Architectural Strategies for Federated Learning in Dynamic Environments

Priyanka Atwani
University of Amsterdam
Amsterdam, Netherlands
priyankaatwani@live.com

Jorrit Stutterheim
University of Amsterdam
Amsterdam, Netherlands
Jorrit.Stutterheim@gmail.com

Anestis Dalgkitsis
University of Amsterdam
Amsterdam, Netherlands
a.dalgkitsis@uva.nl

Aleandro Mifsud
University of Amsterdam
Amsterdam, Netherlands
A.Mifsud@uva.nl

Alexandros Koufakis
University of Amsterdam
Amsterdam, Netherlands
A.M.Koufakis@uva.nl

Leon Gommans
University of Amsterdam
Amsterdam, Netherlands
L.Gommans@uva.nl

Ana Oprescu
University of Amsterdam
Amsterdam, Netherlands
A.M.Oprescu@uva.nl

ABSTRACT

To realize the potential benefits of data-driven research in industry, a crucial step is facilitating data exchange between various parties. A powerful illustrating example comes from the aviation industry, where predictive maintenance could benefit from more fleet data, however many challenges hinder effectively exchanging this data across airlines, such as protection of crew's rights. One promising solution to this problem is Federated Learning, where the learning occurs without data actually being exchanged.

This paper considers scalable and modular design approaches to implement Federated Learning (FL). We showcase how such approaches can be applied when considering predictive aircraft engine maintenance based on machine learning (ML). We address data quality, privacy concerns, and collaborative data sharing among airlines through FL workflows. Our system, built with Docker containerization and leveraging the FABRIC testbed infrastructure, effectively handles large-scale dynamic datasets while ensuring scalability, modularity, and maintainability. Key design strategies include data partitioning, update strategies, adaptive learning, and disagreement handling. Incorporating microservices architecture based patterns enhances scalability and efficiency, allowing independent service deployment, development, testing, and scaling. This integrated approach ensures secure data exchange and seamless integration with existing aviation data management systems.

KEYWORDS

Federated Learning, software architecture, microservices, data aggregation, strategy

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, July 2017, Washington, DC, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Priyanka Atwani, Jorrit Stutterheim, Anestis Dalgkitsis, Aleandro Mifsud, Alexandros Koufakis, Leon Gommans, and Ana Oprescu. 2024. Architectural Strategies for Federated Learning in Dynamic Environments. In . ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Technological advancements are reshaping industries, with the aviation sector at the forefront of innovation [24]. According to the European Union Aviation Safety Agency (EASA), AI-based predictive maintenance, fueled by an enormous amount of fleet data[33], allows for anticipating failures and providing preventive remedies[11]. However, effectively utilizing this data, particularly for predictive maintenance, poses significant challenges. Machine learning algorithms trained on historical data can detect anomalies and predict maintenance needs, improving flight safety and operational efficiency. The availability of quality data, describing anomalies that can point to the need for maintenance apart from normal maintenance needs, is rare. The absence of rare event data affects the accuracy of these algorithms. This absence has also led to interest in collaborative data sharing among airline operators to improve predictive maintenance models [19].
The Independent Data Consortium for Aviation (IDCA) [3], among others, recognizes the potential of collaborative data sharing but faces implementation hurdles, including privacy concerns, intellectual property issues, and regulatory compliance. Additionally, the dynamic and sensitive nature of aviation data necessitates careful consideration of scalability and security. Federated Learning (FL)—a decentralized approach to machine learning—offers a promising solution by allowing model training on distributed data without centralized aggregation, thereby ensuring data sovereignty and protecting privacy [22].
Despite its potential, there is a significant research gap concerning the architectural design and implementation strategies required to transition from centralized machine learning workflows to decentralized FL systems. Existing literature often overlooks practical challenges such as data partitioning, communication overhead

[23][18], model convergence, and client heterogeneity [32]. Addressing these challenges through careful architecture- and solution design descisions is crucial for developing robust, high-performance FL solutions.
In the context of FL systems, it is crucial to understand how to effectively tailor software architecture patterns to support the requirements of these systems. To address this, the research must explore the criteria and metrics that are essential for selecting appropriate software architecture patterns that align with the decentralized nature of FL. This involves examining how existing patterns can be adapted or new patterns developed to handle the distributed data and computation characteristics in FL systems.
Additionally, key design decisions play an important role in the successful implementation of federated learning models. This includes addressing specific challenges such as disagreement resolution among clients, which is critical for ensuring the accuracy and effectiveness of the global model. The research needs to investigate how to effectively manage conflicts among clients, and what design strategies can facilitate this process within the federated learning framework.
To address these challenges, this paper aims to investigate the adaptation of software architecture patterns and critical design decisions using the N-CMAPSS dataset, containing aircraft engine simulator data. The integration of the Fabric framework plays a crucial role in this research, providing the necessary tools and infrastructure to support federated learning and facilitate data sharing. Specifically, the key contributions of our work can be summarized as follows::

- Identification of Optimal Software Architecture Patterns: Investigated and identified software architecture patterns most suitable for supporting federated learning systems, addressing their unique decentralized nature.
- Critical Design Decisions for Federated Learning: Examined and defined key design decisions crucial for enabling effective implementation and operation of federated learning models in practical scenarios.
- Architectural Design Patterns for Federated Learning: Studied architectural design patterns that can accommodate federated learning in a reliable and scalable microservices-based architecture.
- Demonstration in the Aviation Industry: Demonstrated the viability of federated learning in the critical scenario of the aviation industry with the help of the N-CMAPSS dataset, potentially saving thousands of lives by enabling secure inter-airline data sharing for predictive maintenance of turbofan jet engines.
- Performance Evaluation Using JMeter: Conducted extensive testing using JMeter to evaluate the performance and scalability of the implemented architectures, providing insights into their effectiveness.

The remainder of this paper is organized as follows:
In Section 2, we provide the background and context for this research. Section 3, reviews the existing work related to this paper. Section 4 compares different software architectures suitable for federated learning and covers the system design details. Section 5 describes the implementation of the potential federated learning architectures. In Section 6, we elaborate on the key design strategies employed in the system implementation. Section 7 details the experimental setup. Section 8 presents the results obtained from the experiments. The results are discussed in Section 9. Finally, Section 10 offers our concluding remarks and directions for future work.

2 BACKGROUND

This section provides an overview of key concepts and technologies used throughout this paper, focusing on FL, microservices architecture, the open-source N-CMAPSS dataset, and the FABRIC testbed.

2.1 Federated Learning

FL is a decentralized approach to training machine learning models, initially introduced by Google [37]. Unlike traditional centralized training, which aggregates data into a central repository, FL allows multiple clients to collaboratively train a shared, global model while keeping individual data sources local. This approach minimizes data transfer, enhances privacy, and leverages diverse, heterogeneous datasets distributed across different environments. Table 1 outlines the main differences between centralized and federated learning while figure 1 shows the FL process which involves:

Aspect	**Centralized Learning**	**Federated Learning**
Data Distribution and Storage	Data is stored in a central repository.	Data remains on individual devices (or edge nodes).
Model Training	Models are trained using all data in one place.	Models are trained locally on each device (or edge node).
Communication, Data Governance, and Privacy	Simple communication. Data access and usage are governed centrally.	Only model updates are sent, preserving data sovereignty and privacy.
Resource Requirements	High central computational resources.	Distributed computational resources.
Scalability	Scalability challenges due to central infrastructure.	Scales by adding distributed resources.

Table 1: Comparison of Centralized and Federated Learning

- **Initialization:** Clients download the latest global model from a central server.
- **Local Training:** Clients train the model on their local datasets, updating model parameters to minimize the loss function.
- **Model Aggregation:** Clients send updated model parameters to the central server, which aggregates them (e.g., using Federated Averaging) to update the global model.

Key features of FL include [36]:

- **Decentralized Data:** Federated Learning enables machine learning models to be trained on data distributed across multiple devices or servers without centralizing the data .
- **Privacy-Preserving:** Data remains on the user's device, ensuring that sensitive data is not exposed to others. This privacy-preserving nature makes Federated Learning suitable for applications requiring high privacy and security.

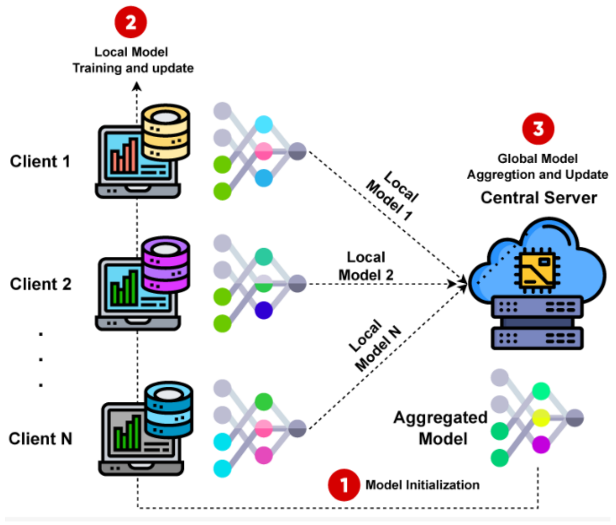

Figure 1: Federated learning process[28]

- **Collaborative Learning:** Multiple devices or servers can collaborate and contribute to the training of a machine learning model, even if they have different datasets.
- **Efficiency:** By reducing the amount of data that needs to be transmitted to a central server, Federated Learning minimizes computational and communication overheads.
- **Iterative Updates:** Federated Learning allows for cumulative updates to the machine learning model over time without the need for full retraining.
- **Flexibility:** Federated Learning is a versatile approach that can be used in various settings, including edge devices, mobile devices, and data centers.

FL can be categorized into different types based on data distribution[36]:

- **Horizontal FL:** Clients have datasets with the same feature space but different samples.
- **Vertical FL:** Clients share the same sample space but have different feature spaces.
- **Federated Transfer Learning:** Datasets differ in both samples and features.

Based on the type of participating clients and their characteristics, federated learning can be classified into [16, 38]:

- **Cross-Silo FL:** Involves a small number of clients (e.g., organizations) with significant computational resources.
- **Cross-Device FL:** Involves a large number of clients (e.g., mobile devices) with limited resources.

Despite its advantages, Federated Learning (FL) faces significant challenges such as data heterogeneity, scalability, communication efficiency, and ensuring privacy and security. Data heterogeneity involves variations in data distributions across different clients, which can be mitigated through personalized learning and hierarchical federated learning. Scalability issues, arising from the increasing number of participating clients, can be addressed with efficient communication protocols and model update compression techniques like quantization and sparsification, reducing the size of updates. Ensuring privacy and security is paramount in FL, with advanced cryptographic methods such as secure multiparty computation (SMC) and differential privacy providing robust frameworks to protect data during training. These solutions collectively aim to make FL a viable approach for collaborative machine learning while preserving data privacy [25].

2.2 Microservices

A microservices architecture based design is an approach where a software system or application is composed of small, independent services that communicate through well-defined APIs [1]. This contrasts with traditional monolithic software architecture, where all functionalities are intertwined and deployed as a single unit.

Technological advancements have facilitated the adoption of microservices architecture, supported by cloud computing, containerization technologies like Docker, orchestration tools like Kubernetes, and DevOps practices such as continuous integration/continuous deployment (CI/CD). The main advantages of a microservices architecture include scalability, fault tolerance, maintainability, and quicker releases [17]. However, it also introduces challenges related to infrastructure complexity and service communication.

2.3 NASA Turbofan Jet Engine Dataset (N-CMAPSS)

The N-CMAPSS dataset, an extension of NASA's earlier CMAPSS dataset[10], was created to enhance predictive maintenance models for aviation engines[7]. It provides high-fidelity simulations of turbofan engines, incorporating realistic flight conditions and component failures. The dataset contains multivariate time-series data from multiple flight cycles, covering climb, cruise, and descent phases. It is divided into subsets (DS01 to DS08), each offering detailed data on operational conditions, measured signals, engine health parameters, and time-to-failure (RUL) labels [7]. Sensor noise is added to simulate real-world variability. This comprehensive dataset is designed to reflect accurate engine degradation patterns, supporting research in predictive maintenance and prognostics[13]. Figure 2 provides a schematic representation of the simulated turbofan engine and its sensors as used in the N-CMAPSS dataset. It illustrates the various components of the engine and the placement of sensors that capture the critical operational parameters, essential for modeling engine degradation and predicting failures.

2.4 FABRIC Testbed

The FABRIC (Framework for Accelerated Built-In Resilience Infrastructure for Computing) testbed is an advanced research infrastructure designed to support next-generation computing systems and networks. It provides a scalable, distributed environment with high-performance computing resources, integration with containerization technologies like Docker and Kubernetes, and advanced networking features [2]. Additionally, the FABRIC Across Borders (FAB) is an extension of the testbed connecting the core North American network with global institutions, enabling international collaboration and accelerating scientific discovery.

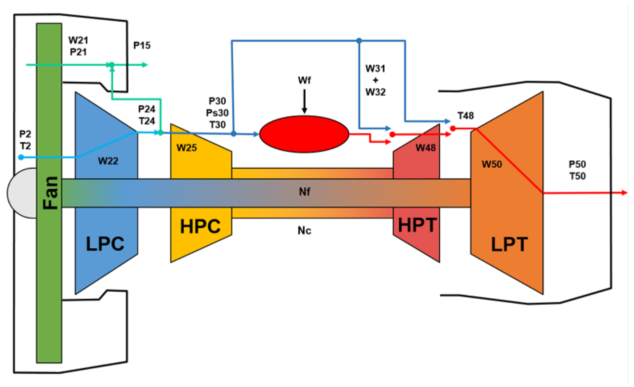

Figure 2: Schematic representation of the CMAPSS model[7]

Citations	Cloud	FL	Industry	Scalability	Privacy	Data Sharing
[26]	✓	✓	Healthcare	-	✓	-
[4]	✓	✓	-	-	✓	-
[35]	-	✓	Aviation	-	✓	-
[34]	✓	✓	-	✓	✓	-
[21]	-	✓	-	-	✓	✓
[31]	-	✓	Healthcare	-	✓	-
[37]	-	✓	Finance	-	✓	-
[15]	-	✓	Mobile Edge Computing	-	✓	-
[20]	-	✓	Healthcare	-	✓	-
[6]	-	-	PdM	-	✓	✓

Table 2: Related work categorization

3 RELATED WORK

Federated learning has emerged as a compelling paradigm in the machine learning domain, enabling collaborative model training across decentralized devices while maintaining data privacy. This section reviews key works related to federated learning, focusing on its foundational concepts and applications across various fields.

3.1 Foundational Concepts and Algorithms

The concept of federated learning was first introduced by McMahan et al. (2017) with the development of the Federated Averaging (FedAvg) algorithm[9]. This work laid the groundwork for training deep neural networks across decentralized devices with minimal communication costs. FedAvg achieves this by averaging model updates from multiple clients, which significantly reduces the need for frequent data exchanges between clients and the central server. Building on this foundation, several variations and improvements to the FedAvg algorithm have been proposed. For instance, [39] addressed the issue of non-IID (non-independent and identically distributed) data across clients by introducing data-sharing strategies to mitigate the impact of skewed data distributions on model performance. Similarly [22] proposed the FedProx algorithm, which adds a proximal term to the local objective function to stabilize training when dealing with heterogeneous client data and computational resources.

3.2 Applications and Case Studies

Federated learning has found applications in a wide range of domains, from healthcare and finance to mobile edge computing and the Internet of Things (IoT). The table below summarizes the concepts explored in the related works.

4 SYSTEM DESIGN & ARCHITECTURE IN FEDERATED LEARNING

In the design and implementation of FL systems, several architectural components and patterns must be carefully considered to ensure efficiency, scalability, and data privacy. This section explores the essential components of FL systems, contrasts centralized and federated architectures, and evaluates different software architecture patterns based on key criteria chosen.

4.1 Architectural components of Federated Learning

Federated Learning (FL) systems consist of several essential components[8] as illustrated in Figure 3:

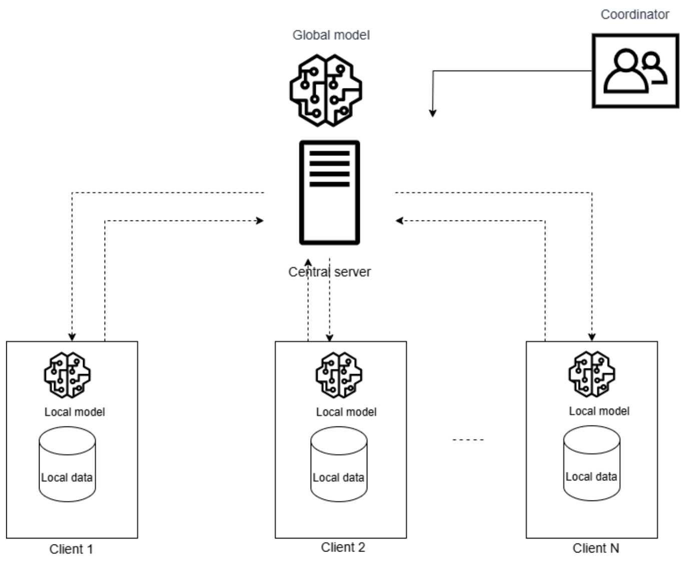

Figure 3: Components in federated learning

- Client Devices: These include mobile devices, Internet of Things (IoT) endpoints, or edge nodes. They perform local training on their data without sharing raw data and send model updates to the server.
- Central Server: Acts as the aggregator, receiving model updates from clients, aggregating them, and redistributing the global model.
- Coordinator (Orchestrator): Manages the FL process, selects clients, distributes initial models, and monitors convergence.

In a Federated Learning (FL) system, the components work in a coordinated manner to ensure the effective training and updating of the global model. The **Coordinator** begins the process by selecting a subset of Client Devices, particularly when dealing with scenarios involving disagreements or specific conditions. This selected subset receives the initial global model from the Coordinator. Each **Client Device** then performs local training on its own data, updating the model parameters based on its local dataset. These updated parameters are sent back to the **Central Server**, which aggregates the received updates to form a new, improved global model. This newly aggregated global model is then redistributed to the Client Devices for further training. This iterative cycle of training, aggregation, and redistribution continues until the global model converges to a desired level of performance.

4.2 Software Architecture Pattern Criteria

Key criteria for selecting software architecture patterns include:

- **Scalability:** Handling many clients and large data volumes.
- **Maintainability:** Ensuring smooth updates and bug fixes.
- **Modularity:** Independent modules simplify development and debugging.

Other important criteria are security, performance, interoperability, and flexibility. In this study, we focus on the criteria of scalability, modularity, and maintainability because they intersect to form a strong foundation for designing and evaluating software architecture patterns in federated learning scenarios. These criteria collectively address critical challenges that are pivotal for the successful deployment and operation of federated learning systems, particularly in dynamic and collaborative environments involving multiple airlines.

4.3 Architecture Selection based on Key Criteria

Three primary architecture patterns are considered: Client-Server, Monolithic, and Microservices. Each has strengths and weaknesses [14, 27, 29, 30]which are presented in Table 3:

Criterion	Client-Server	Monolithic	Microservices
Scalability	High client scalability, server bottlenecks	Limited scalability due to tight coupling	High scalability, each service scales independently
Modularity	Moderate, clear client-server separation	Low, single codebase	High, services are independent and focused
Maintainability	Moderate, easy client-server interactions	Low, complexity grows with the codebase	High, independent services simplify updates

Table 3: Architecture Evaluations

Therefore, for dynamic and collaborative environments like multiple airlines, the Microservices architecture is the most suitable, offering the highest scalability, modularity, and maintainability.

5 PROTOTYPE IMPLEMENTATION

This section outlines the prototype implementation of both client-server and microservices architectures for federated learning. These architectures are designed to enable multiple distributed clients to train local machine learning models on their datasets and collaboratively aggregate these models to build a global model. Monolithic architecture was not considered for the prototype implementation due to its inherent limitations in scalability and modularity as can be seen in Table 3.

5.1 Client-Server Architecture

In this section, we outline the implementation details of a client-server architecture designed for federated learning. This architecture facilitates multiple distributed clients to train local machine learning models on their respective datasets and collaboratively aggregate these models to build a global model.

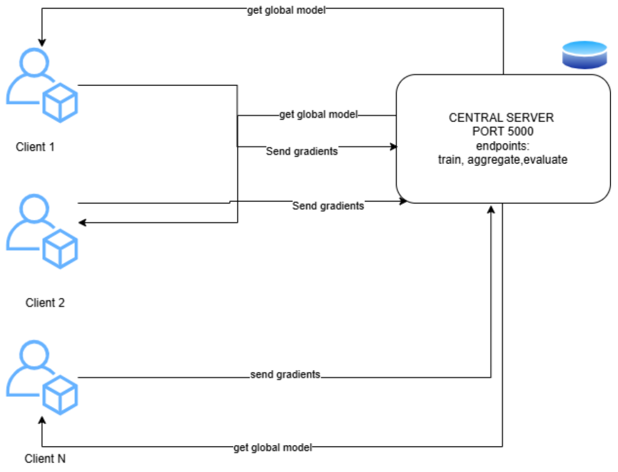

Figure 4: Client-server implementation

Server-Side Implementation: The server utilizes Flask to establish a RESTful API, enabling seamless communication with multiple client devices. Key components and functionalities of the server-side implementation are as follows:

- **Global Model:** The server initializes and updates the global model, employing a LinearRegression model from scikit-learn as the initial model parameters for aggregation.
- **Model Aggregation**: The server receives model updates from clients and aggregates these updates to refine the global model parameters. This includes calculating the mean of model coefficients and intercepts across all participating clients.
- **Model Management:** The server provides endpoints for clients to retrieve the current global model parameters and to update the global model with new parameters from clients.

Client-Side Implementation: The client-side implementation simulates multiple distributed clients participating in federated learning. Key functionalities and interactions include:

- **Data Preparation**: Clients prepare their local datasets (dev_data and test_data) stored in HDF5 format, which are then converted into Pandas DataFrames (df_train and df_test).
- **Model Training**: Clients train their local machine learning models using their respective training data (train_client). The locally trained models are then sent to the server for aggregation.
- **Final Model Evaluation**: Clients evaluate the global model (evaluate_global_model) on their test data to assess its predictive performance using evaluation metrics provided by the server.

5.2 Microservices Architecture

The microservices architecture in this federated learning setup is designed for scalability and modularity, facilitating independent development and deployment of various components. Our proposed architecture is comprised of the following components:

- **Model Service**: Manages global model parameters through endpoints for retrieval and update, ensuring centralized access and synchronization.
- **Training Service**: Allows clients to train local models using their data. It retrieves the current global model parameters, trains a local model, and updates the global model with the learned parameters.
- **Aggregation Service**: Collects model updates from clients and aggregates them to refine the global model. It supports sequential and final aggregation modes to enhance model accuracy.
- **Registration Service**: Manages client registration and participation in the federated learning process. It ensures that only registered clients can contribute updates, maintaining system integrity and security.
- **Evaluation Service**: Evaluates the global model using test data provided by clients. It computes evaluation metrics such as RMSE and R2 score to assess model performance and guide further training efforts.

Flask-RESTful API: The architecture utilizes Flask-RESTful to create and manage API endpoints efficiently. Flask-RESTful promotes modularity by structuring endpoints as resources, facilitating easier maintenance and extensibility across different functionalities like training, evaluation, and aggregation. This structured approach with Flask-RESTful ensures efficient communication and management of microservices in the federated learning system, facilitating robust and scalable deployment across distributed environments [5].

6 DESIGN STRATEGIES FOR FEDERATED LEARNING

Designing effective strategies for federated learning involves addressing challenges such as data partitioning, update management, adaptive learning, handling disagreements among clients, and containerization for scalable deployment. This section discusses the key design decisions made in our federated learning setup, focusing on optimizing model performance and system efficiency across distributed networks.

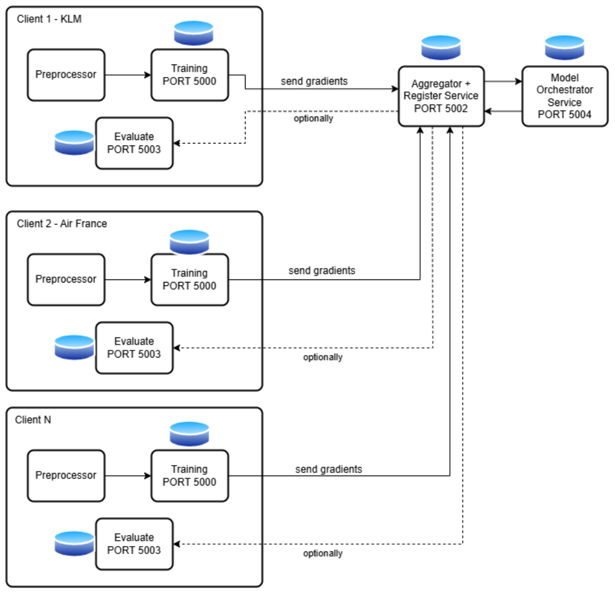

Figure 5: Microservices implementation

6.1 Data Partitioning and Sampling Strategy

In our federated learning architecture, data partitioning plays a crucial role in ensuring efficient utilization of distributed datasets across multiple clients or nodes. We adopt a strategy where the original dataset, stored in HDF5 format, is partitioned into subsets while preserving data integrity and order. Each subset, designated for individual (sovereign) clients, allows them to train local models independently on their specific data segments (dev_data and test_data). This partitioning approach facilitates parallel processing and scalable model training, leveraging the capabilities of each client without compromising data consistency.

6.2 Update Strategy

The update strategy in our federated learning system revolves around enabling seamless collaboration and aggregation of model updates from diverse client environments. Clients autonomously train their local models using their datasets and submit updated model parameters to the central server for aggregation. This iterative process, managed through secure communication channels, ensures that the global model evolves with contributions from all participating clients while maintaining consistency and accuracy. Moreover, to mitigate client drift [32]—wherein local models diverge due to varying data distributions—we employ periodic reinitialization of client models with updated global parameters. This adaptive learning approach enhances model robustness and adaptability over successive training rounds.

6.3 Adaptive Learning and Transfer Learning

Adaptive learning and transfer learning are integral components of our federated learning strategy, designed to optimize model performance and knowledge transferability across distributed clients. Clients initialize their models with pre-trained global parameters received from the server, facilitating rapid adaptation to local datasets while retaining learned knowledge from previous training cycles. This transfer learning initialization reduces the computational burden on clients and accelerates convergence towards an optimized global model. Throughout the training process, clients continuously refine their models based on local data characteristics, contributing unique insights to the global model through iterative updates and aggregations.

6.4 Handling Disagreements

Disagreements among clients pose challenges in federated learning environments, potentially leading to conflicting updates or non-cooperation. Our system implements robust mechanisms for detecting, managing, and resolving disagreements effectively. During client registration, each client specifies any disagreements with other clients, which are recorded and considered during the aggregation phase. Before aggregating model updates, the server checks for conflicting contributions and applies predefined rules to ensure consensus without compromising model integrity. This client-side and server-side handling of disagreements promotes fairness and transparency in model aggregation, fostering collaborative learning across diverse participant networks.

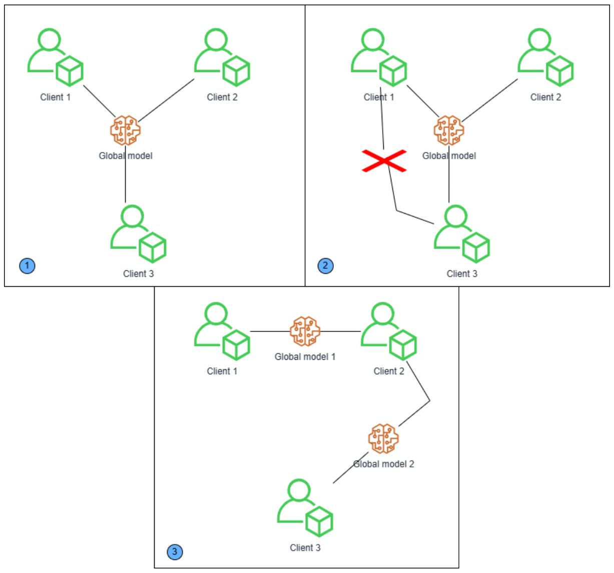

Figure 6: Disagreement scenario with three clients

The figure 6 considers a federated learning process involving three clients—Client 1, Client 2, and Client 3—where Client 1 and Client 3 have a mutual disagreement and do not wish to share data with each other, the system handles this by organizing clients into distinct groups based on their data-sharing preferences. Client 1 and Client 3 are placed in separate groups to avoid conflicts. Consequently, Client 1 and Client 2, who are willing to share data, form one group. Meanwhile, Client 3, who also has no issues with Client 2, forms a group with Client 2. Each group then independently aggregates the model updates from its members, resulting in distinct global models that accurately represent the consensus within each group. This approach ensures that the global models are built on consistent data from clients who agree to collaborate.

6.5 Containerization

To streamline deployment and scalability, our federated learning system is containerized using Docker technology. Each service—such as client training, model aggregation, and API servers—is encapsulated within Docker containers, ensuring isolation, portability, and consistent execution environments. Docker Compose orchestrates the interaction between multiple containers, simplifying deployment across distributed environments and facilitating seamless integration with infrastructure frameworks like the FABRIC testbed. This containerization approach enhances system maintainability, scalability, and resource efficiency, supporting dynamic workload management and rapid deployment of federated learning services.

7 EXPERIMENTAL SETUP

The experimental setup was designed to evaluate the performance of client-server and microservices architectures on the N-CMAPSS dataset (described in Section 2.3) using Apache JMeter, a widely-used tool for performance testing. This setup aimed to provide insights into how each architecture handles simulated user traffic and to compare their performance across various metrics. It facilitated the creation of realistic testing scenarios and provided insights into how each architecture handles varying levels of simulated user traffic through HTTP requests [12].

7.1 Test Environment

Infrastructure: The performance tests were conducted using a single laptop, which served as both the testing and simulation environment. The laptop was configured to run Apache JMeter and execute the performance tests for the client-server and microservices architectures. This setup simulated user interactions and system behavior under controlled conditions, with the laptop's hardware and network settings providing a consistent testing environment.

Software: Apache JMeter was utilized for load generation and performance measurement. The JMeter test plans were designed to simulate a range of user interactions with the client-server and microservices architectures through HTTP requests. Key metrics such as response times, throughput, and maximum latencies were collected and analyzed for comparative analysis.

Testing architecture: In this analysis we are examining the following application architectures:

- **Client server architecture:** Implemented as a monolithic application where a single server instance handled all client requests directly.
- **Microservices architecture:** Configured as a set of independent services communicating through a central server.

Microservices Test Scenarios: We conduct test in two different scenarios for the microservices architecture:

- Non-disagreement scenario: Clients are willing to share data with one another and there are no conflicts.
- Disagreement scenario: Clients have conflicts and are not willing to share their data with one or more clients.

7.2 Test configuration

JMeter Configuration: JMeter test plan was designed to simulate user interactions with the client-server and microservices architectures. The test plans included HTTP requests to the server or services, configured to replicate real-world usage patterns. JMeter listeners were used to collect performance metrics such as response times, throughput, and latency.

Test execution: The tests involved two clients performing a single round of federated learning and then finally the use of an evaluation service to evaluate the global model. This was done to ensure consistency and comparability across different test scenarios.

- Client-Server Tests: The tests involved simulating a controlled number of concurrent requests to a single server instance.
- Microservices Tests: These tests simulated interactions with multiple independent microservices. The performance was measured in two distinct scenarios:
  - Non-Disagreements Scenario: Both clients provided consistent updates to the global model, reflecting no conflicts. This scenario assessed how well the microservices architecture performed when client updates were harmonious.
  - Disagreements Scenario: The two clients provided conflicting updates to the global model, simulating conflicts. This scenario evaluated the impact of conflicting updates on performance metrics such as response time, throughput, and latency.

8 RESULTS

In this section, we present the performance metrics for the client-server and microservices architectures, as evaluated using Apache JMeter. The focus is on understanding how each architecture performs under different scenarios. We assess key performance indicators such as average response time, throughput, and maximum latency. The results are visualized in Figure 7, which shows the mean response time, throughput, and latency with error bars for each architecture.

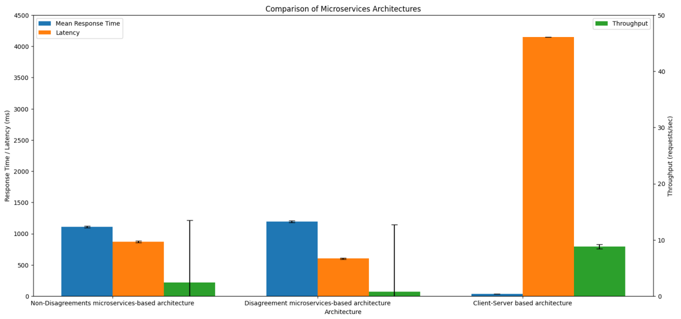

Figure 7: Mean Response Time, Throughput, and Latency with Error Bars for Different Architectures.

8.1 Client-Server Architecture

- Average Response Time: 36.9 ms
- Throughput: 8.8 req/sec
- Max Latency: 4152.3 ms

The client-server architecture demonstrated exceptional performance metrics with low latency and high throughput. Its direct communication model between clients and servers minimized overhead and efficiently handled concurrent requests, making it suitable for applications with predictable workloads.

8.2 Microservices Architecture

Non-Disagreements Scenario

- Average Response Time: 1108.9 ms
- Throughput: 2.4 req/sec
- Max Latency: 869.9 ms

Disagreement Scenario

- Average Response Time: 1192.0 ms
- Throughput: 0.8 req/sec
- Max Latency: 601.5 ms

The performance of the microservices architecture exhibits notable differences between scenarios with and without client disagreements, which can be understood by examining how the architecture handles inter-service communication and model aggregation in federated learning.

9 DISCUSSION

The design strategies employed in our federated learning setup effectively address critical challenges such as data partitioning, update management, adaptive learning, handling disagreements, and containerization. By partitioning data to ensure each client works with a unique subset, we enhance scalability and training efficiency while preserving data integrity. The iterative aggregation of model updates from clients, coupled with periodic aggregation to mitigate client drift, ensures the global model remains accurate and robust. Adaptive and transfer learning techniques facilitate faster convergence and reduce computational burdens, while our mechanisms for handling client disagreements promote fairness and model integrity. Containerization with Docker streamlines deployment and scalability, providing a consistent execution environment and simplifying integration with infrastructure frameworks.
In comparing the client-server and microservices architectures within the context of federated learning, several key observations and insights emerge from the performance metrics obtained using Apache JMeter.

9.1 Performance Metrics

The client-server architecture demonstrates remarkable performance with an average response time of 36.9 ms, high throughput of 8.8 req/sec, and a maximum latency of 4152.3 ms. These metrics highlight the efficiency of the client-server model, which benefits from direct communication between clients and servers. This design minimizes overhead and effectively manages concurrent requests, making it well-suited for applications with predictable workloads where fast response times and high throughput are critical.
In contrast, the microservices architecture exhibits significantly higher average response times in both scenarios compared to the

client-server model. Specifically, the non-disagreements scenario shows an average response time of 1108.9 ms, and the disagreement scenario shows an even higher average response time of 1192.0 ms. Throughput is also lower in the microservices model, with 2.4 req/sec in the non-disagreements scenario and 0.8 req/sec in the disagreement scenario. Maximum latency values are moderate in the non-disagreements scenario at 869.9 ms and lower in the disagreement scenario at 601.5 ms.
These differences are primarily attributed to the added complexity of inter-service communication in the microservices architecture. The need for multiple service calls to fulfill a single request can increase response times and reduce throughput. Despite these performance challenges, microservices offer advantages in terms of modular development and deployment flexibility. The ability to independently scale services helps optimize resource utilization and adapt to varying operational needs, even though the initial performance metrics are less favorable compared to the client-server model.
In the microservices based architecture non-disagreements scenario, clients have aligned objectives and the FL process is done to train one global model, leading to more consistent local model updates. This consistency facilitates several advantages: lower average response time, higher throughput, and moderate maximum latency. With all clients working towards a similar goal, the aggregation of their updates results in a more accurate and stable global model. This accuracy translates to faster response times when the model is deployed and queried, as the predictions are more reliable and require less computational overhead. The efficient aggregation of consistent updates means the system can handle more requests per second, reflecting increased throughput. The relatively lower maximum latency suggests that the system experiences fewer extreme delays, likely due to the smoother integration of client updates.
Conversely, in the microservices based disagreement scenario, clients have divergent objectives and the FL process is done to train more than one global model (one model for each group), leading to more variability in their local model updates. This disagreement introduces complexities such as higher average response time, lower throughput, and lower maximum latency. The central server must perform more aggregation processes from various clients. This often requires additional computation and more sophisticated algorithms, which can increase the overall response time of the system. The need to handle and mitigate conflicts between clients slows down the system's ability to process requests efficiently, indicating reduced throughput.
It is important to note that we did not implement a client-server architecture for the disagreement scenario. In a client-server setup, clients with divergent objectives could potentially disconnect or disengage if they were required to participate in a single global model training. By opting for a microservices-based approach in this scenario, we ensure that clients can work towards separate global models, accommodating their individual objectives without compromising data integrity.

9.2 Error Handling and Resilience

Both architectures exhibit low error rates, with the microservices architecture showing a slightly higher error rate. This is due to its distributed nature, where failures in one service might impact others. However, the microservices model provides better fault isolation, as failures are contained within individual services and do not necessarily affect the entire system. This characteristic enhances the overall resilience of the microservices architecture compared to the more monolithic client-server approach.

9.3 Scalability and Flexibility

The client-server architecture faces limitations in scalability because it requires the entire application to be replicated to scale, which can be resource-intensive and less flexible. On the other hand, the microservices architecture excels in scalability by allowing individual services to scale independently based on demand. This approach optimizes resource usage and provides greater flexibility to adapt to changing requirements, despite the higher response times and lower throughput observed in the initial performance metrics.

9.4 Limitations and Threats to Validity

Use of evaluation metrics: This research underscores the significance of strategic design decisions in developing scalable, modular, and maintainable federated learning systems. By addressing challenges related to data partitioning, update strategies, adaptive learning, disagreement handling, and containerized deployment, the study provides practical insights for implementing federated learning in real-world applications.

Performance metrics limitations: The performance metrics collected, such as response time, throughput, and latency, provide valuable insights but may not encompass all aspects of federated learning performance. Metrics related to model accuracy, convergence rates, and communication efficiency might need further exploration to provide a more comprehensive evaluation.

Testing environment: The performance metrics were obtained in a controlled testing environment using Apache JMeter. Real-world conditions, such as varying network latencies, server loads, and hardware configurations, may differ significantly and impact the observed performance metrics. The results may not fully reflect the performance of the architectures in diverse production environments.

Design decision impacts: The design decisions, including data partitioning and handling disagreements, play a crucial role in the system's performance. While these strategies address key challenges, they may introduce specific limitations or trade-offs that affect overall effectiveness. For instance, while the update strategy helps maintain accuracy, it may also lead to additional computational overhead. Balancing these trade-offs is essential for optimizing federated learning systems.

10 CONCLUSION

In this work, we have explored federated learning, focusing on its application in scenarios requiring robust, decentralized, and privacy-preserving machine learning models. Our research gave insight about the design and implementation of a federated learning

system using a microservices-based architecture, leveraging Flask for API management and Docker for containerization. We addressed the challenges of model training, evaluation, and aggregation in a distributed environment, emphasizing the importance of efficient communication protocols and scalable deployment strategies.
We began by discussing the fundamental concepts of federated learning, highlighting its potential to transform data-intensive industries by enabling collaborative model training without compromising data privacy. Our work on federated learning architectures, particularly in the context of the aviation maintainance industry, demonstrated the viability of this approach in real-world applications. The system we designed showed promise in handling large-scale deployments, managing decentralized model updates, and ensuring efficient data exchanges between clients and servers.
The experimental setup and results provided valuable insights into the performance and scalability of federated learning systems. However, we also identified several limitations and areas for improvement which are highlighted in Section 10.1

10.1 Future Work

Validation of Data and Aggregation Impurity: One critical area for future research is the validation of data and the mitigation of aggregation impurity in federated learning systems. Ensuring the quality and consistency of data across distributed clients is paramount. In future studies, methods for verifying the integrity of data before and after aggregation will be explored. Techniques to detect and correct data inconsistencies or impurities during the aggregation process will be developed, aiming to improve the reliability and performance of the global model.

Reverse Engineering of Global Model Updates: Another promising direction for future work is investigating the security implications of reverse engineering global model updates from local training. Understanding the potential vulnerabilities in the federated learning process can help in developing more robust security measures. Future research will focus on analyzing the feasibility of extracting sensitive information from global model updates and devising strategies to prevent such security breaches. This will enhance the privacy-preserving aspects of federated learning and protect client data from potential adversaries.

Networking and Deployment in Different Data Centers: Future research will also explore the networking and deployment of federated learning systems across different data centers, particularly leveraging the FABRIC testbed infrastructure. Establishing efficient communication protocols and optimizing the networking setup will be crucial for scaling federated learning applications. This involves testing various networking configurations and evaluating their impact on system performance and reliability. Deploying federated learning systems in multiple data centers will also provide insights into handling latency, fault tolerance, and load balancing in a distributed environment, ultimately leading to more robust and scalable federated learning architectures.

ACKNOWLEDGMENTS

We would like to thank our KLM colleagues Jeroen Mulder and Asteris Apostolidis for their support with this research.

REFERENCES

[1] [n. d.]. https://cloud.google.com/learn/what-is-microservices-architecture#section-2
[2] [n. d.]. https://portal.fabric-testbed.net/about/about-fabric
[3] [n. d.]. *Data for Aviation.* https://dataforaviation.org
[4] [n. d.]. Federated learning on Google cloud | cloud architecture center. https://cloud.google.com/architecture/federated-learning-google-cloud
[5] 2018. https://flask-restful.readthedocs.io/en/latest/
[6] Yuan Ao and Yuchen Jiang. 2022. Manufacturing Data Privacy Protection System for Secure Predictive Maintenance. In *2022 5th International Conference on Data Science and Information Technology (DSIT).* IEEE, 1–5.
[7] Manuel Arias Chao, Chetan Kulkarni, Kai Goebel, and Olga Fink. 2021. Aircraft Engine Run-to-Failure Dataset under Real Flight Conditions for Prognostics and Diagnostics. *Data* 6, 1 (Jan 2021), 5. https://doi.org/10.3390/data6010005
[8] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. https://doi.org/10.48550/arXiv.1902.01046
[9] McMahan H Brendan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise. 2016. Communication-Efficient Learning of Deep Networks from Decentralized Data. *arXiv (Cornell University)* (Feb 2016). https://doi.org/10.48550/arxiv.1602.05629
[10] Supratik Chatterjee and Arvind Keprate. 2021. Exploratory Data Analysis of the N-CMAPSS Dataset for Prognostics. , 1114–1121 pages. https://doi.org/10.1109/IEEM50564.2021.9673064
[11] EASA. 2023. EASA Artificial Intelligence Roadmap 2.0 - A human-centric approach to AI in aviation. https://www.easa.europa.eu/en/document-library/general-publications/easa-artificial-intelligence-roadmap-20
[12] Apache Software Foundation. 2019. Apache JMeter-Apache JMeterTM. https://jmeter.apache.org/
[13] Kai Goebel, Jose Celaya, Shankar Sankararaman, Indranil Roychoudhury, Matthew Daigle, and Abhinav Saxena. 2017. *Prognostics: The Science of Making Predictions.*
[14] Konrad Gos and Wojciech Zabierowski. 2020. The Comparison of Microservice and Monolithic Architecture. 150–153. https://doi.org/10.1109/MEMSTECH49584.2020.9109514
[15] Andrew Hard, Kanishka Rao, Ramaswamy Mathews, Sushant Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. In *Proceedings of the 1st Workshop on Privacy-Aware Mobile Computing.* 1–7.
[16] C. Huang, J. Huang, and X. Liu. 2022. Cross-silo federated learning: Challenges and opportunities. *arXiv preprint arXiv:2206.12949* (2022).
[17] Prathap Irudayaraj and Saravanan P. 2019. Adoption Advantages Of Micro-Service Architecture In Software Industries. *International Journal of Scientific & Technology Research* 8 (09 2019), 183–186.
[18] Jan Konečný, Hugh Brendan Mcmahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. *arXiv (Cornell University)* (Oct 2016). https://doi.org/10.48550/arxiv.1610.05492
[19] Panagiotis Korvesis. 2017. *Machine Learning for Predictive Maintenance in Aviation.* Ph. D. Dissertation.
[20] Weimin Lai and Qiao Yan. 2022. Federated Learning for Detecting COVID-19 in Chest CT Images: A Lightweight Federated Learning Approach. In *2022 4th International Conference on Frontiers Technology of Information and Computer (ICFTIC).*
[21] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. 2021. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering* (2021).
[22] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60. https://doi.org/10.1109/MSP.2020.2975749
[23] Dhurgham Mahlool and Mohammed Abed. 2022. A Comprehensive Survey on Federated Learning: Concept and Applications.
[24] Deeksha Negi and Pragya Jaiswal. 2024. Technological Advancement Shaping the Future of Aviation. *International Journal of Research Publication and Reviews* 5, 4 (Apr 2024), 5348–5351. https://doi.org/10.55248/gengpi.5.0424.1064
[25] Adrian Nilsson, Simon Smith, Gregor Ulm, Emil Gustavsson, and Mats Jirstrand. 2018. A Performance Evaluation of Federated Learning Algorithms. *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning* (Dec 2018). https://doi.org/10.1145/3286490.3286559
[26] Suraj Rajendran, Jihad S. Obeid, Hamidullah Binol, Ralph D'Agostino, Kristie Foley, Wei Zhang, Philip Austin, Joey Brakefield, Metin N. Gurcan, and Umit Topaloglu. 2021. Cloud-Based Federated Learning Implementation Across Medical Centers. *JCO Clinical Cancer Informatics* 5 (Dec 2021), 1–11. https://doi.org/10.1200/cci.20.00060

[27] Lendina Rushani and Festim Halili. 2022. Differences Between Service-Oriented Architecture and Microservices Architecture. *International Journal of Natural Sciences: Current and Future Research Trends* 13 (Apr. 2022), 30–48. https://ijnscfrtjournal.isrra.org/index.php/Natural_Sciences_Journal/article/view/1089

[28] Mikael Sabuhi, Petr Musilek, and Cor-Paul Bezemer. 2024. Micro-FL: A Fault-Tolerant Scalable Microservice-Based Platform for Federated Learning. *Future Internet* 16, 3 (Feb 2024), 70–70. https://doi.org/10.3390/fi16030070

[29] Tasneem Salah, M. Jamal Zemerly, Chan Yeob Yeun, Mahmoud Al-Qutayri, and Yousof Al-Hammadi. 2016. The evolution of distributed systems towards microservices architecture. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*. 318–325. https://doi.org/10.1109/ICITST.2016.7856721

[30] Nada Salaheddin and Nuredin Ahmed. 2022. MICROSERVICES VS. MONOLITHIC ARCHITECTURES [THE DIFFERENTIAL STRUCTURE BETWEEN TWO ARCHITECTURES]. *MINAR International Journal of Applied Sciences and Technology* 4 (10 2022), 484–490. https://doi.org/10.47832/2717-8234.12.47

[31] Micah J Sheller, Grant A Reina, Brandon Edwards, Joseph Martin, and Spyridon Bakas. 2020. Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. *Scientific reports* 10, 1 (2020), 1–12.

[32] Yong Shi, Yuanying Zhang, Yang Xiao, and Lingfeng Niu. 2022. Optimization Strategies for Client Drift in Federated Learning: A review. *Procedia Computer Science* 214 (2022), 1168–1173. https://doi.org/10.1016/j.procs.2022.11.292 9th International Conference on Information Technology and Quantitative Management.

[33] James B Simon, Dhruva Karkada, Nikhil Ghosh, and Mikhail Belkin. 2023. More is Better in Modern Machine Learning: when Infinite Overparameterization is Optimal and Overfitting is Obligatory. *arXiv (Cornell University)* (Jan 2023). https://doi.org/10.48550/arxiv.2311.14646

[34] Michal Staňo, Ladislav Hluchý, Martin Bobak, Peter Krammer, and Viet Tran. 2023. Federated Learning Methods for Analytics of Big and Sensitive Distributed Data and Survey. 000705–000710. https://doi.org/10.1109/SACI58269.2023.10158622

[35] Stan Stefanov. 2023. *Automating the Centralized-to-Federated Transition for the NASA C-MAPSS Dataset.* Ph. D. Dissertation.

[36] Sonam Tyagi, Ishwari Singh Rajput, and Richa Pandey. 2023. Federated learning: Applications, Security hazards and Defense measures. (Mar 2023). https://doi.org/10.1109/dicct56244.2023.10110075

[37] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 1–19.

[38] Wenti Yang, Naiyu Wang, Zhitao Guan, Longfei Wu, Xiaojiang Du, and Mohsen Guizani. 2022. A Practical Cross Device Federated Learning Frame-work over 5G N etworks. *IEEE Wireless Communications* (2022), 1–8. https://doi.org/10.1109/mwc.005.2100435

[39] Yue Zhao, Lianqing Liu, Liangzhen Lai, Naveen Suda, Damon Jay Civin, and Vikas Chandra. 2018. Federated Learning with Non-IID Data. *arXiv (Cornell University)* (Jun 2018). https://doi.org/10.48550/arxiv.1806.00582

Received 28 July 2024