. The testbed underscores the necessity of having a central coordinating node or service that can perform aggregation. In a deployment scenario, this could be a base station on the farm, or a cloud server if internet connectivity allows periodic sync. The choice between an on-premise aggregator vs. cloud aggregator has trade-offs in latency and reliability that stakeholders will need to consider. Our work provides data to inform that decision, for example, the bandwidth per round needed, which if small might permit using a cloud server even over a slow link.

Finally, on the implication for sustainable deployment, running training on edge devices means those devices will consume energy and potentially wear out faster. We have shown that for a small model the energy and heat were manageable on our devices, but scaling up may require careful planning in terms of cooling and power supply. This suggests that organisations should monitor device health and perhaps schedule federated training during times that minimise conflict with other critical operations, for instance, not when the device is simultaneously needed for real-time control tasks. In sum, the implications of this work encourage a shift from algorithm-centric research to a more holistic approach where system design, hardware, and context are considered, and they lay out practical considerations that anyone attempting to bring FL to the agricultural field should heed.

Collectively, the above implications point to tangible steps that different stakeholders can take. The next section outlines recommendations for various stakeholders to capitalise on our findings and facilitate the successful adoption of FL in smart agriculture.

7.3 Recommendations for Stakeholders

Building on our results and the above implications, we propose several recommendations for different stakeholders to successfully leverage federated learning in smart agriculture.

For Agricultural Technology Developers and Integrators: Start with pilot deployments of FL on a small number of edge devices to gain experience. Use hardware similar to our testbed to replicate our setup and verify that your specific models can run within resource limits. We recommend using open-source

FL frameworks such as FLIGHT which we found effective, to avoid vendor lock-in and facilitate community support. Also, focus on lightweight models that meet the accuracy needs of the application but are efficient to train on the edge. Complex deep models may need to be simplified, for example, using fewer layers or parameters, for practical on-device training.

For Policymakers and Agricultural Regulators: We encourage the development of guidelines and incentives for privacy-preserving data analytics like FL in agriculture. Policies that presently restrict data sharing could be updated to explicitly recognise federated learning as a compliant method of collaborative analysis, since it does not require raw data to leave the source. Providing grants or subsidies for rural connectivity and edge computing infrastructure can directly impact the success of FL deployments, as our work indicates, a modest improvement in connectivity greatly improves federated model training. Policymakers should also promote standardisation of data formats and interoperability: if sensor data and models adhere to common standards, it becomes easier to plug into a federated learning process across different farms and device types. Support the creation of open testbeds (similar to our prototype) as community resources where new solutions can be tested under realistic conditions before being rolled out broadly.

For the Research Community (AI and Agriculture): We recommend embracing a more deployment-driven evaluation culture for federated learning research. As our Systematic Literature Review highlighted (Section 2.4), much of the current work remains confined to simulations or lab settings (TRL 4-6), underscoring the need for more testing under realistic conditions. Whenever possible, test new FL algorithms on physical devices or at least under emulated edge conditions, for example, using network simulators to impose realistic latency. Researchers could use our testbed blueprint to set up their own experimental environment; we have provided extensive documentation to aid replication. This will ensure that proposed improvements, for example, new aggregation rules or personalisation techniques are grounded in practicality and thus more likely to be adopted. We also encourage interdisciplinary collaboration which involve agricultural scientists or engineers in designing FL experiments so that the chosen ML tasks and conditions reflect real farming needs. By aligning experiments with real-world use cases like specific crop disease datasets or sensor readings, research outcomes will be more directly transferrable.

For Farmers and End-Users: While the technical details of FL may be abstract, it's important for end-users to be educated and engaged about what this technology means for them. We recommend outreach and training for farmers on how their devices and data would be used in a federated learning system. Emphasise the privacy aspect, that their raw data stays on their farm, to build trust. Also, provide easy-to-understand interfaces or summaries of what the AI models are learning and how that benefits farm decisions, for example, a dashboard that shows the model's recommendations or findings. This will make

farmers more willing to participate in data-sharing initiatives via FL, since they remain in control and also see the tangible benefits. Feedback from farmers should be looped into the design of FL systems, for instance, if bandwidth is a concern at certain times, the system could adapt its synchronisation schedule. In short, human factors are as important as technical performance for real-world success, and we recommend stakeholders prioritise ease of use and transparency when deploying federated learning on the farm.

By following these recommendations, stakeholders can collaboratively push federated learning from a promising concept to a routinely deployed technology in agriculture, thereby unlocking new efficiencies and insights while respecting data privacy and local autonomy.

7.4 Technical Limitations of the Current Work

It is important to recognise the limitations of our current testbed and study, as these define the boundaries of our conclusions and suggest avenues for improvement.

First and foremost, the data used in our experiments (FashionMNIST) is a proxy for agricultural data and not actual farm sensor readings or images. While FashionMNIST is a standard benchmarking dataset in ML research, its characteristics (small grayscale images of clothing items) differ from typical agricultural data which might be larger color images of crops, or time-series sensor measurements. Therefore, our results on model accuracy and convergence speed are illustrative but not directly indicative of performance on real agricultural tasks. The patterns of non-IID data we simulated, by partitioning classes among client, may not capture all nuances of farm data distribution. This means the testbed needs further validation with domain-specific datasets to confirm that our findings hold in situ.

Secondly, the current testbed did not integrate physical sensor data streams or real-time data collection. Each client device's data was static and pre-loaded. In a real deployment, new data would arrive continuously which could be each hour or new images from drones each day, and the FL system would need to handle streaming updates and possibly concept drift in the data over time. Our experiments did not account for this temporal aspect. Therefore, the testbed in its present form cannot assess how well FL would cope with evolving data or how to schedule training cycles in tandem with data generation. Incorporating live data feeds remains a challenge for future work.

Another limitation is the scale and heterogeneity of the testbed. We used at most a handful of identical edge devices (up to 6 Jetson Nanos) in our evaluation. Real-world federated networks could involve dozens or hundreds of devices across a region. There are practical limitations in our lab setting that prevented

scaling to that level such as, but not limited to, hardware availability and network setup complexity. As a result, we have not observed how the system behaves at larger scales. Issues like network contention, coordinator overload, or complex scheduling of clients (where only a subset of clients might participate in any given round due to connectivity) were not encountered in our small-scale, managed experiments. Similarly, all our clients were of the same type and had similar computing power; we did not explore heterogeneous hardware where some clients are much slower than others or have different architectures. In real deployments, heterogeneity is likely. The lack of heterogeneity in our testbed means we did not test the robustness of FL algorithms to varying client speeds (straggler issues) or the need for techniques like federated normalisation for different data scales.

Furthermore, our experimental framework was specifically built around the FLIGHT FL library to evaluate its performance, particularly its serverless deployment capabilities and hierarchical support, on physical edge hardware. Therefore, our results primarily highlight the performance and characteristics of FLIGHT within this specific setup. This study does not claim that FLIGHT is the optimal or only viable serverless or hierarchical FL framework; a comparative evaluation against other frameworks was outside the scope. A core goal was to bridge the gap between theoretical performance often seen in simulations and the practical realities observed when running a framework like FLIGHT on resource-constrained physical devices. Our findings are therefore specific to this framework and context, emphasising the importance of physical testbeds for validating simulation-based expectations.

Also, due to time and scope constraints, we did not implement certain advanced FL features that could be relevant. For instance, we did not use any secure aggregation or differential privacy mechanisms in our experiments. All model updates were sent in clear and assumed to be truthful. In practice, if privacy needs to be mathematically guaranteed or if there's a risk of malicious clients, additional protocols would be necessary, which could add computation and communication overhead. Our testbed currently doesn't evaluate those aspects. We also focused on the baseline FedAvg algorithm and did not experiment with more complex FL algorithms. Therefore, our analysis of algorithmic performance is limited to FedAvg's behaviour.

Finally, our evaluation primarily measured technical metrics (accuracy, time, resource usage) and did not incorporate end-user feedback or economic analysis. In a deployment, factors like the cost of devices, ease of maintenance, and user acceptance would play a role. Our work did not cover these human-centric or economic dimensions.

In summary, these limitations delineate the scope of our conclusions. They highlight that while our results are promising, caution must be taken in generalising them directly to all smart farming scenarios. At the same time, each

limitation clearly points to areas for improvement, many of which we address in our recommendations for future research directions described next. By recognising these constraints, we can better understand how to extend the testbed and experiments to further close the gap between this initial prototype and a fully robust real-world system.

7.5 Future Research Directions

Despite the limitations above, our work opens up multiple future research directions to improve the federated learning testbed and to further close the gap towards real-world deployment in agriculture.

1. **Integration of Real-time Sensor Data:** A top priority is to extend the testbed to handle live data streams from physical sensors. In future work, plan should be to connect actual agricultural sensors or sensor simulators to the edge nodes so that data is collected and fed into the local training processes in real time. For example, a Jetson Nano could be interfaced with a camera capturing plant images or with an IoT sensor node streaming soil moisture readings. This would allow studying how the federated learning system performs under continuous data inflow and whether the training schedule can adapt, for example, triggering a new round when sufficient new data has accumulated. It also enables research on concept drift, as models might need to update continuously to reflect seasonal changes or growth stages. Integrating real sensors will bring the testbed environment even closer to on-farm conditions and will likely surface new challenges in data preprocessing, timing of model updates, and handling of irregular or bursty data.

2. **Scaling the Testbed and Network Topology:** One direction is also to scale up the number of edge devices in the testbed to evaluate federated learning at larger scales. This includes adding more Jetson Nanos or other device types, potentially leveraging virtualisation or containerisation to simulate additional clients beyond the physical hardware we possess. A scaled testbed targeting tens of clients would help observe emergent issues like network congestion when many clients send updates simultaneously, or the diminishing returns on accuracy as more low-data clients join the federation. Also, exploring more complex network topologies is a future direction: for instance, implementing a complete hierarchical FL setup where groups of devices located in a different location first aggregate locally, and then a global aggregation occurs. This multi-level federation could reduce communication costs further and better reflect scenarios where a central cloud service is not always reachable, but farm-wide local aggregators, edge gateways, exist. Investigating how such hierarchical or clustered FL performs using our testbed by adding some Jetsons as aggregators would extend its applicability. Moreover, incorporate network emulation tools to simulate various connectivity scenarios including

long-distance low-bandwidth links, or intermittent outages in a controlled manner can be performed to test the robustness of the FL process under true rural network conditions. Scaling the testbed and varying the network conditions will provide deeper insights into the limits of the current FL approach and guide the development of more resilient protocols.

3. **Expanding Application Scenarios and ML Models:** Another direction is to broaden the application scenarios demonstrated on the testbed. While image classification with FashionMNIST was a convenient starting point, future experiments will involve tasks more directly relevant to agriculture. For instance, using a plant disease image dataset where each client could have images of crops from different regions to see how well the testbed handles a more complex, higher-resolution image task. Attempts can also be made to try a time-series prediction task, such as forecasting soil moisture or temperature, where each client has a segment of a sensor time-series. This would test the system's ability to train recurrent or sequence models in a federated way. Along with new tasks, more complex ML models with deeper neural networks or specialised architectures like convolutional LSTMs for spatiotemporal data will be introduced to push the limits of the edge devices. This will likely necessitate optimisation techniques to fit models on the hardware, providing a research avenue on how model optimisation and FL intersect for edge deployment. By expanding the scope of applications, we ensure the testbed's findings are robust across different types of learning problems and we can uncover any domain-specific challenges, for example, federated learning might behave differently for highly imbalanced classification vs. regression tasks.

4. **Incorporating Advanced FL Techniques:** The testbed can serve as a platform to implement and evaluate advanced federated learning techniques that address some of the challenges noted. Future work can include integrating algorithms like Federated Optimisations (FedProx, FedNova) that aim to handle data heterogeneity better, and personalised FL approaches that create slight variations of the global model for different clients which might be useful if farms in different climates need slightly tailored models. Plan can also be made to experiment with communication reduction techniques such as gradient compression, update sparsification, or periodic averaging (skipping rounds) to quantify how much they help in a real network scenario. On the security and privacy front, adding modules for secure aggregation where clients' updates are encrypted and aggregated such that the server cannot see individual contributions or differential privacy adding noise to updates to protect individual data points is an important future step. While these will introduce overhead, the testbed will allow measurement of their cost on accuracy and performance in practice. By incorporating and testing these advanced techniques, the testbed will evolve into a comprehensive tool for evaluating not just vanilla FL, but the full arsenal of methods needed to make FL robust, efficient, and secure in deployment.

5. **Field Deployment and Long-Term Trials:** Ultimately, the goal is to transition from a lab testbed to a field deployment. A prospective future project is to deploy a small fleet of federated learning-enabled devices on an actual farm or across a few farms. This real deployment would use the testbed design as a blueprint but operate under true field conditions for an extended period, for example, an entire growing season. During such a trial, we could assess long-term aspects like model performance over time and whether it plateaus or drifts, device reliability as in how often nodes fail or need maintenance, and user interactions in terms of how farmers use and benefit from the predictions or models. Gathering qualitative feedback from users on the system's usefulness and any barriers to adoption is equally vital. This kind of pilot implementation is the natural next step to validate the research in the wild and would likely generate new research questions, such as how to integrate federated learning outputs with farm management systems or how to maintain models year over year. The insights from a field trial would feed back into refining the testbed and algorithms, in an iterative cycle.

7.6 Concluding Remarks

In conclusion, this thesis has taken an initial but significant step toward bringing federated learning to smart agriculture by creating a tangible testbed and demonstrating its operation. By addressing the outlined future directions, incorporating real sensor data, scaling up, diversifying applications, enhancing methods, and eventually deploying in the field, the aim is to progressively eliminate the barriers that keep FL as largely a theoretical fantasy rather than field-ready practice. Each step in this future work will contribute to transforming FL's promise into reality, allowing AI models to be trained collaboratively on the farm, by the farm, and for the farm. Through continued research and development along these lines, we anticipate a future where federated learning becomes a standard tool in the smart agriculture toolkit, driving innovations in how farming data is utilised for the betterment of agriculture productivity, all while respecting the autonomy and privacy of individual farmers.

References

[1] UN Population Division, "Global agriculture towards 2050: population growth," 2009. Available: `https://www.fao.org/fileadmin/templates/wsfs/docs/Issues_papers/HLEF2050_Global_Agriculture.pdf`

[2] Y. Liu, X. Ma, L. Shu, G. P. Hancke, and A. M. Abu-Mahfouz, "From Industry 4.0 to agriculture 4.0: current status, enabling technologies, and research challenges," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 4322–4334, Jun. 2020, doi: 10.1109/tii.2020.3003910.

[3] M. A. Arıcıoğlu, A. Yılmaz, and N. Gülnar, "4.0 for agriculture," *European Journal of Business Management and Research*, vol. 5, no. 3, Jun. 2020, doi: 10.24018/ejbmr.2020.5.3.364.

[4] R. F. Maia, I. Netto, and A. L. H. Tran, "Precision agriculture using remote monitoring systems in Brazil," *2017 IEEE Global Humanitarian Technology Conference (GHTC)*, pp. 1–6, Oct. 2017, doi: 10.1109/ghtc.2017.8239290.

[5] R. T. Reas, D. L. Carcoza, and M. J. S. Hernandez, "Application of wireless sensor network for photosynthetically active radiation monitoring in coconut-cacao intercrop model with applied internet of things," *Innovative Technology and Management Journal*, vol. 2, Mar. 2020, doi: 10.70954/itmj.v2i1.77.

[6] A. Poonia, T. Garg, O. Mishra, E. Batra, and G. R, "Agriculture 4.0 - Integrated Smart Irrigation System," *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pp. 1–8, Jun. 2024, doi: 10.1109/icccnt61001.2024.10724377.

[7] Z. Kapetanovic, D. Vasisht, J. Won, R. Chandra, and M. Kimball, "Experiences deploying an always-on farm network," *GetMobile Mobile Computing and Communications*, vol. 21, no. 2, pp. 16–21, Aug. 2017, doi: 10.1145/3131214.3131220.

[8] J. Freeman, S. Park, and C. Middleton, "Technological literacy and interrupted internet access," *Information Communication & Society*, vol. 23, no. 13, pp. 1947–1964, Jun. 2019, doi: 10.1080/1369118x.2019.1623901.

[9] C. Feng, A. Arafa, Z. Chen, M. Zhao, T. Q. S. Quek, and H. H. Yang, "Towards Understanding Federated Learning over Unreliable Networks," *IEEE Transactions on Machine Learning in Communications and Networking*, p. 1, Jan. 2024, doi: 10.1109/tmlcn.2024.3511475.

[10] K. Le, N. Luong-Ha, M. Nguyen-Duc, D. Le-Phuoc, C. Do, and K.-S. Wong, "Exploring the Practicality of Federated Learning: A survey towards the Communication Perspective," *arXiv (Cornell University)*, May 2024, doi: 10.48550/arxiv.2405.20431.

[11] M. E. Sykuta, "Big data in agriculture: property rights, privacy and competition in AG data services," *The International Food and Agribusiness Management Review*, vol. 19, pp. 57–74, Jun. 2016, doi: 10.22004/ag.econ.240696.

[12] M. E. Sykuta, "Big data in agriculture: property rights, privacy and competition in AG data services," *The International Food and Agribusiness Management Review*, vol. 19, pp. 57–74, Jun. 2016, doi: 10.22004/ag.econ.240696.

[13] S. S. L. Chukkapalli, P. Ranade, S. Mittal, and A. Joshi, "A Privacy Preserving Anomaly Detection Framework for Cooperative Smart Farming Ecosystem," *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pp. 340–347, Dec. 2021, doi: 10.1109/tpsisa52974.2021.00037.

[14] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *arXiv (Cornell University)*, Jan. 2016, doi: 10.48550/arxiv.1602.05629.

[15] N. K. Ray, D. Puthal, and D. Ghai, "Federated learning," *IEEE Consumer Electronics Magazine*, vol. 10, no. 6, pp. 106–107, Jul. 2021, doi: 10.1109/mce.2021.3094778.

[16] P. Qi, D. Chiaro, A. Guzzo, M. Ianni, G. Fortino, and F. Piccialli, "Model aggregation techniques in federated learning: A comprehensive survey," *Future Generation Computer Systems*, vol. 150, pp. 272–293, Sep. 2023, doi: 10.1016/j.future.2023.09.008.

[17] N. Truong, K. Sun, S. Wang, F. Guitton, and Y. Guo, "Privacy preservation in federated learning: An insightful survey from the GDPR perspective," *Computers & Security*, vol. 110, p. 102402, Jul. 2021, doi: 10.1016/j.cose.2021.102402.

[18] Y. Chen, "Advancing federated learning by addressing data and system heterogeneity," *Proceedings of the AAAI Symposium Series*, vol. 3, no. 1, p. 294, May 2024, doi: 10.1609/aaaiss.v3i1.31214.

[19] H. Zhu, J. Xu, S. Liu, and Y. Jin, "Federated Learning on Non-IID Data: a survey," *arXiv (Cornell University)*, Jan. 2021, doi: 10.48550/arxiv.2106.06843.

[20] A. K. Gaur and J. A. Valan, "Impact of Federated Learning in Agriculture 4.0: Collaborative Analysis of Distributed Agricultural Data," *2024 2nd World Conference on Communication & Computing (WCONF)*, pp. 1–6, Jul. 2024, doi: 10.1109/wconf61366.2024.10692286.

[21] A. Blanco-Justicia, J. Domingo-Ferrer, S. Martínez, D. Sánchez, A. Flanagan, and K. E. Tan, "Achieving security and privacy in federated learning systems: Survey, research challenges and future directions," *Engineering Applications of Artificial Intelligence*, vol. 106, p. 104468, Sep. 2021, doi: 10.1016/j.engappai.2021.104468.

[22] X. Ma, H. Sun, R. Q. Hu, and Y. Qian, "A new implementation of federated learning for privacy and security enhancement," *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pp. 4885–4890, Dec. 2022, doi: 10.1109/globecom48099.2022.10001614.

[23] G. R. Russo, V. Cardellini, and F. Lo Presti, "Serverless Functions in the Cloud-Edge Continuum: Challenges and Opportunities," *2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 321–328, Mar. 2023, doi: 10.1109/pdp59025.2023.00056.

[24] R. Sahay, "Serverless computing," in *Apress eBooks*, 2020, pp. 433–496. doi: 10.1007/978-1-4842-6200-9_13.

[25] Z. Li *et al.*, "FUNCX: Federated Function as a Service for Science," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4948–4963, Sep. 2022, doi: 10.1109/tpds.2022.3208767.

[26] F. Edverton, "Digitalization in manufacturing and industry 4.0," *SSRN Electronic Journal*, Jan. 2024, doi: 10.2139/ssrn.4723314.

[27] C. Cimini, G. Pezzotta, R. Pinto, and S. Cavalieri, "Industry 4.0 Technologies Impacts in the manufacturing and Supply chain landscape: An Overview," in *Studies in computational intelligence*, 2018, pp. 109–120. doi: 10.1007/978-3-030-03003-2_8.

[28] D. Sinha and R. Roy, "Reviewing Cyber-Physical System as a part of smart factory in industry 4.0," *IEEE Engineering Management Review*, vol. 48, no. 2, pp. 103–117, May 2020, doi: 10.1109/emr.2020.2992606.

[29] J. J. Ferreira, J. M. Lopes, S. Gomes, and H. G. Rammal, "Industry 4.0 implementation: Environmental and social sustainability in manufacturing multinational enterprises," *Journal of Cleaner Production*, vol. 404, p. 136841, Mar. 2023, doi: 10.1016/j.jclepro.2023.136841.

[30] S. K. S. Durai and M. D. Shamili, "Smart farming using Machine Learning and Deep Learning techniques," *Decision Analytics Journal*, vol. 3, p. 100041, Apr. 2022, doi: 10.1016/j.dajour.2022.100041.

[31] M. J. Smith, "Getting value from artificial intelligence in agriculture," *Animal Production Science*, vol. 60, no. 1, p. 46, Nov. 2018, doi: 10.1071/an18522.

[32] A. Subeesh and C. R. Mehta, "Automation and digitization of agriculture using artificial intelligence and internet of things," *Artificial Intelligence in Agriculture*, vol. 5, pp. 278–291, Jan. 2021, doi: 10.1016/j.aiia.2021.11.004.

[33] A. Yousaf, V. Kayvanfar, A. Mazzoni, and A. Elomri, "Artificial intelligence-based decision support systems in smart agriculture: Bibliometric analysis for operational insights and future directions," *Frontiers in Sustainable Food Systems*, vol. 6, Jan. 2023, doi: 10.3389/fsufs.2022.1053921.

[34] J. D. Leaver, "Global food supply: a challenge for sustainable agriculture," *Nutrition Bulletin*, vol. 36, no. 4, pp. 416–421, Nov. 2011, doi: 10.1111/j.1467-3010.2011.01925.x.

[35] O. Bertoglio and S. Sehnem, "Industry 4.0 in the context of Agribusiness: A Systematic Literature review," *Procedia Computer Science*, vol. 232, pp. 107–116, Jan. 2024, doi: 10.1016/j.procs.2024.01.011.

[36] J. Karas, R. Skýpalová, and P. Tomšík, "Human Capital in Agriculture: Barriers to Industry 4.0," *Journal of Interdisciplinary Research*, vol. 12, no. 2, 2020, doi: 10.33543/1202.

[37] T. Dibbern, L. A. S. Romani, and S. M. F. S. Massruhá, "Main drivers and barriers to the adoption of Digital Agriculture technologies," *Smart Agricultural Technology*, vol. 8, p. 100459, Apr. 2024, doi: 10.1016/j.atech.2024.100459.

[38] V. Bellon-Maurel, I. Piot-Lepetit, N. Lachia, and B. Tisseyre, "Digital agriculture in Europe and in France: which organisations can boost adoption levels?," *Crop and Pasture Science*, vol. 74, no. 6, pp. 573–585, Mar. 2023, doi: 10.1071/cp22065.

[39] S. J. Fielke, R. Garrard, E. Jakku, A. Fleming, L. Wiseman, and B. M. Taylor, "Conceptualising the DAIS: Implications of the 'Digitalisation of Agricultural Innovation Systems' on technology and policy at multiple levels," *NJAS - Wageningen Journal of Life Sciences*, vol. 90–91, no. 1, pp. 1–11, May 2019, doi: 10.1016/j.njas.2019.04.002.

[40] J. D. Van Der Ploeg, "A History of World Agriculture. From the Neolithic age to the current crisis - by Marcel Mazoyer and Laurence Roudart," *Journal of Agrarian Change*, vol. 11, no. 2, pp. 265–268, Mar. 2011, doi: 10.1111/j.1471-0366.2010.00302.x.

[41] P. Steenwyk, M. K. Heun, P. Brockway, T. Sousa, and S. Henriques, "The contributions of muscle and machine work to land and labor productivity in world agriculture since 1800," *Biophysical Economics and Sustainability*, vol. 7, no. 2, Mar. 2022, doi: 10.1007/s41247-022-00096-z.

[42] L. B. Sayre, "The pre-history of soil science: Jethro Tull, the invention of the seed drill, and the foundations of modern agriculture," *Physics and Chemistry of the Earth Parts a/B/C*, vol. 35, no. 15–18, pp. 851–859, Jan. 2010, doi: 10.1016/j.pce.2010.07.034.

[43] L. T. Evans, "Geopolitics and the Green Revolution: Wheat, genes, and the Cold War," *Crop Science*, vol. 38, no. 6, pp. 1712–1713, Nov. 1998, doi: 10.2135/cropsci1998.0011183x003800060053x.

[44] Y. Liu, X. Ma, L. Shu, G. P. Hancke, and A. M. Abu-Mahfouz, "From Industry 4.0 to agriculture 4.0: current status, enabling technologies, and research challenges," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 4322–4334, Jun. 2020, doi: 10.1109/tii.2020.3003910.

[45] N. Khan and M. A. Babar, "Innovations in precision agriculture and smart farming: Emerging technologies driving agricultural transformation," *Innovation and Emerging Technologies*, vol. 11, Jan. 2024, doi: 10.1142/s2737599424300046.

[46] D. Mulla and R. Khosla, "Historical Evolution and Recent Advances in Precision Farming," in Soil-Specific Farming, 1st Edition., 2015, pp. 1–36. doi: 10.1201/b18759-2.

[47] M. F. McCabe, R. Houborg, and A. Lucieer, "High-resolution sensing for precision agriculture: from Earth-observing satellites to unmanned aerial vehicles," *Proceedings of SPIE, the International Society for Optical Engineering/Proceedings of SPIE*, vol. 9998, p. 999811, Oct. 2016, doi: 10.1117/12.2241289.

[48] P. J. Zarco-Tejada, "A new era in remote sensing of crops with unmanned robots," *SPIE Newsroom*, Jan. 2008, doi: 10.1117/2.1200812.1438.

[49] E. Nakasone, M. Torero, and B. Minten, "The Power of Information: The ICT Revolution in Agricultural development," *Annual Review of Resource Economics*, vol. 6, no. 1, pp. 533–550, Jul. 2014, doi: 10.1146/annurev-resource-100913-012714.

[50] D. S. Paraforos and H. W. Griepentrog, "Digital farming and field robotics: internet of things, cloud computing, and big data," in *Agriculture automation and control*, 2021, pp. 365–385. doi: 10.1007/978-3-030-70400-1_14.

[51] N. Y. Kim, R. G. Evans, and W. M. Iversen, "Remote sensing and control of an irrigation system using a distributed wireless sensor network," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 7, pp. 1379–1387, May 2008, doi: 10.1109/tim.2008.917198.

[52] N. Sales, O. Remedios, and A. Arsenio, "Wireless sensor and actuator system for smart irrigation on the cloud," *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec. 2015, doi: 10.1109/wf-iot.2015.7389138.

[53] M. El-Hadi, "Overview of the IOT that Meeting Societal Challenges," *Compunet (the Egyptian Information Journal)*, vol. 28, no. 28, pp. 5–23, Jul. 2022, doi: 10.21608/jstc.2022.252017.

[54] S. Sonka, "Big Data: fueling the next evolution of agricultural innovation," *Journal of Innovation Management*, vol. 4, no. 1, pp. 114–136, May 2016, doi: 10.24840/2183-0606_004.001_0008.

[55] A. Holzinger, I. Fister, I. Fister, H.-P. Kaul, and S. Asseng, "Human-Centered AI in smart farming: Towards Agriculture 5.0," *IEEE Access*, vol. 12, pp. 62199–62214, Jan. 2024, doi: 10.1109/access.2024.3395532.

[56] J. M. Al-Khayri and T. Khan, "Revolutionizing agriculture: Exploring the potential of digital technologies and controlled environmental agriculture," *Sylwan*, Jan. 2024, doi: 10.59879/o1lor.

[57] A. K. Rai *et al.*, "Unlocking productivity potential: The promising role of agricultural robots in enhancing farming efficiency," *International Journal of Plant & Soil Science*, vol. 35, no. 18, pp. 624–633, Jul. 2023, doi: 10.9734/ijpss/2023/v35i183327.

[58] K. Grgic, D. Zagar, J. Balen, and J. Vlaovic, "Internet of Things in smart agriculture — Possibilities and challenges," *2022 International Conference on Smart Systems and Technologies (SST)*, pp. 239–244, Oct. 2020, doi: 10.1109/sst49455.2020.9264043.

[59] A. Castrignanò, G. Buttafuoco, R. Khosla, A. Mouazen, D. Moshou, and O. Naud, *Agricultural internet of things and decision support for precision smart farming*. 2020. doi: 10.1016/c2018-0-00051-1.

[60] J. P. Becoña, M. Grané, M. Miguez, and A. Arnaud, "LORA, SiGFOX, and NB-IoT: An Empirical Comparison for IoT LPWAN technologies in the Agribusiness," *IEEE Embedded Systems Letters*, vol. 16, no. 3, pp. 283–286, Apr. 2024, doi: 10.1109/les.2024.3394446.

[61] P. Sommer, Y. Maret, and D. Dzung, "Low-Power Wide-Area Networks for Industrial Sensing Applications," *2018 IEEE International Conference on Industrial Internet (ICII)*, pp. 23–32, Oct. 2018, doi: 10.1109/icii.2018.00011.

[62] M. Narasimharao, B. Swain, P. P. Nayak, and S. Bhuyan, "Cloud Based Automated Low Power Long Range Smart Farming Modular IoT Architecture," *2022 2nd Odisha International Conference on Electrical Power Engineering, Communication and Computing Technology (ODICON)*, pp. 1–5, Nov. 2022, doi: 10.1109/odicon54453.2022.10010231.

[63] M. Ayaz, M. Ammad-Uddin, Z. Sharif, A. Mansour, and E.-H. M. Aggoune, "Internet-of-Things (IoT)-Based Smart Agriculture: Toward making the fields talk," *IEEE Access*, vol. 7, pp. 129551–129583, Jan. 2019, doi: 10.1109/access.2019.2932609.

[64] M. Pincheira, F. Shamsfakhr, J. Hueller, and M. Vecchio, "Overcoming Limitations of IoT Installations: Active Sensing UGV for Agricultural Digital Twins," *2023 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*, pp. 319–324, Nov. 2023, doi: 10.1109/metroagrifor58484.2023.10424235.

[65] M. J. O'Grady, D. Langton, and G. M. P. O'Hare, "Edge computing: A tractable model for smart agriculture?," *Artificial Intelligence in Agriculture*, vol. 3, pp. 42–51, Sep. 2019, doi: 10.1016/j.aiia.2019.12.001.

[66] A. Kargar, M. P. Wilk, D. Zorbas, M. T. Gaffney, and B. Q'Flynn, "A Novel Resource-Constrained Insect Monitoring System based on Machine Vision with Edge AI," *2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS)*, Dec. 2022, doi: 10.1109/ipas55744.2022.10052895.

[67] N. I. Vatin, S. K. Joshi, P. Acharya, R. Sharma, and N. Rajasekhar, "Precision Agriculture and Sustainable Yields: Insights from IoT-Driven Farming and the Precision Agriculture Test," *BIO Web of Conferences*, vol. 86, p. 01091, Jan. 2024, doi: 10.1051/bioconf/20248601091.

[68] V. Chintamaneni, A. Janipalli, J. Rajaram, T. A. Devi, K. V. K. Vajjala, and V. Vivekanandhan, "Revolutionizing Farming: An Analysis of IoT-based Smart Agriculture Monitoring Systems," *2024 5th International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pp. 486–492, Aug. 2024, doi: 10.1109/icesc60852.2024.10689990.

[69] P. Patros *et al.*, "Rural AI: Serverless-Powered Federated Learning for Remote applications," *IEEE Internet Computing*, vol. 27, no. 2, pp. 28–34, Sep. 2022, doi: 10.1109/mic.2022.3202764.

[70] J. G. Pauloski *et al.*, "Accelerating Communications in Federated Applications with Transparent Object Proxies," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23)*, pp. 1–15, Oct. 2023, doi: 10.1145/3581784.3607047.

[71] N. Hudson *et al.*, "FLIGHT: A FAAS-Based Framework for complex and hierarchical federated learning," *arXiv (Cornell University)*, Sep. 2024, doi: 10.48550/arxiv.2409.16495.

[72] K. Bonawitz *et al.*, "Practical secure aggregation for Privacy-Preserving machine learning," *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, Oct. 2017, doi: 10.1145/3133956.3133982.

[73] J. Wu, F. Dong, H. Leung, Z. Zhu, J. Zhou, and S. Drew, "Topology-aware Federated Learning in Edge Computing: A Comprehensive survey," *ACM Computing Surveys*, vol. 56, no. 10, pp. 1–41, Apr. 2024, doi: 10.1145/3659205.

[74] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous Online Federated Learning for Edge Devices with Non-IID Data," *arXiv (Cornell University)*, Jan. 2019, doi: 10.48550/arxiv.1911.02134.

[75] C. Huang, J. Huang, and X. Liu, "Cross-Silo Federated Learning: Challenges and opportunities," *arXiv (Cornell University)*, Jan. 2022, doi: 10.48550/arxiv.2206.12949.

[76] K. R. Žalik and M. Žalik, "A review of Federated Learning in agriculture," *Sensors*, vol. 23, no. 23, p. 9566, Dec. 2023, doi: 10.3390/s23239566.

[77] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *arXiv (Cornell University)*, Jan. 2018, doi: 10.48550/arxiv.1812.06127.

[78] M. Moshawrab, M. Adda, A. Bouzouane, H. Ibrahim, and A. Raad, "Reviewing federated learning aggregation algorithms; strategies, contributions, limitations and future perspectives," *Electronics*, vol. 12, no. 10, p. 2287, May 2023, doi: 10.3390/electronics12102287.

[79] H. Devaraj *et al.*, "Ruralai in tomato farming: Integrated sensor system, distributed computing, and Hierarchical Federated Learning for Crop Health Monitoring," *IEEE Sensors Letters*, vol. 8, no. 5, pp. 1–4, May 2024. doi:10.1109/lsens.2024.3384935

[80] A. Hard *et al.*, "Federated Learning for Mobile keyboard prediction," *arXiv (Cornell University)*, Jan. 2018, doi: 10.48550/arxiv.1811.03604.

[81] A. Dwarampudi and M. K. Yogi, "Application of federated Learning for smart Agriculture System," *International Journal of Information Technology and Computer Engineering*, no. 43, pp. 36–47, Apr. 2024, doi: 10.55529/ijitc.43.36.48.

[82] L. Cui, X. Su, Y. Zhou, and Y. Pan, "Slashing communication traffic in federated learning by transmitting clustered model updates," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2572–2589, Jun. 2021, doi: 10.1109/jsac.2021.3087262.

[83] R. E. Mokadem, Y. B. Maissa, and Z. E. Akkaoui, "Federated Learning Communications optimization using sparse Single-Layer updates," *Procedia Computer Science*, vol. 236, pp. 168–176, Jan. 2024, doi: 10.1016/j.procs.2024.05.018.

[84] V. Huang *et al.*, "Keep It Simple: Fault Tolerance Evaluation of Federated Learning with Unreliable Clients," *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, pp. 1–3, Jul. 2023, doi: 10.1109/cloud60044.2023.00024.

[85] M. J. Sheller *et al.*, "Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data," *Scientific Reports*, vol. 10, no. 1, Jul. 2020, doi: 10.1038/s41598-020-69250-1.

[86] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv (Cornell University)*, Jan. 2018, doi: 10.48550/arxiv.1812.00564.

[87] H. R. Roth *et al.*, "Federated Learning for Breast Density Classification: a Real-World Implementation," in *Lecture notes in computer science*, 2020, pp. 181–191. doi: 10.1007/978-3-030-60548-3_18.

[88] S. M. Attya *et al.*, "Harnessing Federated Learning for Secure Data Sharing in Healthcare Systems," *2024 36th Conference of Open Innovations Association (FRUCT)*, pp. 390–399, Oct. 2024, doi: 10.23919/fruct64283.2024.10749928.

[89] K. Sozinov, V. Vlassov, and S. Girdzijauskas, "Human Activity Recognition Using Federated Learning," *2018 IEEE Intl Conf on Parallel & Distributed Processing With Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/-SustainCom)*, pp. 1103–1111, Dec. 2018, doi: 10.1109/bdcloud.2018.00164.

[90] H. K. Kondaveeti, G. B. Sai, S. A. Athar, V. K. Vatsavayi, A. Mitra, and P. Ananthachari, "Federated Learning for Smart Agriculture: Challenges and Opportunities," *2024 Third International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)*, pp. 1–7, Apr. 2024, doi: 10.1109/icdcece60827.2024.10548604.

[91] N. Singh and M. Adhikari, "Edge-centric collaborative federated learning for irrigation management of paddy fields using agriculture sensor data processing," *IEEE Sensors Letters*, vol. 8, no. 8, pp. 1–4, Aug. 2024. doi:10.1109/lsens.2024.3422417

[92] S. Bolettieri and R. Bruno, "Edge-Assisted Resource Management for Data-Centric IoT Applications in Shared Sensor Networks," *2020 IEEE 21st International Symposium on "a World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pp. 137–146, Aug. 2020, doi: 10.1109/wowmom49955.2020.00034.

[93] M. S. Islam, S. Javaherian, F. Xu, X. Yuan, L. Chen, and N.-F. Tzeng, "FedClust: Optimizing Federated Learning on Non-IID Data through Weight-Driven Client Clustering," *arXiv (Cornell University)*, Mar. 2024, doi: 10.48550/arxiv.2403.04144.

[94] X. Ma, J. Zhu, Z. Lin, S. Chen, and Y. Qin, "A state-of-the-art survey on solving non-IID data in Federated Learning," *Future Generation Computer Systems*, vol. 135, pp. 244–258, May 2022, doi: 10.1016/j.future.2022.05.003.

[95] P. Kairouz *et al.*, "Advances and open problems in federated learning," *arXiv.org*, Dec. 10, 2019. https://arxiv.org/abs/1912.04977

[96] J. Kim, S. Chang, and N. Kwak, "PQK: model compression via pruning, quantization, and knowledge distillation," *arXiv.org*, Jun. 25, 2021. https://arxiv.org/abs/2106.14681

[97] J. Su, X. Wang, X. Chen, and R.-G. Cheng, "Joint Sparsification and Quantization for Wireless Federated Learning under Communication Constraints," *2023 IEEE 24th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 401–405, Sep. 2023, doi: 10.1109/spawc53906.2023.10304559.

[98] Y. Oh, N. Lee, Y.-S. Jeon, and H. V. Poor, "Communication-Efficient federated learning via quantized compressed sensing," *arXiv (Cornell University)*, Jan. 2021, doi: 10.48550/arxiv.2111.15071.

[99] P. Prakash, J. Ding, M. Shu, J. Wang, W. Xu and M. Pan, "SQuaFL: Sketch-Quantization Inspired Communication Efficient Federated Learning," *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, San Jose, CA, USA, 2021, pp. 350-354, doi: 10.1145/3453142.3491415.

[100] S. Kim, J. Woo, D. Seo, and Y. Kim, "Communication-Efficient and Drift-Robust federated learning via Elastic Net," *arXiv (Cornell University)*, Jan. 2022, doi: 10.48550/arxiv.2210.02940.

[101] A. -m. Kermarrec, L. Massoulie, and A. J. Ganesh, "Probabilistic reliable dissemination in large-scale systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 3, pp. 248–258, Mar. 2003, doi: 10.1109/tpds.2003.1189583.

[102] M. Naseri, University College London, J. Hayes, DeepMind, E. De Cristofaro, and UCL & Alan Turing Institute, "Local and central differential privacy for robustness and privacy in federated learning," journal-article, 2022. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2022-54-paper.pdf

[103] K. Jung, S. Biswas, and C. Palamidessi, "Mitigating membership inference vulnerability in personalized federated learning," *arXiv.org*, Mar. 12, 2025. https://arxiv.org/abs/2503.09414

[104] S. Vlaski, C. Schroth, M. Muma, and A. M. Zoubir, "Robust and efficient aggregation for distributed learning," *arXiv (Cornell University)*, Jan. 2022, doi: 10.48550/arxiv.2204.00586.

[105] I. Kholod *et al.*, "Open-Source Federated Learning Frameworks for IoT: A Comparative Review and analysis," *Sensors*, vol. 21, no. 1, p. 167, Dec. 2020, doi: 10.3390/s21010167.

[106] A. Ziller *et al.*, "PySyft: a library for easy federated learning," in *Studies in computational intelligence*, 2021, pp. 111–139. doi: 10.1007/978-3-030-70604-3_5.

[107] D. J. Beutel *et al.*, "Flower: a friendly federated Learning research framework," *arXiv (Cornell University)*, Jan. 2020, doi: 10.48550/arxiv.2007.14390.

[108] B. Et-Taibi, M. R. Abid, E.-M. Boufounas, and D. Benhaddou, "Cloud and Edge based Smart Agriculture: A Real-World Deployment," *2024 International Conference on Circuit, Systems and Communication (ICCSC)*, pp. 1–6, Jun. 2024, doi: 10.1109/iccsc62074.2024.10617136.

[109] A. Triantafyllou, D. C. Tsouros, P. Sarigiannidis, and S. Bibi, "An Architecture model for Smart Farming," *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, May 2019, doi: 10.1109/dcoss.2019.00081.

[110] F. Borelli, G. Biondi, F. Horita, and C. Kamienski, "Architectural software patterns for the development of IoT smart applications," *arXiv.org*, Mar. 10, 2020. https://arxiv.org/abs/2003.04781

[111] C. Seo *et al.*, "Exploring the Role of Software Architecture in Dynamic and Fault Tolerant Pervasive Systems," *First International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments (SEPCASE '07)*, p. 9, May 2007, doi: 10.1109/sepcase.2007.6.

[112] J. Baur, J. Pfaff, H. Ulbrich, and T. Villgrattner, "Design and development of a redundant modular multipurpose agricultural manipulator," 2022 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), pp. 823–830, Jul. 2012, doi: 10.1109/aim.2012.6265928.

[113] N. B. Nagamalla, "Architecting Reliable data Systems for Smart agriculture: A Big Data and SRE perspective," *International Journal of Scientific Research in Computer Science Engineering and Information Technology*, vol. 11, no. 1, pp. 556–563, Jan. 2025, doi: 10.32628/cseit25111253.

[114] P. S. Khatoon and M. Ahmed, "Semantic Interoperability for IoT Agriculture Framework with Heterogeneous Devices," in *Advances in intelligent systems and computing*, 2020, pp. 385–395. doi: 10.1007/978-981-15-7234-0_34.

[115] R. R. D. Pereira *et al.*, "ISO 11783 Standard: Procedures for Serial Data Communication between the Implement ECU with the Task Controller," Sep. 2009.

[116] L. A. S. Romani *et al.*, "AgroAPI platform: An initiative to support digital solutions for agribusiness ecosystems," *Smart Agricultural Technology*, vol. 5, p. 100247, May 2023, doi: 10.1016/j.atech.2023.100247.

[117] M. Sine, H. Theo-Paul, and E. Emeric, "API - AGRO: An Open Data and Open API platform to promote interoperability standards for Farm Services and Ag Web Applications," *Journal of Agricultural Informatics*, vol. 6, no. 4, Oct. 2015, doi: 10.17700/jai.2015.6.4.209.

[118] J. M. K. Sri, V. G. Narendra, and V. Pai, "Implementing and testing of internet of things (IoT) technology in agriculture And compare the application layer protocols: message Queuing Telemetry Transport (MQTT) and Hyper Text Transport Protocol (HTTP)," in *Communications in computer and information science*, 2019, pp. 320–333. doi: 10.1007/978-981-15-0111-1_29.

[119] M. Treiber and H. Bernhardt, "NEVONEX—The Importance of Middleware and Interfaces for the Digital Transformation of Agriculture," *Engineering Proceedings*, p. 3, Nov. 2021, doi: 10.3390/engproc2021009003.

[120] A. J. Romera, M. Sharifi, and S. Charters, "Digitalization in agriculture. Towards an integrative approach," *Computers and Electronics in Agriculture*, vol. 219, p. 108817, Mar. 2024, doi: 10.1016/j.compag.2024.108817.

[121] O. Debauche, S. Mahmoudi, P. Manneback, and F. Lebeau, "Cloud and distributed architectures for data management in agriculture 4.0: Review and future trends," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 9, pp. 7494–7514, Oct. 2021, doi: 10.1016/j.jksuci.2021.09.015.

[122] S. E. Jaouhari and E. Bouvet, "Secure firmware Over-The-Air updates for IoT: Survey, challenges, and discussions," *Internet of Things*, vol. 18, p. 100508, Feb. 2022, doi: 10.1016/j.iot.2022.100508.

[123] A. Chava, "CI/CD and automation in DevOps engineering," *Asian Journal of Research in Computer Science*, vol. 17, no. 11, pp. 73–80, Nov. 2024, doi: 10.9734/ajrcos/2024/v17i11520.

[124] N. Peladarinos, D. Piromalis, V. Cheimaras, E. Tserepas, R. A. Munteanu, and P. Papageorgas, "Enhancing smart Agriculture by Implementing Digital Twins: A Comprehensive review," *Sensors*, vol. 23, no. 16, p. 7128, Aug. 2023, doi: 10.3390/s23167128.

[125] L. Bollini, A. Caccamo, and C. Martino, "Interfaces of the Agriculture 4.0," *15th International Conference on Web Information Systems and Technologies*, pp. 273–280, Jan. 2019, doi: 10.5220/0008164800002366.

[126] S. E. Brennan and Z. Munn, "Prisma 2020: A reporting guideline for the next generation of Systematic Reviews," *JBI Evidence Synthesis*, vol. 19, no. 5, pp. 906–908, May 2021. doi:10.11124/jbies-21-00112

[127] A. T. Balafoutis, F. K. Van Evert, and S. Fountas, "Smart Farming Technology trends: economic and environmental effects, labor impact, and adoption readiness," Agronomy, vol. 10, no. 5, p. 743, May 2020, doi: 10.3390/agronomy10050743.

[128] J. Mankins and Artemis Innovation Management Solutions LLC, "Technology Readiness Level – a white paper," Technology Readiness Levels, Apr. 1995.

[129] L. H. Talero-Sarmiento, D. T. Parra-Sanchez, and H. L. Diaz, "Opportunities and Barriers of Smart Farming adoption by farmers based on a systematic literature review," Proceedings INNODOCT/19. International Conference on Innovation, Documentation and Education, pp. 53–64, Apr. 2023, doi: 10.4995/inn2022.2022.15746.

[130] R. Dembani, I. Karvelas, N. A. Akbar, S. Rizou, D. Tegolo, and S. Fountas, "Agricultural data privacy and federated learning: A review of challenges and opportunities," Computers and Electronics in Agriculture, vol. 232, p. 110048, Feb. 2025, doi: 10.1016/j.compag.2025.110048.

[131] C. S. Sullivan, M. Gemtou, E. Anastasiou, and S. Fountas, "Building trust: A systematic review of the drivers and barriers of agricultural data sharing," Smart Agricultural Technology, vol. 8, p. 100477, May 2024, doi: 10.1016/j.atech.2024.100477.

[132] M. Aggarwal, V. Khullar, N. Goyal, and T. A. Prola, "Resource-efficient federated learning over Ioat for Rice Leaf Disease Classification," *Computers and Electronics in Agriculture*, vol. 221, p. 109001, Jun. 2024. doi:10.1016/j.compag.2024.109001

[133] S. Choubey and Divya, "Lightweight Federated Transfer Learning for plant leaf disease detection and classification across multiclient cross-silo datasets," *BIO Web of Conferences*, vol. 82, p. 05018, 2024. doi:10.1051/bioconf/20248205018

[134] Md. Fahim-Ul-Islam *et al.*, "A comprehensive approach toward wheat leaf disease identification leveraging transformer models and Federated Learning," *IEEE Access*, vol. 12, pp. 109128–109156, 2024. doi:10.1109/access.2024.3438544

[135] K. Thonglek and A. Maipradit, "Hierarchical Federated Learning for Predicting Water Levels: A case study in Thailand," *2024 16th International Conference on Computer and Automation Engineering (ICCAE)*, pp. 218–222, Mar. 2024. doi:10.1109/iccae59995.2024.10569296

[136] D. Mamba Kabala, A. Hafiane, L. Bobelin, and R. Canals, "Image-based crop disease detection with Federated Learning," *Scientific Reports*, vol. 13, no. 1, Nov. 2023. doi:10.1038/s41598-023-46218-5

[137] M. Aggarwal *et al.*, "Federated Transfer Learning for Rice-leaf disease classification across multiclient cross-silo datasets," *Agronomy*, vol. 13, no. 10, p. 2483, Sep. 2023. doi:10.3390/agronomy13102483

[138] L. Praharaj, M. Gupta, and D. Gupta, "Hierarchical Federated Transfer Learning and Digital Twin Enhanced Secure Cooperative Smart Farming," *2023 IEEE International Conference on Big Data (BigData)*, pp. 3304–3313, Dec. 2023. doi:10.1109/bigdata59044.2023.10386345

[139] V. Sharma *et al.*, "Weedgan: A novel generative adversarial network for cotton weed identification," *The Visual Computer*, vol. 39, no. 12, pp. 6503–6519, Dec. 2022. doi:10.1007/s00371-022-02742-5

[140] K. Thonglek and P. Rattanatamrong, "Decentralized Federated Learning for agricultural plant diseases identification on edge devices," *2023 18th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP)*, pp. 1–6, Nov. 2023. doi:10.1109/isai-nlp60301.2023.10354694

[141] A. Kaushal *et al.*, "Shield: A secure heuristic integrated environment for load distribution in rural-ai," *Future Generation Computer Systems*, vol. 161, pp. 286–301, Dec. 2024. doi:10.1016/j.future.2024.07.026

[142] M. Pincheira, F. Shamsfakhr, J. Hueller, and M. Vecchio, "Overcoming limitations of IOT installations: Active Sensing UGV for Agricultural Digital Twins," *2023 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*, pp. 319–324, Nov. 2023. doi:10.1109/metroagrifor58484.2023.10424235

[143] A. Garcia-Perez, R. Miñón, A. I. Torre-Bastida, and E. Zulueta-Guerrero, "Analysing edge computing devices for the deployment of embedded AI," Sensors, vol. 23, no. 23, p. 9495, Nov. 2023, doi: 10.3390/s23239495.

[144] O. Jouini, K. Sethom, A. Namoun, N. Aljohani, M. H. Alanazi, and M. N. Alanazi, "A survey of Machine learning in edge Computing: techniques, frameworks, applications, issues, and research directions," *Technologies*, vol. 12, no. 6, p. 81, Jun. 2024, doi: 10.3390/technologies12060081.

[145] B. Shubyn *et al.*, "Resource Consumption of Federated Learning Approach Applied on Edge IoT Devices in the AGV Environment," in *Lecture notes in computer science*, 2023, pp. 492–504. doi: 10.1007/978-3-031-36030-5_39.

[146] S. K. Lo, Q. Lu, C. Wang, H.-Y. Paik, and L. Zhu, "A Systematic Literature Review on Federated Machine Learning," ACM Computing Surveys, vol. 54, no. 5, pp. 1–39, May 2021, doi: 10.1145/3450288.

[147] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," arXiv (Cornell University), Jan. 2017, doi: 10.48550/arxiv.1708.07747.

[148] T. Liu, Delft University of Technology, M. Yang, Delft University of Technology, Q. Wang, and Delft University of Technology, "FEDREG: Recouping the global model in personalized federated learning," journal-article, 2021.

[149] F. Lai, Y. Dai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "FedScale: Benchmarking model and system performance of federated Learning at scale," *arXiv (Cornell University)*, Jan. 2021, doi: 10.48550/arxiv.2105.11367.

[150] H. Woisetschläger, A. Erben, R. Mayer, S. Wang, and H.-A. Jacobsen, "FLEdge: Benchmarking Federated Learning Applications in Edge Computing Systems," *Proceedings of the 25th International Middleware Conference (Middleware '24)*, pp. 88–102, Nov. 2024, doi: 10.1145/3652892.3700751.

[151] B. Li, N. Su, C. Ying, and F. Wang, "Plato: An Open-Source Research Framework for Production Federated Learning," *Proceedings of the ACM Turing Award Celebration Conference - China 2023 (ACM TURC '23)*, pp. 1–2, Jul. 2023, doi: 10.1145/3603165.3607364.

[152] E. Dritsas and M. Trigka, "Federated Learning for IoT: A survey of techniques, challenges, and applications," *Journal of Sensor and Actuator Networks*, vol. 14, no. 1, p. 9, Jan. 2025, doi: 10.3390/jsan14010009.

[153] S. Caldas *et al.*, "LEAF: a benchmark for federated settings," *arXiv (Cornell University)*, Jan. 2018, doi: 10.48550/arxiv.1812.01097.

[154] T. M. Antico, L. F. R. Moreira, and R. Moreira, "Evaluating the Potential of Federated Learning for Maize Leaf Disease Prediction," *Anais Do XIX Encontro Nacional De Inteligência Artificial E Computacional*, Nov. 2022, doi: 10.5753/eniac.2022.227293.

[155] J. Božič, A. R. Faustino, B. Radovič, M. Canini, and V. Pejović, "Where is the Testbed for My Federated Learning Research?," *2024 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 249–264, Dec. 2024, doi: 10.1109/sec62691.2024.00027.

[156] Y. Wang, F. Zobiri, M. A. Mustafa, J. Nightingale, and G. Deconinck, "Consumption prediction with privacy concern: Application and evaluation of Federated Learning," *Sustainable Energy Grids and Networks*, vol. 38, p. 101248, Dec. 2023, doi: 10.1016/j.segan.2023.101248.

Appendices

A Complete Results Table

A.1 Simulation Results Table

Number of workers	% of dataset	Rounds	Completion Time (min)	Test accuracy	Train accuracy	CPU usage	Data Transferred (MB)
1	50%	1	1.66	80.16%	81.09%	13.7%	0.34
1	50%	3	3.62	84.25%	85.43%	16.5%	1.02
1	50%	5	5.44	87.45%	89.08%	16.1%	1.69
1	50%	10	10.13	88.88%	91.71%	15.4%	3.39
1	100%	1	1.90	83.98%	85.08%	19.8%	0.34
1	100%	3	4.92	87.85%	89.7%	15.2%	1.02
1	100%	5	8.86	89.01%	91.63%	14.2%	1.69
1	100%	10	17.28	89.6%	94.46%	15.3%	3.39
2	50%	1	1.20	71.18%	71.83%	20.3%	0.68
2	50%	3	2.91	82.57%	83.68%	21.4%	2.03
2	50%	5	4.77	85.32%	86.32%	20.3%	3.39
2	50%	10	8.06	86.78%	88.43%	24.8%	6.78
2	100%	1	1.89	79.09%	80.24%	16.8%	0.68
2	100%	3	4.50	87.11%	87.78%	15.8%	2.03
2	100%	5	7.14	88.05%	89.71%	15.6%	3.39
2	100%	10	12.67	86.69%	88.96%	17.1%	6.78
3	50%	1	1.25	72.21%	73.71%	22.9%	1.02
3	50%	3	2.81	82.84%	83.62%	26.5%	3.05
3	50%	5	4.61	85.66%	86.78%	27.2%	5.08
3	50%	10	9.35	87.32%	88.65%	26.6%	10.17
3	100%	1	1.71	73.1%	74.46%	21.7%	1.02
3	100%	3	4.98	84.75%	85.91%	16.4%	3.05
3	100%	5	7.47	86.77%	88.53%	18.4%	5.08
3	100%	10	13.61	88%	90.1%	20%	10.17
4	50%	1	1.12	67.34%	67.5%	24.1%	1.36
4	50%	3	3.11	80.07%	81.59%	30.2%	4.07
4	50%	5	5.32	82.98%	83.74%	29.8%	6.78
4	50%	10	9.84	86.13%	87.22%	34.6%	13.56
4	100%	1	2.02	71.67%	72.22%	22.6%	1.36
4	100%	3	4.86	84.91%	86.28%	23.2%	4.07
4	100%	5	7.52	86.2%	87.47%	23.9%	6.78
4	100%	10	13.65	87.98%	89.45%	25.1%	13.56

A.2 Complete Testbed Results Table

Number of workers	% of dataset	Rounds	Completion Time (min)	Test accuracy	Train accuracy	CPU usage	Data Transferred (MB)
1	50%	1	7.67	81.65%	82.37%	18.8%	0.34
1	50%	3	21.36	85.06%	86.03%	18.4%	1.02
1	50%	5	34.74	87.46%	88.93%	18.6%	1.69
1	50%	10	69.04	88.53%	91.33%	18.3%	3.39
1	100%	1	13.78	85.19%	85.88%	16.6%	0.34
1	100%	3	36.04	87.82%	90.19%	14.4%	1.02
1	100%	5	58.43	89.31%	92.16%	10.7%	1.69
1	100%	10	109.46	89.38%	93.88%	12%	3.39
2	50%	1	8.57	74.03%	75%	19.2%	0.68
2	50%	3	23.75	81.98%	82.94%	18.6%	2.03
2	50%	5	35.03	84.88%	85.98%	19.8%	3.39
2	50%	10	60.21	86.87%	88.71%	22.5%	6.78
2	100%	1	10.01	78.66%	79.43%	16.6%	0.68
2	100%	3	27.07	85.43%	86.37%	16.4%	2.03
2	100%	5	46.03	87.68%	89.8%	15.5%	3.39
2	100%	10	101.80	89.28%	93.15%	14.3%	6.78
3	50%	1	8.01	72.74%	73.77%	21.1%	1.02
3	50%	3	19.97	83.07%	83.9%	22.8%	3.05
3	50%	5	27.48	83.69%	84.5%	27.2%	5.08
3	50%	10	54.93	86.39%	87.31%	26.8%	10.17
3	100%	1	10.64	76.47%	77.75%	16.9%	1.02
3	100%	3	27.86	86.07%	87.03%	17%	3.05
3	100%	5	39.17	85.87%	87.14%	19.2%	5.08
3	100%	10	81.46	88.03%	89.88%	18.3%	10.17
4	50%	1	6.08	69.63%	69.98%	28.3%	1.36
4	50%	3	16.67	74.74%	75.9%	28.3%	4.07
4	50%	5	28.77	82.01%	82.99%	27.3%	6.78
4	50%	10	59.41	87.4%	88.72%	25.8%	13.56
4	100%	1	10.19	73.43%	74.68%	18.2%	1.36
4	100%	3	22.92	83.78%	84.82%	22.5%	4.07
4	100%	5	39.78	86.77%	88.03%	20.1%	6.78
4	100%	10	75.85	87.29%	88.69%	21%	13.56

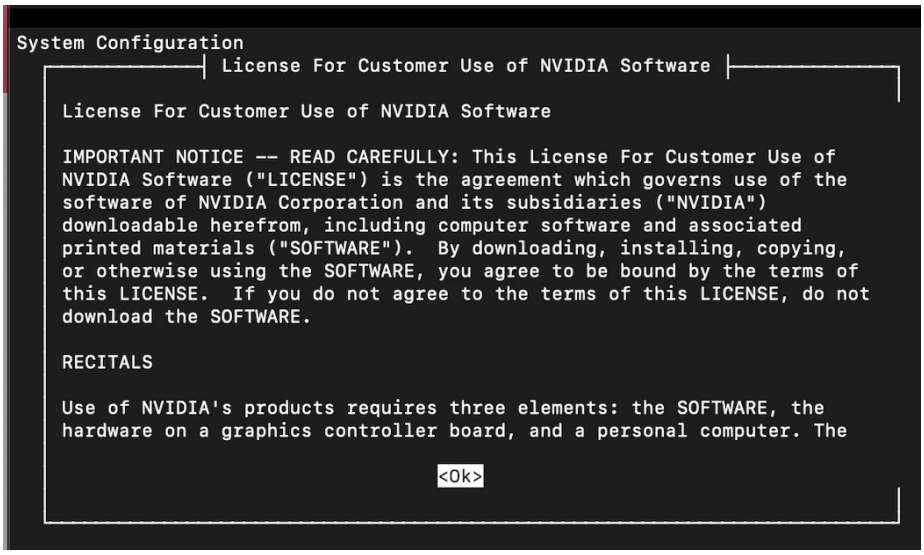
B Testbed Setup: Wireless Setup and SSH Access for Jetson Nano

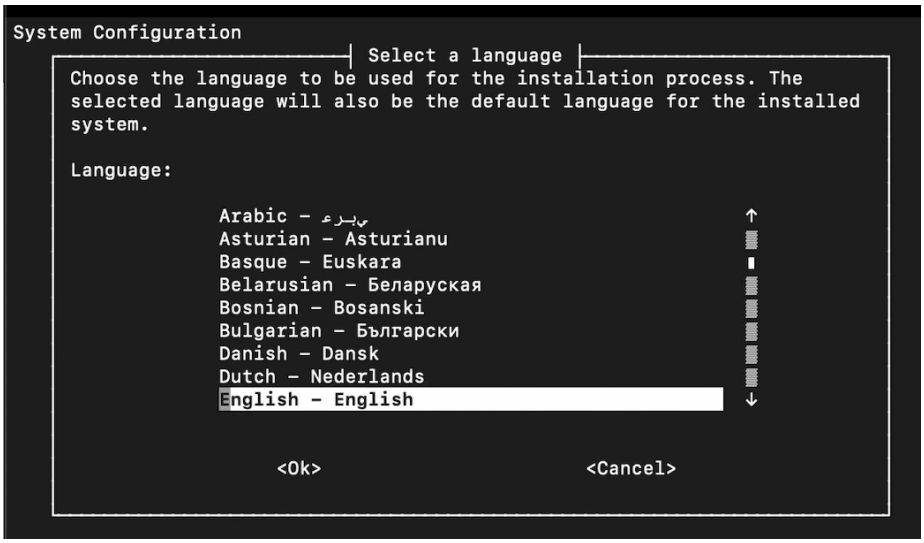
The goal is to set up the Jetson Nano wirelessly and enable SSH access. Below are the detailed steps to achieve this:

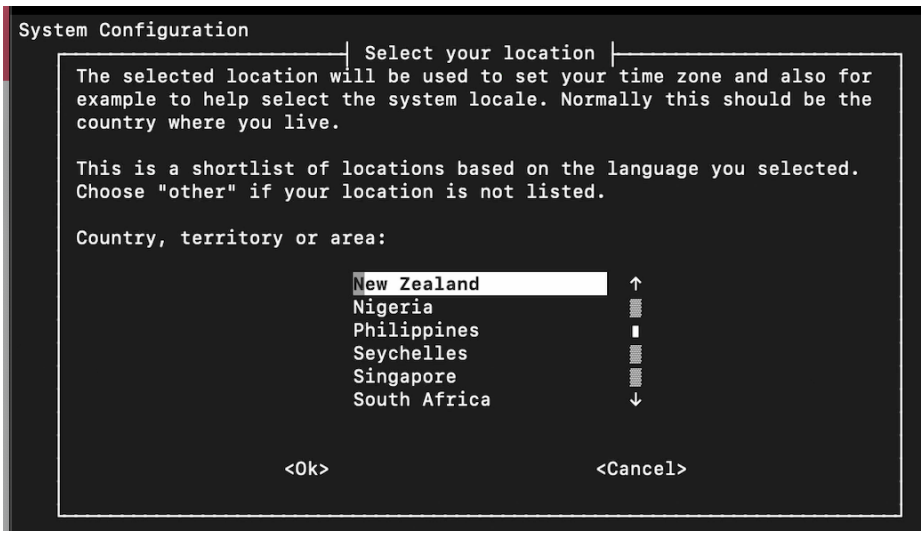
Step 1 : **Connect to the Jetson Nano via USB Serial Console**

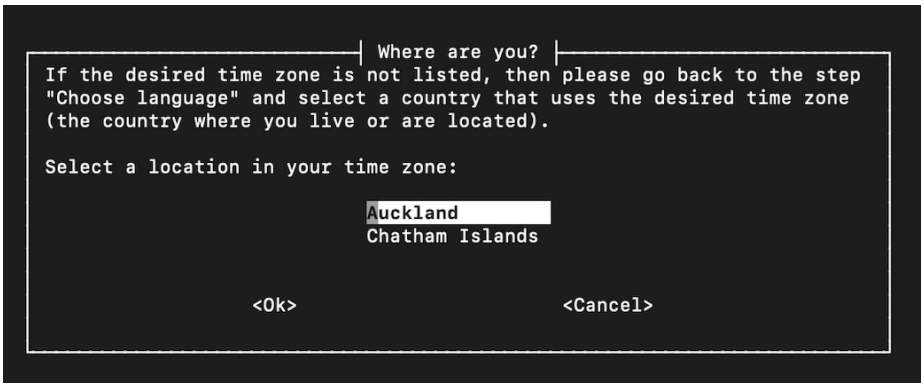
1. Connect the Jetson Nano to your laptop using a USB cable
2. Open a terminal on your laptop and identify the USB serial port (e.g. `/dev/cu.usbmodemXXXX` on macOS or `/dev/ttyUSB0` on Linux)

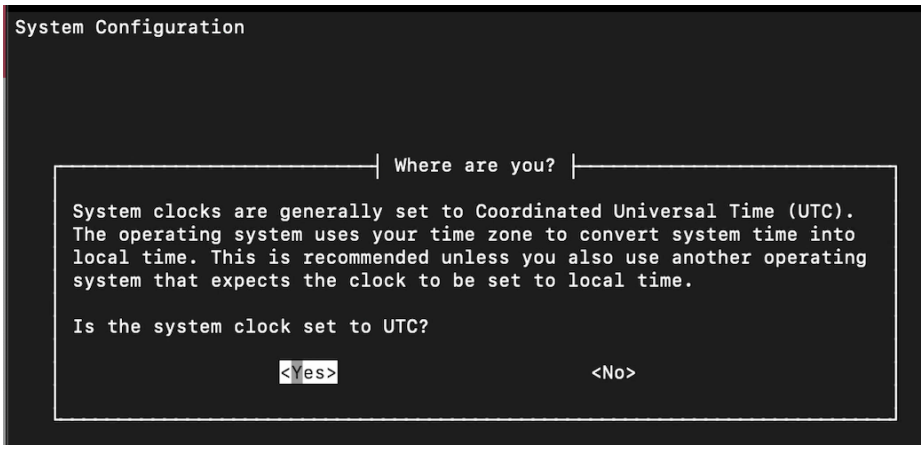
3. Follow the initial setup prompts and Configure the following settings:
   - Language : Choose your preferred language.
   - Location : Select your country/region.
   - Time Zone : Set the correct time zone.
   - User Account : Create a new user account representing your machine (e.g. username: w3 (for worker 3)).
   - Network Interface : Select wlan0 as the primary network interface.
   - Wireless Network : Connect to the Wi-Fi network using the provided password available to you.

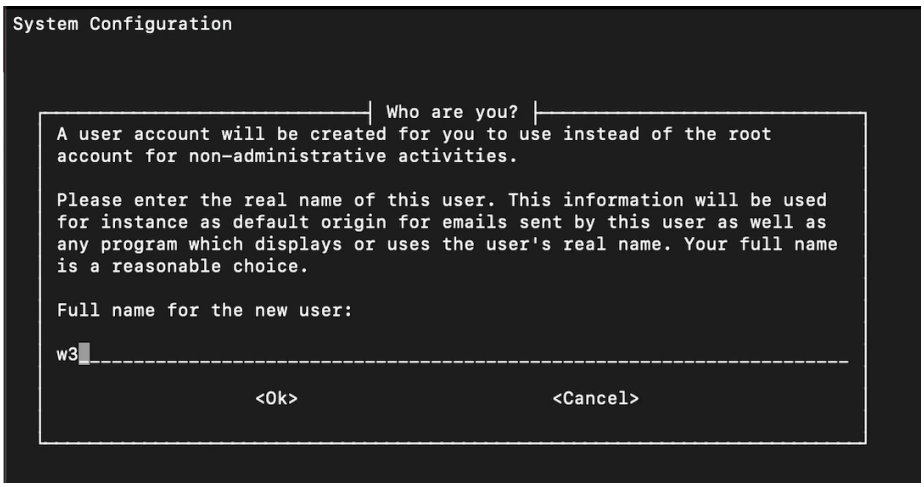


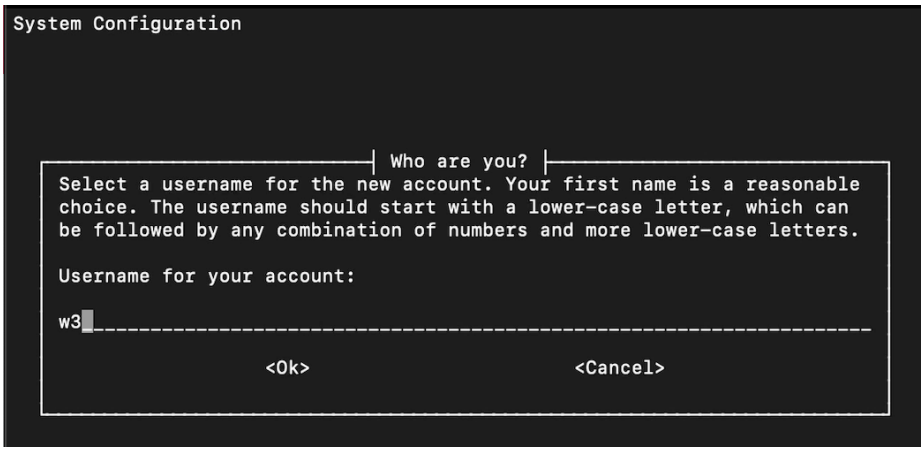

System Configuration
Select a language
Choose the language to be used for the installation process. The selected language will also be the default language for the installed system.
Language:
Arabic - عربي
Asturian - Asturianu
Basque - Euskara
Belarusian - Беларуская
Bosnian - Bosanski
Bulgarian - Български
Danish - Dansk
Dutch - Nederlands
English - English
<Ok>
<Cancel>

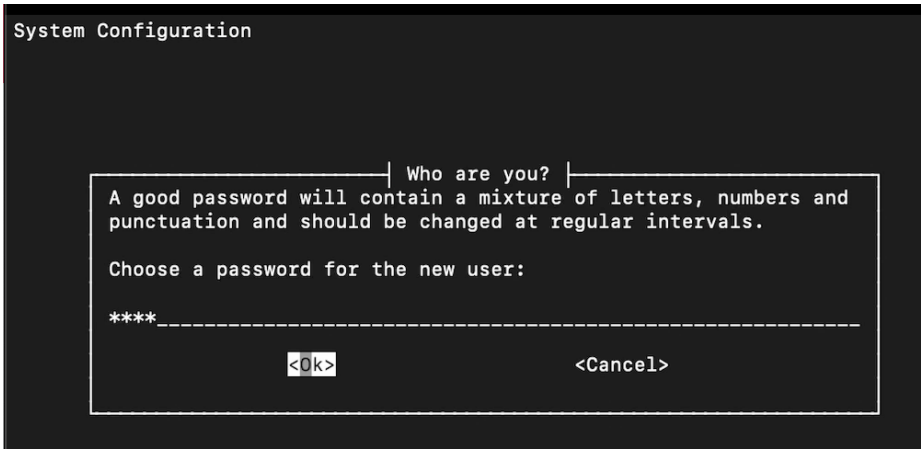

System Configuration
Select your location
The selected location will be used to set your time zone and also for example to help select the system locale. Normally this should be the country where you live.
This is a shortlist of locations based on the language you selected. Choose "other" if your location is not listed.
Country, territory or area:
New Zealand
Nigeria
Philippines
Seychelles
Singapore
South Africa
<Ok>
<Cancel>

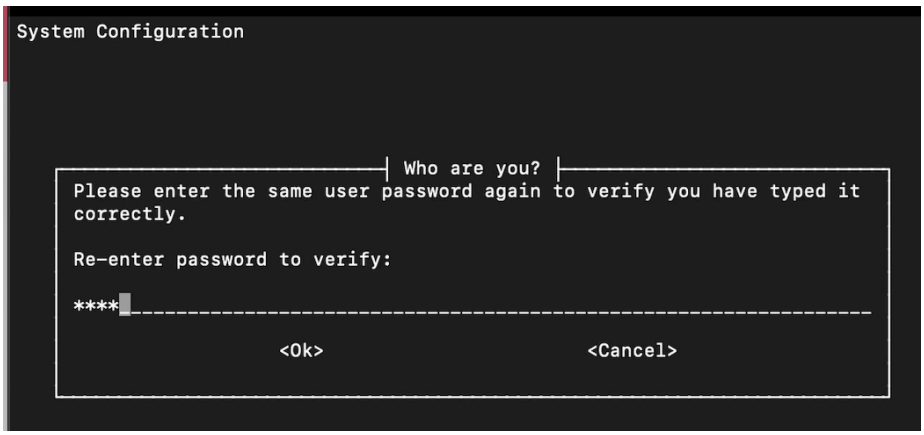

Where are you?
If the desired time zone is not listed, then please go back to the step
"Choose language" and select a country that uses the desired time zone
(the country where you live or are located).
Select a location in your time zone:
Auckland
Chatham Islands
<Ok>
<Cancel>

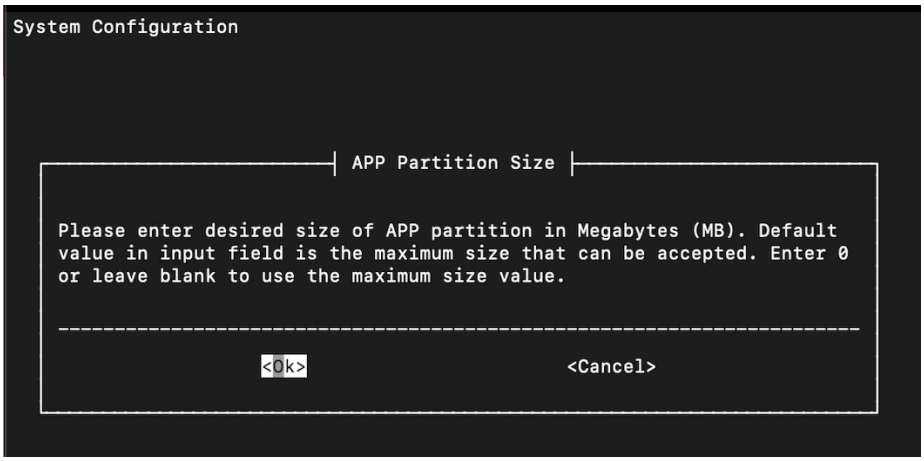

System Configuration
Where are you?
System clocks are generally set to Coordinated Universal Time (UTC).
The operating system uses your time zone to convert system time into
local time. This is recommended unless you also use another operating
system that expects the clock to be set to local time.
Is the system clock set to UTC?
<Yes>
<No>

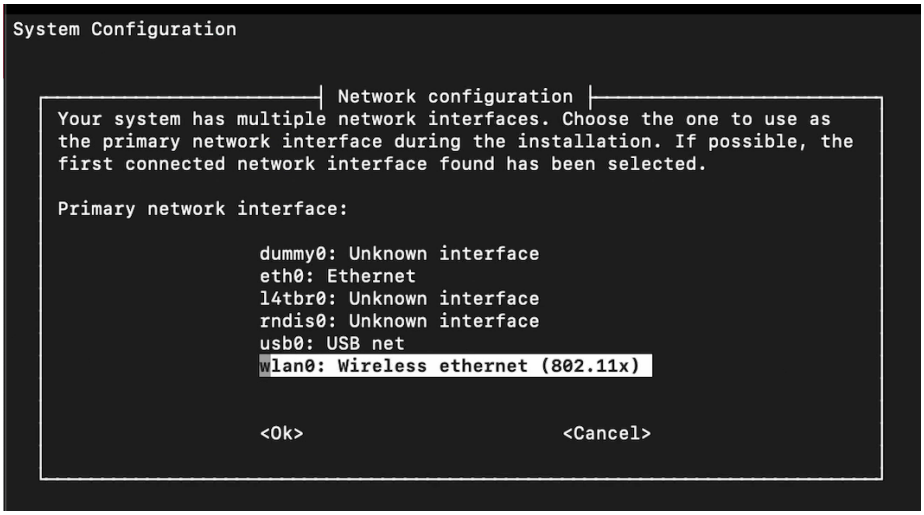

System Configuration
Who are you?
A user account will be created for you to use instead of the root
account for non-administrative activities.
Please enter the real name of this user. This information will be used
for instance as default origin for emails sent by this user as well as
any program which displays or uses the user's real name. Your full name
is a reasonable choice.
Full name for the new user:
w3
<Ok>
<Cancel>

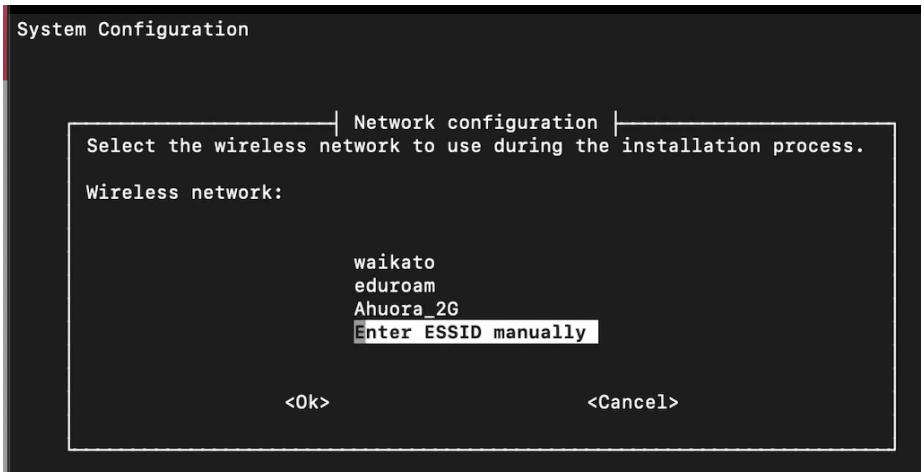

System Configuration
Who are you?
Select a username for the new account. Your first name is a reasonable choice. The username should start with a lower-case letter, which can be followed by any combination of numbers and more lower-case letters.
Username for your account:
w3
<Ok>
<Cancel>

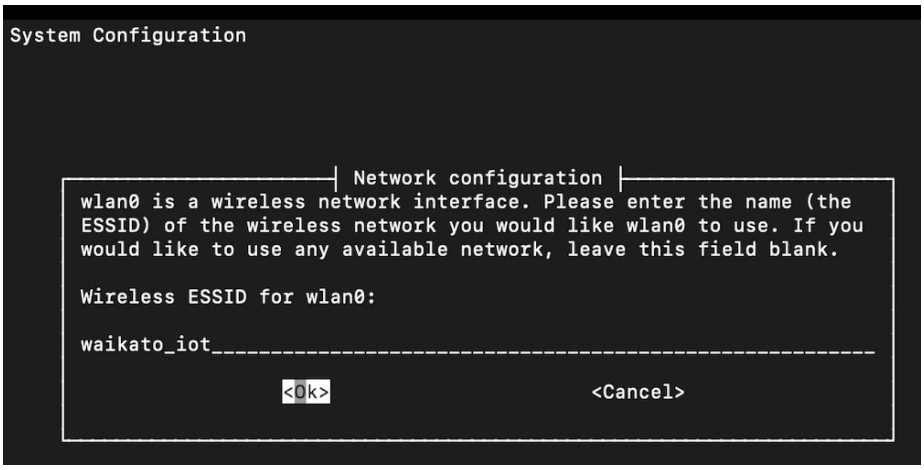

System Configuration
Who are you?
A good password will contain a mixture of letters, numbers and punctuation and should be changed at regular intervals.
Choose a password for the new user:
****
<Ok>
<Cancel>

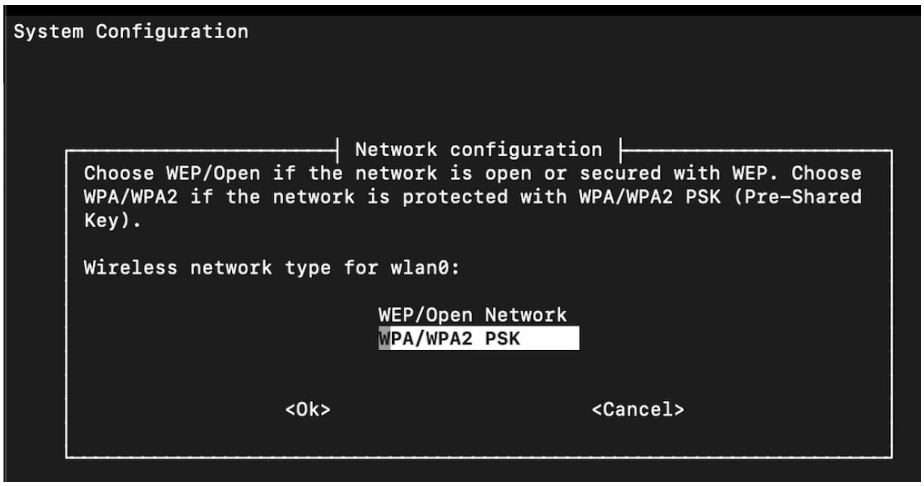

System Configuration
Who are you?
Please enter the same user password again to verify you have typed it correctly.
Re-enter password to verify:
****
<Ok>
<Cancel>

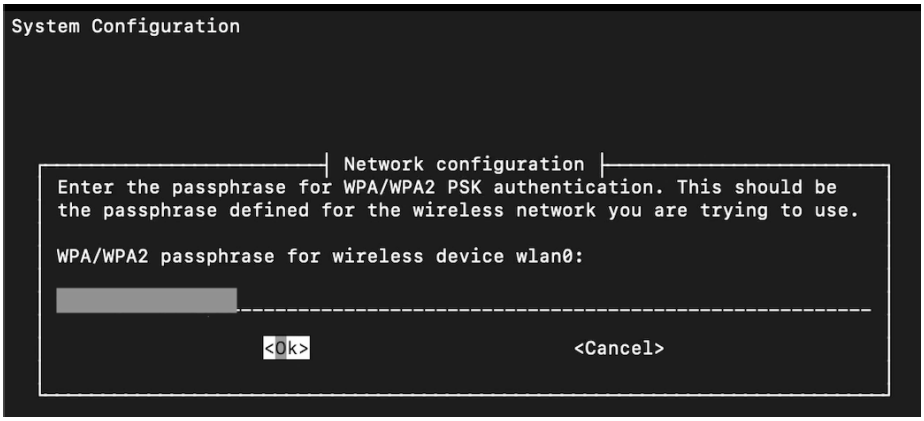

System Configuration
APP Partition Size
Please enter desired size of APP partition in Megabytes (MB). Default value in input field is the maximum size that can be accepted. Enter 0 or leave blank to use the maximum size value.
<Ok>
<Cancel>


System Configuration
Network configuration
Your system has multiple network interfaces. Choose the one to use as the primary network interface during the installation. If possible, the first connected network interface found has been selected.
Primary network interface:
dummy0: Unknown interface
eth0: Ethernet
l4tbr0: Unknown interface
rndis0: Unknown interface
usb0: USB net
wlan0: Wireless ethernet (802.11x)
<Ok>
<Cancel>

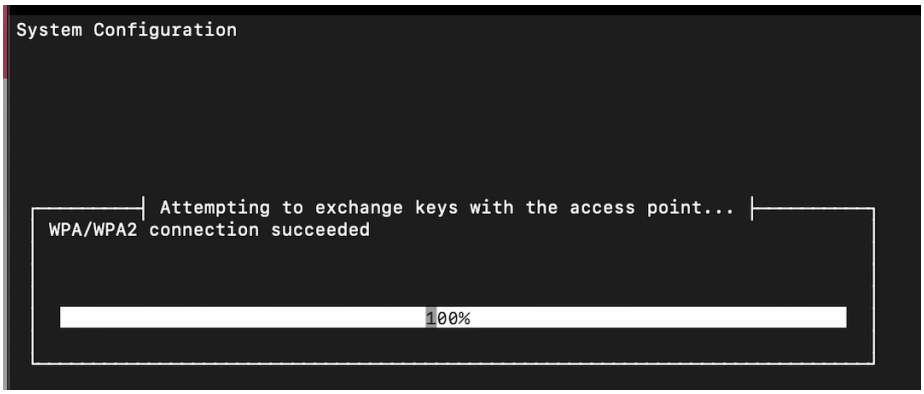

System Configuration
Network configuration
Select the wireless network to use during the installation process.
Wireless network:
waikato
eduroam
Ahuora_2G
Enter ESSID manually
<Ok>
<Cancel>


System Configuration
Network configuration
wlan0 is a wireless network interface. Please enter the name (the ESSID) of the wireless network you would like wlan0 to use. If you would like to use any available network, leave this field blank.
Wireless ESSID for wlan0:
waikato_iot
<Ok>
<Cancel>

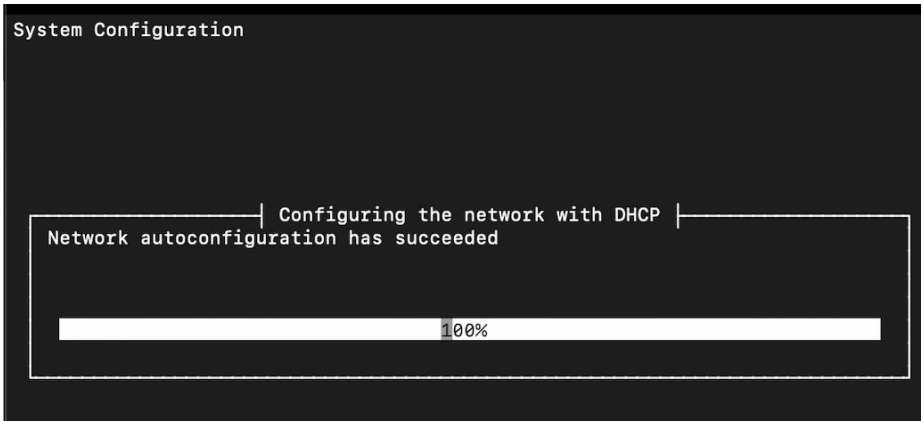

System Configuration
Network configuration
Choose WEP/Open if the network is open or secured with WEP. Choose WPA/WPA2 if the network is protected with WPA/WPA2 PSK (Pre-Shared Key).
Wireless network type for wlan0:
WEP/Open Network
WPA/WPA2 PSK
<Ok>
<Cancel>

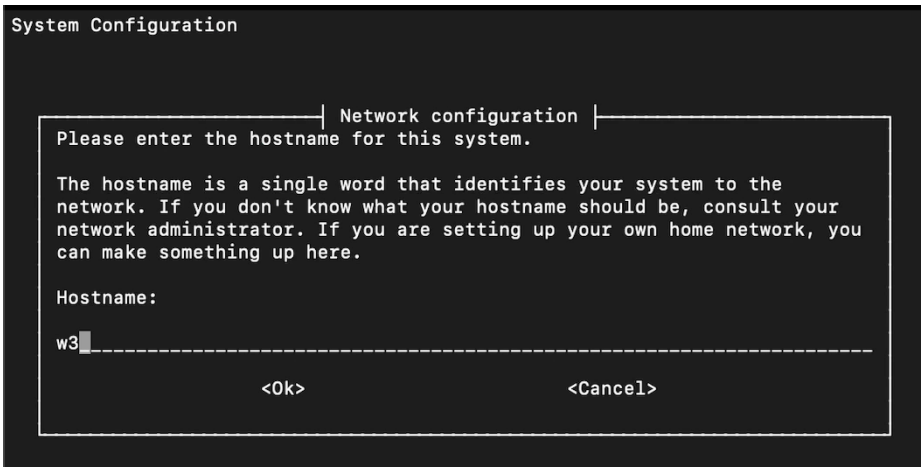

System Configuration
Network configuration
Enter the passphrase for WPA/WPA2 PSK authentication. This should be the passphrase defined for the wireless network you are trying to use.
WPA/WPA2 passphrase for wireless device wlan0:
<Ok>
<Cancel>

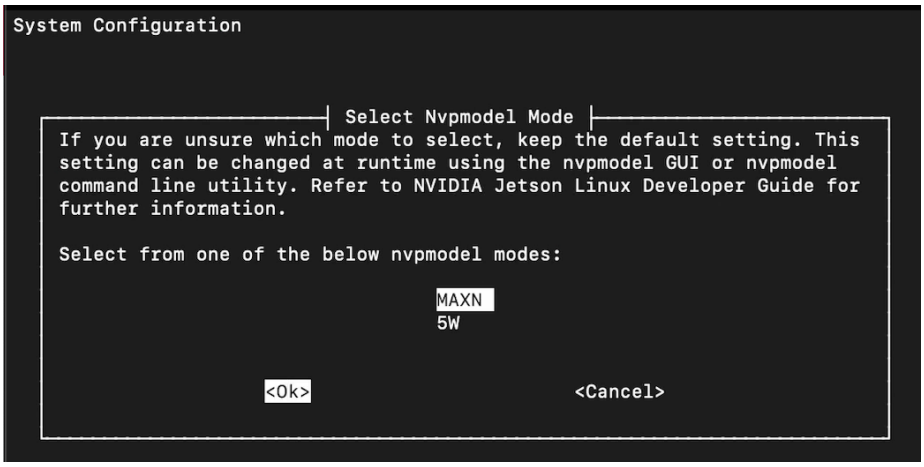

System Configuration
Attempting to exchange keys with the access point...
This may take some time.
31%

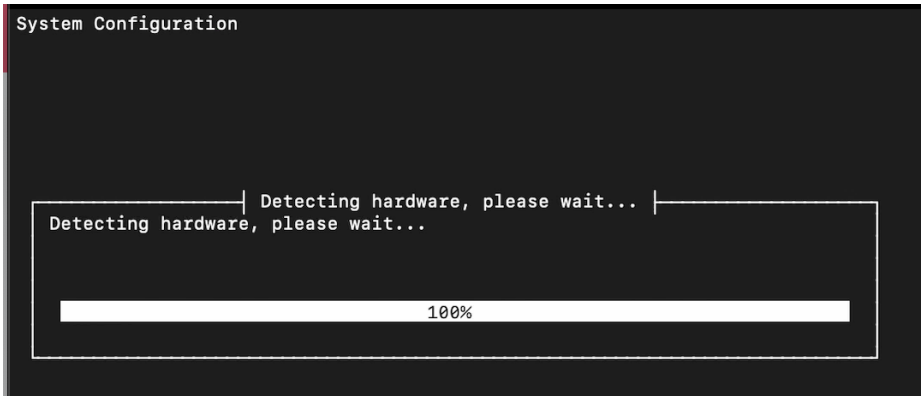

System Configuration
Attempting to exchange keys with the access point...
WPA/WPA2 connection succeeded
100%

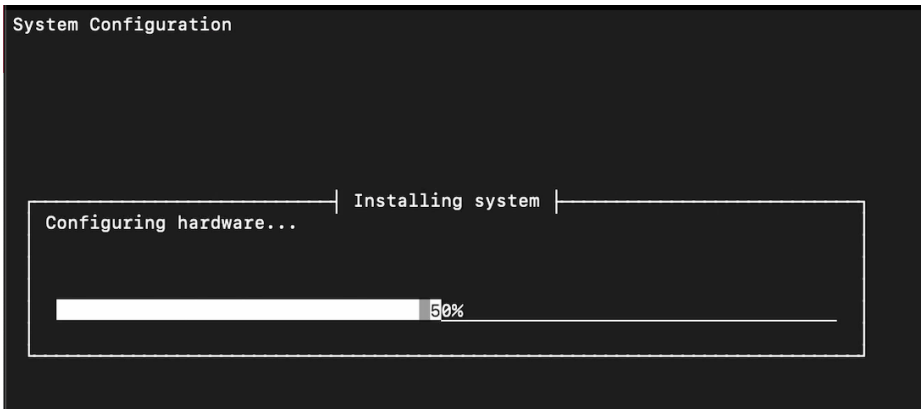

System Configuration
Waiting for link-local address...
75%

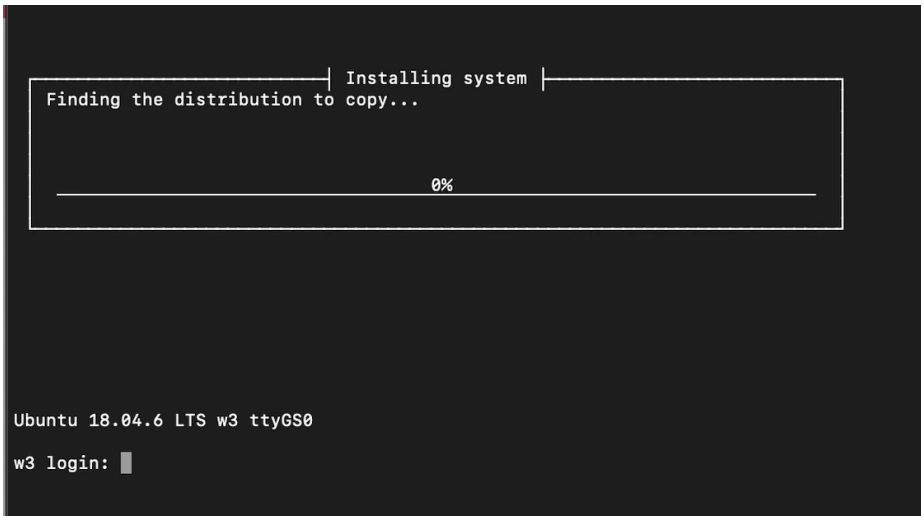

System Configuration
Configuring the network with DHCP
Network autoconfiguration has succeeded
100%


System Configuration
Network configuration
Please enter the hostname for this system.
The hostname is a single word that identifies your system to the network. If you don't know what your hostname should be, consult your network administrator. If you are setting up your own home network, you can make something up here.
Hostname:
w3
<Ok>
<Cancel>


System Configuration
Select Nvpmodel Mode
If you are unsure which mode to select, keep the default setting. This setting can be changed at runtime using the nvpmodel GUI or nvpmodel command line utility. Refer to NVIDIA Jetson Linux Developer Guide for further information.
Select from one of the below nvpmodel modes:
MAXN
5W
<Ok>
<Cancel>


System Configuration
Detecting hardware, please wait...
Detecting hardware, please wait...
100%





Step 2 : Verify Wi-Fi Connection

1. Log in to the Jetson Nano
   - After the Jetson Nano reboots, log in using the credentials you created during the setup

```
Ubuntu 18.04.6 LTS w3 ttyGS0

w3 login: w3
Password:
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.9.253-tegra aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

22 updates can be applied immediately.
16 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

w3@w3:~$
```

2. Manually connect to the Wi-Fi network
   - Use the nmcli command to connect to the Wi-Fi network:
     `sudo nmcli dev wifi connect your_wifi password your_password hidden yes ifname wlan0`

```
Device 'wlan0' successfully activated with 'b32ae8fd-b614-467b-968a-ff781203d59e'.
```

3. Verify the connection and note down the UUID of the Wi-Fi connection (e.g. b32ae8fd-b614-467b-968a-ff781203d59)

```
NAME                UUID                                  TYPE      DEVICE
docker0             21a6e505-21ec-4370-b0d8-0fdb4fd865ee  bridge    docker0
l4tbr0              591c3c9e-6cf5-473b-bfdb-f4ef7e94f059  bridge    l4tbr0
waikato_iot         b32ae8fd-b614-467b-968a-ff781203d59e  wifi      wlan0
Wired connection 1  1d741816-7255-31d4-a342-d506bf90648a  ethernet  --
```

4. Enable auto-connect for the Wi-Fi connection :
   - Modify the connection to ensure it auto-connects on boot:
     `sudo nmcli connection modify 723c68d9-d625-4e63-8bdc-d1bb1f7db822 connection.autoconnect yes`
   - Verify the auto-connect setting:
     `nmcli connection show 723c68d9-d625-4e63-8bdc-d1bb1f7db822 | grep autoconnect`

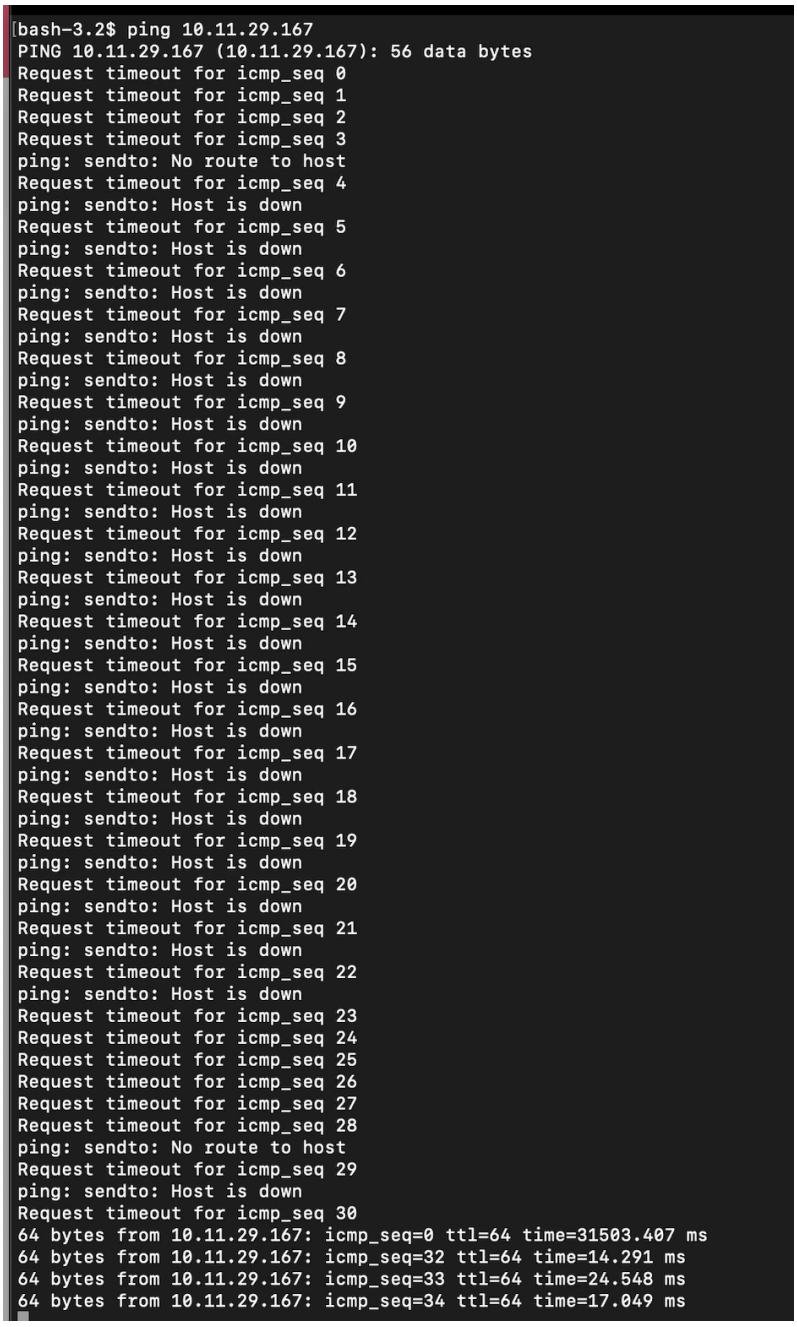
Step 3 : Obtain the IP Address

1. Run `ifconfig` to find the IP address by looking for the wlan0 interface to note down the IP address assigned to it (e.g., 10.11.29.167)

```
w3@w3:~$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:33:3c:a2:f9  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

l4tbr0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.55.1  netmask 255.255.255.0  broadcast 192.168.55.255
        inet6 fe80::1  prefixlen 128  scopeid 0x20<link>
        inet6 fe80::e8df:8dff:febc:7d1  prefixlen 64  scopeid 0x20<link>
        ether ea:df:8d:bc:07:d1  txqueuelen 1000  (Ethernet)
        RX packets 77  bytes 31860 (31.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 96  bytes 12376 (12.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 4405  bytes 297804 (297.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4405  bytes 297804 (297.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

rndis0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::e8df:8dff:febc:7d1  prefixlen 64  scopeid 0x20<link>
        ether ea:df:8d:bc:07:d1  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::e8df:8dff:febc:7d3  prefixlen 64  scopeid 0x20<link>
        ether ea:df:8d:bc:07:d3  txqueuelen 1000  (Ethernet)
        RX packets 93  bytes 38773 (38.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 82  bytes 19392 (19.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.11.29.167  netmask 255.255.255.0  broadcast 10.11.29.255
        ether 40:ae:30:47:91:60  txqueuelen 1000  (Ethernet)
        RX packets 128  bytes 58879 (58.8 KB)
        RX errors 0  dropped 23  overruns 0  frame 0
        TX packets 68  bytes 9423 (9.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Step 4 : Test Wi-Fi Connectivity

1. Disconnect the Jetson Nano from your laptop
2. Power the Jetson Nano using an external power supply
3. Ping the Jetson Nano from your laptop. If the Jetson Nano is reachable, you should see responses like below.



Note :-

- Ensure that both your laptop and the Jetson Nano are on the same Wi-Fi network.

- Test SSH connectivity by executing commands on the Jetson Nano remotely.

C Testbed Setup: Instructions for Coordinator

C.1 System Preparation and Swap File Configuration

Since federated learning can be memory-intensive, a 16GB swap file is created for stability in resource-limited environments.

Task	Commands
Allocate swap space	`sudo fallocate -l 16G /swapfile`
Set permissions	`sudo chmod 600 /swapfile`
Format swap file	`sudo mkswap /swapfile`
Enable swap	`sudo swapon /swapfile`
Go to /etc/fstab file	`sudo gedit /etc/fstab`
Make swap persistent	Add this line at the end of /etc/fstab: `/swapfile swap swap defaults 0 0`

C.2 Miniconda Installation

Task	Commands
Install conda	`bash`
	`wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-aarch64.sh`
	`bash Miniconda3-latest-Linux-aarch64.sh`
	`source ~/miniconda3/etc/profile.d/conda.sh`
	`sudo gedit ~/.bashrc`
	Add this line at the end of ~/.bashrc: `source ~/miniconda3/etc/profile.d/conda.sh`

C.3 Repository Cloning and Environment Creation

Task	Commands
Clone the repository and move into the relevant folder.	`git clone https://github.com/h-flox/sc24-flox-artifacts.git`
	`cd sc24-flox-artifacts/a4-ec2-tests/`
Create and activate a python 3.11.8 environment	`conda create -name=a4-env python=3.11.8`
	`conda activate a4-env`

C.4 Dependencies Installation

Task	Commands
Install system Packages	`sudo apt-get install gcc python3-dev`
Run requirement file (see Appendix. H)	`pip install -r requirements.txt`
Install python package	`pip install torchmetrics`
	`pip install pyarrow`
	`pip install git+https://github.com/h-flox/sc24-flox.git@964ed6c39a528eec50f6d01212ea0177bdfc0bd9\`
Installations to prevent serialisation issues	`pip install -no-deps pydantic==2.6.3`
	`pip install dill==0.3.8`

C.5 Globus Compute and ProxyStore Configuration

Task	Commands
Create and start globus compute endpoint (replace <NAME> with endpoint name)	`globus-compute-endpoint configure <NAME>`
	`globus-compute-endpoint start <NAME>`
Create and start proxystore with Globus credentials	`proxystore-globus-auth login`
	`proxystore-endpoint configure <NAME>`
	`proxystore-endpoint start <NAME>`

C.6 Final Steps and Running

Task	Commands
Download sataset	`python3 download_data.py -root .`
Modify `flock.yaml` file with UUID of the configured endpoints	View the endpoints list UUIDs using: `globus-compute-endpoint list` `proxystore-endpoint list`
Copy and paste the `run.py` script	Script is located in Appendix G
Run the script after all the nodes have been set up	`python run.py -root '.'`

D Testbed Setup: Instructions for Aggregator/-Worker

All steps from Appendix. C.1 to Appendix. C.5 should be followed to setup both aggregator and worker nodes.

E Simulation Setup: Instructions for Coordinator

To configure your VM, first install VMware Fusion (or Workstation)[1], download the Ubuntu Server ISO[2], create a new virtual machine using that ISO, and then follow the steps in the table below.

[1] `https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion`
[2] `https://ubuntu.com/download/server`

E.1 VM Setup: Basic Configuration

Task	Command
Update package index and upgrade all packages	`sudo apt update && sudo apt upgrade -y`
Install VMware desktop integration	`sudo apt-get install open-vm-tools-desktop`
Install the Ubuntu Desktop environment	`sudo apt install ubuntu-desktop`
Extend the root logical volume to use all free space	`sudo lvextend -l +100%FREE /dev/mapper/ubuntu-vg-ubuntu-lv`
Resize the filesystem to fill the enlarged volume	`sudo resize2fs /dev/mapper/ubuntu-vg-ubuntu-lv`

E.2 Post-VM Configurations

After completing the initial configuration above, follow the instructions from Appendix. C.2 to Appendix C.6 to complete the setup.

F Simulation Setup: Instructions for Aggregator/Worker

All steps in Appendix. E.1 and then from C.2 to Appendix. C.5 should be followed to setup both aggregator and worker nodes.

G run.py Script

```
import logging
import os
import time

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchmetrics
from flox import federated_fit
from flox.data import federated_split
from flox.flock import Flock
from flox.nn import FloxModule
from torchvision import transforms
from torchvision.datasets import FashionMNIST

# For CPU usage
import psutil

logging.basicConfig(
    format="(%(levelname)s  - %(asctime)s) > %(message)s", level=
    logging.INFO
)


class SmallConvModel(FloxModule):
```

```
    def __init__(self, lr: float = 0.01, device: str | None = None)
    :
        super().__init__()
        self.lr = lr
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(256, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.accuracy = torchmetrics.Accuracy(task="multiclass",
    num_classes=10)
        self.last_accuracy = None

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def training_step(self, batch, batch_idx):
        inputs, targets = batch
        preds = self.forward(inputs)
        loss = F.cross_entropy(preds, targets)
        self.last_accuracy = self.accuracy(preds, targets)
        return loss

    def configure_optimizers(self) -> torch.optim.Optimizer:
        return torch.optim.SGD(self.parameters(), lr=self.lr)

def estimate_data_transferred(model, num_workers, num_rounds):
    """
    Roughly estimate total data transferred by model updates.
    Assumes each round: aggregator -> worker, then worker ->
    aggregator
    with a full model state dict each time (float32).
    """
    total_params = sum(p.numel() for p in model.parameters())
    # float32 -> 4 bytes per param
    model_size_bytes = total_params * 4

    # aggregator -> worker + worker -> aggregator = 2 transmissions
    total_bytes = model_size_bytes * num_workers * 2 * num_rounds

    # convert to MB
    total_mb = total_bytes / (1024 * 1024)
    return total_mb


def main(root_dir):
    os.environ["TORCH_DATASETS"] = root_dir
    flock = Flock.from_yaml("flock.yaml")

    # Start timers & CPU usage
```

```
start_time = time.time()
start_cpu = psutil.cpu_percent(interval=None)

# Load training data
data = FashionMNIST(
    root=root_dir,
    train=True,
    download=False,
    transform=transforms.Compose([transforms.ToTensor(),
transforms.Normalize(0.5, 0.5)]),
)
fed_data = federated_split(data, flock, 10, 3.0, 1.0)

logging.info("Starting federated fitting.")
num_rounds = 10

# Federated training
module, history = federated_fit(
    flock,
    SmallConvModel(),
    fed_data,
    num_global_rounds=num_rounds,
    strategy="fedavg",
    kind="sync-v2",
    launcher_kind="globus-compute",
    debug_mode=False,
    logging=False,
)
history.to_feather("edge_run.feather")

# Training is done - measure total time
end_time = time.time()
completion_time = end_time - start_time

# CPU usage difference (basic snapshot)
end_cpu = psutil.cpu_percent(interval=None)
cpu_usage_percent = end_cpu  # This is a simple instantaneous
measure

# Evaluate final global model on the test set
test_data = FashionMNIST(
    root=root_dir,
    train=False,
    download=True,
    transform=transforms.Compose([transforms.ToTensor(),
transforms.Normalize(0.5, 0.5)]),
)
test_loader = torch.utils.data.DataLoader(test_data, batch_size
=64, shuffle=False)
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
module.to(device)
module.eval()

correct_test = 0
total_test = 0
with torch.no_grad():
```

```
        for imgs, labels in test_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            outputs = module(imgs)
            _, preds = torch.max(outputs, dim=1)
            correct_test += (preds == labels).sum().item()
            total_test += labels.size(0)
    final_test_accuracy = correct_test / total_test

    # Evaluate final global model on the *training* set
    train_loader = torch.utils.data.DataLoader(data, batch_size=64,
     shuffle=False)
    correct_train = 0
    total_train = 0
    with torch.no_grad():
        for imgs, labels in train_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            outputs = module(imgs)
            _, preds = torch.max(outputs, dim=1)
            correct_train += (preds == labels).sum().item()
            total_train += labels.size(0)
    final_train_accuracy = correct_train / total_train

    # Estimate total data transferred
    # Make sure 'module' is your final aggregated model
    # or create a fresh instance if needed.
    # If you have 10 workers in your YAML file, set num_workers=10
    num_workers = 4
    transferred_mb = estimate_data_transferred(module, num_workers,
     num_rounds)

    # Print results
    print("----- FEDERATED LEARNING RESULTS -----")
    print(f"Completion Time (s): {completion_time:.2f}")
    print(f"Test Accuracy:       {final_test_accuracy:.4f}")
    print(f"Train Accuracy:      {final_train_accuracy:.4f}")
    print(f"CPU Usage (%):       {cpu_usage_percent:.1f}")
    print(f"Data Transferred (MB, approx.): {transferred_mb:.2f}")
    print("Finished learning!")


if __name__ == "__main__":
    import argparse

    args = argparse.ArgumentParser()
    args.add_argument("--root", "-r", required=True)
    parsed_args = args.parse_args()
    main(parsed_args.root)
```

H Requirements File

Below is the content of `requirements.txt` used for pip installations

```
aiofiles
aioice
aiortc
```

```
aiosqlite
annotated-types
anyio
async-timeout
av
bcrypt
blinker
certifi
cffi
charset-normalizer
click
cloudpickle
contourpy
cryptography
cycler
# dill
dnspython
docutils
exceptiongroup
filelock
Flask
fonttools
fsspec
globus-compute-common
globus-compute-sdk
globus-sdk
google-crc32c
h11
h2
hpack
httptools
Hypercorn
hyperframe
idna
ifaddr
itsdangerous
Jinja2
kafka-python
kiwisolver
lazy-object-proxy
lockfile
MarkupSafe
matplotlib
mpmath
networkx
numpy
packaging
pandas
paramiko
parsl
pika
pillow
priority
psutil
pycparser
pyee
PyJWT
```

```
pylibsrtp
PyNaCl
pyOpenSSL
pyparsing
pystun3
python-daemon
python-dateutil
python-dotenv
pytz
PyYAML
pyzmq
Quart
redis
requests
scipy
setproctitle
six
sniffio
sympy
taskgroup
tblib
texttable
tomli
tomli_w
torch
torchvision
tqdm
typeguard
typing_extensions
tzdata
urllib3
uvicorn
uvloop
watchfiles
websockets
Werkzeug
wsproto
globus-compute-endpoint
proxystore
```

I Sample flock.yaml Script

```
Coordinator:
  kind: leader
  children: [ agg1]
  globus_compute_endpoint: 73315379-dd22-4c7a-82ea-90a93e9d4c5e
  proxystore_endpoint: 25ec54fe-e7a5-4d7f-af2e-39bed89bda4a

agg1:
  kind: aggregator
  children: [ worker1.1, worker1.2, worker1.3, worker1.4]
  globus_compute_endpoint: f1809ea1-0981-4195-b1c5-f90829e1c212
  proxystore_endpoint: d4c01984-2162-4cf3-96bf-3d40db91635f
```

```
worker1.1:
  kind: worker
  children: [ ]
  globus_compute_endpoint: 4039adde-bea8-413d-8991-b7a43a005d81
  proxystore_endpoint: 779ef14a-8648-4b35-967c-d4d659c21a88

worker1.2:
  kind: worker
  children: [ ]
  globus_compute_endpoint: 13a09d40-1857-402d-9300-8dec98887e42
  proxystore_endpoint: 64e92d8c-7080-4224-a3d7-8d52e17c37b2

worker1.3:
  kind: worker
  children: [ ]
  globus_compute_endpoint: dcb2510c-c233-4225-96a3-7b82f03b6d78
  proxystore_endpoint: bb333883-ab46-4a6c-b794-ccb26ef6af7d

worker1.4:
  kind: worker
  children: [ ]
  globus_compute_endpoint: 0ad311d8-3c67-4a8b-a98a-627c448cdfa0
  proxystore_endpoint: 03a83818-fddc-48d4-b902-28af28507491
```