Informatics

Federated Learning Systems in the Industrial Internet of Things

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Thomas Blumauer-Hießl
Matrikelnummer 01126115

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Dr.-Ing. Stefan Schulte

Diese Dissertation haben begutachtet:

Javid Taheri

Lin Wang

Wien, 17. März 2025

Thomas Blumauer-Hießl

Informatics

Federated Learning Systems in the Industrial Internet of Things

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Thomas Blumauer-Hießl
Registration Number 01126115

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr.-Ing. Stefan Schulte

The dissertation has been reviewed by:

Javid Taheri	Lin Wang

Vienna, March 17, 2025

Thomas Blumauer-Hießl

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Thomas Blumauer-Hießl

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 17. März 2025

Thomas Blumauer-Hießl

Acknowledgements

During my PhD studies, I received immense support from numerous individuals who shared their professional expertise and scientific abilities. This greatly aided my growth in the research and development field.

I want to thank Prof. Dr.-Ing. Stefan Schulte for being my supervisor and providing valuable feedback and advice at all times. This not only helped to improve my scientific output but also fostered my personal growth.

Deep gratitude is sincerely dedicated to Dr. Daniel Schall, who supported me with his professional experience. Leading the Distributed AI Research Group at Siemens, Daniel facilitated access to invaluable resources and engaging projects. His guidance and inspiration have been pivotal in my research process and the development of industrial prototypes. As a researcher in this group, I want to thank my colleagues and co-authors Jana, Safoura, Thomas, Elias, Alex, and Michael for great discussions, brainstorming sessions, and support in implementing and running our systems.

I am very grateful to Prof. Javid Taheri, PhD (Queen's University Belfast and Karlstad University) and Prof. Dr. Lin Wang (Paderborn University) for reviewing this thesis and providing very valuable input.

I want to thank my family for supporting my educational journey from the very beginning and for always encouraging me to pursue my interests with dedication and resilience.

Many thanks to my friends, who accompany me through important phases of my life, including my time at school and university.

Finally, my PhD studies would not have been possible without the immense emotional support from my wife, Victoria. Every paper that forms the foundation of this thesis required not only scientific contributions and innovative ideas but also a significant amount of work to deliver the results. I wish to express my deepest gratitude to Vicy for her unwavering mental support, her inspiring motivational speeches, and her unconditional support throughout my research work and my life in general.

The work presented in this thesis was primarily funded by Siemens AG. Additionally, Chapters 6 and 7 have been co-funded by the Austrian Research Promotion Agency (FFG).

Kurzfassung

Im industriellen Internet der Dinge (engl. Industrial Internet of Things (IIoT)) liefern eine Vielzahl von Sensoren und Geräten Daten von Maschinen und Produktionsprozessen. Die Anwendung von Analysemethoden und trainierten Machine Learning (ML)-Modellen auf den Daten liefert aufschlussreiche Informationen zur Optimierung der Prozesse und somit zur Steigerung der Produktivität. Mit dem Aufkommen von Federated Learning (FL) werden dezentrale Geräte in die Lage versetzt, ML-Modelle kollaborativ zu trainieren, ohne Daten zu teilen. Hierfür aggregiert ein Server die von den Geräten gelieferten Modellupdates (d. h. Mittelwertbildung der Parameter). Dieser Ansatz wahrt die Privatsphäre der lokalen Geräte und deren Benutzer und bietet die Möglichkeit, die Qualität der ML-Modelle zu steigern, da Wissen zwischen den beteiligten FL-Teilnehmern geteilt wird, ohne Rohdaten offenzulegen.

Die Anwendung von FL in der Industrie und die Entwicklung von ganzheitlichen FL-Systemen für industrielle Benutzer führen jedoch zu offenen Fragen. Insbesondere besteht ein Bedarf an Werkzeugen und Diensten zur Entwicklung, Bereitstellung und Ausführung von FL-Lösungen und deren Integration in Industrieanwendungen. Aufgrund der unterschiedlichen Konfiguration der einzelnen Geräte und Variationen in den zugrundeliegenden Prozessen und überwachten Maschinen, folgen die gesammelten Daten oft auch heterogenen Datenverteilungen. Daher kann das Training eines globalen Modells durch Mittelwertbildung zu einer unzureichenden Modellqualität führen. Ein weiteres Problem ist die optimale Bereitstellung von Clients (Software auf den dezentralen Geräten) und ihre Auswahl für FL-Prozesse. Dabei ist es für effiziente FL-Systeme entscheidend, die Latenz zu minimieren und die Modellqualität zu steigern. Ebenso gibt es noch kaum Integrationen von FL-Lösungen in Geschäfts- und Fertigungsprozessen, da viele Voraussetzungen für den operativen Einsatz noch nicht erfüllt sind.

Diese Arbeit präsentiert Systeme und Konzepte, die FL-Dienste für Problemstellungen im IIoT-Bereich anbieten, optimieren und deren Einsatz in betriebsübergreifenden Setups erläutert. Des Weiteren präsentiert diese Arbeit FL-Aktivitäten entlang des gesamten Lebenszyklus von der Entwicklung bis zur Anwendungsintegration und dem Betrieb. Zur Optimierung der Bereitstellung von FL-Lösungen stellen wir Ansätze für das Deployment und die Selektion von FL-Clients vor. Dabei berücksichtigen wir Edge-, Fog- und Cloud-Plattformen. Schließlich werden Voraussetzungen für FL-Lösungen und bestehende

Problemstellungen aus der Praxis erfasst, um geeignete FL-Blueprints für die industrielle Kollaboration abzuleiten.

Die vorgestellten FL-Systeme und Algorithmen werden anhand von Industriedaten aus verteilten Datenquellen validiert. Dabei werden Verbesserungen der trainierten FL-Modelle im Vergleich zum isolierten und lokalen Training von ML-Modellen und bereits bestehenden FL-Ansätzen gezeigt. Die Ergebnisse der Client-Selektions-Optimierung zeigen, dass die relevanten Metriken (z. B. Modellqualität und Latenz) zum jeweilig besten Ansatz der Klasse konvergieren. Unsere FL-Blueprints adressieren Problemstellungen von 13 Unternehmen und liefern Architekturen zur Lösung praxisrelevanter Künstlicher Intelligenz (KI)-Probleme.

Abstract

In the Industrial Internet of Things (IIoT), a multitude of sensors and devices deliver data from machines and production processes. Applying data analysis and trained Machine Learning (ML) models on the data provides insightful information for optimizing the processes and therefore increasing productivity. With the rise of Federated Learning (FL), decentralized devices are enabled to collaboratively train ML models without sharing data. For this, a server aggregates (i.e., averages) the model updates delivered by devices. This approach preserves privacy of the local clients and offers the potential to boost the performance of ML models, since knowledge is shared between involved FL participants without revealing raw data.

The application of FL in industry and the development of holistic FL systems for industrial clients still face open issues. In particular, there is a need for providing tools and services to develop, deploy, and run FL solutions and integrate them into industry applications. Due to the diverse setup of individual devices and variations in underlying processes and monitored machines, the collected data often follows heterogeneous data distributions as well. Consequently, the training of a global model through averaging can result in insufficient model quality. Another problem is the optimal deployment of clients and their selection for FL runs. Essentially, it is crucial for efficient FL systems to minimize the response time and boost the performance of the model. Furthermore, there is still a gap to integrate FL solutions into business and manufacturing processes, since there is only limited information on practical implementation guidelines available for industry.

This thesis presents systems and concepts that provide FL services to clients in the IIoT considering strategies for optimizing models towards individual data distributions. We support FL activities along the whole lifecycle from development to application integration. To optimize deployments of FL solutions, we present approaches for client placement and selection for FL clients by the server. For this, we consider edge, fog, and cloud platforms. Finally, we survey the prerequisites and requirements for FL solutions in practice and collect respective pain points to derive suitable FL blueprints for industrial collaboration.

We evaluate the FL system(s) and proposed algorithms on industrial data from decentralized data sources and show improvements in the performances of the trained FL models as compared to isolated local ML and plain FL. The results of our deployment and client selection optimization demonstrate flexibility and convergence to the relatively best platform for FL clients aiming for optimizing e.g., model performance and response time. Our FL blueprints address the surveyed pain points of 13 companies and provide architectures for solving identified Artificial Intelligence (AI) problems with FL.

Contents

Publications

The research presented in this thesis is based on published work as listed in the following.

- Hiessl, T., Schall, D., Kemnitz, J., Schulte, S. (2020). Industrial federated learning – requirements and system design. In: De La Prieta, F., et al. Highlights in Practical Applications of Agents, Multi-Agent Systems, and Trust-worthiness. The PAAMS Collection. PAAMS 2020. Communications in Computer and Information Science, vol 1233. Springer.
- Hiessl, T., Lakani, S. R., Kemnitz, J., Schall, D., and Schulte, S. (2022). Cohort-based federated learning services for industrial collaboration on the edge. Journal of Parallel and Distributed Computing, 167, 64-76.
- Hiessl, T., Lakani, S. R., Ungersboeck, M., Kemnitz, J., Schall, D., and Schulte, S. (2023). Lifecycle management of federated learning artifacts in industrial applications. In 2023 IEEE 7th International Conference on Fog and Edge Computing (ICFEC) (pp. 7-15). IEEE.
- Blumauer-Hiessl, T., Schulte, S., Rezapour Lakani, S., Keusch, A., Pinter, E., Kaufmann, T., and Schall, D. (2024). Federated learning deployments of industrial applications on cloud, fog, and edge resources. In 2024 IEEE 8th International Conference on Fog and Edge Computing (ICFEC) (pp 19-26).
- Blumauer-Hiessl, T., Fessl, A., Breitfuss, G., Schall, D., and Schulte, S. (2024). Federated learning solution blueprints for use cases surveyed in Austrian industries. In 2024 IEEE 26th Conference on Business Informatics (CBI) (pp 80-89).

Additional published work can be found in the list below.

- Hiessl, T., Karagiannis, V., Hochreiner, C., Schulte, S., and Nardelli, M. (2019). Optimal placement of stream processing operators in the fog. In 2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC) (pp. 1-10). IEEE.
- Hiessl, T., Hochreiner, C., and Schulte, S. (2019). Towards a framework for data stream processing in the fog. Informatik Spektrum, 42, 256-265.

- Schall, D., Hiessl, T., and Kemnitz, J. (2021). Method and system for operating a technical installation with an optimal model. U.S. Patent Application No. 17/237,157.

- Ding, A. Y., Peltonen, E., Meuser, T., Aral, A., Becker, C., Dustdar, S., Hiessl, T., ... and Wolf, L. (2022). Roadmap for edge AI: A Dagstuhl perspective. ACM SIGCOMM Computer Communication Review, 52(1), 28-33.

- Holly, S., Hiessl, T., Lakani, S. R., Schall, D., Heitzinger, C., and Kemnitz, J. (2022). Evaluation of hyperparameter-optimization approaches in an industrial federated learning system. In Data Science–Analytics and Applications: Proceedings of the 4th International Data Science Conference–iDSC2021 (pp. 6-13). Springer Fachmedien Wiesbaden.

- Ungersböck, M., Hiessl, T., Schall, D., and Michahelles, F. (2023). Explainable federated learning: A lifecycle dashboard for industrial settings. IEEE Pervasive Computing, 22(1), 19-28.

- Keusch, A., Hiessl, T., Joksch, M., Sündermann, A., Schall, D., and Schulte, S. (2023). Edge intelligence for detecting deviations in batch-based industrial processes. In 2023 IEEE 21st International Conference on Industrial Informatics (INDIN) (pp. 1-8). IEEE.

- Kemnitz, J., Weissenfeld, A., Schoeffl, L., Stiftinger, A., Rechberger, D., Prangl, B., ... and Schall, D. (2023). An edge deployment framework to scale AI in industrial applications. In 2023 IEEE 7th International Conference on Fog and Edge Computing (ICFEC) (pp. 24-32). IEEE.

- Lakani, S. R., Blumauer-Hießl, T., Eder, J., and Schall, D. (2024). Computer-implemented method and system for operating a technical device. 18/641,572.

- Keusch, A., Blumauer-Hiessl, T., Furutanpey, A., Schall, D., and Dustdar, S. (2024). Platform-agnostic MLOps on edge, fog and cloud platforms in industrial IoT. International Conference on Web Information Systems and Technologies (pp. 71-79). SciTePress.

- Lakani, S. R., Blumauer-Hießl, T., Meixner, S., and Schall, D. (2025). Computer-implemented method for operating a technical device using a model. US Patent App. 18/711,992.

CHAPTER 1

Introduction

In the Internet of Things (IoT), an ever-increasing number of heterogeneous objects (things) is connected to the Internet providing applications for multiple domains like industrial manufacturing, smart cities, smart grids, healthcare, retail, and logistics [GBMP13]. For this, data is generated by sensors and analyzed accordingly to derive insightful information for improving the life of modern societies. In recent years, Machine Learning (ML) has improved industrial manufacturing and process automation significantly, e.g., in fault classification, quality estimation, and soft sensing [GSDH17]. For instance, value-added and ML-based services and applications like Condition Monitoring (CM) for production machines can be used to facilitate timely and cost-efficient maintenance actions throughout the lifetime of a machine [CNG+18, BLS+13].

To achieve these benefits, high-quality ML models require a significant amount of training and testing data. This data is often considered privacy-sensitive and needs to be protected from outside parties [SS15]. In addition, there is often a lack of ground truth data, which is referred to as the missing labeled data problem [AB14]. This is especially the case in industrial scenarios since some labels can only be assigned to a dataset, if critical and potentially rare events (e.g., a machine breakdown) are observed [DODGS19].

Since the required large and labeled datasets should not be shared with centralized servers for ML, a privacy-preserving way of knowledge sharing between collaborating devices is desired. This is the goal of Federated Learning (FL), as introduced by McMahan et al. [MMR+17]. FL is an approach for transferring knowledge as model parameters (e.g., weights of neural networks) between (edge) devices without revealing raw data. For this, models are trained locally on edge devices and are then uploaded to an aggregation server that fuses model parameters e.g., by averaging. The aggregation can be secured so that the server only learns the sum of all updated models instead of individual model contributions from clients using secure aggregation [BIK+17]. After aggregation, the model is returned to the clients for evaluation. This process is executed repeatedly either until a pre-defined number of communication rounds or a provided level of quality (e.g., classification accuracy) is reached.

To optimize collective model training, various FL algorithms, e.g., [MMR+17, YCKB18, PKH19, YWL20], have been researched, with a specific focus on model aggregation and client selection. Applying the concept of FL for industrial machines that are distributed over multiple factories and facing heterogeneous environmental and operational conditions is referred to as Industrial Federated Learning (IFL) [HSKS20].

At this time, still many challenges [KMA+21b] exist in FL which apply especially to industrial clients [HSKS20] as described in the following subsections.

1.1 Problem Statement

1.1.1 Federated Learning Services for Independent Edge Devices

The initiation of FL training procedures usually involves the selection of clients by a server, which functions as a central authority [KMA+21b, MMR+17]. Considering an edge computing environment, clients are typically deployed on edge devices that process data (i.e., train models based on local data) in close proximity to data sources to ensure data privacy and low latency [SD16]. Servers are utilized to oversee and monitor the distributed training process, managing the connected edge devices [MPP+21]. Figure 1.1 depicts how FL can be applied considering a central model aggregation and several edge devices that train on data generated by machines to e.g., improve ML model quality for detecting anomalies in a production process.

In most FL approaches, the central authority defines the learning task, for instance, choosing the ML model, hyperparameters, and the FL algorithm [KMA+21b]. However, in industrial applications, users such as machine operators on production lines have unique needs concerning business collaborations when partnering with other organizations to enhance and sustain machine performance [HSKS20]. For example, FL-based applications can be deployed to a significant number of edge devices that can be located at multiple sites of a given organization or even multiple organizations. Hence, although the same type of problems are solved using a given application, diverse clients with their heterogeneous edge devices, respective users, operating conditions, and data sources yield individual restrictions that need to be considered in FL. Furthermore, clients can impose limitations to only collaborate with certain partners or ensure collaboration with a specified minimum number of partners to enhance the likelihood of actual enhancements in the ML model. For this, approaches using central authorities are not fully applicable in industrial setups and need to be adapted according to the participating clients.

A service-oriented system is required that enables individual clients to develop and submit ML models to the server, facilitating FL-as-a-Service (FLaaS) [KKP20]. This approach allows a group of independent, collaborating clients to apply FL on these models and to consider individual restrictions. Finally, the resulting model can be used on their devices.

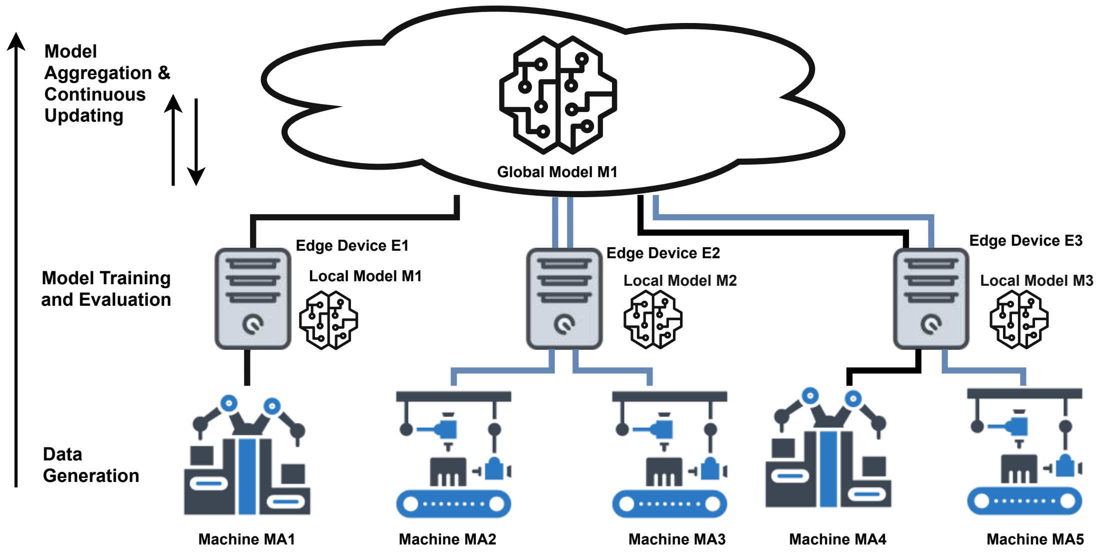


Figure 1.1: FL with industrial machines; Machines generate data that are used for training and evaluating ML models on edge devices; Models are periodically sent to the server for aggregation and distribution to other edge devices for knowledge sharing.

1.1.2 Skewed Data

A significant issue in FL is the problem of heterogeneous data distributions [KMA+21b], a situation particularly prevalent in industrial sectors when machines function under diverse operational conditions with different configurations [HSKS20].

For instance, considering different liquids that are pumped in industrial processes, one can observe different error cases occurring over time detected by models using vibration patterns as input data [ZHLZ10]. Typically, one can observe that input data (e.g., vibration data) is varying across clients which is referred to as *feature distribution skew* [HPMG20]. Additionally, varying labels (e.g., error cases) can often be observed as well, which is referred to as *label distribution skew*. This phenomenon corresponds to the non-Independent and Identically Distributed (IID) data problem, which is a general issue in ML and especially in FL [KMA+21b]. In non-IID settings, poor model quality can be observed by individual clients as the model is validated on their local data after FL has been applied for all clients [KMA+21b]. Therefore, FL systems need to provide mitigation strategies in implemented FL algorithms.

1.1.3 Federated Learning in Machine Learning Lifecycles

Model engineers usually start a ML model lifecycle by identifying the underlying problem that needs to be solved by learning a high quality model. To achieve this, typically a lifecycle considers multiple stages like client instrumentation, model training (including hyperparameter optimization), (federated) model evaluation, and deployment [KMA+21b].

Hence, stages in ordinary ML lifecycles need to be updated if FL comes into play as additional complexity is added due to decentralization of tasks, heterogeneous resources and privacy preservation.

Bonawitz et al. [BEG+19] propose to start the FL process by centrally creating FL plans that define the model and the instructions to be executed on every client. In the proposed lifecycle, they consider an extended review and safety check of FL plans and an evaluation of resources available to edge devices even before training is started. This ensures a safe and robust execution as the training process is started on every involved edge device.

Furthermore, architectural patterns for client selection, clustering, model aggregation, deployment selection of suitable edge devices, etc. can be applied in FL lifecycles [LLZ+22].

In contrast to the centralized initiation of FL, FLaaS-based approaches allow clients to upload models and to trigger FL with other clients by using an Application Programming Interface (API) provided by the FL server [KKP20]. So, training, validation, and deployment is influenced by individual clients that are not only responsible for their local ML lifecycle anymore. Hence, the quality of local data and the model resulting from FL also affects other clients. This poses challenges when it comes to adopting and validating models that have been created by possibly unknown clients with different data distributions and potentially different contributions to the overall FL model.

Based on these considerations, there is a need to incorporate local ML lifecycle aspects (e.g., model validation, pre- and postprocessing of data) into the overall management of an FL lifecycle. In particular, customization steps for involved clients are needed to e.g., adapt the FL model or its outputs to local use cases before it is applied on the client's IoT data. This can be relevant to provide quality-assured outputs as the FL model is integrated into client applications. Overall, a suitable FL lifecycle facilitates model quality, customization, and integrability for FL applications. For this, a holistic lifecycle management (including individual lifecycle steps, the extension of existing ones with new architectural patterns [LLZ+22], or a novel composition of steps) are needed.

1.1.4 FL Deployments in Heterogeneous Environments

To solve generic or domain-specific problems on a larger scale, data scientists create FL solutions and provide reusable models across multiple applications and deployments [ARPS20, KZDB16]. These applications can be downloaded and deployed to available compute resources (e.g., edge devices). This enables users (e.g., machine operators, facility managers) to run the models and to react to outputs like identified anomalies and critical conditions of connected things.

However, considering that FL is implemented in respective applications, (i.e., several decentralized deployments of the application collaborate on a common model) additional complexity is introduced, since the ML model is distributed to heterogeneous deployment locations [ATOS+21]. Diverse resource capabilities result in differences in metrics like response time (caused by changing communication and training time) [CAZ+20], energy consumption [RMB23], and model performance. To optimize these metrics, it is necessary to investigate how client deployments can be managed, taking into consideration available compute resources (e.g., cloud, fog, edge), as well as application constraints, in order to achieve best results.

1.1.5 Implementation of FL in Industries

Multiple FL systems, approaches and frameworks have been proposed and use cases in industry have been demonstrated [KMA+21a]. For example, applications are used in pharmaceutical research and development [OÁP+23], quality inspection in the manufacturing industry [BHKS+23], or text prediction while typing on mobile keyboards [HRM+18]. In manufacturing, robots can learn from each other to better execute their tasks by applying FL on underlying ML models, which also holds for use cases in cooperative driving, where detection and localization of safety-relevant events is important [SNB+21]. However, the implementation for practical use in real-world environments, such as industry, is lagging behind. Most approaches presented in the literature are applied in lab settings, while the transfer of FL into practice is not thoroughly investigated. This can be concluded from the survey of Pandya et al. [PSJ+23], where the surveyed approaches from the several application fields (e.g., power grids, industries, healthcare) focus on various technical aspects (e.g., communication, privacy) but not on the actual embedding into businesses. For this, the current state of applying AI in a collaborative setup need to be surveyed across different businesses. Based on that, identified use cases and implementation challenges need be addressed with FL designs and implementation plans for given industry types.

1.2 Research Questions

Summarizing the described aspects from Section 1.1, challenges of FL services for clients in heterogeneous deployment locations, dealing with non-IID data, FL lifecycles, and industrial implementation strategies are identified as core issues in real-world scenarios. To tackle these challenges, this work considers the research questions below. We will revisit them in Chapter 8 and assess how our contributions address the research questions with suitable solutions.

RQ1 *How can systems and algorithms improve the operation of FL clients in real-world environments with skewed data and independent edge devices?*

To facilitate collaboration of multiple clients and improve their local ML models, an FL system need to provide suitable services and apply algorithms for addressing the non-IID problem. Additionally, individual and independent edge devices need to be considered with respective requirements and a possibility for controlling the overall FL process instead of have it controlled by a central authority. For this, we aim for designing an FL system and evaluate it on real-world data.

RQ2 *How should ML/FL lifecycles (individual stages and overall composition) be updated, enhanced, or managed by FL systems, to ensure high-quality models, customizability for individual clients, and integrability of FL into applications?*

The adaptation of ML lifecycles to decentralized and privacy-preserving FL lifecycles requires a system that enforces the FL specifics discussed in Section 1.1. In particular, data scientists and machine operators need to be supported to develop, deploy, integrate and use FL solutions. Hence, we seek a solution by providing end-to-end support starting with the development of FL solutions until the application can make use of trained FL models.

RQ3 *How can FL deployments be optimized on heterogeneous compute resources with respect to response times, energy consumption and model performance?*

As an FL system is about to be deployed across multiple locations, various deployment options with different capabilities still exist. Available compute resources (e.g., on-premise edge devices) can be selected or new ones can be used (e.g., on-demand cloud instances). Trade-offs in response time and energy consumption need to be investigated and potential implications to the model performance should be understood as FL is applied. Hence, we aim to analyze different client deployments and seek for algorithms to optimize the respective metrics.

RQ4 *What are the current pre-requisites for FL in industry and how can FL solutions be provided to overcome existing challenges?*

Since FL is primarily utilized in research and has limited implementation in industry, our goal is to identify and understand the barriers to its adoption. We want to understand the pre-requisites and challenges of different companies in multiple industries to derive suitable and actionable architectures that can be implemented. Furthermore, we seek for collaboration models that demonstrate how different FL parties are involved in the implementation and operation of FL systems.

1.3 Scientific Contributions

C1 *IFL requirements and system design*

The first contribution considers the analysis and discussion of requirements for an FL system operating in the context of the IoT in industrial environments. Based on identified requirements, a system design is proposed to faciliate collaborative ML model training. The design comprises a domain model, an architecture of FL components and a description of supported workflows. Contribution C1 has originally been published in [HSKS20] and is presented in Chapter 3.

C2 *FL services for independent edge devices with non-IID data*

The second contribution includes the presentation of the *IFL Process*, a multi-step approach that is provided as a service which encompasses algorithms for on-demand FL requests and the subsequent building of groups of clients (= cohorts) with similar data to address the non-IID data problem. The system is evaluated on industrial datasets demonstrating improved model performance in classification accuracy when FL is applied to groups of similar clients. We presented contribution C2 originally in [HRK$^+$22] and provide an extensive description in Chapter 4.

C3 *Lifecycle management for FL artifacts*

The third contribution consists of the design of an FL lifecycle (i.e., the *IFL Lifecycle*) that supports the software development, publishing, deployment, and execution of FL solutions using customizable software artifacts introduced as *IFL Templates*. The implementation of the *IFL Lifecycle* includes key components such as the *IFL Core*, a Python library for FL, and a distributed clustering algorithm called Federated Clustering (FedClust). The approach is integrated into an industrial CM application, enabling the provision and consumption of extensible FLaaS. The *IFL Lifecycle* is evaluated with exemplary clustering scenarios using industrial data. We demonstrate the function of the lifecycle management and the impact of different levels of privacy in FL. Contribution C3 has originally been presented in [HRLU$^+$23] and is considered in Chapter 5 in this thesis.

C4 *Optimization of client selection on cloud, fog, and edge resources*

The fourth contribution introduces an IFL deployment architecture that enables FL execution in a multi-location setup, utilizing cloud, fog, and edge platforms for client training. We propose an optimization approach called *IFL Opt*, based on Integer Linear Programming (ILP), for client selection in FL. This approach takes into account factors such as energy consumption, time, and model performance. Furthermore, for model performance optimization, we propose *XMeanCohorting*, an FL algorithm where cohorts of clients from different deployment locations are created to faciliate the learning process. The evaluation considers a real-world industrial dataset from the electronics manufacturing industry, comparing different FL client deployments on cloud, fog, and edge nodes with the optimized approach. We provide a detailed description of contribution C4 in Chapter 6 which has originally been published in [BHSRL$^+$24].

C5 *FL blueprints for industry personas*

The fifth contribution provides a collection of use cases, challenges, and requirements for industrial AI applications based on interviews conducted with 13 companies across multiple industries in Austria. This is used to create *industry personas* that represent different business types and their specific fields of industrial AI applications that can benefit from FL. Additionally, FL blueprints, which include system architectures and implementation steps, are designed to offer solutions tailored to the identified industry personas. This contribution has originally been presented in [BHFB⁺24] and is described in detail in Chapter 7.

1.4 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 provides background information and introduces main concepts that are used in subsequent chapters. Chapters 3 through 7 constitute the main matter and correspondingly address the five contributions. In Chapter 3, we present identified requirements and propose a system design for general industrial use of FL. Chapter 4 provides the detailed design, implementation and evaluation of a service-based FL system. To demonstrate how developers and operators benefit from FL lifecycle management, we provide details in Chapter 5. In Chapter 6, we propose an FL deployment architecture and evaluate optimizations of energy consumption, model performance, and response times. FL blueprints are presented in Chapter 7 for implementing FL for different industry personas. In Chapter 8, we conclude the thesis with a summary of the scientific contributions, a discussion of the research questions and an overview of the future work. The Appendix A includes a summary of the (I)FL-specific terms introduced in this thesis.

CHAPTER 2

Background

In this thesis, FL approaches and systems are presented to address challenges in an environment of decentralized data and compute resoures with the main goal to provide high-quality ML models. The following sections describe the background of this work considering basic FL concepts, compute platforms for executing FL jobs, and the IoT as main data source for FL tasks.

2.1 Federated Learning

FL is a decentralized learning approach based on ML initially proposed by McMahan et al. [MMR$^+$17]. Multiple devices are involved with the goal to train a ML model in a privacy-preserving way. For this, each device holds local data samples that are not shared with other parties (i.e., servers, other devices). This eliminates the need to exchange potentially sensitive data. The original algorithm, named Federated Averaging (FedAvg) (see Algorithm 1), considers a server and multiple clients (devices). A parametric base model m_0 is passed as input. This can either be provided by the server, acting as the central authority, or uploaded by individual clients. The server iterates over a number of communication rounds R, where a subset S_r of clients are selected and requested to train and provide an updated model (line 5). For this, the client executes an ML training procedure (line 8), considering input data X^t, the number of local iterations (epochs) E and the local model. The latter one is a parametric model with a weight matrix that is updated (trained) by applying a loss function l on the previous iteration's model and labels (expert input), provided in every batch $b \in B$ (see line 12). As the client returns the local model to the server (line 15), the server averages the model parameters (line 7) of all clients. The server weights the respective client contribution using the number of local data samples n_i. In an alternative approach, instead of iterating for R rounds, the algorithm continues to iterate until the loss function converges for all involved clients [YLCT19]. With this approach, the resulting model after R communication rounds is called the global model (m_R) considering the learning results of all involved clients.

Algorithm 1 `Federated Averaging (FedAvg)`

```
Input: parametric base model $m_0$, number of communication rounds $R$, set of clients
    $K$, fraction of clients to be considered in each round $C$. Input for each client: input
    features $X^t$, number of local data samples for the i-th client $n_i$, sum of all local data
    samples $n$, local epochs $E$, learning rate $\alpha$
    Server execution:
 1: for round $r = 1, ..., R$ do
 2:     $c \leftarrow \max(C \cdot K, 1)$
 3:     $S_r \leftarrow$ (random subset of $c$ clients)
 4:     for client $t_i \in S_r$ do
 5:         $m_{r+1}^{t_i} \leftarrow ClientUpdate(t_i, m_r)$
 6:     end for
 7:     $m_{r+1} \leftarrow \sum_{i=1}^{S_r} \frac{n_i}{n} m_{r+1}^{t_i}$
 8: end for

    ClientUpdate(t,m) // Run task $t$ on client
 9: $B \leftarrow$ ($X^t$ split into batches of size)
10: for epoch $e \in 1..E$ do
11:     for batch $b \in B$ do
12:         $m \leftarrow m - \alpha \nabla l(m, b)$
13:     end for
14: end for
15: return $m$ to server
```

The use of FL provides many advantages for the involved parties [KMA+21a, MMR+17]. First, the aspect of privacy preservation is considered by design, since no raw data of the clients leave the device. The model updates shared with the server are parameters that are focused on the learning task rather than the raw data. Second, the performance of the underlying model can be improved as compared to isolated ML training with only the local data as training input. In these local setups, often insufficient data and labels are present, which could lead to poor model quality [AB14]. To avoid this, FL is used as a collaborative approach that allows to share the learning results between clients. Hence, performance metrics used in training and validation can significantly be improved. For example, one commonly used metric for classification tasks is *accuracy*, that is the number of correctly classified data samples divided by the number of overall data samples [PF13]. Third, the workload in the training process is parallelized by using the compute resources provided by client devices. Hence, scaling up the number of involved devices can lead to significant improvements in a short period of time, instead of using only a few devices. Fourth, no raw data upload to a central instance is needed, which reduces network load [TBZ+19].

2.1.1 FL Settings

In a setting without FL, individual learners apply ML and in most cases Deep Learning (DL) techniques to train parametric models (e.g., neural networks). For this, training procedures are executed on local compute resources to derive a model that infers a target variable with sufficient quality. The inference can be carried out on other resources. Another common learning setting is to upload data from a decentralized fleet of devices (e.g., data sources) to a central data center, which distributes the learning process to a cluster of compute nodes in the same data center [DCM+12, KMA+21a]. This is referred to as *data center distributed learning* that benefits from parallelization of the training workload and typically very fast networks.

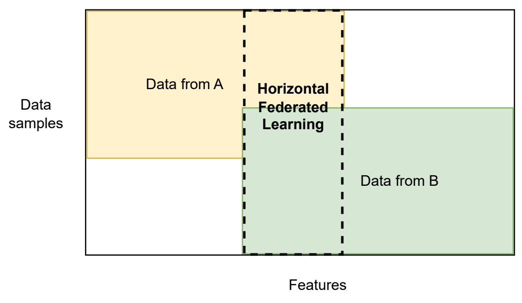
In FL, there are two important settings that are recognized, *cross-silo FL* and *cross-device FL* [KMA+21a]. *Cross-silo FL* considers a few (2-100) clients with an own data silo (e.g., database or larger data centers storing an organization's data). The involved clients are almost always available and typically connect to a central server to participate in FL. In contrast to data center distributed learning, cross-silo FL never discloses data of individual clients or nodes. *Cross-device FL* makes use of a large distributed fleet of IoT or mobile devices (up to 10^{10}) [KMA+21a]. The availability of the involved devices is decreased since only a fraction of the devices are online at any given time. The primary bottleneck is often the communication, since network conditions can fluctuate (e.g., for mobile devices), which is more stable in the cross-silo FL case. Approximately 5% of the devices involved in an FL communication round drop out due to battery, network, or idleness requirements [KMA+21a]. In both FL settings, typically, the data is very heterogeneous following different data distributions considering the multitude of devices. Each client can only access the local data and no data samples from other clients. This is not given in data center distributed learning, where data can be shuffled and balanced across the nodes.

In our approaches presented in this work, we primarily consider the cross-silo FL setup with less than 100 clients and industrial IoT data.

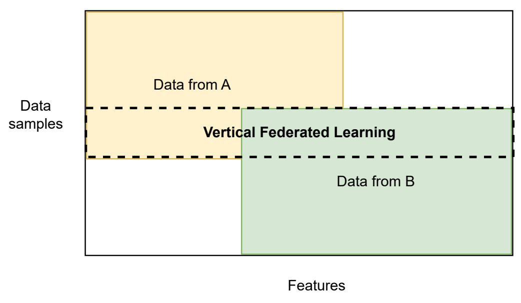
2.1.2 Horizontal and Vertical FL

To train on local data, the FL system need to be aware of the data structure and data partitioning on the client's side [YLCT19]. For this, two types of data partitioning approaches are distinguished i.e., Horizontal Federated Learning (HFL) and Vertical Federated Learning (VFL).

HFL considers the same structure (schema) of data samples on every device. The term HFL is derived from the idea of splitting a dataset horizontally and therefore providing partitions of different data samples with the same features [ZXB+21]. For example, considering a schema of vibration data measured from an IoT sensor mounted on an industrial pump, we would have three dimensions $<x, y, z>$ and potentially a label $<$*anomaly status*$>$ created by an expert telling whether the data sample is



(a) Horizontal Federated Learning



(b) Vertical Federated Learning

; ;

Figure 2.1: Data partitioning types in FL [YLCT19]

anomalous [KBG+22]. Hence, every client can train on the same type of ML model (e.g., a neural network predicting the anomaly status based on x, y, z).

Figure 2.1a depicts the idea of HFL, showing two parties A and B with an overlapping area of features.

In contrast, VFL mainly considers different data schemes [YLCT19] but overlapping data samples as depicted in Figure 2.1b. The term VFL is based on the idea of splitting a dataset vertically and therefore providing partitions of different features with the same data samples [ZXB+21]. For instance, two companies operate in the same region or have a business partnership and therefore share data from the same persons. While one company focuses on retail transactions of a person, the other company considers bank account data. In the industrial domain, this can be data from a partial product (e.g., test data from an engine) for one party and quality inspection data for the integrated end product (e.g., test data from a vehicle) [BHFB+24]. The learning process considers two different models (e.g., neural networks with different structure and output) [YLCT19]. Both parties train their model on local data and share (encrypted) outputs with each other (often with a trusted third-party collaborator). Depending on the architecture of the models and how they are connected (e.g., output of the model from company A is considered as input for the model of company B), the backpropagation is initiated accordingly so that both parties can adapt their models to reduce the loss from the overall loss function. For model inference, both parties need to collaborate as well to share (encrypted) model output and provide a global result (e.g., overall product quality).

In this thesis, we mainly consider HFL settings, and show how FL models perform as multiple clients with the same data schema collaborate. In Chapter 7, we provide FL solution architectures for industrial collaboration covering also VFL.

2.1.3 Non-IID, Personalization and Cohorts

In Section 1.1.2, we have addressed the problem of non-IID data and identified poor model quality as an issue if a global FL model is trained on clients with heterogeneous data distributions.

To provide the mathematical background, we highlight the definition of IID in the following [Kle06]. The random variables $X_1, X_2, ..., X_n$ are IID if they satisfy two conditions:

- Independence
  The joint probability distribution of any subset of these variables is the product of the individual distributions. i.e., $[P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i)]$
- Identity
  All random variables follow the same probability distribution, i.e., $[P(X_i) = P(X_j)]$ for all $i, j \in 1, 2, ..., n$

The setup of IID is rarely given in real-world data as for example IoT data from industrial domains [HSKS20]. To address the non-IID problem in FL and to derive an FL model that is suitable for being applied at the client's side, the concept of *personalization* is used [TYCY23]. In general, personalization is about updating the global FL model to better fit to local client data distribution so that the model is close to the local optima. Since the local (edge) client device often belongs to a user, the term personalization has been used to express the adaptation to a personal setup [AASC19]. Assuming a non-IID setup, the global model (i.e., the weights) can drift towards the local optima of a subset of clients and therefore move away from another subset of clients that have different data distributions [TYCY23]. Furthermore, the global model (weights) can move towards an average of local optimas (by the definition of FedAvg), which is not suitable for all of them. For this, it is the task of an FL system to provide strategies to personalize the model to come as close as possible to the local optimum of each client, still facilitated by suitable knowledge sharing using FL.

Throughout this thesis, we make use of the concept of *FL Cohorts*, or in short, cohorts. *FL Cohorts* address model personalization by grouping together clients with similar data distributions within the same *FL Population*, or in short, population, which is the set of all clients joining an FL server using the same ML model for learning [HSKS20, HRK+22]. To be precise, an *FL Cohort* groups the *FL Tasks*, or in short, tasks, that represent the client requests to participate in FL. Since a task refers to exactly one client, it is valid to describe an *FL Cohort* as a group of clients, which we do in this work. Similarly, this holds for *FL Populations*.

The grouping enhances the model accuracy by allowing clients to share updates only within a specific subset of clients. Thus, *FL Cohorts* prevent potential inaccuracies from knowledge transfer among dissimilar clients. In our presented approaches (see

Chapters 3-6), we provide cohort-based strategies for personalizing FL models so that the validation metrics (e.g., accuracy) for involved clients are improved as compared to plain vanilla approaches such as global FL model with weight averaging or individual ML.

2.1.4 Privacy Preservation Approaches

As mentioned earlier, FL provides the advantage of supporting a certain level of privacy preservation by design since no raw data is shared with other parties. Although only model updates are shared with the server, potential attackers could try to infer data samples by analyzing the previous model, the model update, and the resulting model after one communication round [KMA+21a]. For this, approaches with suitable counter measures have been developed that are commonly used in FL systems.

A prominent concept is *differential privacy*, that enables privacy preservation in FL by allowing model learning without disclosing personal information [WLD+20]. Differential privacy has been developed in the context of statistical reports and queries with the goal to allow for querying statistical aggregates and still keeping individual information private [DR+14]. A typical approach to ensure privacy is to add noise to the data. In FL settings, clients perturb their information (i.e., model weights that should be shared) and send the randomized versions to the server [WLD+20]. The aggregation on the server leads to a valid FL model to be used by the involved parties. Nevertheless, the model performance can be impacted by the noise since there is a trade-off between the level of privacy (which increases as more noise is added) and the time of converging to a desired model performance.

Another approach for preserving privacy in FL is provided by *secure aggregation* protocols [BIK+17]. For this, the client's encrypt the models before they are shared with the server. The server can still perform the aggregation and relevant operatons on the masked models, without knowing the actual parameters. This reduces the risk that adversaries can leak information.

2.1.5 FL Model Lifecycle

Similar to ML, FL models follow a certain lifecycle starting with *initiation* and ending with *deployment* and *monitoring* [LLZ+22]. Figure 2.2 depicts the operations that are done with the model along the lifecycle. In contrast to ML, the decentralized nature of FL requires to *broadcast* the base model to all clients. Subsequently, all clients start to execute the *training* and *transmission* step and the server performs model *aggregation*. If the *evaluation* shows that the model has not converged, another training iteration is executed. Otherwise, the deployment and monitoring of the FL model can be started by the clients. As the model is in use (i.e., processing data tuples and providing inference results), the monitoring activity can initiate a new FL model lifecycle iteration if the model performance degrades.

In Section 5, we consider a broader definition of the FL lifecycle by regarding a lifecycle of model-generating objects (FL code artifacts), instead of FL models. This helps to

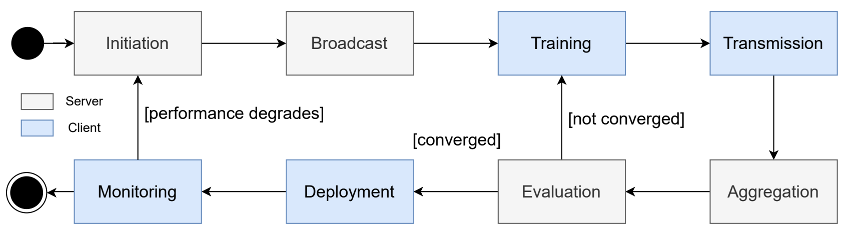


Figure 2.2: FL model lifecycle [LLZ+22]

address the development and integration of FL solutions.

2.1.6 Basic FL Architectures

To implement FL systems, typically, client-server-based architectures are chosen for distributed training and central model aggregation [BEG+19]. For this, the communication style is mostly synchronous, as the server waits for the contributions of selected clients. However, with the rapid increase of FL research in recent years, various systems have been proposed and several architectural patterns arised. Lo et al. [LLZ+22] surveyed important patterns for practical use, as summarized in the following.

To manage clients, the *client registry* pattern provides the possibility to hold the state of clients and their FL tasks on the server after initial registration. The *client selector* pattern enables the server for selecting clients to increase the performance and system efficiency. To separate clients into groups with respect to e.g., data distribution, available resources or geolocation, the *client cluster* pattern has been proposed. Regarding model aggregation, multiple patterns exist, i.e., *asynchronous aggregator* for asynchronous global model updating whenever a client provides an update without waiting for all contributions in a communication round, *decentralized aggregator* for settings without a central authority (server) in place, and *hierarchical aggregator* for adding additional aggregators on the edge layer to perform partial aggregation and therefore improve efficiency. Further patterns have been applied in the area of training and model management, i.e., *training configurator* for providing a platform to the user to run and control the FL training processes, *heterogeneous data handler* for providing strategies to handle skewed data, and *message compressor* to reduce the size of exchanged messages increasing communication efficiency.

In this thesis, we design FL systems that implement a subset of these patterns (e.g., client cluster in Chapter 4 or client selector in Chapter 6) and demonstrate how to tackle the underlying challenges with them.

2.1.7 FLaaS

In this work, we provide FL systems supporting the concept of FLaaS. For this, we consider definitions of Kourtellis et al. [KKP20] and Mazzocca et al. [MRM+23], who specify relevant properties associated with FLaaS. We also add additional properties to the definition of FLaaS as it was originally published in [HRK^{+}22].

To motivate FLaaS, Kourtellis et al. [KKP20] state that without a dedicated service model for FL, independent third parties are unable to collaborate on shared ML models while protecting user privacy. Additionally, the lack of a service model makes it challenging for developers and data scientists to deploy FL solutions quickly and easily, which hinders the broader adoption of FL.

Hence, we describe FLaaS as a service model for FL supporting the following properties of Kourtellis et al. [KKP20]: 1) FLaaS eliminates the need for algorithm development and tuning (e.g., in Chapter 4, a service provides pre-defined algorithms that can be selected by the client). 2) Collaborative model building for single and multiple applications is supported (i.e., models can be shared between different clients via API if permitted). 3) Clients can describe the data types that are considered as model input in the training stage. 4) FLaaS addresses the management of privacy and permissions (e.g., defining access to the models and data).

Mazzocca et al [MRM^{+}23] add the following property: 5) Clients are enabled to define their specific requirements for FL training, e.g., quality metrics, aggregation strategies, and the number of nodes involved.

Finally, we consider two more properties from the work on cohort-based FL to complete the idea of FLaaS (see Chapter 4): 6) Clients can explicitly request participation in FL rounds (e.g., by using an API), instead of automatically being invoked by a central authority (server). 7) Clients can be grouped into populations or cohorts via API considering client requirements (e.g., selected algorithm, business partner criteria) and data types if requested. With this, collaboration with other parties can be controlled by the client.

2.2 Internet of Things

In the IoT, physical things connect to the Internet, which enables to access sensor data remotely and control parts of the physical world with suitable actuators [KS22]. The technology already exists for more than a decade, but the increase in the number of deployed things (often referred to as *smart objects*) enables more applications and large data collections that can be processed with powerful compute platforms. Things are considered to be either directly equipped with sensor technology or can be identified by devices in the near proximity by using e.g., Radio Frequency Identification (RFID) tags or (bar)codes [KS22].

Integrating data from various things into applications enables a multitude of use cases that span across logistics, energy savings, physical security and safety, the medical domain, and industry [KS22]. The latter for example, is often about predictive maintenance and CM use cases, where machines are monitored with the goal to predict critical conditions and potential breakdowns [CBZ20]. If identified correctly, suitable maintenance actions are derived, which could save high costs that can occur in cases of severe mechanical issues or production outages.

IoT sensors are often characterized by technical constraints as limitations in e.g., processing power, battery power, storage and memory [KKG+20]. Furthermore, IoT solutions and platforms are characterized by providing capabilities for *real-time processing* of sensor data to gain valuable insights, *scalability* to provide reliable systems as more sensors are introduced, and the capability to handle *heterogeneity*, since multiple different device types might be integrated.

2.2.1 Protocols

To enable connectivity of things, a number of protocols are considered to communicate data [LXZ15]. For example, *Bluetooth*, *ZigBee*, and *WLAN* exist for communication in short-range (10m-100m) wireless networks. As things data is captured by compute resources with basic data processing capabilities (e.g., IoT gateways), several protocols are used to upstream data to application services (e.g., in the cloud). These protocols include *WebSocket* as simultaneous two-way communication approach, the request-response-based Constrained Application Protocol (CoAP) and MQTT. In this thesis, we use MQTT in our FL system implementations to collect data from sensors (e.g., vibration sensors) and to implement an FL protocol between client and servers for exchanging model updates. For this, we make use of the publish-subscribe protocol provided by MQTT. MQTT enables reliable communication based on the Transmission Control Protocol (TCP) [LXZ15]. Furthermore, the protocol offers comparably low transmission times and energy consumption, especially for larger message sizes.

2.2.2 IoT Data Processing and Analysis

To process and analyze the generated data of sensors and other IoT devices, multiple processing techniques and platforms can be considered as depicted in Figure 2.3 [KKG+20]. While basic data processing like data denoising, outlier detection, missing data imputation, and data aggregation can be executed directly by resource-constrained sensors or IoT/edge devices, higher level data analysis and learning techniques often require more powerful compute resources provided e.g., by cloud platforms or dedicated Virtual Machines (VMs) as part of fog or edge computing environments. For this, we provide details on compute platforms in Section 2.3 and also highlight the characteristics when it comes to apply FL.

In the IoT, most of the collected data is time series data that is sampled in fixed-time intervals or irregularly [SDO18, CMF20]. The temporal context of IoT data implies that there can be a dependency between different points in time (e.g., data at time t correlates with data from time t-n)

Given a time series of IoT data, a prominent use case is to detect anomalies [CMF20]. For this, techniques reach from simple approaches like threshold-based outlier detection (either setting manual thresholds or adding statistical mean and variance as corridor for acceptable values) to complex statistical and ML-based models. For this, regression

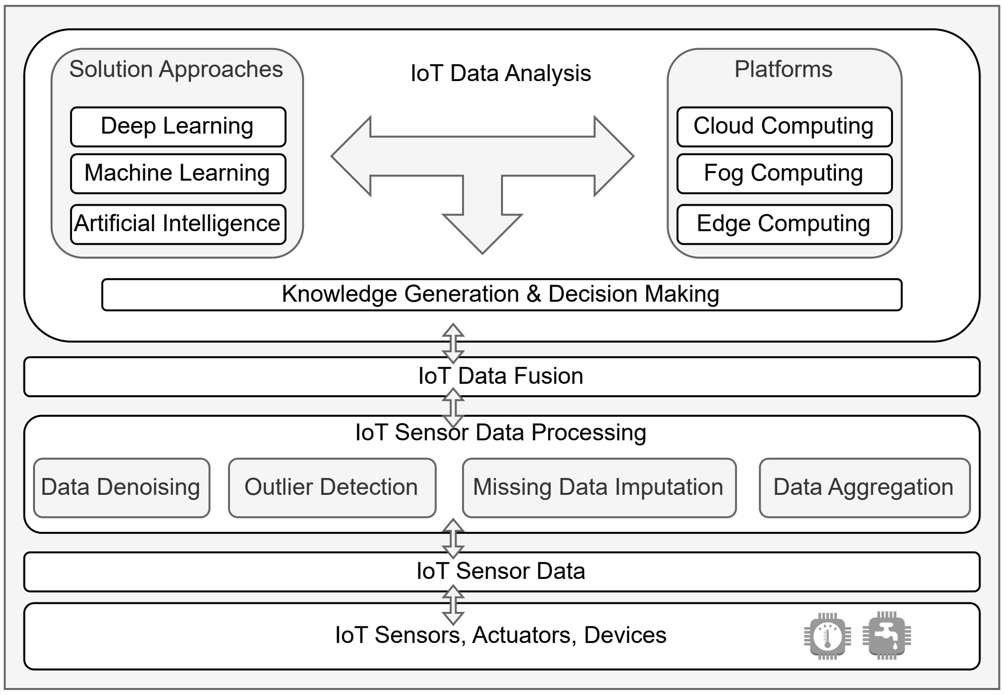


Figure 2.3: IoT data processing [KKG+20].

models can be defined to predict future time series values based on the historical series. This can identify trends towards future anomalies and reveals potentially unacceptable outliers. Supervised ML-based approaches can be applied by assigning labels to historical spans of time series values. For example (based on a similar case in Chapter 4), spans of 10 seconds of time series data can be labeled with a specific machine condition (e.g., failure, upcoming issue, idle). These labels can be used for training e.g., a neural network to classify the machine condition. For this, characteristical variables (e.g., mean, variance, kurtosis, frequencies, amplitude) are computed from the 10 seconds signal and used as features for the training.

2.2.3 Industrial IoT

Similarly, time series analyses can be beneficial for the industry, as IoT devices collect data from various assets (machines) and production processes. This field is known as IIoT, which Boyes et al. [BHCW18] provide a comprehensive definition for:

> "A system comprising networked smart objects, cyber-physical assets, associated generic information technologies and optional cloud or edge computing platforms, which enable real-time, intelligent, and autonomous access, collection, analysis, communications, and exchange of process, product and/or service information, within the industrial environment, so as to optimise

overall production value. This value may include: improving product or service delivery, boosting productivity, reducing labour costs, reducing energy consumption, and reducing the build-to-order cycle"

Given an IIoT setup and the pervasion of edge computing resources in the proximity of the production process, it seems natural to apply FL to further improve locally trained models. For example, FL can facilitate anomaly detection as individual clients operate on *data islands* [LGN$^+$21]. Data islands are isolated data collections that cannot be centralized as edge devices are not willing to share the potential privacy sensitive data.

In this thesis, we train and apply models on IoT time series data from industrial data sources (see Chapters 4, 5 and 6). With FL, we aim for improving model quality (e.g., classification accuracy) in a collaborative way considering multiple distributed data sources as e.g., vibration sensors mounted on pumps. The value for the underlying industrial machines is a timely detection of issues that can be inferred from the measured vibration signals.

2.3 Compute Platforms: Cloud, Fog, and Edge Computing

To execute FL jobs, different decentralized compute resources can be utilized for client (training) and server (aggregation) jobs. For this, multiple platforms can be chosen if not already available on-premise resources are used. In the following sections, we introduce the conceptual basics of *cloud, fog* and *edge computing* and discuss their suitability for executing data processing jobs like FL.

2.3.1 Cloud Computing

Cloud computing is a paradigm that enables on-demand access to networked compute resources with rapid (de)provisioning [MG$^+$11]. Different deployment models exist to host and to provide these resources i.e., *private cloud* for individual use for a single organization, *community cloud* for a collective use between multiple organizations, *public cloud* for general use for the public, and *hybrid cloud* that is a mix between two or more of the aforementioned deployment models. The well-known public cloud model (e.g., AWS[1], Azure[2], Google Cloud Platform[3]), provides compute resources in large data centers and multiple regions world-wide on the premises of the cloud provider. On the contrary, private cloud solutions often need to be operated by the using company itself.

One of the main advantages of cloud computing over on-premise resources is the efficent resource management [QLDG09]. For this, resources are often pooled to be efficiently shared with other customers as they are needed on-demand. To use compute resources,

[1]`https://aws.amazon.com`, accessed 2024-07-22
[2]`https://azure.microsoft.com/`, accessed 2024-07-22
[3]`https://cloud.google.com/gcp`, accessed 2024-07-22

often VMs are used (Infrastructure-as-a-Service (IaaS)) to address compute loads demanded by e.g., application developers. Any of the used products is typically paid per use, and as demanded resources need to be increased (e.g., memory, CPU, storage) the price per unit of time is updated as well.

Similarly, one can conclude that the role of application developers and e.g., the role of data scientists, also seek to directly utilize Platform-as-a-Service (PaaS) [MG+11] resources as databases, data lakes, analytics and ML workflows to provide e.g., FL training procedures on data collected in the cloud. The PaaS model supports to abstract from the underlying infrastructure like the operating system. Hence, easy usage of software components for data processing underlines the suitability for FL jobs in the cloud. In particular, the scalability of resources addresses varying training loads in FL. However, some FL organizations might hesitate to upload training data to storages apart from on-premise locations.

2.3.2 Edge Computing

To move compute resources closer to the potential end user, the concept of edge computing has been introduced [KAH+19]. Edge computing provides high-bandwidth, low-latency and realtime access to data from the close proximity. One aspect that differentiates edge computing from cloud computing is the location of the servers. While in cloud computing they are located in larger data centers reachable over the Internet, for edge computing, the servers are located at the edge of the network. Having compute and storage resources available at the edge of the network, enables latency-sensitve applications for use cases dealing e.g., with (video) analytics, data caching, or location-specific services. In particular, there can be a benefit for applications that facilitate near realtime preprocessing on the edge and partial forwarding of compute loads and data to the cloud for highly scalable processing [Sat17].

Edge devices also complement the IoT, since data generated by nearby edge devices can be processed on the edge with 95% less latency and energy consumption [SD16]. IoT devices (e.g., air quality sensors, electrical devices) benefit from the edge servers as data processing entities (data consuming and data producing). This is because a massive load in the core network is avoided as compared to transferring large volumes of IoT data continuously to cloud servers and back [AATA+21]. Using edge devices in these scenarios can significantly save costs, since data transfer is typically charged by cloud providers.

Furthermore, edge computing also integrates heterogeneous and potentially resource-constrained devices [LHL+21]. These devices can also dynamically join the network and disconnect at a later stage. For this, additional resource management capabilities are needed to e.g., efficiently assign resources to the tasks that need to be executed.

Two examples for edge systems are the open source platform *EdgeX Foundry*[4] and the

[4] `https://www.edgexfoundry.org/`, accessed 2024-07-22

commercial *Siemens Industrial Edge* platform[5]. Both platforms provide connectors to read from various IoT data sources (e.g., industrial machines) and to forward data to target systems (e.g., cloud storages or other IoT devices) [VSP+23]. Furthermore, they provide the capability to host custom software applications in containers via Docker[6]. In general, a central management server (i.e., *Industrial Edge Management (IEM)* and *EdgeX Server*) is used to distribute and deploy applications to the distributed edge devices, each running *Industrial Edge Device (IED)* or *EdgeX*. Device platform services provide capabilities for monitoring applications, storing data, and controlling data flows.

The suitability for executing FL jobs on edge devices is inherently given by the design, since edge nodes are close to data sources and IoT data can be streamed into the storage of edge devices for training. However, the compute capabilities are often heterogeneous throughout different device types [KAH+19], which can be a bottleneck if resource-intensive training jobs are executed, and therefore delay the model training [LLZ+22].

Edge Intelligence

Since FL is designed to train and apply AI where the data is collected, often close to the data sources on the edge, it can be categorized as an edge intelligence approach. Taheri et al. [TDZD23] define edge intelligence as “running AI on the edge” aiming for optimal model performance, cost, privacy, and reliability. This involves the four pillars of edge intelligence that are caching, training, inference, and offloading. As such, edge systems are tasked with coordinating and executing these tasks on the edge instead of in the cloud, which can be used for offloading tasks if necessary.

Service Placement

Edge computing often faces the service placement problem, where service entities (data processing and storage) need to be deployed to nodes in the edge network [WJH+21]. This is a complex problem since potentially conflicting factors are present such as e.g., communication latency (to users and other nodes), heterogeneous maintenance and operational costs, and available resources such as GPU or CPU power. The goal is to optimize the Quality of Service (QoS) or resulting costs for the deployed application(s). Specifically, in the case of FL, a Collaborative Edge Service Placement (CESP) problem arises when considering multiple dependent edge nodes that need to share model updates as identified by Wang et al. [WJH+21]. This could involve a server or multiple clients running on the edge and collaborating on a global model. In Chapter 6, we deploy FL clients to various platforms (i.e., edge system) and focus on optimizing selected clients. For this, we aim to optimize e.g., energy consumption (cost-related) and model performance (QoS-related).

[5] `https://www.siemens.com/global/en/products/automation/topic-areas/industrial-edge.html`, accessed 2024-07-22

[6] `https://www.docker.com/`, accessed 2024-07-22

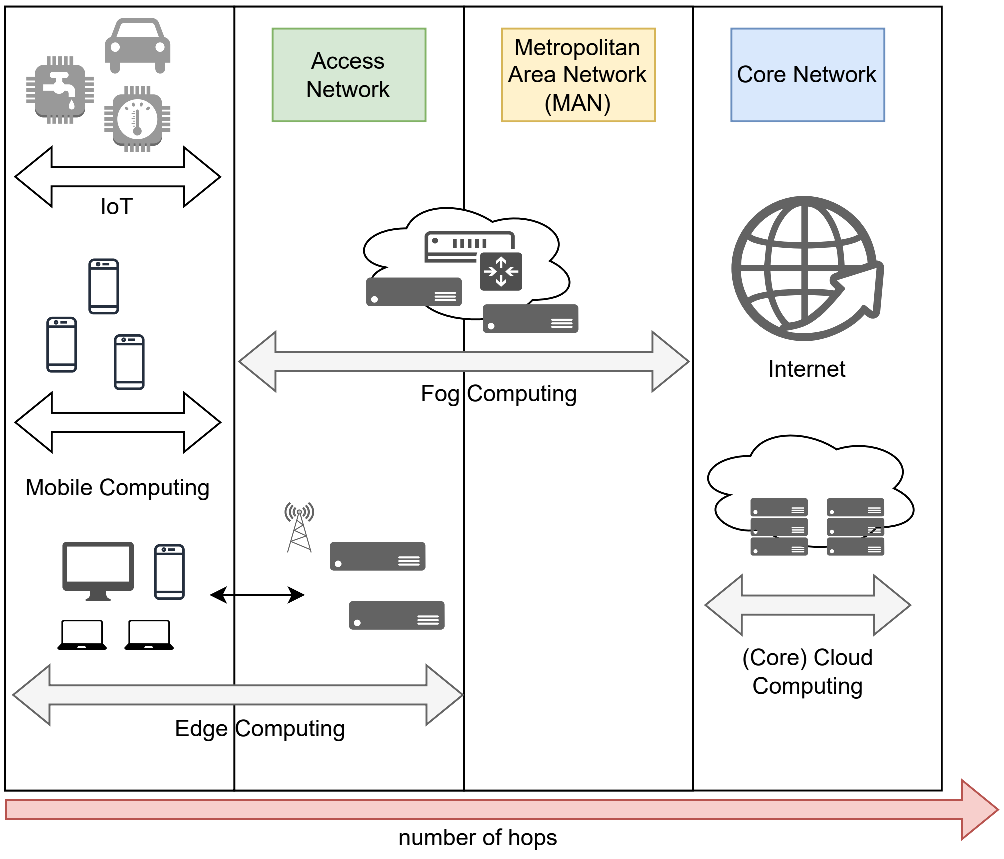


Figure 2.4: Mapping of edge, fog and cloud computing in the network context [YFN+19].

2.3.3 Fog Computing

In 2012, Bonomi et al. [BMZA12] initially coined the term *fog computing* as highly virtualized platform that provides storage and compute resources somewhere between end devices and cloud data centers. As the term edge computing has arised in the mid 2010s, the boundary to the term fog computing has been refined [YFN+19]. In principle, edge computing considers resources in the local network and access network, while fog computing resources are located in the access network and the higher-level Metropolitan Area Network (MAN) as depicted in Figure 2.4. Hence, fog computing provides resources anywhere from things (IoT) to cloud.

A fog system follows design goals such as low latency, efficiency (energy), and generality [YHQL15]. The latter refers to general APIs and services that are provided to users for hosting and operating applications. This is enabled by virtualization and APIs for e.g., system monitoring, network management, VM scheduling, load balancing, authentication, location services, communication services to support IoT sensor data ingest, and cloud agents for connecting to cloud platforms.

Basic architectures for fog computing are hierarchical and flat [KS20]. A hierarchical architecture incorporates two, three, or even more layers, with resource-constrained nodes positioned on layers near IoT devices, resulting in low latency. More powerful resources are located on the top layers, which results in higher latency. This pattern can be generalized to consider cloud nodes in the top layer and edge nodes on the lowest layer. A flat architecture is organized in neighbourhoods with diverse capacities [KSLP19]. In this configuration, even powerful nodes are placed close to IoT devices, facilitating the execution of resource-demanding tasks while still maintaining low latency.

Given a set of (fog) nodes and IoT-based services, the problem of service placement needs to be addressed [SNS+17]. The available nodes need to be selected and services need to be deployed. For example, a suitable placement (i.e., deployment) of the services would be to minimize expected execution times and communication latencies.

In FL, potentially FL servers run in geospatially distributed fog nodes, while individual clients are deployed close to the data-generating sensors on edge devices [SMD21]. For this, the fog can be used to dynamically select the best fog nodes for model aggregation, considering geo distances, response times, and availability of sufficient compute resources. Furthermore, even multiple fog nodes can be used for providing redundant aggregation services, avoiding a single point of failure and facilitating the reliability of the system. In other settings, one can apply fog resources to temporarily store data which is used for client training.

In Chapter 6, we analyze the impact of deploying and selecting clients from cloud, fog, and edge platforms. We investigate trade-offs between the platforms and analyze the execution of FL jobs on heterogeneous resources. For this, we consider IIoT data, three real-world compute platforms and multiple scenarios to provide a realistic comparison.

CHAPTER 3

Industrial FL – Requirements and Systems Design

3.1 Introduction

Industrial manufacturing systems often consist of various operating machines and automation systems. High availability and fast reconfiguration of each operating machine is key to frictionless production resulting in competitive product pricing [VDP96]. To ensure high availability of each machine, often CM is realized based on ML models deployed to edge devices e.g., indicating anomalies in production [HPH18]. The performance of these ML models clearly depends on available training data, which is often only available to a limited degree for individual machines. Increasing training data might be realized by sharing data within the company or with an external industry partner [CKD+04]. The latter approach is often critical as vulnerable business or private information might be contained.

FL enables to train a ML model on multiple local datasets contained in local edge devices without exchanging data samples [SS15]. As introduced in Chapter 2, in this privacy-preserving approach, typically a server receives parameters (e.g., gradients or weights of neural networks) from local models trained on decentralized devices and averages these parameters to build a global model [MMR+17].

To solve the aforementioned challenges of successfully applying ML models in industrial domains, FL needs to be adapted. Therefore, the integration of operating machines and its digital representations named *assets*[1] need to be considered as depicted in Figure 3.1. Assets generate data on the shop floor during operation. Edge devices record this data to enable training of ML models e.g., in the field of anomaly detection aiming to identify

[1] `https://documentation.mindsphere.io/MindSphere/apps/asset-manager/introduction.html`, accessed 2024-07-22

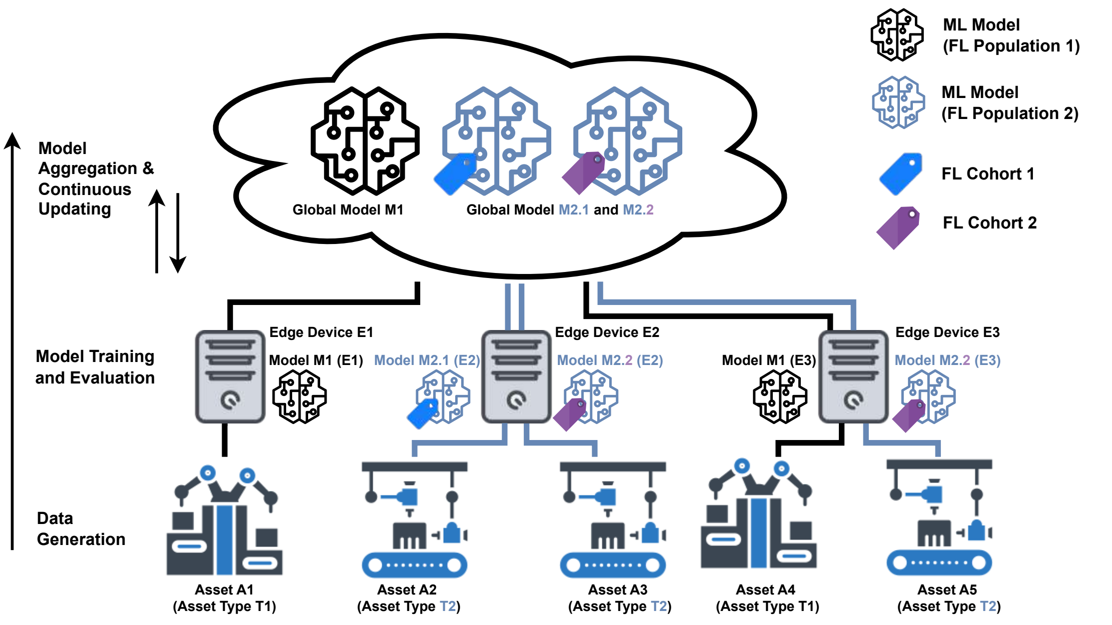


Figure 3.1: FL with industrial assets; Assets generate data that are used in learning tasks for *ML Models* executed on edge devices; Learning tasks for *ML Models* based on the same asset type are part of an *FL Population*; Learning tasks for *ML Models* with similar data are part of an *FL Population* subset named *FL Cohort*; Knowledge transfer in continuously evaluated and updated *FL Cohorts* ensures optimal collaboration with respect to model performance and business partner criteria

abnormal behavior of machines in production. To improve the model quality, FL is applied by aggregating model parameters centrally in a global model e.g., in the cloud, and sending out updates to other edge devices. Typically, all models of local learning tasks corresponding to the same ML problem are updated. This set of tasks is called an *FL Population*. In the depicted industry scenario, an *FL Population* is a group of all learning tasks (or clients), whereas the models are trained on asset data with the same data scheme, which is typically ensured if assets are of the same asset type, e.g., learning tasks of models *M2.1 (E2)*, *M2.2 (E2)*, and *M2.2 (E3)* belong to *FL Population 2*, since they are based on assets of *Asset Type T2*. In contrast, learning tasks of models *M1 (E1)* and *M1 (E3)* belong to *FL Population 1*. However, assets even of same asset type could face heterogenous environmental and operation conditions which affect recorded data. Due to these potential dissimilarities in asset data, negative knowledge transfer can be caused by the model updates which decreases model performance [TS10]. For this, industrial FL systems need to consider *FL Cohorts* as subsets of an *FL Population*. This enables knowledge sharing only within e.g., *FL Cohort 2* including *M2.2* models using similar asset data.

This chapter has originally been published in 2020 [HSKS20]. At that time, applying FL in industrial settings had not been addressed profoundly. Therefore, this work has aimed to provide an initial foundation in terms of requirements, a basic notation, a generic system design and a set of workflows (WFs) that should be supported by IFL systems to meet industrial requirements. These concepts are extended, implemented and evaluated in Chapter 4.

To establish this foundation, the proposed *IFL System* aims to improve collaboration on training and evaluating ML models in industrial environments. In particular, the system supports knowledge exchange in *FL Cohorts* involving industrial clients that train on asset data. Furthermore, it needs support for continuous adaption of *FL Cohorts* as ML models evolve over time. To additionally support efficient FL with high quality of asset data, we aim for resource optimization of involved edge devices and appropriate consideration of Quality of Information (QoI) metrics [LSKW02].

Our contribution in this chapter comprises:

- List of requirements for an IFL system.
- System design with a domain model, a system architecture, and supported WFs for IFL

For this, we build on FL systems and approaches from [BEG+19, CNSR20, LWLX19, MMR+17, SS15] and incorporate industry concepts as well as experience from industrial projects.

The remainder of this chapter is organized as follows: In Section 3.2 we refer to the basic notation of FL. We review related work in Section 3.3 and subsequently present requirements of IFL in Section 3.4. The design of the *IFL System* is presented in Section 3.5 with respect to supported WFs, domain models, and architectures. We conclude in Section 3.6.

3.2 IFL Notation

Based on the discussion in Section 2.1 and the motivational example in Section 3.1, we introduce the basic notation of an IFL system. We extend the FL notation by Bonawitz et al. [BEG+19] that defines *device*, *FL Server*, *FL Task*, *FL Population* and *FL Plan*. Devices are hardware platforms as e.g., industrial edge devices or mobile phones, running *FL Clients* to execute the computation necessary for training and evaluating *ML Models*. To use FL, an *FL Client* communicates to the *FL Server* to run *FL Tasks* for a given *FL Population*. An *FL Population* is a globally unique name that identifies a learning problem which multiple *FL Tasks* have in common. An *FL Task* represents the client's request to participate in the FL process with a specified *ML Model*. The *FL Server* aggregates results (i.e., model updates), persists the global model, and provides it to *FL Clients* of a given *FL Population*. an *FL Plan* corresponds to an FL Task and represents its federated execution instructions for the *FL Server* and involved *FL Clients*. It consists of sequences of ML steps as e.g., data pre-processing, training, and evaluation to be executed by *FL Clients* and instructions for aggregating *ML Models* on the *FL Server*. Furthermore, we detail *FL Cohorts* that group multiple clients (i.e., *FL Tasks*) within the same *FL Population* and with similarities in their underlying asset data.

3.3 Related Work

3.3.1 FL Systems

In 2019, most of the FL studies had focused on federated algorithm design and efficiency improvement [ARPS20]. Besides that, Bonawitz et al. [BEG+19] built a scalable production system for FL aiming to facilitate learning tasks on mobile devices using *TensorFlow*[2]. Furthermore, *NVIDIA Clara*[3] provides an SDK to integrate custom *ML Models* in an FL environment. This system has been evaluated with data from the medical domain, considering a scenario with decentralized image datasets located in hospitals [BKKZ22].

Between 2019 and 2024, the FL systems research has been progressing and has mostly organized around the following taxonomy provided by Liu et. al [LLGL24]: Aggregation optimization, heterogeneous FL, secure FL, and fair FL. The latter one addresses fair client selection, fair model optimization and fair contribution evaluation. A few FL frameworks for running an FL system have already been existing in 2019, i.e., open source frameworks such as *PySyft*[4], *TensorFlow Federated (TFF)*[5], and *FATE*[6]. *PySyft* is a Python framework that enables operating (i.e., training) on remote data, which we use as underlying framework in Chapter 4. *FATE* is a framework developed by WeBank (China). It implements several secure computing protocols and provides a mechanism for deploying the framework (i.e., clients) in a distributed way [BKKZ22]. *TFF* is based on *TensorFlow* and enables researchers and developers to test and simulate FL algorithms on their data [BKKZ22].

In 2020, *Flower*[7] has been released. Flower supports multiple aggregation algorithms, is language- and communication-agnostic, and addresses heterogeneous clients [BTM+20].

NVIDIA FLARE[8] has been developed as part of NVIDIA Clara[9] and has been considered as its own open source project since 2021. *NVIDIA FLARE* supports the simulation, provisioning, orchestration, and monitoring of FL runs.

Open FL[10] has originally been founded by Intel and was released as open source project by VMWare, University of Pennsylvania and Flower Labs in 2023. The framework

[2]`https://www.tensorflow.org/`, accessed 2024-07-22
[3]`https://devblogs.nvidia.com/federated-learning-clara/`, accessed 2024-07-22
[4]`https://github.com/OpenMined/PySyft`, accessed 2024-07-22
[5]`https://www.tensorflow.org/federated`, accessed 2024-07-22
[6]`https://fate.fedai.org/`, accessed 2024-07-22
[7]`https://flower.ai/`, accessed 2024-07-22
[8]`https://developer.nvidia.com/flare`, accessed 2024-09-14
[9]`https://docs.nvidia.com/clara/index.html`, accessed 2024-09-14
[10]`https://openfl.io/`, accessed 2024-07-22

provides secure communication through certificates, a command line interface and a Python API as well as extensible interfaces to ensure model training with arbitrary ML libraries [BKKZ22].

3.3.2 Client Selection

Nishio et al. [NY19] optimize model training duration in FL by selecting only a subset of *FL Clients*. Since they face heterogeneous conditions and are provisioned with diverse resource capabilities, not all *FL Clients* will manage to deliver results in decent time. For this, only those who deliver before a deadline are selected in the current training round. To achieve the best accuracy for the global model, the *FL Server* may select *FL Clients* based on their model evaluation results on held-out validation data [BEG+19]. This allows to optimize the configuration of *FL Tasks* such as centrally setting hyperparameters for model training or defining the optimal number of involved *FL Clients*. These approaches are relevant to IFL systems as well. However, the *IFL System* additionally selects *FL Clients* based on collaboration criteria with respect to potential FL business partners.

Client selection has further been addressed in many approaches after 2020. Most of them optimize the model accuracy, convergence rate, fairness, and resource usage, taking into account heterogeneous FL devices [FZG+23]. For example, Abdel et al. [ASCF23], propose to select clients so that overall resource wastage (training time of clients that do not contribute) is minimized.

3.3.3 Continuous Federated Learning

Liu et al. [LWLX19] propose a cloud-based FL system for reinforcement tasks of robots navigating around obstacles. Since there exist robots that train much and therefore update *ML Models* continuously, the authors identify the need for sharing these updates with other federated robots. These updates are asynchronously incorporated in the global model to eventually enhance navigation skills of all involved robots. Based on that, in IFL the continuous updates are used to re-evaluate data similarity that is needed to ensure high model quality within an *FL Cohort*.

3.4 Requirements

In this section, we present requirements that should be covered by an *IFL System*. Based on FL system features discussed in Chapter 2, we add requirements with respect to industrial data processing and continuous adaptation of the system.

3.4.1 Industrial Metadata Management

To support collaboration of *FL Clients*, we identify the requirement of publishing metadata describing the organization and its devices. Based on this, *FL Clients* can provide criteria for collaborating with other selected *FL Clients*. Although actual raw data is not shared in FL, FL enables to adhere to company policies for interacting with potential partners. Asset models as provided by Siemens MindSphere[11] describe the data scheme for IIoT data. Since industrial *FL Clients* target to improve ML models using asset data, metadata describing the assets builds the basis for collaborating in suitable *FL Populations*.

3.4.2 FL Cohorts

As discussed in Sections 2.1.3 and 3.3.2, *FL Client* selection plays a role in FL to reduce the duration of e.g., training or evaluation [NY19]. Furthermore, client selection based on evaluation using held-out validation data can improve accuracy of the global model [BEG$^+$19]. In our experience, these approaches do not sufficiently address data generated by industrial assets and processed by *FL Clients*. For this, our approach aims for considering asset data characteristics for achieving optimal accuracy and performance for all individual client models. To this end, we identify the requirement of evaluating models in regards to similarities of asset data influenced by operating and environmental conditions. This is the basis for building *FL Cohorts* of *FL Tasks* using asset data with similar characteristics. *FL Cohorts* enable that *FL Clients* only share updates within a subset of *FL Clients*, whose submitted *FL Tasks* belong to the same *FL Cohort*. These updates probably improve their individual model accuracy better, as if updates would be shared between *FL Clients* that face very heterogeneous data due to e.g., different environmental or operating conditions of involved assets. In manufacturing industries there are situations where assets are placed in sites with similar conditions, as e.g., placing production machines onto shop floors with similar temperature, noise and other features considered in the model prediction. Nevertheless, the conditions might be different in other shop floors or factories and still there is a need to apply FL between them to improve model performance. This can especially be the case if multiple different companies collaborate. In such cases, the *IFL System* needs to build *FL Cohorts*.

3.4.3 Quality of Information

Since each *FL Client* trains and evaluates on its local dataset, aggregated global models result from datasets with diverse QoI. Furthermore, due to different agents operating in the industry as e.g., fully autonomous control systems as well as semi-autonomous ones with human interaction [Jen94], different data recording approaches can influence QoI of asset datasets. Lee et al. [LSKW02] discuss different dimensions of QoI as e.g., *free-of-error*, *relevancy*, *reputation*, *appropriate amount*, *believability*, *consistent representation* and *security*. Based on that, we derive that there is the need to evaluate QoI on *FL Clients* and use resulting metrics on the *FL Server* to decide on the extent of contribution of an individual *FL Client* in the parameter aggregation process. Storing QoI metrics next to existing industrial metadata of participating organizations further enhances building and updating suitable *FL Cohorts*.

[11]`https://documentation.mindsphere.io/MindSphere/apps/asset-manager/introduction.html`, accessed 2024-07-22

3.4.4 Continuous Learning

AI increasingly enables operation of industrial processes to realize flexibility, efficiency, and sustainability. For this, often domain experts have to repeatedly understand new data with respect to its physical behavior and the meaning of parameters of the underlying process [LDSP18]. Continuously involving domain experts and data scientists in updating *ML Models* by e.g., providing labels to recently recorded time series data, is a resource-intensive process that can be faciliated by continuously collaborating in FL. Based on that, we identify the need of supporting continuously re-starting FL processes and cohort reorganization over time to consider major changes in asset time series data.

3.4.5 Scheduling and Optimization

Executing *FL Plans* can cause heavy loads on edge devices, as e.g., training of *ML Models* on large datasets [BEG+19]. Bonawitz et al. [BEG+19] identified the need for device scheduling. This involves that e.g., multiple *FL Plans* are not executed in parallel on single devices with little capacities, or that repeated training on older datasets is avoided while training on *FL Clients* with new data is promoted. For industry purposes, it further needs optimization of cohort communication. This means that *FL Tasks* linked to an *FL Cohort* can be transferred to other cohorts if this improves communication between involved *FL Clients* with respect to e.g., latency minimization [HKH+19]. We believe this decreases model quality due to preferring communication metrics over model quality metrics. However, IFL systems need to consider this trade-off in an optimization problem and solve it to maximize overall utility. Furthermore, collaboration restrictions of *FL Clients* need to be considered in the optimization problem. This ensures that no organization joins *FL Cohorts* with other organizations that they do not want to collaborate with.

3.5 System Design

3.5.1 Domain Model

To establish a domain model for IFL, we consider FL terminology [BEG+19] as well as concepts from industrial asset models as discussed in Section 3.4.1. For this, Figure 3.2 depicts *FL Population*, *FL Server*, *FL Client*, *FL Task* and *FL Plan* as discussed in Section 3.2. Herein, we consider to deploy and run the *FL Server* either in the *Cloud* or on an *Edge Device*. The *FL Client* is hosted on an industrial edge device, that is a hardware device on a given *location*. To support scheduling and optimization decisions of the *FL Server*, the *Edge Device* contains *resource usage* metrics and hardware specifications (*hwConfig*). An *FL Task* refers to an *ML Model* that needs to be trained with an *algorithm* on a given *Dataset* consisting of time series *values*. The scheme of the *Dataset* is defined by an *Aspect Type*, which contains a set of *Variables*. Each Variable has *name*, *unit*, *dataType*, *defaultValue* and *length* attributes to define the content of the corresponding time series values. The *qualityCode* indicates whether a variable supports

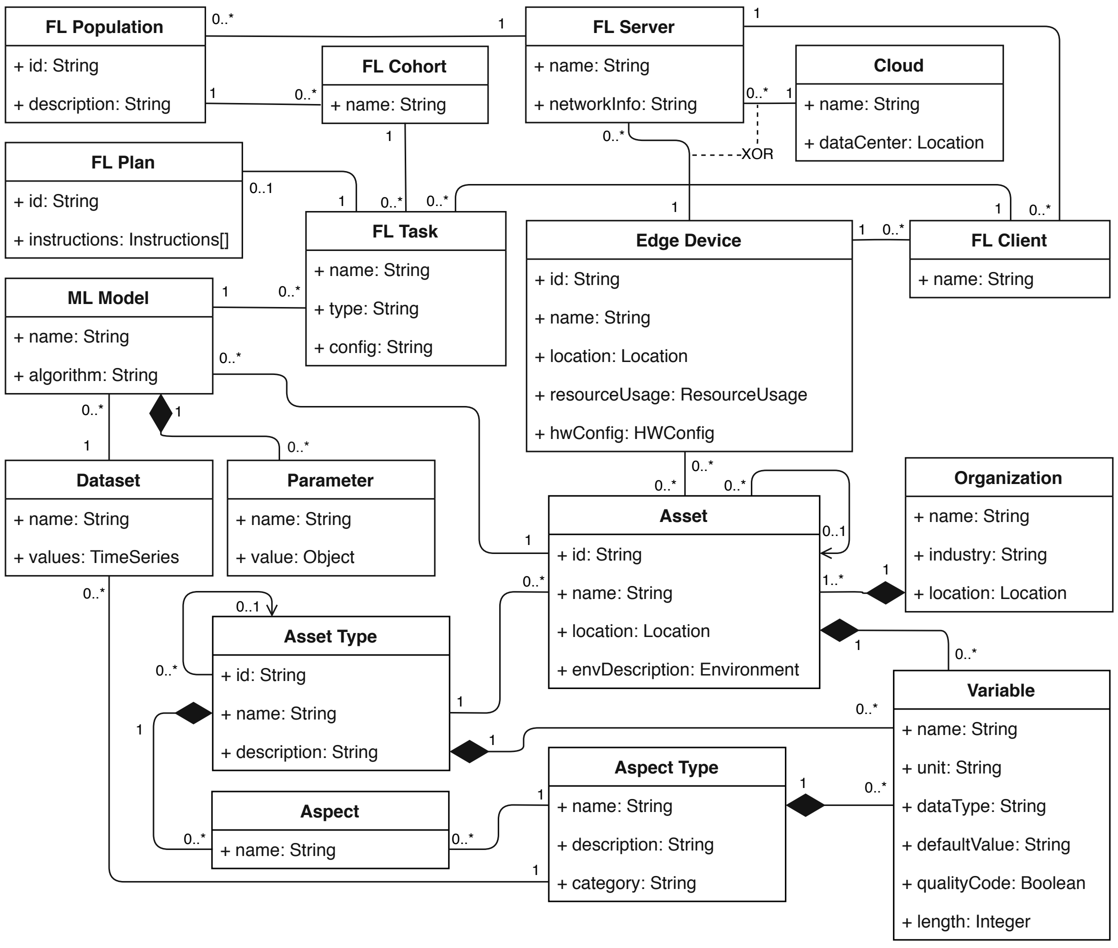


Figure 3.2: Domain model

Open Platform Communications (OPC) Quality Codes[12]. This enables to record and evaluate QoI metrics on the *FL Client* as discussed in Section 3.4.3. Since industrial *FL Tasks* typically consider data from industrial assets, we define an *Asset* (e.g., a concrete engine) operating on a given *location* facing environmental conditions (*envDescription*). The asset is an instance of an *Asset Type* (e.g., an engine) that collects multiple *Aspects* (e.g., surface vibrations) of corresponding *Aspect Types* (e.g., vibration) again collecting variables (e.g., vibrations in x,y,z dimensions). The asset is connected to an Edge Device which is recording data for it. To express the complexity of industrial organizations, hierarchical asset structures can be built as it is depicted with recursive associations of *Assets* and related *Asset Types*, considering nesting of e.g., overall shop floors, their assembly lines, involved machines and its parts. Finally, we introduce *FL Cohorts* as groups of *FL Tasks*. an *FL Cohort* is built with respect to similarities of *Assets* considered

[12]https://www.opcsupport.com/s/article/What-are-the-OPC-Quality-Codes, accessed 2024-10-01

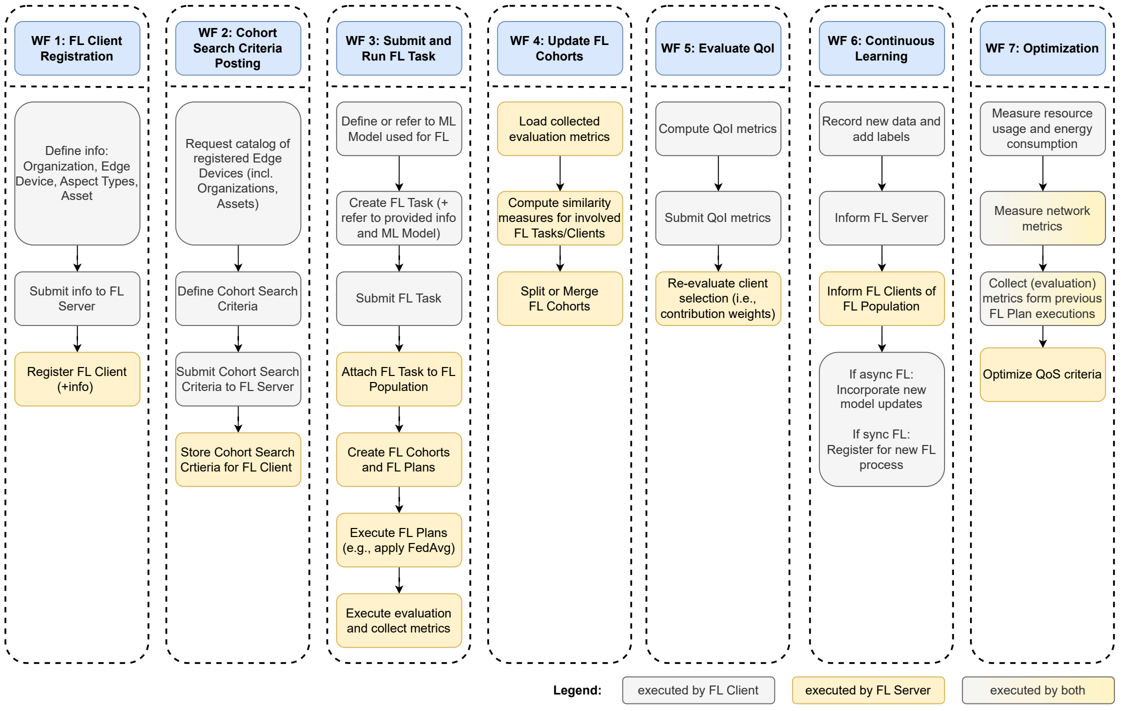

WF 1: FL Client Registration
Define info: Organization, Edge Device, Aspect Types, Asset
Submit info to FL Server
Register FL Client (+info)
WF 2: Cohort Search Criteria Posting
Request catalog of registered Edge Devices (incl. Organizations, Assets)
Define Cohort Search Criteria
Submit Cohort Search Criteria to FL Server
Store Cohort Search Crtieria for FL Client
WF 3: Submit and Run FL Task
Define or refer to ML Model used for FL
Create FL Task (+ refer to provided info and ML Model)
Submit FL Task
Attach FL Task to FL Population
Create FL Cohorts and FL Plans
Execute FL Plans (e.g., apply FedAvg)
Execute evaluation and collect metrics
WF 4: Update FL Cohorts
Load collected evaluation metrics
Compute similarity measures for involved FL Tasks/Clients
Split or Merge FL Cohorts
WF 5: Evaluate QoI
Compute QoI metrics
Submit QoI metrics
Re-evaluate client selection (i.e., contribution weights)
WF 6: Continuous Learning
Record new data and add labels
Inform FL Server
Inform FL Clients of FL Population
If async FL: Incorporate new model updates
If sync FL: Register for new FL process
WF 7: Optimization
Measure resource usage and energy consumption
Measure network metrics
Collect (evaluation) metrics form previous FL Plan executions
Optimize QoS criteria
Legend:
executed by FL Client
executed by FL Server
executed by both

Figure 3.3: IFL workflows

in the attached *ML Model*. So, creating *FL Tasks* intents to typically solve ML problems based on *Asset* data, whereas the *Aspect Type* referred in the *Dataset* of the *ML Model* are used in the linked *Asset*.

3.5.2 Workflows

To regard the requirements of Section 3.4, we propose seven WFs to be supported by the *IFL System* as depicted in Figure 3.3. Each WF covers multiple steps to achieve a defined task (e.g., client registration on the server). These WFs can be executed independently but also combined to provide the functionalities of the *IFL System*. In Chapter 4, we propose the *IFL Process*, providing an API and services for the combined end-to-end execution of what we present in WFs 1-4. WF 7 (Optimization) is conceptually addressed in Chapter 6.

WF 1: FL Client Registration

Assuming the *FL Server* to be in place, the *FL Client* starts participation in the *IFL System* by registering itself. For this, the *FL Client* has to submit a request including organization and *Edge Device* information to the *FL Server*. Furthermore, *Aspect Types* are handed in, describing the data scheme based on which the organization is willing to collaborate in FL processes with other organizations. Additionally, the assets enabled for

FL are posted to the *FL Server*, to provide an overview to other organizations and to ensure that IFL can build *FL Cohorts* based on respective environmental conditions.

WF 2: Cohort Search Criteria Posting

After *FL Client* registration, other *FL Clients* can request a catalog of *Edge Devices*, *Organizations* and connected *Assets*. Based on this, *Cohort Search Criteria* can be created which potentially include organizations, industries, and *Asset Types* as well as *Aspect Types* that the user wants to filter for. This enables the creation of suitable *FL Tasks* that should be part of a specific *FL Population*. If the underlying data distribution is similar between the created *FL Task* and the assigned *FL Tasks* in the *FL Population*, it can be even matched to a desired *FL Cohort*, which is handled in WF 3.

WF 3: Submit and Run FL Tasks

The *FL Client* creates an *FL Task* including references to the *ML Model* without revealing the actual dataset and submits it to the *FL Server*. If *FL Tasks* target the same problems, i.e., reference to the same *Aspect Types* and corresponding *ML Model*, the provided *FL Task* is attached to an existing *FL Population*, otherwise a new *FL Population* is created. IFL then builds *FL Cohorts* of *FL Tasks* based on metadata provided during registration and posted *Cohort Search Criteria*. If no *Cohort Search Criteria* is provided by the *FL Client*, the submitted *FL Tasks* are initially considered in the *default FL Cohort* of the given *FL Population*. To actually start FL, an *FL Plan* is created including server and client instructions to realize e.g., *FedAvg* [MMR+17] on the server and training of *ML Models* on every involved *FL Client*. The *configuration* of *FL Tasks* allows for defining parameters for supported algorithms of IFL for e.g., setting break-up conditions for FL or defining the number of repeated executions over time. *FL Tasks* are either realized as training or evaluation plan. Therefore, the exchanged data between *FL Client* and *FL Server* are different. While training plans typically include the sharing of model parameters as e.g., gradients or weights of neural networks, evaluation plan execution results in metrics that are stored by the *FL Server* to further enable *FL Cohort* reconfiguration and optimization.

WF 4: Update FL Cohorts

Collected metrics in the FL process enable to update *FL Cohorts* with respect to splitting and merging *FL Cohorts*. Furthermore, moving *FL Tasks* between cohorts is considered in IFL. The respective metrics include information like the environmental changes of assets and model accuracy. Furthermore, similarity measures of *ML Models* are computed based on possible server-provided data. If such evaluation data is present, a strategy for updating *FL Cohorts* includes to put *FL Tasks* in the same *FL Cohort*, where its *ML Model* predicts ideally the same output based on provided input samples.

WF 5: Evaluate QoI

The QoI of raw data used by each *FL Client* is computed on Edge Devices and mapped to OPC Quality Codes. Besides using submitted QoI for e.g., updating *FL Cohorts*, IFL considers QoI in the client selection and updating of contribution weights of *FL Clients*. The latter one is applied when it comes to weighted averaging of model parameters as defined in [MMR+17]. Hence, the model parameters delivered by individual clients can be weighted differently according to their QoI metrics. This enables that clients with high quality data can contribute more to the global model.

WF 6: Continuous Learning

After time series data is updated and if needed properly labelled, the affected *FL Client* informs the *FL Server*. For this, either synchronous [MMR+17] or asynchronous [CNSR20] FL can be used to integrate these new data and labels. In the asynchronous case, IFL notifies *FL Clients* of a given *FL Population* to update ML models according to recent improvements of one *FL Client*. In the synchronous case, the *FL Population* is notified to enable assigned *FL Clients* to register for a new synchronous FL process.

WF 7: Optimization

First, the *FL Client* collects resource usage (e.g., CPU, memory) and energy consumption data from the underlying *Edge Device* to enable the aggregation of these metrics on the *FL Server*. Second, network statistics (e.g., latency) are identified as recorded for model update sharings between *FL Clients* and the *FL Server*. Third, statistics and evaluation metrics of past *FL Plan* executions e.g., duration of processing is loaded to be incorporated into an optimization model. Finally, this model optimizes future *FL Plan* executions considering QoS criteria [HKH+19] as processing cost, network latency, energy consumption, and cohort reconfiguration cost.

3.5.3 Architecture

To realize the WFs presented in the previous section, we propose the IFL architecture depicted in Figure 3.4.

Considering two types of parties involved in IFL, we present the *FL Application* and the *FL Server*, whereas the former is a container for a *Industry Application* that is a domain-dependent consumer of IFL. Furthermore, the *FL Application* contains the *FL Client* that interacts with the *FL Server*.

We now discuss the main components of the *IFL System* and its responsibilities. First, the *FL Client* registration WF involves the *Device Manager* of the *FL Client*. It provides an API to the *Industry Application* to register for FL. The *Industry Application* provides a list of participating *Edge Devices* and general information of the organization. Forwarding this to the *Client Registry* allows persistence in the *Device & Asset Metadata Catalog* stored on the *FL Server*. *Cohort Search Criteria* posting is supported by the *Device*

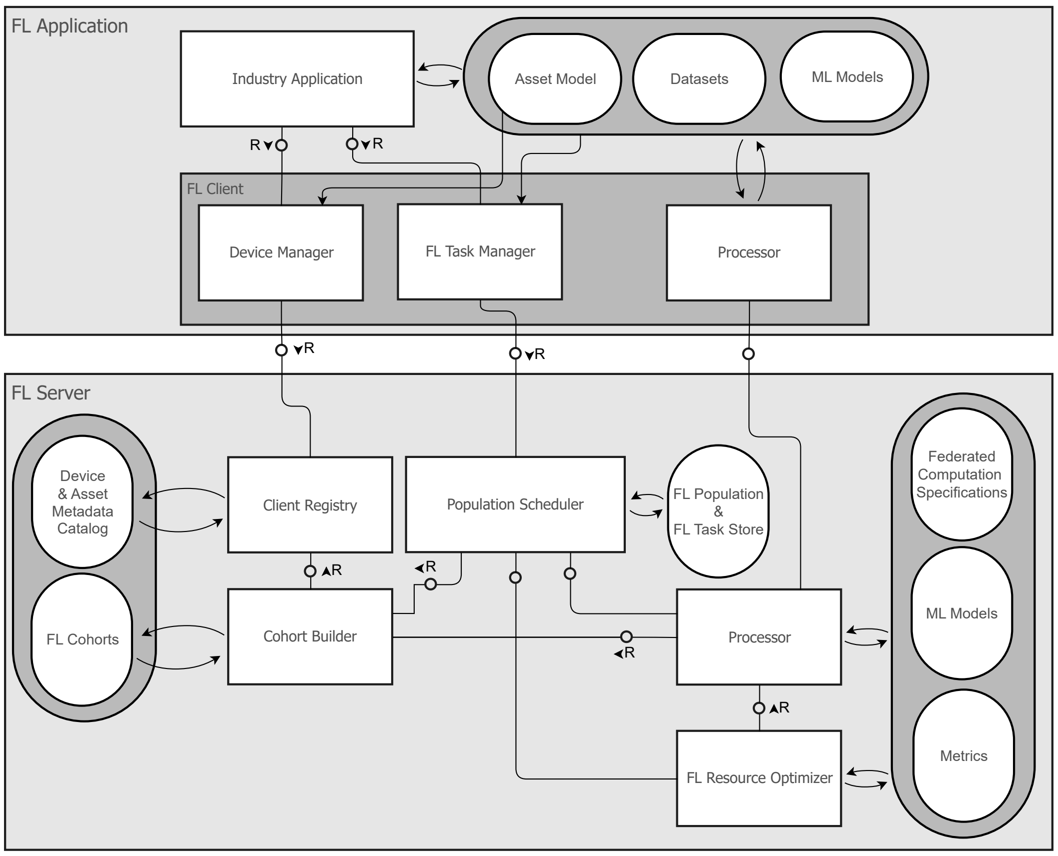


Figure 3.4: FL client and server architecture

Manager and *Client Registry* too, with additionally exposing an interface to the *Cohort Builder* to provide the device & asset metadata catalog and the *FL Cohort Search Criteria* for creating *FL Cohorts*.

Submitting new *FL Tasks* is initiated by invoking the *FL Task Manager* which is in charge of enriching the information provided by the *FL Task* with information of the associated *ML Model* and targeted *Asset*. After forwarding the *FL Task* to the server-side *Population Scheduler*, it is mapped to the corresponding *FL Population* and persisted. Furthermore, the *Population Scheduler* attaches scheduling information to timely trigger execution of all *FL Tasks* of an *FL Population*. To actually run an *FL Task*, the *Population Scheduler* hands it over to the *Processor*. It translates the *FL Task* to an *FL Plan* and corresponding instructions as defined in *Federated Computation Specifications*. Subsequently, it creates the corresponding global *ML Model* and starts the FL process for a given *FL Cohort* by connecting to all *FL Clients* that have *FL Tasks* in the same *FL Cohort*. This information is provided by the *Cohort Builder*. Analogously to *FL Plans*, there exists a client counterpart of the *Processor* too. It invokes the client instructions specified in the

FL Plan to e.g., train or evaluate *ML Models* on local *Edge Devices*. *Metrics* resulting from evaluation plans are provided by the *Processor* to the *Cohort Builder* to update cohorts continuously. Further metrics from e.g., continuous learning approaches or QoI evaluations are stored and used directly by the *Processor* e.g., during model aggregation. The *FL Resource Optimizer* connects to the metrics storage to incorporate parameters in optimization models. After solving the optimization task, the solution is returned to the *Population Scheduler* to trigger, e.g., cohort reorganization and to update the schedule of *FL Plan* executions.

3.6 Summary

In this chapter, we identified the need for IFL and provided a structured collection of requirements and WFs covered in an IFL architecture. Due to diverse conditions of assets operating in industry, *FL Clients* are not advised to exchange *ML Model* parameters with the global set of FL participants. For this, we concluded to consider *FL Tasks* grouped in *FL Cohorts* aiming to share knowledge resulting from similar environmental and operating conditions of involved assets. Furthermore, we highlighed that FL can decrease the amount of resource-intensive work of domain experts considering less continuous updates of datasets and labeling to be done. Additionally, making use of metrics resulting from QoI and *ML Model* evaluations can be used for *FL Cohort* reorganizations and weighting in the FL process.

The IFL concepts that have been introduced in this chapter are used as a foundation for the work that is presented in the next chapters. Hence, we extend these concepts and implement the *IFL System* and the respective WFs (i.e., WF 1-4) in Chapter 4. Furthermore, we evaluate how *FL Cohorts* perform with different cohort building algorithms. To use FL for industrial applications, we implement a framework to facilitate the development, integration and deployment of the *FL Clients* and the *FL Server*. In Chapter 6, we evaluate the impact of deploying an *IFL System* on heterogeneous platforms and test how cohort building can optimize model performance (WF 4 and WF 7). WF 5 (Evaluate QoI) and WF 6 (Continuous Learning) are left as future work.

This chapter was originally published as a paper at the 18th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS) in 2020:

Hiessl, T., Schall, D., Kemnitz, J., Schulte, S. (2020). Industrial federated learning – requirements and system design. In: De La Prieta, F., et al. Highlights in Practical Applications of Agents, Multi-Agent Systems, and Trust-worthiness. The PAAMS Collection. PAAMS 2020. Communications in Computer and Information Science, vol 1233. Springer.

CHAPTER 4

Cohort-based FL Services for Industrial Collaboration on the Edge

4.1 Introduction

In our exploration of FL systems within the industrial context, we consider concepts such as the client registration with suitable criteria and cohort building, as presented in Chapter 3. Various FL algorithms have been investigated with a focus on model aggregation and client selection [MMR+17, YCKB18, PKH19, YWL20]. We extend this investigation to model aggregation for clients operating on industrial machines that encounter heterogeneous operational conditions. Particular attention is paid to the formation and evaluation of cohorts, assessing how model quality can be enhanced compared to FL applied to the overall set of participating clients. Our intention in this chapter is to extend and update the concept of the *IFL System* that we have introduced in the previous chapter. We implement the *IFL System* as multiple interacting services, and discuss its capabilities and limitations. Furthermore, we evaluate the effectiveness within an industrial setting by comparing different FL scenarios using real-world data. Specifically, we demonstrate how to optimize collective model training with respect to the accuracy of two underlying classification tasks for machines such as saws and pumps. In the following, we motivate the main challenges as drivers for the proposed *IFL System* and integrated algorithms.

Many of the challenges prevalent in FL, as outlined in Section 1.1 and the study by Kairouz et al. [KMA+21b], are particularly relevant to industrial clients [HSKS20]. First, in the majority of FL approaches the central authority defines the learning task by deciding e.g., on the used ML model, hyperparameters, and the FL algorithm [KMA+21b]. However, as discussed in Chapters 1 and 3, in industrial applications, users (e.g., machine operators in production lines) have individual requirements regarding business partnerships when it comes to collaborations with other companies on improving and maintaining machine performance [HSKS20]. Hence, clients dependent on the learning task definition provided by the server are restricted to centrally managed ML models, therefore applications, and have no influence on selecting partners for FL. For example, pump manufacturers provide their products to customers (clients) from various industries like manufacturing, food and beverage, pharmaceutical engineering or water supply. Different use cases and therefore operational conditions are present in these domains, which may require adaptations of used ML models and hyperparameters for groups of clients. Moreover, even clients from the same field have restrictions to collaborate only with selected partners or with at least a defined minimum number of partners to increase the chance for actual ML model improvements [DPM+22].

Hence, there is a need for a service-based system, empowering independent clients to create and submit ML models to the server and thereby enable FLaaS [KKP20]. Using a FLaaS approach allows a group of collaborating and independent clients to apply FL on these models and subsequently use it on their machines. Furthermore, considering client restrictions for collaboration partners and the applied FL algorithm on the server provides flexibility to the clients.

Second, model updates for operating machines often need to be triggered explicitly in industry under human supervision. This can be relevant in industry when ML models are used to assist human users [KHY+19]. Potential use cases are, e.g., condition classification of factory machines, failure detection, or even optimization of production processes. To update the used ML model with FL, a client may want to explicitly request participation in FL rounds, instead of automatically getting invoked in periodical execution plans by the server. This enables manual testing before deciding whether to use the model in production. Although explicit participation in FL is not unique to IFL, this is significant for industrial applications.

Third, a prominent challenge in FL is the problem of non-IID data (see Section 2.1.3), which is especially present in industrial domains when machines operate with different configurations under varying environmental and operational conditions [HSKS20]. A suitable example to explain this issue is addressed in Section 1.1.2, which considers the pumping of different liquids. This process causes heterogeneous vibration patterns resulting in skewed data. To not receive poor model quality as multiple clients apply FL on their (skewed) local data, FL systems need to provide mitigation strategies in respective FL algorithms.

This chapter addresses the aforementioned challenges of (i) enabling clients to individually and independently select the used ML models and to define client criteria for collaboration in FL, (ii) enabling clients to explicitly participate in FL on an on-demand basis, and (iii) non-IID data distributions by proposing an FLaaS system. Notably, the contributions are motivated and evaluated using scenarios from the industrial domain, but as discussed above, similar problems also occur in FL in general.

The contributions of this chapter can be summarized as follows:

- Presentation of an FL process, the *IFL Process*, for industrial clients including two algorithms dealing with individual and on-demand requests, and non-IID data.
- Design and implementation of a service-based system, the *IFL System*, covering the *IFL Process* and providing FLaaS.
- Evaluation of the *IFL System* using two time series-based industrial datasets, providing several physical clients (machines) and derived virtual ones. The results on the model performance after FL are presented and compared in IID and non-IID scenarios.

We show that the *IFL System* considers on-demand participation of clients and yields significant improvements in classification accuracy applying FL on cohorts of similar clients rather than on the overall population.

The remainder of this chapter is organized as follows: We describe the *IFL System* design in Section 4.2. In Section 4.3, we demonstrate the results of the FL experiments, and we review related work in Section 4.4. We conclude in Section 4.5.

4.2 System Design

In order to present the design of our system, we first introduce the basic notation used in this chapter in Section 4.2.1. For this, we extend the domain model from the previous chapter (Section 3.5.1) We reuse the concepts of clients, assets, asset types, models, populations, cohorts and tasks and add concepts for (FL) algorithms, and cohort building algorithms. Therefore, we address the dynamic aspects (cohort building and model aggregation) that are described by the *IFL Process*. The *IFL Process* (see Section 4.2.2) is the central approach that applies algorithms to build cohorts and execute FL on respective similar clients. The architecture of the implemented service-based system that supports the execution of the *IFL Process*, is described in Section 4.2.3.

4.2.1 Basic Notation

To describe our *IFL System* model formally, we introduce the notation presented in Table 4.1. For this, we consider that a client $c \in C$ manages an asset $a \in A$ (e.g., a concrete heating pump) of a given asset type $type(a)$ (e.g, a centrifugal pump). Every asset type defines a data scheme $u(type(a))$ that needs to match with the data scheme $v(m)$ required by the ML model $m \in M$ that client c wants to train.

To participate in IFL, the i-th client c_i submits a task t_i to the *IFL Server*. For this, the client needs to specify asset a^{t_i}, model m^{t_i}, the individually selected FL algorithm $alg^{t_i} \in ALG$ for aggregating model weights, a cohort building algorithm $cb^{t_i} \in CB$, and federation criteria $crit^{t_i} \in CRIT$ before submitting t_i to the server.

The federation criteria $crit^{t_i}$ correspond to a set of requirements, where each need to be fulfilled by the system to consider t_i for upcoming FL rounds. So, we consider $t_i \in T$ with $T \subseteq A \times M \times CRIT \times CB \times ALG$.

A population p is a set of tasks that refer to the same asset type $type^p$, model m^p, FL algorithm alg^p and cohort building approach cb^p, where $p \subseteq T$ with $type^p = type(a^{t_i})$, $m^p = m^{t_i}$, $alg^p = alg^{t_i}$, and $cb^p = cb^{t_i}$ holds for all $t_i \in p$.

A cohort coh is a set of tasks with similar data distributions with $coh \subseteq p$. For this, the cohort building approach cb^p is used to assign every task within a population to a cohort. Furthermore, we consider COH^p as the set of all cohorts of population p. Hence, for $coh^p_1, \ldots, coh^p_{|COH^p|} \in COH^p$ it holds that $coh^p_j \subseteq p$ for all $j \in \{1, \ldots, |COH^p|\}$ with $coh^p_j \cap coh^p_k = \{\}$ and $j \neq k$.

Finally, FL is applied on population p using the FL algorithm alg^p to train one model per cohort $coh_j \in COH^p$. For this, we consider m^{coh_j} with $m^{coh_j} = alg^p(coh_j)$ for all $coh_j \in COH^p$.

Note that in the previous chapter, we introduced the concept of *Cohort Search Criteria* for matching suitable cohorts for a given client. This concept is now generalized as federation criteria, as these criteria are used not directly for cohort building but rather for the overall participation in IFL.

Table 4.1: *IFL System* entities

Notation	Description
C	Set of clients $c_i \in C$ participating in IFL
A	Set of assets $a \in A$
M	Set of ML models $m \in M$
$CRIT$	Set of federation criteria $crit \in CRIT$
CB	Set of cohort building approaches $cb \in CB$
ALG	Set of FL algorithms $alg \in ALG$
T	Set of tasks $t_i \in T$ submitted by clients
P	Set of populations $p \in P$
COH^p	Set of cohorts $coh \in COH^p$ of a given population p

4.2.2 IFL Process

In our solution, we address the discussed challenges and propose the *IFL Process* depicted in Figure 4.1. The process is executed by the *IFL System*, which consists of the *IFL Client* (our implementation and extension of an *FL Client* from Chapter 3, referred to as client for short) and the *IFL Services* hosted by the *IFL Server* (referred to as server for short). The client can be deployed to edge devices to train and operate ML models based on data generated by connected machines. The *IFL Services* offer an API to the clients providing knowledge aggregation and distribution on a central server.

To support FL for edge-based industrial clients, the *IFL Client* and *IFL Services* provide a four-step process (Figure 4.1). We have conceptually integrated WF 1 - WF 4 from

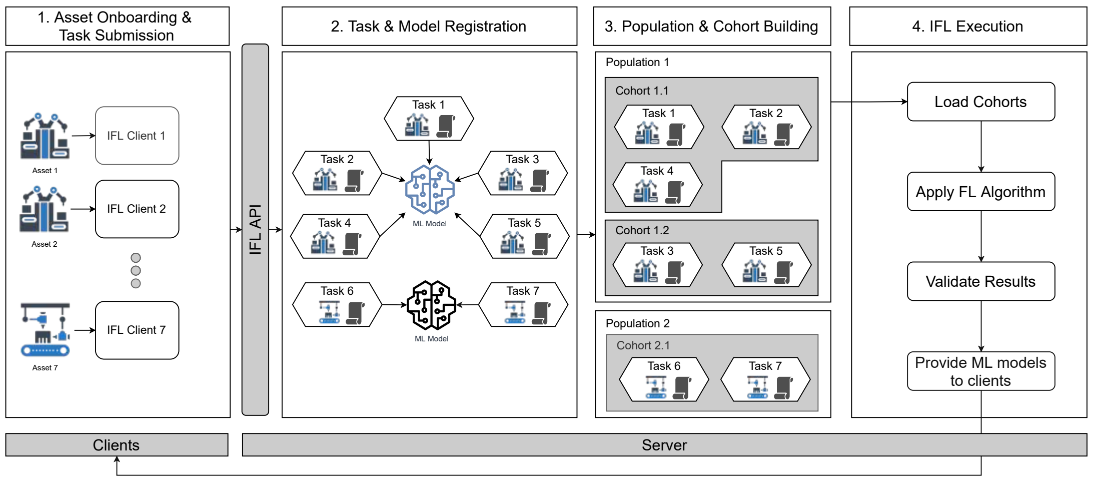


Figure 4.1: *IFL Process* with 4 phases: 1. Clients are connected to their assets and submit tasks using the IFL API to participate in FL. 2. Submitted tasks are registered on the server referring to ML models used as base for FL. 3. Populations of tasks with same asset types are created. Cohorts further split populations into clusters of tasks with similar data distributions. 4. FL is executed for each cohort by applying the algorithm selected by the clients. Finally, validating results and providing the ML model to clients.

Chapter 3 for client registration, federation criteria posting, and task handling (including cohort building). The *IFL Process* covers the end-to-end process starting with the registration of clients and resulting in an FL model for each cohort. As a prerequisite, we consider deployed client applications that invoke the *IFL Client* to establish connectivity to the *IFL Services*. Data is recorded from the asset and stored on the device. For this, we assume a classification problem with input data provided as matrix $X \subseteq R^{N_S \times N_V}$ with N_V variables and N_S samples, and a N_S-dimensional target vector $y \subseteq R^{N_S}$.

Asset Onboarding and Task Submission

The first step involves the *IFL Client* that needs to specify metadata that is referenced in a task. This metadata contains the used asset with the corresponding asset type, whereas the asset type can be reused, if other clients have already published this to the server. Similarly, an ML model is created or selected from the server to be applied to the data. This model is created upfront, based on the asset types' corresponding data structure and the ML task that needs to be solved.

Based on that, the client selects a cohort building approach. For this, the *IFL Services* provide two approaches. The first approach applies a cluster algorithm based on input data X to address potential feature distribution skew. The second approach clusters based on target data y to consider label distribution skew. In both cases, the respective cohort building approach is selected to reduce skewness within cohorts and to improve performance of models that are trained in a cohort by applying FL.

Furthermore, clients specify the knowledge aggregation algorithm, e.g., FedAvg [MMR+17], and individual federation criteria e.g., the minimum number of clients in a population or the minimum dataset size. This enables the client to control the knowledge aggregation process, e.g., to avoid that knowledge is transferred only between a small number of clients, which may lead to insufficient training results. Furthermore, this prevents that knowledge from an individual model is transferred to only a few other clients that may not contribute to the global model. To participate in IFL, clients then submit a task with the mentioned specifications to the server using the IFL API.

Task and Model Registration

In the second step, the server stores the defined assets, uploaded ML models and the tasks. Subsequently, the server verifies that the data scheme of the asset fits to the data scheme required by the referenced ML model, i.e., $u(\mathit{type}(a)) = v(m)$. Hence, it is ensured that clients can participate in FL rounds when the model is trained on their local data.

Population and Cohort Building

The third step assigns tasks to populations and further splits populations into cohorts. This facilitates FL to be executed within small groups of similar clients. For this, we consider the server to execute Algorithm 2 to assign tasks accordingly as they are submitted by clients. First, to build populations, we consider tasks with equal configurations. Particularly, we search for a population p with the same asset type, ML model, cohort building algorithm and FL algorithm as referred to in task t_i. This is necessary to consider a valid FL setting, whereas the same algorithms and model need to be applied on a common data scheme. If any population matches the task configuration as checked in line 3, the task is added. Otherwise, a new population p_{new} is created and added to the population store P on the server (lines 13 and 14).

Furthermore, it is required to reach a consensus regarding the federation criteria, i.e., all criteria $crit^{t_i}$ have to hold for all tasks $t_i \in p$ as verified in line 6. For instance, we consider a criterion for requiring a minimum number of tasks in a population as a precondition before starting FL. To formally describe this exemplary federation criterion, let $q(t_i)$ be a function with $q(t_i) < |p|$ for all $t_i \in p$, where $q(t_i)$ returns the minimum number of tasks that is required by t_i. Based on that, the server eventually starts cohort creation in line 7 to improve model accuracy when FL is executed.

For this, we consider that data of individual clients can be non-IID. As we identified in [HSKS20], non-IID data can be observed when assets operate in heterogeneous industrial environments. Hence, in `CreateCohorts(p)` (lines 16-33), the server makes use of aggregated data that describe the clients data distribution.

For this, we consider two cohort building approaches i.e., *Target Distribution* and *Input Distribution*. Applying the former one, the server requests the client to compute the statistical moments mean, variance, skewness and kurtosis of the target data y^{t_i} in line 19.

Algorithm 2 `PopulationCohortBuilding`

Input: new task t_i received from client, populations P stored on the server

```
    **Update populations**:
 1: $added_{t_i} \leftarrow False$
 2: for $p \in P$ do
 3:   if $type^p = type(a^{t_i})$, $m^p = m^{t_i}$, $alg^p = alg^{t_i}$, and $cb^p = cb^{t_i}$ then
 4:     $p \leftarrow p \cup t_i$
 5:     $added_{t_i} \leftarrow True$
 6:     if $crit^{t_i}$ holds for every $t_i \in p$ then
 7:       $COH^p \leftarrow CreateCohorts(p)$
 8:       break
 9:     end if
10:   end if
11: end for
12: if not $added_{t_i}$ then
13:   $p_{new} \leftarrow \{t_i\}$
14:   $P \leftarrow P \cup p_{new}$
15: end if

    **CreateCohorts**(p):
16: Initialize $F$ as $n \times m$ matrix for $n$ tasks and $m$ features
17: for $t_i \in p$ do
18:   if $cb^p$ = Target Distribution then
19:     $r \leftarrow \{mean(y^{t_i}), var(y^{t_i}), skew(y^{t_i}), kurt(y^{t_i})\}$
20:   end if
21:   if $cb^p$ = Input Distribution then
22:     $r \leftarrow \{mean(X^{t_i}), var(X^{t_i}), skew(X^{t_i}), kurt(X^{t_i})\}$
23:   end if
24:   add row $r$ to $F$
25: end for
26: for feature $f \in F$ do
27:   if $std(f) < \epsilon$ then
28:     remove $f$ from $F$
29:   end if
30: end for
31: $k \leftarrow elbow(F)$
32: $COH^p \leftarrow kMeans(F, k)$
33: return $COH^p$
```

These measures provide information on the shape of the data's underlying distribution function, i.e., location (mean), dispersion (variance), asymmetry (skewness) and the form of tails (kurtosis). We use these features to capture the target distribution of every client's dataset as precise as possible, which is the basis for accurately assigning the corresponding task to a cohort of tasks with similar target distributions. Therefore, we can reduce label distribution skew in a cohort. In line 24, the features are returned to the server and added as a new row to the feature matrix F.

The *Input Distribution* approach computes the mentioned moments based on all n variables of the input matrix X^{t_i} (line 22) and adds them to F. This feature matrix F consists of $|p|$ rows and u columns, where u is the number of features. Hence, using *Target Distribution* we have $u = 4$, while for *Input Distribution* we consider $u = 4 \times n$. Analogous to *Target Distribution*, using *Input Distribution* we can reduce feature distribution skew.

Since F may contain many features with limited information, we remove all features f from F with $std(f) \leq \epsilon$ in line 28, where $std(f)$ computes the standard deviation and ϵ is a pre-configured parameter on the server.

To eventually create the cohorts COH^p, we apply the *k-means* [KMN+02] cluster algorithm based on the reduced feature matrix F. The *k-means* cluster algorithm is an iterative approach that considers a fixed number of k clusters, whereas data points are assigned to cluster centers that minimize variance within clusters. Optimally, we want clusters where all the data in a cluster are close to each other, and the distance between two clusters is as large as possible. To identify k, we apply the *elbow* method [Tho53, SAIR11] in line 31 to find the optimal value with respect to the *silhouette* [Rou87] score. For this, the elbow method iterates over increasing values of k applying k-means and stopping when improvements of the silhouette score are no longer worth further computation. The silhouette score is maximized if samples (= clients) have a relatively short distance to other samples in the assigned cluster and a relatively large distance to samples from other clusters. In line 32, we finally apply k-means with the identified k to assign all tasks $t_i \in p$ to cohorts COH^p. We use the combination of k-means and the elbow method since it can automatically be applied by the *IFL Service* without the need for human validation of the number of built cohorts and still avoiding under- and over-fitting of the trained cluster model.

IFL Execution

The fourth and final step in the *IFL Process* applies the selected FL algorithm alg^p on all created cohorts COH^p as described in Algorithm 3. To start the execution, the server loops through all $coh \in COH^p$ and initializes the models $m_0^{t_i}$ for all tasks $t_i \in coh_j$ in line 2. For this, the model architecture of the underlying neural network is created by building all layers as defined in the base model m^p of the population.

To actually share knowledge between the involved clients, FL algorithms can be invoked by the server. In this work, we evaluate the integration of two FL algorithms in the *IFL Services* to be applied on cohorts, i.e., *FedAvg* [MMR+17] and *Sequential FL*

Algorithm 3 `IFL Execution`

```
Input: $COH^p$ received from population and cohort building step
    Server execution:
 1: for $coh_j \in COH^p$ do
 2:    initialize $m_0^{t_i}$ for all tasks $t_i \in coh_j$
 3:    for round $r = 1, ..., R$ do
 4:       if $alg^p = FedAvg$ then
 5:          for task $t_i \in coh_j$ do
 6:             $m_{r+1}^{t_i} \leftarrow ClientUpdate(t_i, m_r^{t_i})$
 7:          end for
 8:          $m_{r+1}^{coh_j} \leftarrow \frac{1}{|coh_j|} \sum_{i=1}^{|coh_j|} m_{r+1}^{t_i}$
 9:       end if
10:       if $alg^p = SeqFL$ then
11:          for task $t_i \in coh_j$ do
12:             $m_r^{t_{i+1}} \leftarrow ClientUpdate(t_i, m_r^{t_i})$
13:          end for
14:          $m_{r+1}^{coh_j} \leftarrow m_{r+1}^{t_0} \leftarrow m_r^{t_{|coh_j|}}$
15:       end if
16:    end for
17:    validate and send $m_R^{coh_j}$ to all clients $c_i$ of tasks $t_i \in coh_j$
18: end for

    ClientUpdate(t,m) // Run task $t$ on client
19: $B \leftarrow$ ($X^t$ split into batches of size $B$)
20: for epoch $e \in 1..E$ do
21:    for batch $b \in B$ do
22:       $m \leftarrow m - \alpha \nabla l(m, b)$
23:    end for
24: end for
25: return $m$ to server
```

(*SeqFL*) [CBL+18]. We selected the two algorithms since they are well-established and often considered as a benchmark for FL.

Both algorithms trigger client training by iterating over tasks t_i of the processed cohort coh_j, and calling `ClientUpdate(t,m)` (lines 6 and 12). In this function (lines 19-25), the dataset X^t is split into batches B and is iterated several times as defined in the number of epochs E. In line 22, the model is updated using one step of gradient descent optimization, considering a loss function l and a learning rate *alpha*. After iterating the defined number of epochs, the optimized model is returned to the server in line 25, to exchange the gained knowledge with other clients.

In FedAvg, this is achieved by summing up model weights from all clients of a given round and dividing it by the number of tasks in the cohort $|coh_j|$ as presented in line 8. In our approach, we integrated a simplified equally weighted averaging, whereas in the original FedAvg approach client models are weighted with an additional factor expressing the proportion of the number of examples N_{S_i} of client c_i with respect to the total number of examples. Using our approach does not reveal this information of the dataset to the server. In the next round, the aggregated model is again distributed to all clients.

In SeqFL, the model is passed sequentially from one client to another, whereas the subsequent client optimizes the model that the previous client has optimized before. The result of the last client in a given round r is used as input for the first client of round $r + 1$ (line 14).

After training using either of the aforementioned algorithms, the resulting model $m_R^{coh_j}$ is validated by involved clients on their local test datasets as stated in line 17. This yields validation metrics (i.e., test set accuracies) that are passed to the clients along with the model, which concludes the *IFL execution*. As a result, the client is enabled to decide whether to operate the IFL-based model or an individually trained model on the edge device.

4.2.3 System Architecture

To run the *IFL Process* described in Section 4.2.2, the *IFL System* provides the service-based architecture depicted in Figure 4.2. This architecture is an extension of the general IFL architecture introduced in the previous chapter (Section 3.5.3) and provides further implementation details. As compared to the previous chapter, no resource optimization and associated metrics are considered in the implementation architecture, since this will be the focus in Chapter 6.

In the FLaaS approach, we consider two main locations i.e., the shop floor and the *IFL Server*. On the shop floor, assets are operated and generated data (e.g., measured vibrations) are sent to an *Edge Device* to train an ML model that can be used for e.g., CM. The *IFL Server* consists of the *IFL Services* that provide an interface to onboard assets, submit tasks, and apply FL on cohorts of clients. This architecture allows multiple clients from different locations (i.e., shop floors) to use the *IFL Services* and participate in FL.

IFL Client on Edge Devices

To support the asset onboarding and task submission step in the *IFL Process* as described in Section 4.2.2, the *IFL Client* is deployed to *Edge Devices* and invoked by the *Industry Application*. The *Industry Application* can include arbitrary business logic that makes use of FLaaS to train ML models on recorded asset data. To support this, the *IFL Client* connects to the *IFL Services* and creates an *IFL Task*. The corresponding metadata (e.g., asset, ML model) is stored along with the used dataset on a local data storage to ensure transparency on the training history.

After submitting a task, the *IFL Client* starts a local *Worker Node* that can run e.g., the `ClientUpdate(t,m)` function (see lines 19ff in Algorithm 3) to be invoked by the server.

For this, the *IFL Client* provides the training and test dataset to the *Worker Node* to optimize the model m in the respective communication round. As a prerequisite, the *Worker Node* needs to register at the *Worker Network* that is located on the *IFL Server*. This enables that *Worker Nodes* can be found, as a lookup on the server is initiated to invoke clients. *Worker Nodes* are relevant in this architecture, since they enable isolated training per task. This training can even be outsourced to trusted *Edge Devices* by spawning *Worker Nodes* on remote locations if local resources (e.g., memory) are already fully utilized.

IFL Services

To provide FLaaS, the *IFL Process* is supported on the server side, which considers *Client Registry*, *Population Scheduler*, *Cohort Builder*, and the *Processor* as independent services. We further consider a database for assets, associated asset types, and ML models. The latter are considered as base ML models, reflecting a DL neural network architecture with untrained weights.

Using the *Client Registry*, the ML models can be created by clients and used by other clients by referring to them in the submitted tasks. For this, the *Client Registry* provides an API that accepts assets, asset types, ML models and tasks. After data scheme validation (see Section 4.2.2), the task is forwarded to the *Message Broker*. This broker supports a publish/subscribe messaging architecture to share processed output (i.e., validated tasks, built populations, and cohorts) between the services. This enables loosely-coupled services, whose instances can be scaled up and down considering varying loads of task submissions.

To support the population and cohort building as addressed in Section 4.2.2, the population scheduler consumes tasks from the *Message Broker* and assigns them to a population. If provided federation criteria hold for all clients, the population is published to the *Message Broker* for cohort building.

The *Cohort Builder* queries the *Worker Network* for registered *Worker Nodes* of clients that have submitted tasks to the currently processed population. The query mechanism

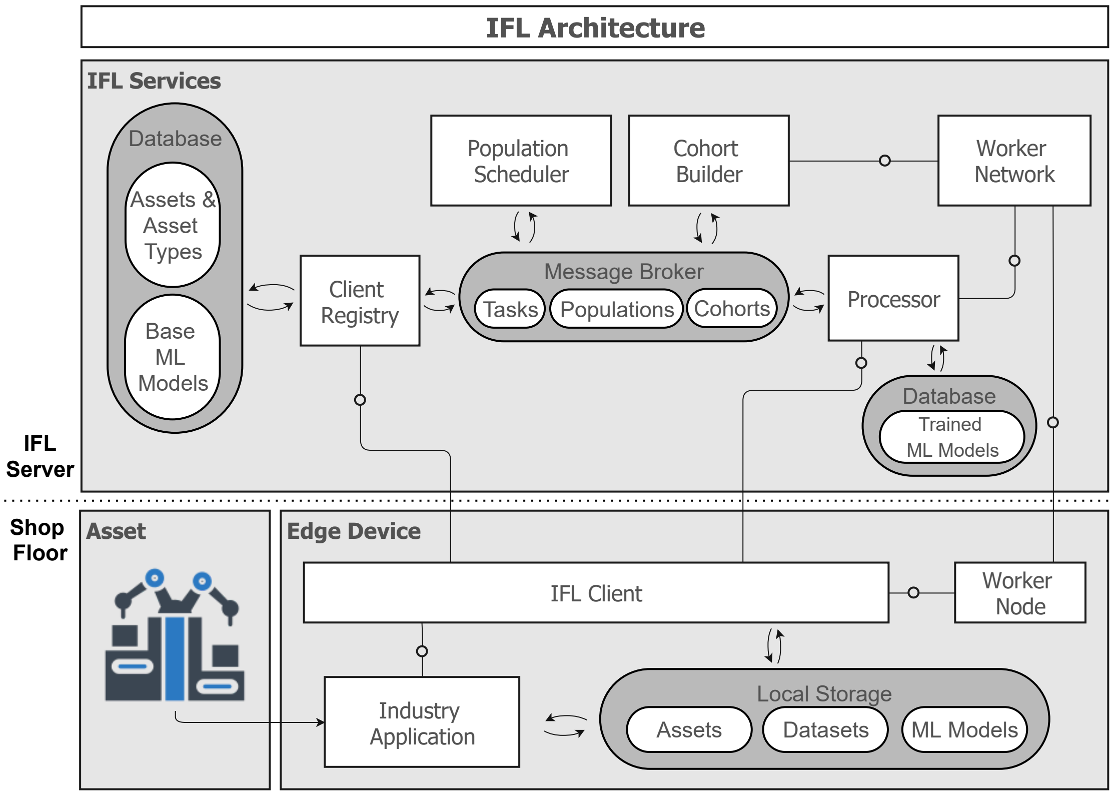


Figure 4.2: IFL architecture: edge-based *IFL Client* and *IFL Services*

is useful since the selective approach does not invoke clients of other populations blocking their resources. Next, the client statistics are retrieved from the *Worker Nodes*, to build cohorts (see Algorithm 2).

To address the *IFL Execution* process step, as described in Section 4.2.2, the *Processor* service subscribes to created cohorts on the *Message Broker* and applies the selected FL algorithm. For this, the *Processor* sends the ML model to respective *Worker Nodes* and retrieves the updated model after training. After the last communication round, the trained model is validated on test data by the *Worker Nodes*. To deploy the model to the shop floor, participating clients can download the model as provided by the *Processor* API.

With this architecture and the *IFL Process*, we directly address the FLaaS properties 1-4 that we have introduced in Section 2.1.7, since 1) no algorithm development is required, 2) collaborative model building is supported through an API, 3) clients describe the input data types via specifying the asset, and 4) permissions for collaboration (and therefore access to models) can be specified in the federation criteria. Properties 5-7 are covered, since 5) clients can specify federation criteria, 6) explicit participation of clients is supported through APIs, and 7) cohorts of collaborating clients can be built.

4.3 Evaluation

In this section, we show the applicability of the *IFL Process* on different datasets including industrial data. We evaluate our *IFL System* approach on IID and non-IID datasets.

In the following, we explain our evaluation setup (Section 4.3.1). The characteristics of the datasets used for our evaluation and the scenarios designed from these datasets are described in Section 4.3.2. Our experimental design is explained in Section 4.3.3. Finally, the evaluation results based on the designed scenarios are reported in Section 4.3.4.

4.3.1 Experimental Setup

All the experiments are run on a Windows machine with 3.0 GHz Intel Xeon Scalable Processor with 8 CPU cores and 32 GiB RAM, hosting the *IFL Services*. *IFL Clients* run on a Windows machine with 3.3 GHz Intel Xeon Scalable Processor with 2 CPU cores and 4 GiB RAM.

The implementation of the *IFL System* uses PySyft[1], an open source framework for private DL to operate on data that resides on remote locations, i.e., the server invokes PySyft to connect to the clients and to train ML models on their local datasets. The *Worker Nodes* and *Worker Network* communication, as described in Section 4.2.3, is implemented using PyGrid[2]. PyGrid is based on PySyft and adds the functionality for registering nodes and searching for nodes in a network to eventually establish connection between the *IFL Services* and the *Worker Nodes* on the *Edge Devices*.

4.3.2 Datasets

We evaluate our proposed approach using two real-world datasets from industry. In this section, we explain the characteristics of each dataset and elaborate on the ML approaches used on these datasets.

Pump Condition Classification Dataset

This dataset contains acceleration data from five pumps. The data is obtained from multiple measurements [KBG+22] using an IIoT sensor, namely the *SITRANS multi sensor* [BGvDH19]. The sensor provides 512 acceleration samples at three dimensions every minute, thus providing a time series representation.

The time series data is labeled based on the conditions of the pumps. Figure 4.3 shows a schematic view of the virtual Z-axis from 70,000 samples obtained from one of the sensors. As it can be seen, there are six classes indicating anomalous or healthy conditions. The healthy conditions are healthy stationary normal load (the load of pumped water at the range of $50\frac{m^3}{h}$), healthy stationary partial load (the load is $[12.5\frac{m^3}{h}, 37.5\frac{m^3}{h}]$), and idle state (the pump is turned off). The anomalous conditions are hydraulic blockade failure

[1]`https://github.com/OpenMined/PySyft`, accessed 2024-07-22
[2]`https://github.com/OpenMined/PyGrid`, accessed 2024-07-22

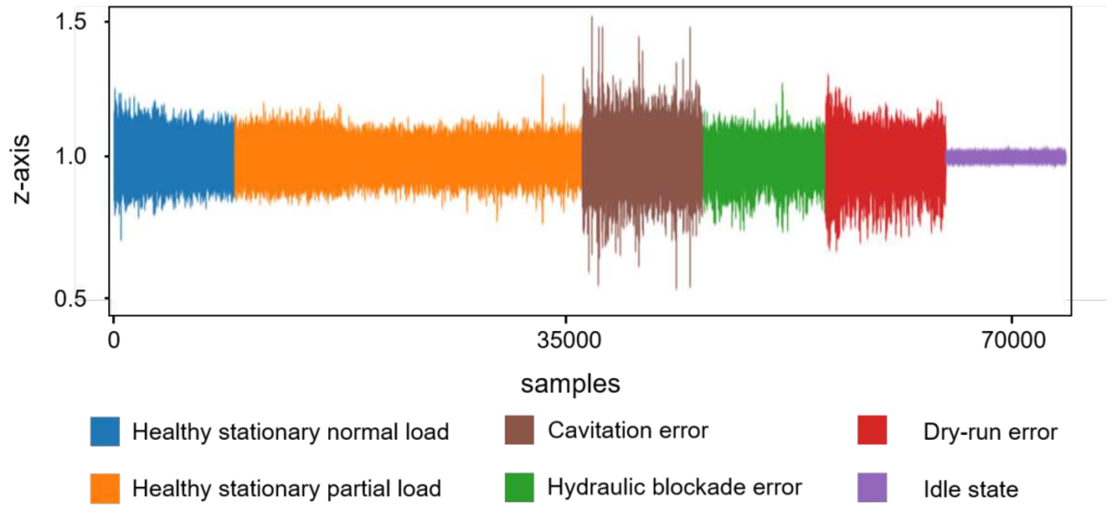


Figure 4.3: Pump classification dataset: a schematic view of samples from one sensor.

(nothing was pumped at the start of pumping), dry-run error (nothing was pumped at the end of pumping), and cavitation error (the water in a pump turns to a vapor at low pressure).

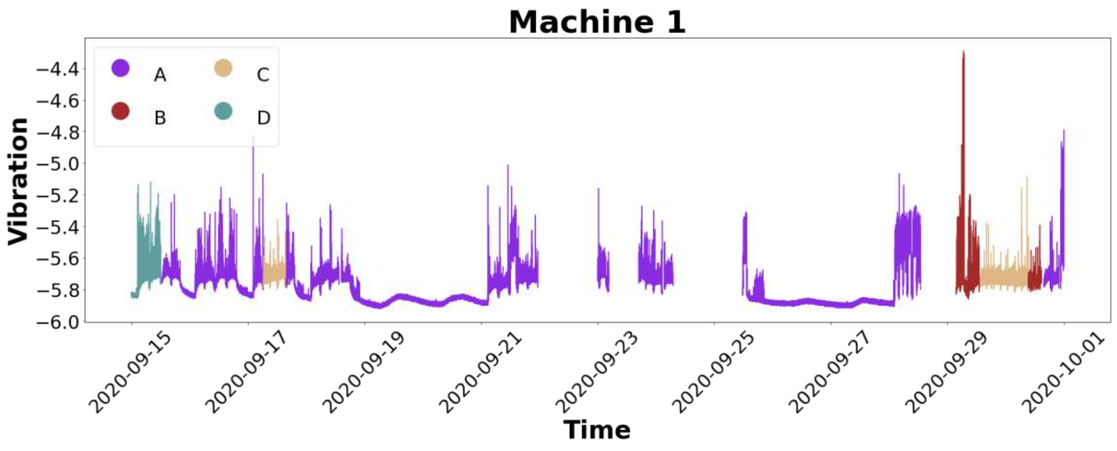
We follow a sliding window approach for collecting data with a window size of 512 and a step size of 256. We then extract Mel-frequency Cepstral Coefficients (MFCC) as features (20) from the samples in each window. MFCCs are widely used as features for audio classification. We employ it here for acceleration data as both have strong relation due to air-borne and structure-borne sound transmission [CH13].

We consider two evaluation scenarios for this dataset: 1) *pump classification IID* where the distribution of target data (i.e., pump conditions) is IID, and 2) *pump classification non-IID* with a non-IID target distribution. In both cases, we consider the five pumps as data sources for the used clients. For the IID case, we artificially create nine clients, with each client being assigned a subset of data of 1-3 pumps such that all class labels are represented. The clients receive on average 26,554 samples. For the non-IID case, we artificially create ten clients, where each pump dataset is divided between two clients on average with respect to independent and time-separated measurements. These measurements were performed on different days considering even adjustments in between like, e.g., dismantling and rebuilding of pumps, or removing and reattaching screws to address feature and label skews [KBG$^+$22]. With this, each of the non-IID clients receives 212,738 samples on average.

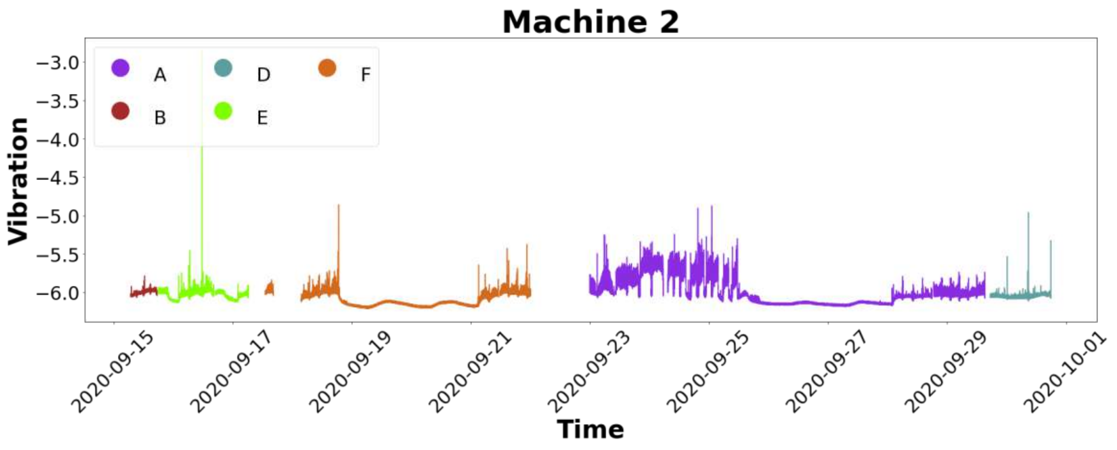
As a learning approach, we use an Artificial Neural Network (ANN) with two dense layers with 64 units each using Rectified Linear Units (ReLu) activations and followed by 40% dropout, and a last output layer without any activation function (5,894 total parameters). We use this dropout-rate to avoid that the model is biased towards a single client. We consider just two dense layers to limit the number of parameters, which increase training time and would require larger datasets. However, to facilitate classification (i.e., linear separability), the number of units per layer (64) is chosen to be significantly greater than the input features (20). This model is then considered as our base model m^p that is uploaded to the server and used by tasks of population p for evaluation.

To train all the clients based on the same scale of data, the input data is normalized using a Gaussian distribution before being fed into the model.

Material Classification Dataset



(a)



(b)

Figure 4.4: Material hardness labels for two machine tools. Each row shows vibration time series data for one machine, colors indicate different material hardness.

This dataset contains vibration data and material hardness labels from two machine tools that process metals. The vibration data is obtained at a frequency of 1Hz, thus providing 60 samples per minute. The objective here is to classify material hardness based on vibration data. As it can be seen in Figure 4.4, there are six material hardness classes in this dataset.

It can be observed that some materials are only machined from one of the machine tools, thus the distribution of hardness labels between the machines is not uniform. Furthermore, the dataset is imbalanced, for example, the material with hardness label A has been observed more than the other material hardnesses. FL is reasonable and applicable for this dataset because 1) both machines are similar in construction and 2) non-IID distribution of material hardness labels makes transferring one model for both machines not applicable.

In this use case, we demonstrate how IFL can be applied on a small scale, considering only two assets (i.e., machine tools) generating non-IID data, and still enabling ML model improvements.

The input vibration time series data is split into sequences of maximum two continuous hours during the operation time of the machines. These sequences are normalized and divided into training and test data, keeping 70% of sequence data for training. Our data is obtained from these sequences following a sliding window approach, with a window size of 120 (i.e., samples of every two minutes). For training sequences, we have overlapping windows to cover the entire data with a stepsize of 60, whereas for testing sequences, we do not use any overlapping. We then compute Fast Fourier Transform (FFT) [BB86] on the samples and consider the magnitude of the computed FFTs as our features. This yields a 120-dimensional feature vector.

We consider one evaluation scenario for this dataset using two clients, each representing a real (physical) machine tool. The target data distribution (i.e., material hardness) is non-IID and the data is imbalanced. We have a total number of 5,985 samples for the first client and 7,136 samples for the second client.

As a learning approach, we employ an ANN with two dense layers, each with 256 units, each using ReLu activations and followed by 40% dropout, and a last output layer without any activation function (98,310 total parameters).

4.3.3 Experimental Design

All the scenarios as discussed in Section 4.3.2 are provided in a JavaScript Object Notation (JSON) format. The scenario JSON contains configuration for clients, tasks, assets, and the used model.

The client configuration provides settings for the client's name, the path to a client dataset, the tasks associated with a client, and additional organizational descriptions.

The task configuration has settings for the selected FL algorithm alg^{t_i} (FedAvg or SeqFL), the cohort strategy cb^{t_i} (if cohorts of similar clients should be built for federation and the algorithm to be used for building cohorts), and federation criteria $crit^{t_i}$ (e.g., the minimum number of required clients that need to join a population to start the training).

The asset configuration gives settings for name, description, location, and environment description of an asset.

The model configuration provides settings for the path of the base model to be used for federation, and the training parameters (e.g., number of communication rounds, number of epochs per communication round, learning rate, batch size, etc).

Based on the aforementioned configuration parsed from the scenario JSON, the individual client processes independently create their assets, refer to a common ML model, and eventually submit tasks to the *Client Registry* API to join for FL.

After FL is finished, resulting ML models are validated using classification accuracy. If applicable, we consider classification accuracy on cohort test data (i.e., test data from all the clients in a cohort) to ensure comparability between models trained only on client data, models trained on central data, and models trained with FL on cohorts. To further compare cohort-based FL with FL not using cohort building (i.e., FL algorithm is applied on the overall population), we consider classification accuracy on the overall population. Finally, the clients download the model using the *Processor* API to conclude the *IFL Process* which terminates the evaluation scenario.

4.3.4 Results

In this section, we report on experiments evaluating the performance of our *IFL System* on multiple scenarios from three datasets (as described in Section 4.3.2) using FedAvg and SeqFL algorithms.

In all the experiments, our base model for training is an ANN model. We use PyTorch [PGM+19] for training and prediction. We compute loss using the cross-entropy loss function [Bis07]. Since we deal with non-IID data, we use a weighted cross-entropy loss function. The loss l is computed for each class c separately and can be written as:

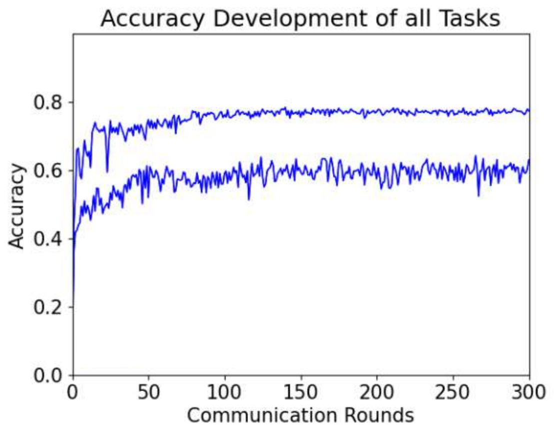
$$\begin{aligned} l(x,c) &= weight[c] \times -\log \frac{\exp x[c]}{\sum_j \exp x[j]} \\ &= weight[c](-x[c] + \log \sum_j \exp x[j]) \end{aligned} \tag{4.1}$$

where x is an observation (i.e., the output logit of an ANN for a sample whose size is equal to the number of classes), and $weight[c]$ is the class weight computed for each client independently considering a balanced heuristic inspired by [KZ01].

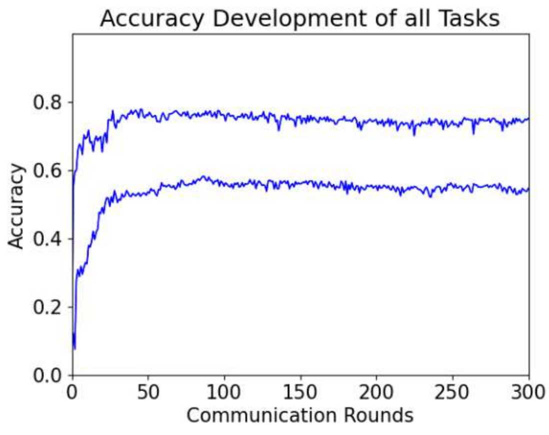
The class weight can be described as:

$$weight[c] = \begin{cases} \frac{N_S}{N_C \times N_{S_c}}, & \text{if } N_{S_c} > 0, \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

where N_S is the total number of samples for each client, N_C is the total number of classes provided as a training parameter to each client because there can be classes which are



(a) FedAvg



(b) SeqFL

Figure 4.5: Accuracy of the federated model trained on the two clients (one per two machine tools) using the material classification dataset.

not present for a particular client, and N_{S_c} is the number of samples for this class in a client dataset.

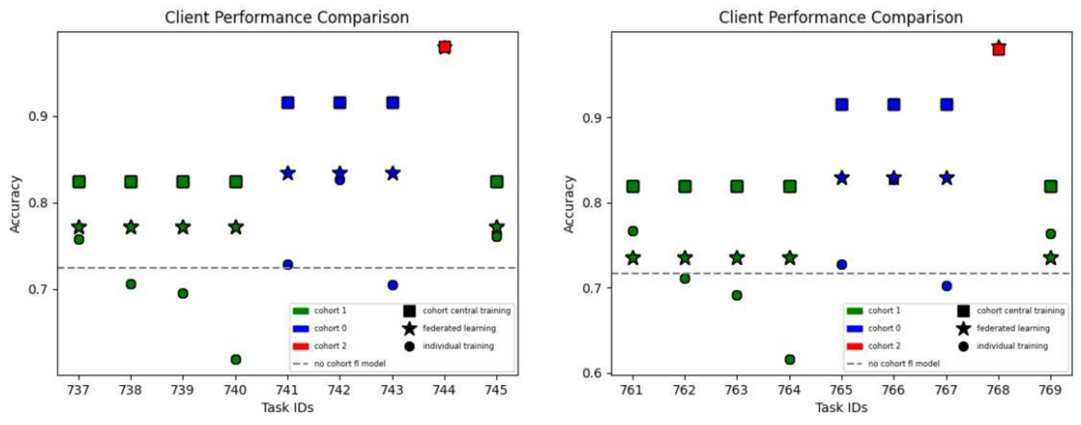
We perform batch optimization for training where training loss is computed on batches of data instead of the entire data due to memory efficiency and overfitting problem. For optimization, we use the Adam optimizer [KB14] and a batch size of 128. In order to compromise between time and performance, we keep the learning rate $1e-3$ in our experiment. A lower learning rate makes training much slower and a higher learning rate might deteriorate the performance.

Impact of IFL on Non-IID Data

The material classification dataset as discussed in Section 4.3.2 has non-IID data and target distribution. We provided two scenarios for each FL algorithm with a default cohort strategy (when no cohort is built). We trained a federated model with two clients (one for each machine tool). The training is done for 300 communication rounds and one epoch per communication round for each client. The average accuracy on the test data of this dataset is shown in Figure 4.5. As it can be seen, the accuracy of the federated model improves after each communication round from initial accuracies of 50% and less. We obtain for both clients a slightly higher accuracy using the FedAvg algorithm. But in both cases, we get the best accuracy of 76% in spite of non-IID data distribution which underlines the usability of our *IFL System*.

Impact of Cohort-based IFL on IID Data

We have shown the applicability of our *IFL System* on industrial data with two clients. Next, we evaluate our approach on the *pump classification IID* dataset (Section 4.3.2)



(a) FedAvg (b) SeqFL

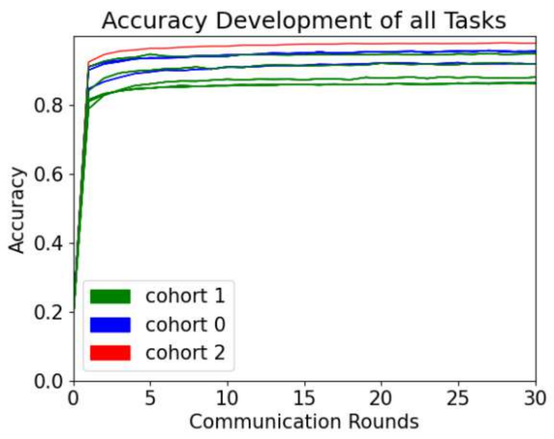
Figure 4.6: Client performance comparison on the pump classification dataset with IID target distribution. Every client is represented by a task ID and has four accuracy metrics resulting from different ways of model training: FL, individual training, cohort central training, and global FL (no cohort FL model).

with 9 clients where the data has an IID target distribution. Our federation criteria in both scenarios is the minimum number of clients for starting federation which we set as 9 (the total number of clients).

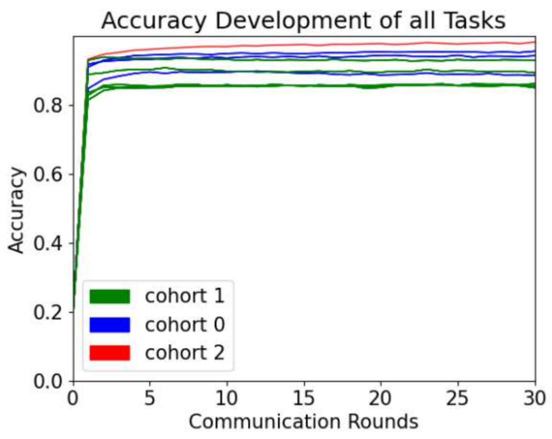
Figure 4.6 shows evaluation results of this scenario after 30 communication rounds and 5 epochs per communication round. The performance of the federated model is compared with three other evaluation approaches:

- No cohort FL model: A federated model is trained on the training data of all the clients and tested on the test data of all the clients.
- Cohort central training: Training and test datasets of cohort members are aggregated by the server which results in one joint training and test dataset per cohort. For each cohort, one model is trained and tested on the corresponding cohort dataset without applying FL.
- Individual training: A model is trained on each client and transferred to the other clients (i.e., testing on cohort test data).

So, for each cohort and approach all clients share one model, except for individual training, which considers one model per client (task id). It can be seen in Figure 4.6 that the accuracy of the cohort-based FL model is higher than the global FL model. The cohort building approach in this experiment is based on input data. Although, the result as depicted in Figure 4.6 considers a target distribution that is IID, the input data of pump conditions does not have IID distribution. More precisely, the input data of some pump conditions are more similar to each other than other pump conditions. Therefore,



(a) FedAvg



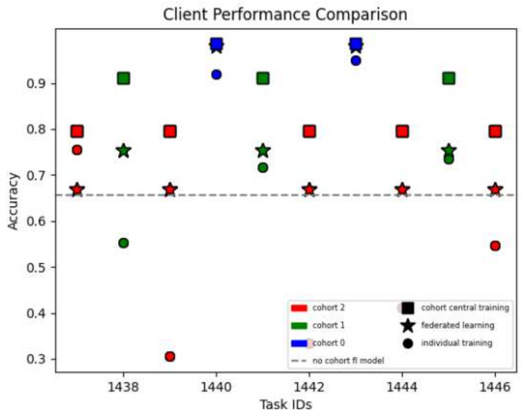
(b) SeqFL

Figure 4.7: Accuracy of the federated model after each communication round on the pump classification dataset for 9 clients with IID target distribution.

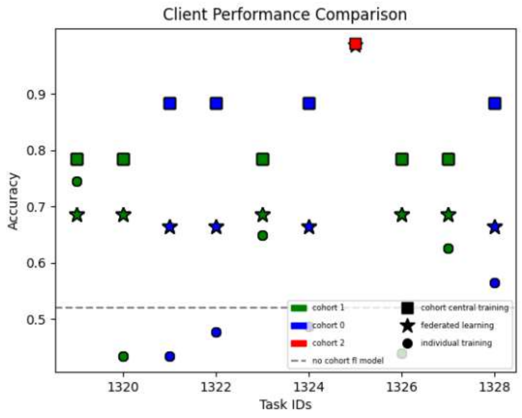
we obtained a higher performance by learning separate FL models on similar groups of clients. This result highlights our contribution using a cohort-based IFL approach with either FedAvg or SeqFL algorithms.

The best result achieved using federation on each cohort is shown in Figure 4.6 as cohort central training. It can be observed that we can achieve an accuracy close to cohort central training using our *IFL System*. In order to show the importance of using FL in this scenario, we compare the accuracy of the federated model on each cohort with the individual training. We can observe that our federated model on average improves the accuracy of each client compared to the individual training. Using FedAvg, this improvement can be seen more clearly than for SeqFL. This emphasizes the applicability of FL for this experiment.

The average accuracy on the test data after each communication round is shown in Figure 4.7. It can be seen that the federated model reaches an accuracy above 80% just after a few communication rounds. We can see that cohort 2 has the highest accuracy. The reason is that this cohort has only one client, therefore FL performs like a central model (without federation) for this cohort. The accuracy of FL for the clients in cohort 1 is slightly lower than others. As it can be seen in Figure 4.6, the cohort central training accuracy for this cohort is also lower than the other cohorts. Therefore, FL cannot achieve a higher accuracy. One reason for the lower performance of cohort 1 is that it has more clients compared to the other cohorts and some of these clients (e.g., task 740) are not fitting the cohort very well. Therefore, it increases the risk of contributing model updates with below-average quality clients.



(a) FedAvg



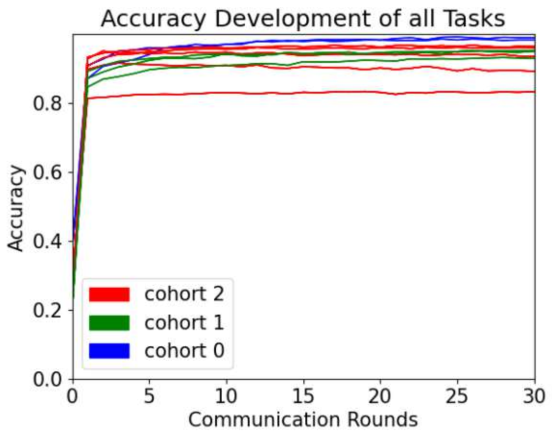
(b) SeqFL

Figure 4.8: Client performance comparison on the pump classification dataset with non-IID target distribution. Every client is represented by a task ID and has four accuracy metrics resulting from different ways of model training: FL, individual training, cohort central training, and global FL (no cohort FL model).

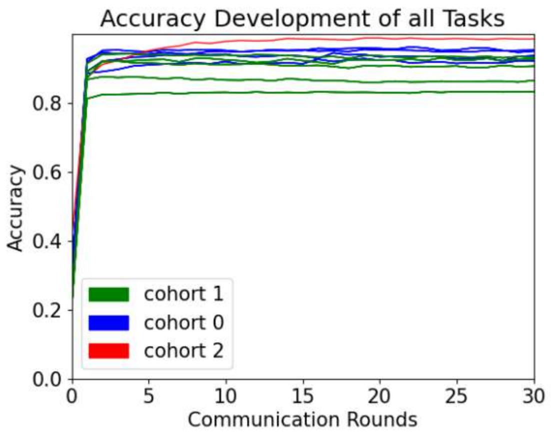
Impact of Cohort-based IFL on Non-IID Data

We showed that cohorts can improve the performance of the *IFL System* on datasets with IID target distribution. We now go one step further and evaluate the performance of a cohort-based IFL system on the *pump classification non-IID* (Section 4.3.2) with non-IID target distribution with 10 clients. For this experiment, we also provided two scenarios, each considering one FL algorithm. The cohort strategy in both scenarios is based on input data distribution. And the federation criteria in both scenarios is the minimum number of clients for starting federation which is set as 10 (the total number of clients). Figure 4.8 shows the evaluation results after 30 communication rounds and five epochs per communication round. It can be seen that the federated models achieve a high accuracy compared to their respective cohort central models. Furthermore, the cohort-based approach gives on average a higher performance than the no cohort federated model. This result also emphasizes the impact of finding similarity between clients and using them in federation.

Figure 4.9 shows the average accuracy on the client test data of this dataset after each communication round. As it can be observed, the federated model reaches at least 80% accuracy just after a few communication rounds. It can be seen that in cohorts with more clients such as cohort 2 in Figure 4.9a or cohort 1 in Figure 4.9b, the accuracy of some clients is lower than the other clients in the same cohort. The reason is that we used the individual client test datasets and the k-means clustering approach for building cohorts. The clusters are initialized randomly and the clients get assigned to the clusters with the minimum mean squared distance. The random cluster initialization may result in outliers during assignment.



(a) FedAvg



(b) SeqFL

Figure 4.9: Accuracy of federated model after each communication round on the pump classification dataset for 10 clients with non-IID target distribution.

4.3.5 Discussion

As we have demonstrated in the evaluation, the *IFL Process* with the integrated cohort building and FL algorithms can be applied to improve ML models for industrial scenarios. Our FLaaS-based approach provides FL to client processes by considering their IFL tasks and defined metadata. So, the clients explicitly submit the tasks using the *Client Registry* API, and download the resulting model using the *Processor* API. This invocation of the service-based architecture and the *IFL Process* addresses the challenges of (i) individual, independent, and (ii) explicit participation in FL with (iii) non-IID data. The individual implications of the presented solutions are discussed in the following:

Impact of datasets on defining clients The respective experiments are limited to two industrial datasets with a small number of assets. Therefore, the findings may not be generalizable. For example, the pump condition classification dataset only contains acceleration data from five pumps, but resulted in 9-10 different clients. Furthermore, the measurements and conditions were recorded by different experts and due to the asset complexity, partial load and error creation resulted in additional data skews.

Impact of cohort-based FL Our experimental evaluation proves the applicability of our cohort-based IFL system especially on non-IID data. It can be seen that the accuracy of a cohort-based approach is on average higher than for FL models without considering cohorts. However, a limitation is shown in Figures 4.9a and 4.9b. The clusters are initialized randomly and the clients get assigned to the clusters with the minimum mean squared distance. The random cluster initialization may result in outliers during assignment. Therefore, providing a dynamic cohort-assignment approach that enables changes in cohort assignment between communication rounds, is part of our future work, which we discuss in Chapter 8.

FL versus model transfer The results reported on industrial datasets (i.e., pump classification and material classification datasets) indicate that FL is applicable and reasonable for these datasets. Based on our results, transferring a model from one client to the others is not applicable for these datasets and does not give higher accuracy than FL. However, as it can be seen in Figure 4.6b, the individual training result of cohort 1 for task 761 performs slightly better than the FL model. The potential reason might be that this client has either more data or more variety of data in this cohort. Making an adaptive approach, for deciding if FL or model transferring should be used in a cohort is an interesting research question we have not covered yet.

Impact of FL algorithm on model performance FedAvg performs better than SeqFL for all the evaluated scenarios. The potential reason is that in FedAvg, the parameters of all the clients are averaged in every communication round. Whereas in SeqFL, the model is passed from one client to the other. All the clients contribute, but the federated model of SeqFL is significantly impacted by the training of the last client.

Impact of FL algorithms on fair model sharing The selection of FL algorithms could also prevent clients from just waiting for federated model updates without individual contributions. If clients select FedAvg, each client needs to provide the locally trained model to the server before a federated model is aggregated and distributed. So, if clients do not contribute, subsequent model updates would not be received after aggregation. In contrast, selecting SeqFL could lead to transfer models from client A to B, although B might not have contributed something before.

Impact of resource-constrained edge devices on FL In our evaluation, all the clients are running on machines with similar resource capabilities. Yet, we have not considered heterogeneous resource-constrained *Edge Devices* and volatility in the utilization of e.g., CPU, GPU, memory, and network. This information can be used to avoid inefficient training e.g., due to overloaded *Edge Devices*. Since our current focus was on the applicability of the *IFL System* on *Edge Devices*, we considered optimizing accuracy rather than optimizing resources consumption. Our *IFL Services* will be extended in Chapter 6 to address the optimization of clients on several platforms and heterogeneous devices.

Realistic applicability of the IFL system Although independent clients can always register for FLaaS as provided by our *IFL System*, they can also drop out due to e.g., missing resources, volatile connectivity, or intended stopping since the model is already mature enough. Therefore, the remaining clients in the cohort miss potential valuable contributors and the model quality could stagnate, which may cause the server to restart FL with a new cohort organization considering new clients as well. Furthermore, the *IFL*

Services do not provide rewards for clients with already mature models to join or stay in the system. This is also relevant in the early stages of the *IFL Process*, as only a limited number of clients might have joined. Since our system creates a subset structure (tasks in cohorts in populations), only a few clients collaborate initially, which can slow down the training progress. For this, introducing a reward system for providing model updates could solve the cold start problem and could prevent drop-outs.

4.4 Related Work

Despite being a recent research topic, FL has been studied widely in various applications [BCM+18, SRE+18, FVH19]. The basic idea of FL as proposed in [MMR+17] is to federate a global FL model among all the clients. However, one of the main challenges in FL is that the client's data especially in real-world scenarios are non-IID [ZXLJ21]. Thus, a global FL model might not perform well for all the clients.

To overcome this problem, clustered FL approaches are proposed where clusters (or cohorts) of similar clients are identified and the model is personalized for each cluster. Clustered FL approaches can be classified into two categories: centralized approaches where the server identifies a client's assignment to a cluster [GHYR19, SMS21], and decentralized approaches where the clients choose and update the model parameters that best fit them [GCYR22, MMRS20]. We followed a centralized cohort (clustered) FL approach by considering features (statistical moments) extracted from the client's data. However, the *IFL Clients* can influence the cohort building by defining the cohort building approach and by defining the federation criteria that affect prior population building. In our work, the server also identifies the number of required cohorts. Building cohorts (clusters) on the server in our work is efficient because only few features from clients are used. As it is shown in Section 4.3.3, a cohort-based FL approach yields a higher performance than a global FL especially for scenarios with non-IID data distribution.

The FedAvg algorithm aggregates model parameters by averaging the value of local parameters from clients in each communication round [MMR+17]. Clients often have different dataset sizes and data distributions, thus training an effective global model with a good convergence using FedAvg is challenging. Therefore, the clients' contribution in aggregation operation of FedAvg is weighted either proportional to their local dataset size [MMR+17] or using multiple criteria such as the diversity of the dataset, data quality, and dataset size of clients' local datasets [BEG+19]. In our work, all clients are weighted equally. This way, we do not use any information about clients' dataset sizes on the server. In our experiments, we still obtained good convergence during training with equal contribution weights.

Selecting edge-based clients for participating in FL rounds is an important aspect for reducing communication cost and potentially long-lasting training times. This can be due to heterogeneous compute capabilities provided on resource-constrained edge devices. Clients may be selected randomly [MMR+17, NY19] or based on pre-defined criteria such as availability of their resources [ATMT21]. In our approach, we follow a clustered

FL approach, using all the available clients for building a population. As discussed in Section 4.3.1, clients' resources are similar in our evaluation setup. Therefore, we did not need to include any resource-based criteria on the server for selecting the clients. However, in the *IFL System*, we consider client-defined federation criteria for building populations. This could be used as a basis for defining requirements on resources that are provided on edge devices of potential FL partners.

Focusing on edge-based FL, Feraudo et al. [FYS$^+$20] provide an architecture that enables asynchronous FL for edge devices using a publish/subscribe pattern. For this, the client registers for FL by sending a message to a *Message Broker*, which notifies the server that waits until the end of a pre-defined timespan to consider the start of FL rounds. Similarly, Bonawitz et al. [BEG$^+$19] propose an FL system enabling edge devices to declare their training intention on a central server that starts FL as a required number of clients have joined. Furthermore, the authors consider selection and rejection of clients on the server. Hence, both approaches address the challenge of a varying number of edge devices, particularly unavailable edge devices. Our *IFL Services* enable clients to define e.g., a minimum number of FL partners, using custom federation criteria. This democratic approach provides more power to the edge clients, e.g., by reducing the required minimum number of FL partners if too little clients joined in previous FL attempts due to potential unavailability.

To provide FLaaS to edge and mobile devices, Kourtellis et al. [KKP20] provide high-level APIs that can be invoked by mobile apps to collaborate on common ML problems. The authors propose a hierarchical scheme for collecting model updates on local device services considering potentially multiple apps. The updates can be forwarded to e.g., edge nodes or central servers for aggregation. Furthermore, the authors address challenges like the design of collaboration across (hierarchical) network topologies, permission and privacy management, and forms of providing FL-based ML models to clients. However, the approach does not provide a service for building cohorts based on the client data distribution, which is relevant for non-IID datasets.

Since the original publication of this chapter in 2022, research in the area of FLaaS has of course further advanced. Mazzocca et al. [MRM$^+$23] propose a FlaaS-based approach that addresses the trustworthiness of the FL process. For this, clients authorize at the server with an identifier and associated claims. The claims are verified by the server to check whether the client is allowed to participate in an FL training. If the participation is allowed, the model that is trained by the client is validated on a server-side validation dataset. The resulting validation metrics influence the reputation of the client. The reputation is then considered as contribution weight to control to which extent a client should contribute to the global FL model. For example, if malicious clients want to contribute with a model that is not properly trained on a suitable dataset or certain weights have been manipulated (poisoned model), the reputation is poor and the contribution to the global FL model is avoided or decreased accordingly. With this approach, the authors ensure trustworthiness with respect to access and model contribution.

Zheng et al. [ZLL+23] focus on secure and communication-efficient FLaaS. To achieve this, the clients provide obscured model updates that can only be decrypted by the server using a public-key/private-key approach. This secure aggregation algorithm is extended with a model quantization approach. Therefore, the model updates are quantized (i.e., the numerical precision of the weights is reduced) on the client side after training and before they are sent to the server. The authors have demonstrated that these secured FL model updates can maintain both model accuracy and communication speed, competing with unencrypted approaches. In this chapter, we have not addressed security and communication efficiency. However, we see this as a crucial aspect to provide trustworthy FLaaS.

Han et al. [HWJH24] address the optimization of both FL training and inference services. In the proposed setup, the model is served by the clients and therefore the inference is provided by them at the edge of the network (e.g., for IoT-based applications or autonomous vehicles) The authors identify a trade-off between the training and the inference part, since both need compute and communication resources. Hence, to maintain hiqh-quality inference results, the model need to be trained and updated by FL continuously, which can in turn lead to a backlog of inference requests that need to be served. To optimize the inference performance, both with respect to speed and quality, an online decision-making algorithm (FedLS) is proposed to adaptively balance the resources between the model training and inference. In comparison to most of the published approaches (including our FLaaS-based approach), the inference service optimizations are added as relevant and novel features that need to be further investigated in the future as discussed in Chapter 8.3.

4.5 Summary

In this chapter, we have presented the *IFL System*, consisting of the *IFL Services* and the *IFL Client* that provide FLaaS for edge-based clients connected to industrial machines. The machine data is used by potentially multiple deployments of the *IFL Client* to collaboratively train ML models. For this, we introduced the *IFL Process* that includes a four-step approach enabling cohort-based FL and therefore addressing the challenge of non-IID data. We proposed a service architecture that supports the *IFL Process* by providing APIs to clients for participating in FL rounds.

As discussed, our *IFL System* has three main contributions, (i) explicit participation in FL on an on-demand basis, (ii) an architecture supporting individual and independent collaboration on shared ML models, and (iii) handling non-IID data distributions using cohort-building.

To the best of our knowledge, we provided the first cohort-based FL service system considering the characteristics and requirements of industrial clients. Furthermore, using client metadata and statistical moments for cohort building is a unique approach that both considers client preferences and local data distribution to handle non-IID data.

Evaluation on diverse and real-world industrial data is missing in current FL literature. We have shown the potential of our *IFL System* by applying it on two completely diverse and large real-world industrial datasets, used for pump condition classification and material classification. Our results also show the importance of a cohort-based FL approach, yielding higher accuracies on average compared to executing FL algorithms on the overall population.

This chapter has originally been published in the Journal of Parallel and Distributed Computing in 2022:

Hiessl, T., Lakani, S. R., Kemnitz, J., Schall, D., and Schulte, S. (2022). Cohort-based federated learning services for industrial collaboration on the edge. Journal of Parallel and Distributed Computing, 167, 64-76.

CHAPTER 5

Lifecycle Management of FL Artifacts in Industrial Applications

5.1 Introduction

In recent years, high volumes of decentralized IoT data have been collected in industry [MVM16]. In many cases, sensors are involved in recording this data, which is often analyzed in ML applications to e.g., classify critical machine conditions in production to provide timely maintenance services [KBG+22]. In these cases, privacy-preserving approaches such as FL became a promising factor [MMR+17], due to private or confidential data [SS15].

With the *IFL System*, we have presented an approach for FL in industrial setups in the previous chapters. So far, we have mainly addressed the FL interactions between clients and the server to provide a high-quality model. For this, aspects like the domain model, workflows, and architecture of IFL systems have been provided to support FLaaS. In contrast, this chapter delves into the development and execution support provided for FL-based applications. It encapsulates the journey from creating FL software artifacts, integrating them into client applications and using resulting FL models for inference. Hence, we present an approach that addresses the full lifecycle of an FL software artifact and propose a framework for development and execution. This framework reuses the *IFL Process* (see Chapter 4) as part of a Python library (*IFL Core*) that we introduce in this chapter. The *IFL Core* can be used by both the server and clients to run FLaaS and extend it with custom behavior. To illustrate the applicability, we demonstrate the integration of a FL-based clustering algorithm into a CM app.

To motivate our contributions for IFL, we discuss the challenges for providing FL services at scale and integrating applications that successfully solve industrial use cases.

First, typically, models resulting from FL are supposed to serve many clients [KKP20]. Therefore, a scalable use of FL-powered applications is intended. However, not all clients operate in the same context. For example, data structure, data quality, and requirements for how the models are integrated into applications and services can be different [RBP+22, SYL21]. Hence, there is a need for customization, as for example, client-specific pre-processing before the model is trained to align the input data with the global model. Furthermore, as the model is typically used in prediction, client-specific post-processing may be applied before the output is served to applications. Combining scalable deployments with customization, one should aim for serving as many clients as possible with a given AI solution and still addressing local client contexts.

Second, the development and deployment of scalable and still customizable IFL-based solutions [HSKS20] is a multi-step process often involving data scientists, IT professionals and operators [LK18]. Partially, this is addressed by ModelOps approaches [HMR+19]. These approaches support the operationalization of models by executing a lifecycle with automated pipelines covering e.g., data processing, (re-)training, validation, and deployment. This bridges the gap from model development to model operation in production. To provide this for FL, the clients on the edge layer need to be involved in the lifecycle as well [RBWA21]. However, managing the lifecycle of a model resulting from FL does not address client-specific customization and adaptation to the local context. We identify the need for end-to-end lifecycle management of software artifacts that address client context-aware processing steps for this purpose. This enables customized and extensible FLaaS, which in turn can be integrated into edge-based client applications.

Third, running analytic edge applications like CM on the shop floor demands repeated updates to react to changing operation modes or new business requirements [HSKS20]. Presuming that edge apps make use of FL to e.g., improve classification of asset conditions, FL capabilities may need to be updated as well. This may include algorithmic updates of client or server training for optimizing model performance, updates of communication patterns and protocols due to security reasons, or simply changes in the format of input data or resulting predictions. For this, in FLaaS scenarios, operators on the shop floor need to have the flexibility to either explicitly pull updated artifacts or have them pushed by the central authorities that manage the service.

Fourth, in industrial scenarios, labels are often missing entirely and hence unsupervised problems like clustering are faced [SSA+21]. For instance, to cluster operating conditions in production processes, one can use a cluster model trained on the data of other sites to improve the detection of previously unseen anomalies. Similarly to supervised FL scenarios, one cannot assume having access to third-party data for improving local models. Therefore, there is a need for deploying a privacy-preserving FL artifact and applying a collaborative clustering algorithm to provide FLaaS to edge clients.

This chapter provides the following contributions:

- The design of an *IFL Lifecycle* that supports the software development, publishing, deployment and execution of FL solutions based on customizable software artifacts (*IFL Templates*, or in short, templates).
- Implementation of the *IFL Lifecycle*, including the main components such as the *IFL Core* (Python library for FL), and a distributed clustering algorithm, named FedClust. The *IFL Lifecycle* enables the *IFL Process* from Chapter 4, as it is used in the *IFL Core*-based FL solutions.
- Integration of the approach in industrial CM applications for providing and consuming extensible FLaaS.
- Evaluation of the *IFL Lifecycle* with clustering scenarios using industrial data to show the potential for improvements to the underlying ML model and the system's ability to facilitate collaboration among multiple clients.

For this, we make use of a small subset of IFL components from the previous chapter (Section 4.2.3) in the proposed *IFL Core*, i.e., to register clients, store and manage base ML models, as well as to aggregate models. The modular *IFL Core* library is subject to be integrated in FL solutions that can be managed along the proposed *IFL Lifecycle* and integrated into applications. Compared to the *IFL Services* in the previous chapter, we ensure extensibility of IFL functions (e.g., client training, server aggregation) and customizability of holistic FL solutions. This FL solution-centered approach includes the customization of steps as e.g., pre-processing of input data, FL, model validation, and post-processing of FL results.

The remaining parts of this chapter are organized as follows: In Section 5.2, we describe the *IFL Lifecycle* management and its main components. In Section 5.3, we demonstrate the results of the clustering experiment that we integrated into a CM application. We review related work in Section 5.4 and conclude in Section 5.5.

5.2 System Design

5.2.1 Basic Notation

To describe our approach, we define and describe the main entities used in the *IFL Lifecycle*. First, the template is a software artifact e.g., a Python module or Jupyter notebook with dependencies, that can be created to implement ML/FL models in dedicated functions considering e.g., data preprocessing, training, aggregation, validation, and prediction. The template can be packaged and published to a server. Based on that, the template can be shared with clients for distributed execution of FL. For this, the server executes the *Server Stage* part of the template as defined in a dedicated function. This function contains logic for running the FL services using the *IFL Core*. The *IFL Core* is a library and a framework that covers a significant part of FL lifecycles, i.e., client registration, local training, model and knowledge aggregation, and distributed model validation. Clients also use the library during the *Client Stages*. These stages are specific to client parts of FL lifecycles and therefore implement local data preprocessing, registration requests to the server, training, or validation based on local held-out test datasets. Clients download the templates from the template registry and run the corresponding stages to participate in FL. In both *Server Stage* and *Client Stages*, the *Template Runtime* executes the respective functions. The *Template Runtime* provides interfaces to load (training) data as input for the templates, return trained models for local use, and apply model inference (prediction) using the trained model based on ingested features.

5.2.2 Templates

To address the customizability of FL solutions with distributed client contexts (i.e., client data), we present how templates can be defined for a distributed FL deployment. Listing 5.1 depicts a basic FL template implemented in Python. To identify *Server Stage* and *Client Stages* (i.e., training and prediction), the *t.exec* decorator is defined accordingly. This is necessary for the server and client runtimes to perform related functions. The *Server Stage* is implemented in lines 1-3, which run the FL services provided by the *IFL Core*. The client training stage is defined in lines 5-8. As this is executed on client locations, local data is passed to the function to be used in FL training rounds in line 8. Note that this is the part where the *IFL Process* (Chapter 4) is executed to train and provide an FL model. The context of the client can be used to inject local configurations and to e.g., store models to be applied in prediction (line 10-13).

Based on that, data scientists, for example, can enhance the template with client-context-aware pre- and post-processing behavior. Finally, the template can be packaged and uploaded to a server for scalable use in multiple deployments and applications, as addressed in Section 5.2.4.

Listing 5.1: *IFL Template* definition in Python

```
1   @t.exec(Stage.SERVER)
2   def start():
3       ifl_core.start_server()
4
5   @t.exec(Stage.TRAINING)
6   @template.inject_context()
7   def train(data, context):
8       context['model'] = ifl_core.train_FL(data)
9
10  @t.exec(Stage.PREDICTION)
11  @t.inject_context()
12  def prediction(features, context):
13      return context['model'].predict(features)
```

5.2.3 IFL Core and FL-enabled Templates

The FL capabilities on which *IFL Templates* are based, are provided by the modular and extensible architecture depicted in Figure 5.1. The library is not limited to being used just by templates. As part of the API, the classes *IFL Client API* and *IFL Server API* provide access for client participation in FL rounds, and for hosting *IFL Services* respectively. Flexibility is provided for different means of implementing FL. On the one hand, an FL solution can be defined centrally, i.e., as a template that can be deployed to a distributed fleet of edge clients. On the other hand, FL can be provided by hosting a server and providing the *IFL Client API* to different applications for collaboratively solving ML tasks.

Note that in contrast to the *IFL System* in Chapter 4, the *IFL Core* is a portable library for both server and client instead of providing a static FL server. This facilitates the integration into *IFL Templates* that can be deployed as holistic FL solutions.

To start, the client invokes the function *start_FL(...)* of *IFL Client API* and provides information of the task the system needs to execute. This includes the data, the FL algorithm to be applied, collaboration criteria (e.g., minimum number of clients) and the model to be trained (see Chapter 4). To define the mean of communication (e.g., MQTT) used for client server interaction, a respective *Communicator* needs to be selected or customized. For this, *IFL Core* provides MQTT support out-of-the-box by defining host, port and potential credentials for a given MQTT broker. The *Registry* is an internal store of metadata used by the server to manage submitted tasks. According to these tasks, the server executes the corresponding protocol of collaboration, which we term flow. In particular, *FLFlow* covers basic supervised FL execution with FedAvg [MMR+17] which averages weights of parametric models and shares the resulting global model with clients. The *FedClustFlow*, which we present in Section 5.2.5 in detail, solves distributed clustering problems by facilitating knowledge exchange between the clients. The currently supported flows are compositions of Commands (CMDs), which are chained to cover e.g., *Registration* of clients, *Training* of local models, *Upload Local Training Results*, *Global Training* of a global model, sharing the global model with *Global Model Update*, and *Validation*. The *Trainer* module provides interfaces and respective default implementations that are invoked by the CMDs to realize client training (*Default Client Trainer*), model aggregation and global model training (*Default Server Trainer*).

The trainer interfaces provide extensibility to the *IFL Core* consumers, since individual implementations can be passed to override behavior in the defined flows. Furthermore, communication protocols can be integrated with the communicator interfaces, to consider potential security and performance aspects specific to an edge client's environment. Based on that, this addresses the need for extensible and modifiable FL capabilities required by industrial operators.

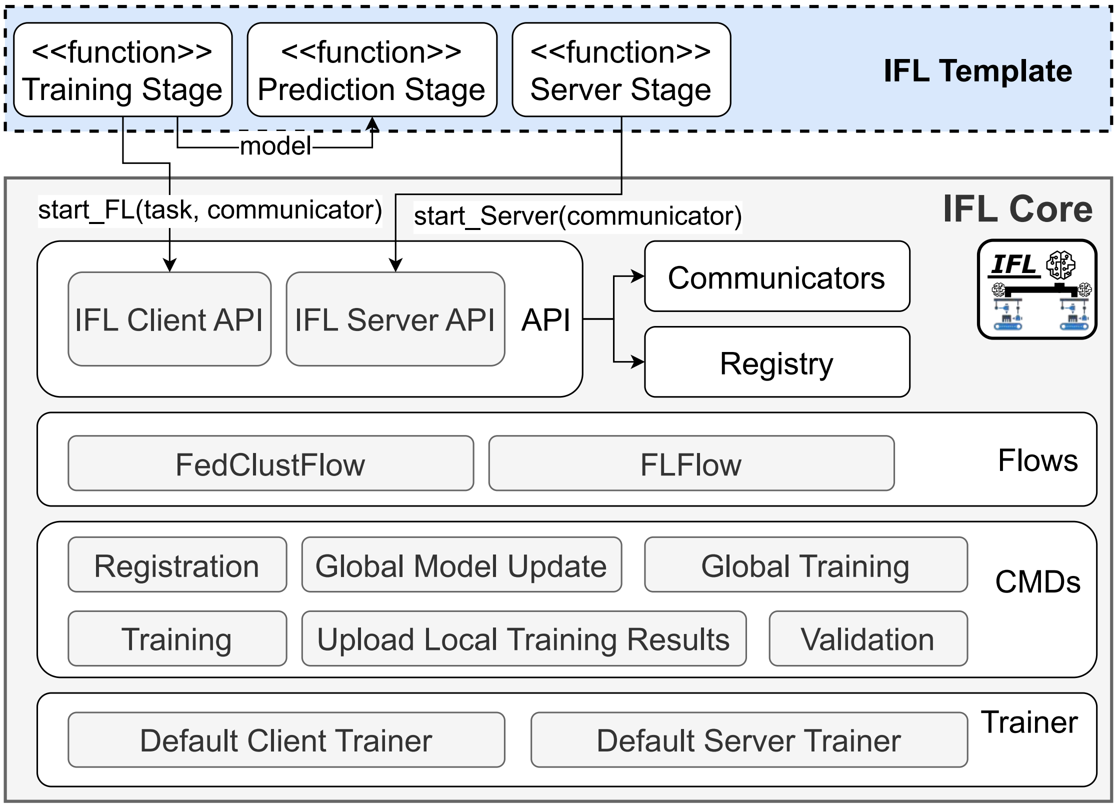


Figure 5.1: *IFL Core* architecture with template as consumer

5.2.4 IFL Lifecycle

To provide end-to-end support for *IFL Templates*, reaching from development to continuous operation, we present the *IFL Lifecycle* in Figure 5.2. Note that the *IFL Process* (Chapter 4), which includes selected WFs of Chapter 3, is only a part of the *IFL Lifecycle*. The *IFL Process* covers the client-server interplay in a FLaaS style as the template is executed.

Based on Section 5.2.3, the first steps in the template lifecycle involve the development, testing and packaging of templates in the development environment of a data scientist. These phases are supported by the template framework that comprises function decorators as shown in Figure 5.2 and the *Template Runtime* for executing developed templates. The next step is to publish the packaged template to a server (e.g., via file upload), which is then triggered to deploy one template instance to a server-hosted *Template Runtime*. This executes the *Server Stage* and therefore makes use of the *IFL Core*. To execute the corresponding *Client Stage*, the server could either deploy the template to the runtimes of clients that are connected to the server (push), or clients could download the template on-demand (pull).

With this, the control of FL participation can be centralized using the push approach, or

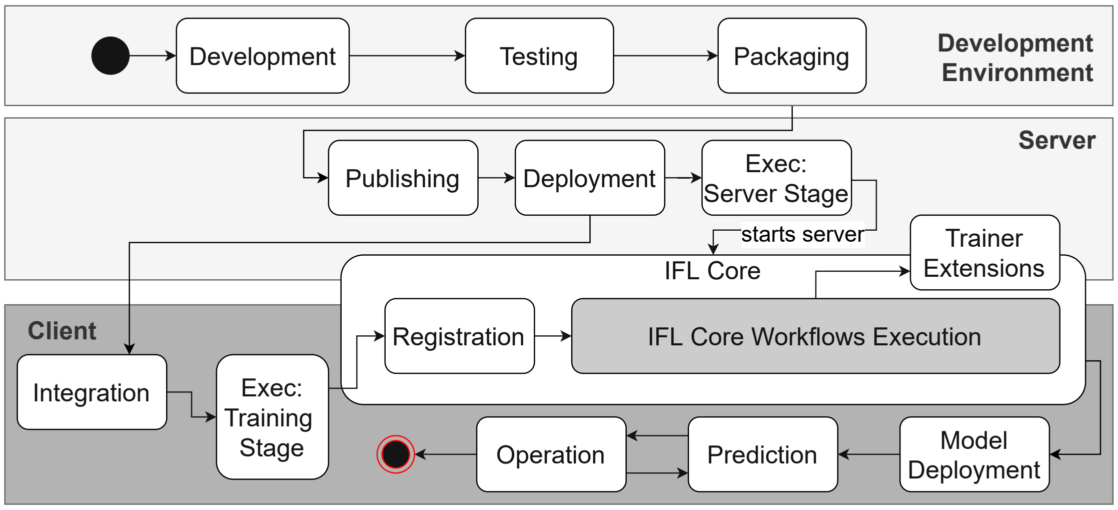


Figure 5.2: *IFL Lifecycle*

participation can be chosen by the decentralized clients using the pull approach. The push approach can be used to train a global model on data that cannot be accessed e.g., by a data scientist who operates on the server side. In contrast, the pull approach is more suitable for data scientists or operators who are working on the edge client side and want to explicitly consume FLaaS as presented in Chapter 4.

To address the extensibility of scalable edge-based client applications making use of FL, Figure 5.2 considers an integration step. For this, the application consumes models of FL templates after executing the training stage with the client runtime. Since new templates can be added and existing ones can be updated with new deployments on an ongoing basis, the application is continuously updated with e.g., model training or pre- and post-processing logic.

Finally, the template execution results in new model deployments that can be used in continuously executed prediction to serve the client (application) with results during operation.

5.2.5 Federated Clustering

For distributed industrial edge clients, labels are often not present at every client location. Therefore, by applying cluster algorithms in a collaboration with multiple clients, cluster models can be shared and used to improve the local identification of clusters. Hence, to address this style of FL and to present a concrete algorithm that is realized in *IFL Core* to be used in *IFL Templates*, we introduce FedClust in Algorithm 4.

FedClust consists of a client part (lines 1-10) and a server part (lines 11- 15). The client starts with local clustering using k-means [KMN$^+$02] and the elbow method [Tho53, SAIR11]. The k-means algorithm is a method of grouping data points into a fixed number of clusters, based on minimizing variance within each cluster. The elbow method is used to determine the optimal number of clusters by iteratively applying k-means with increasing values of k, and stopping when the improvement in cluster optimization is no longer significant.

For each cluster obtained with k-means, the client computes its centroid (line 5) and adds it to the set of data to be disclosed to the server (line 6). Then, the client randomly selects some data points from the cluster according to the privacy disclosure parameter *pd* and adds them to the disclosed dataset (line 7). The number of samples can be controlled with *pd*, ranging from 0% (privacy preserving) to 100% (disclosing all data samples) In the case of sharing 0% of data samples, only the centroids are shared. In line 9, the client sends the disclosed data to the server and receives a trained global model.

With this, the resulting clusters are equal for all clients, which we refer to as cluster alignment. With aligned clusters, a given cluster label can be interpreted equally at every client location. This facilitates the scalable use of applications deployed to multiple client locations, each potentially relying on cluster results as served by a template.

On the server side, the algorithm first calculates the optimal number k_{global} of clusters using the elbow method on the disclosed data received from all clients (line 11). If k_{global} is less than or equal to 1, the server defaults to the maximum number of clusters provided by any client (line 13). This avoids trivial clustering with one single cluster, which can be the case for small numbers of disclosed data samples. Then, the server applies k-means clustering on the disclosed data using the determined number of clusters to obtain the global model (line 15) that is shared with all clients (line 16).

5.3 Evaluation

In this section, we demonstrate how FedClust is executed as a template in the *IFL Lifecycle* integrated into an industrial CM system. We evaluate the applicability of the overall system and the effects of collaboration on an unsupervised problem with real-world vibration time series data from pumps. Compared to the previous chapter, we make use of the same dataset but consider only four pumps and no labels for training, since the underlying ML problem is about clustering similar vibration patterns. The scenarios cover different numbers of clients and privacy settings.

5.3.1 Experimental Setup

The experiments are executed on a server and multiple clients, each running on a Linux machine with a 3.3Ghz Intel Xeon Scalable Processor with 2 CPU cores and 4 GiB RAM. Server and client applications are implemented with Python[1] using a common MQTT broker for asynchronous message exchange. To execute data pre-processing, clustering, model-sharing and validation, the implemented FedClust template and the invoked *IFL*

[1] `https://www.python.org/`, accessed 2024-07-22

Algorithm 4 `FedClust`

```
Input: training data $TRAIN$, privacy disclosure $pd \in [0, 1]$
    Clients executes:
 1: $D \leftarrow \{\}$ // data disclosed for server clustering
 2: $k \leftarrow elbow(TRAIN)$
 3: $C \leftarrow kMeansClusters(TRAIN, k)$
 4: for $c \in C$ do
 5:     $centroid_c \leftarrow \frac{1}{|c|} \sum_j^{|c|} a_j$
 6:     $D \leftarrow D \cup centroid_c$
 7:     $D \leftarrow D \cup randomSelect(c, pd)$
 8: end for
 9: $m \leftarrow getFedClustModelFromServer(D, k)$
10: return $m$

Input: client data $CD \leftarrow \bigcup_i^n D_i$, cluster counts $K \leftarrow \bigcup_i^n k_i$
    Server executes:
11: $k_{global} \leftarrow elbow(DC)$
12: if $k_{global} \ngtr 1$ then
13:     $k_{global} = \max_i^{|K|} k_i$
14: end if
15: $m \leftarrow kMeansModel(DC, k_{global})$
16: $shareWithClients(m)$
```

Core make use of PyTorch[2]. The template code is developed in a Jupyter notebook[3], then compiled to a corresponding Python class containing all the implemented functions and stages, and finally packaged with all dependencies to be provided for decentralized execution.

5.3.2 Data

The FedClust template is applied to vibration time series data that consists of three dimensions with 512 samples per minute measured from four pumps. Multiple measurements [KBG+22] have been carried out using the *SITRANS Multisensor* [BGvDH19]. The dataset considers six conditions (healthy and anomalous) of the underlying pumps as described in Section 4.3.2. The healthy conditions comprise healthy stationary normal load, healthy stationary partial load, and idle state (pump is turned off). In contrast, hydraulic blockade (rotation in the pump is not possible), dry-run error (air in the pump), and cavitation error (pumped water vaporized at low pressure) are the anomalous conditions.

The pumps use case is suitable to demonstrate IFL scenarios to support industrial collaboration, since there are several conditions to be clustered and assets generating relatively similar data. Clients can improve local condition clustering, using the global

[2]`https://pytorch.org/`, accessed 2024-07-22
[3]`https://jupyter.org/`, accessed 2024-07-22

model resulting from the *IFL Lifecycle*. However, our approach is not limited to pumps, and can be applied to other data since dataset-specific processing can be covered in templates and *Trainer* functions.

5.3.3 Scenarios and Experimental Design

We consider two scenarios, each with a different number of involved clients, to test collaboration on small (four clients) and larger (33 clients) scales.

Scenario 1 - Four Clients

Scenario 1 involves four clients with data from a measurement of one pump with four sensors attached at different locations. Each sensor corresponds to one client. The distribution of conditions (labels) is similar across all clients. However, we consider two subscenarios, each with six conditions, where Scenario 1a has an unbalanced dataset with one dominant label (healthy stationary partial load), while Scenario 1b considers a balanced dataset. This is relevant to investigate the impact of label distribution per client. Although there are generally no privacy restrictions among the four clients operating on the same pump, data sharing is avoided in this scenario. This is done to demonstrate the applicability of FL for clients with sensors located at different locations. In this scenario, we exclude the extension of handling a default k as defined in line 12 of Algorithm 4.

The clients train on 20% of the local data and validate the cluster models on 80% of the local data. The data is split randomly to ensure similar data distributions in training and test set.

Scenario 2 - 33 Clients

The larger scale Scenario 2 is based on 33 clients, with heterogeneous label distributions to reflect real-world operations in different environments. For this, all four pumps are involved in eleven measurements with three sensors used per measurement. Three subscenarios are considered to compare the impact of balancing datasets and a variation in the cluster algorithm. The latter considers setting a default number of clusters to be identified on the server's global clustering if the elbow method does not find an optimal k. With this change, all samples should no longer be assigned to one cluster. So, Scenario 2a considers unbalanced datasets on every client location, Scenario 2b additionally considers the algorithmic extension using the default number of clusters, and Scenario 2c uses balanced data and the default number of clusters.

Variants

For any of the aforementioned scenarios, we further investigate how FedClust impacts the clustering of (unseen) classes that are not part of the training set. In Scenario 1, we remove 50% (three out of six) of the labels, including input data, from the training set. In Scenario 2, we remove only 16.6% (one out of six) since some of the 33 clients have no more than three conditions in the dataset. We additionally investigate the impact of disclosing privacy (i.e., clients upload raw data samples to the server) on multiple levels considering 10%, 20%, 50%, and 100% of data sample uploads.

Performance Metric: Purity

To compare the scenario runs as they are executed in the *IFL Lifecycle*, performance metrics are computed in the validation phase of the *IFL Core*. For this, the FedClust template makes use of the purity metric [Man08], which is defined as $purity = \frac{1}{N}\sum_{i=1}^{k} max_j|c_i \cap l_j|$ where N is the number of samples, k is the number of clusters, c_i corresponds to the i-th cluster, and l_j is the set of elements with the label j representing the ground truth. The purity can range from 0 to 1 and is computed on every client based on local test data and shared with the server to compute the average purity. We compare individual clustering with global clustering sharing centroids and global clustering sharing additional data samples. For individual clustering, purity is computed based on the individually trained model, while for the other two approaches, the global model is used for computing the purity. To enable more generalizable conclusions, we run all scenarios three times and report the average of the collected (average) purities.

5.3.4 Scenario Execution using the IFL Lifecycle

To demonstrate how the *IFL Lifecycle* can be integrated to facilitate FL (i.e., FedClust) on IIoT data, we describe the process of executing the scenarios with an implemented IIoT-based application for CM. The task of the CM system (see Figure 5.3) is to execute templates on ingested sensor data (i.e., cluster pump conditions) on every client with the help of the *IFL Core*. The CM system provides the *IFL Lifecycle Cockpit* to publish and distribute templates to a fleet of client devices. The client devices host the CM app for collecting sensor data, creating datasets and starting the training via the runtime to finally make use of the resulting model for continuously predicting conditions based on streaming data. As depicted in step 1 in Figure 5.3, we assume the role of a data scientist and develop the FedClust template, invoking Algorithm 4 as provided by the *IFL Core*. To provide FedClust to multiple clients, we publish the template to the *IFL Lifecycle Cockpit* Web application via file upload in step 2.

To run the *Server Stage* that makes use of lines 11-15 in Algorithm 4, we start the template deployment in step 3a. This causes the *IFL Lifecycle Cockpit* to deploy the template to the runtime on the server in step 3b. Assuming the role of the operator of the local CM process, we start with step 4a by onboarding the sensors (define MQTT topics for upstreaming sensor data) and assigning the data stream to a logical entity, namely an asset, representing the pump.

In the execution of the scenarios, the sensor data ingest to the respective client MQTT brokers is started in step 4b, making use of a Python script that simulates the data upstream from pumps. This is relevant to enable the CM app to collect a dataset for the training. To apply clustering to the data, as defined in the client part of the

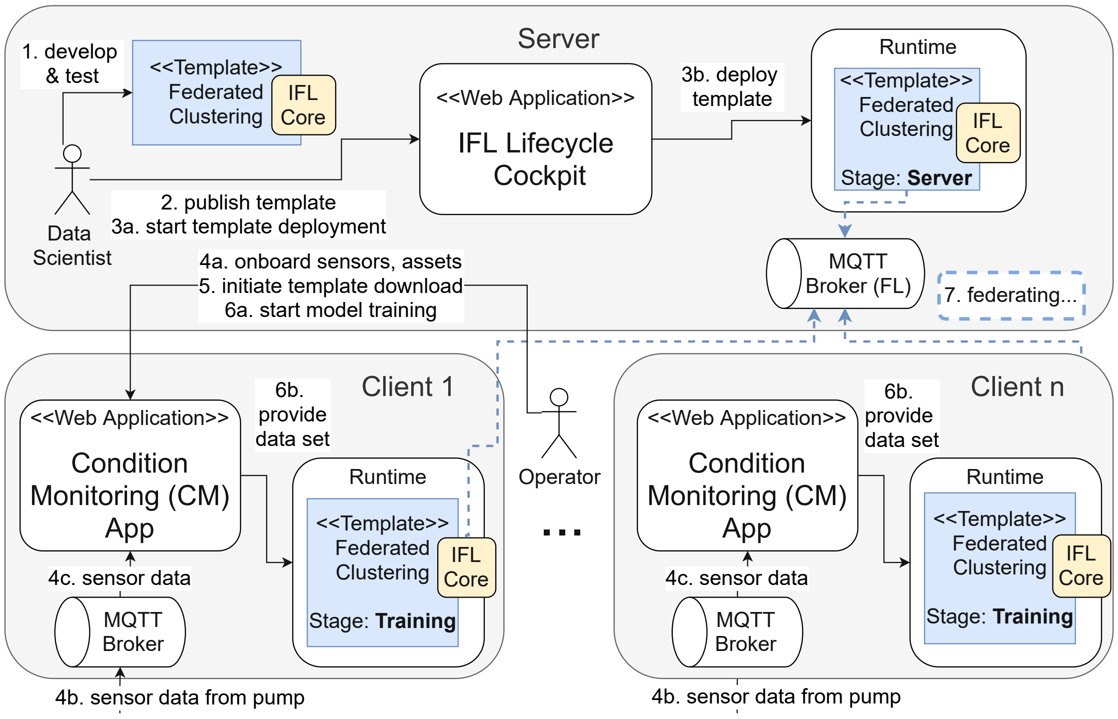

Server
1. develop & test
<<Template>> Federated Clustering
IFL Core
<<Web Application>> IFL Lifecycle Cockpit
3b. deploy template
Runtime
<<Template>> Federated Clustering
IFL Core
Stage: Server
Data Scientist
2. publish template
3a. start template deployment
4a. onboard sensors, assets
5. initiate template download
6a. start model training
MQTT Broker (FL)
7. federating...
Client 1
<<Web Application>> Condition Monitoring (CM) App
6b. provide data set
Runtime
<<Template>> Federated Clustering
IFL Core
Stage: Training
4c. sensor data
MQTT Broker
4b. sensor data from pump
Operator
...
Client n
<<Web Application>> Condition Monitoring (CM) App
6b. provide data set
Runtime
<<Template>> Federated Clustering
IFL Core
Stage: Training
4c. sensor data
MQTT Broker
4b. sensor data from pump

Figure 5.3: Evaluation: Condition monitoring system integrating the *IFL Lifecycle* to perform FedClust of pump conditions

template (invoking lines 1-9 of Algorithm 4), we initiate the template download to the client device in step 5 and start the training in step 6a. This triggers the CM app to provide the dataset to the runtime and starts the template execution. Finally, the FedClust knowledge exchange between the clients is performed (step 7: federating) and the resulting global models are returned to the involved client CM app deployments. The validation is performed in the template and the CM app stores the resulting purity metrics that are presented in the next section.

5.3.5 Results of the FedClust Template Execution

The resulting purity metrics of the FedClust template execution are now presented for each of the defined scenarios to demonstrate the effect of collaboration on local cluster problems facilitated by the *IFL Lifecycle*. As a result, we compare the purity of individual training (clients train on their own data without federation), centroid sharing (federation with only client centroids shared), and federation with various disclosing factors.

Scenario 1: Four Clients

The average purity of federated clustering with centroid sharing is close to that of individual training, as shown in Figure 5.4a and Figure 5.4c. The reason why centroid sharing is not outperforming individual training is that clients face the same classes in the validation set. Hence, no new knowledge is transferred between the clients. However, for individual training, there is no cluster alignment over the involved clients, which is the case for centroid sharing.

In Figs. 5.4b and 5.4d, we observe higher (unbalanced case) and slightly higher purity (balanced case) for centroid sharing as compared to individual training. Since we have randomly removed 50% of the labels (incl. corresponding input data) from the training set, clients require knowledge from collaborators to correctly cluster test data samples with unseen conditions. This can improve CM scenarios that use clustering, as not all conditions might be present at one location.

We observe that disclosing more data does not yield better results; centroid sharing is enough for federated clustering. As shown in Figs. 5.4a and 5.4b, when dealing with unbalanced data, disclosing more data (disclosure factor greater than 0) may result in lower purity than simply sharing the centroid. This is due to the fact that data is selected randomly for disclosure, and data items might be from the most dominant cluster. As can be seen, the results for higher disclosure factors are similar to the centroid sharing in the case of a balanced dataset (Figs. 5.4c and 5.4d).

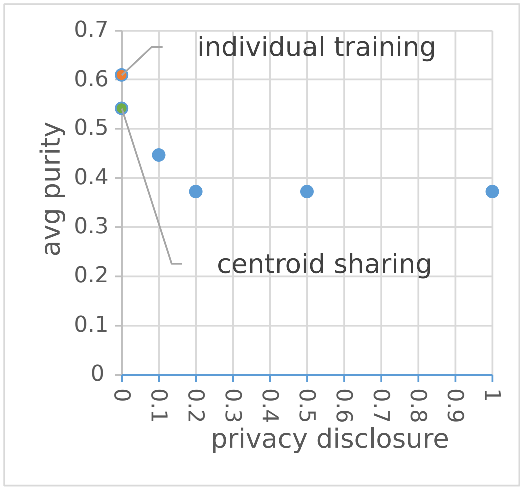
However, the problem of not finding a proper k with the elbow method on the server (see line 11 in Algorithm 4) still persists for some lower privacy disclosure levels. Hence, only one cluster was built globally, and the purity decreases dramatically as validating the global model for the clients. For this, we investigate the impact of using a minimum default k in Section 5.3.5.

Scenario 2: 33 Clients

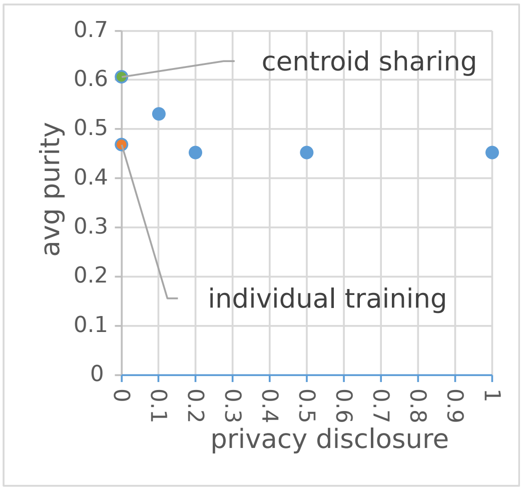
We now evaluate FedClust with 33 clients to investigate a large-scale CM setup for clustering and the effect of default k used on the server with both unbalanced and balanced data.

In Figs. 5.5a and 5.5b, we see that collaborative runs (centroid sharing and privacy disclosure) produce relatively poor results when compared to individual training. This changes when the default k is used, as shown in Figs. 5.5c-5.5f. Without that, only one global cluster is built in the global model, which decreases purity. Furthermore, we can see that balancing data does not further increase purity if the default k is already set. This suffices for the used dataset, but in cases where more conditions are present, the default k might just be a lower boundary and balancing could help to identify more clusters.

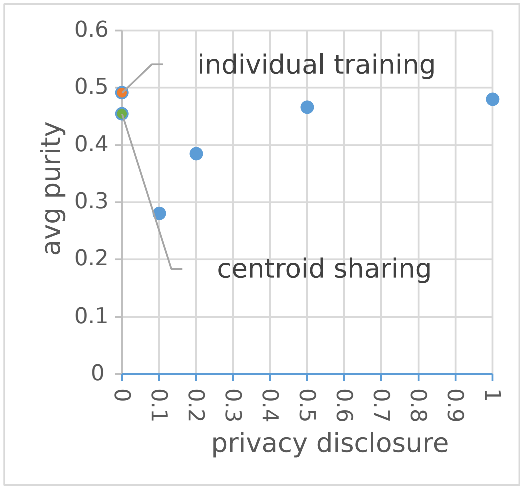
The centroid sharing approach shows competitive performance especially in cases with unseen labels in test data. In Figure 5.5f, centroid sharing outperforms individual training, which was harder than in the four client scenario, since we now only consider one unseen label. This underlines the positive impact of sharing knowledge in a collaborative approach.



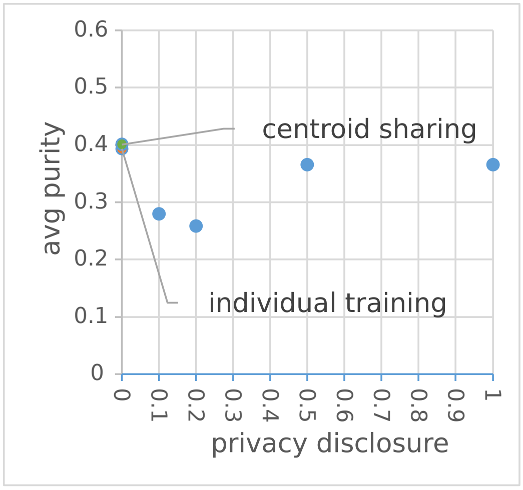
(a) unbalanced data



(b) unbalanced data and unseen labels



(c) balanced data



(d) balanced data and unseen labels

Figure 5.4: Scenario 1: Average purity results of FedClust with four clients

When we use the default k, we see that increasing the privacy disclosure factor does not significantly increase purity. Hence, centroid sharing is sufficient in the presented scenario and no raw data is revealed to the server. Therefore, there is no natural trade-off between privacy preservation and performance.

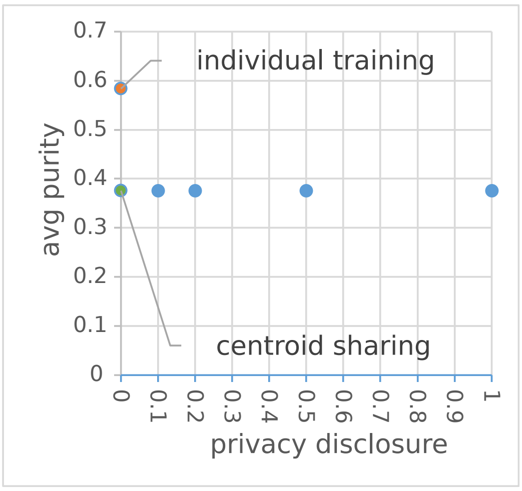
The centroid sharing results from the 33 clients scenario with unbalanced data (Figs. 5.5a and 5.5b) are now compared with the four clients scenario with unbalanced data (Figs. 5.4a and 5.4b). As it can be seen, the four clients scenario provides higher purities in centroid sharing than the 33 clients scenario. Although there are more collaborators in the latter scenario, the heterogeneous data distribution makes it hard to leverage the potential. Here, both the unbalanced data at the client locations and data distributions, which are relatively non-IID between involved clients, can have an impact. The phenomenon of non-IID data is a general issue in FL [HPMG20], and the presented results show that this is valid for FedClust as well. In particular, given a group of many clients with similar data distributions, the group might dominate the global clustering with their shared cluster centroids (and potentially shared data samples). Hence, centroids of clients that have different data distributions could be seen as outliers on the server and are not considered as separate clusters. Therefore, increasing the number of collaborators requires non-IID-aware approaches, such as grouping clients with similar data distribution into cohorts and restricting global model training to occur only within these cohorts, as presented in Chapter 4.

5.3.6 Discussion on Scalability in Industrial Setups

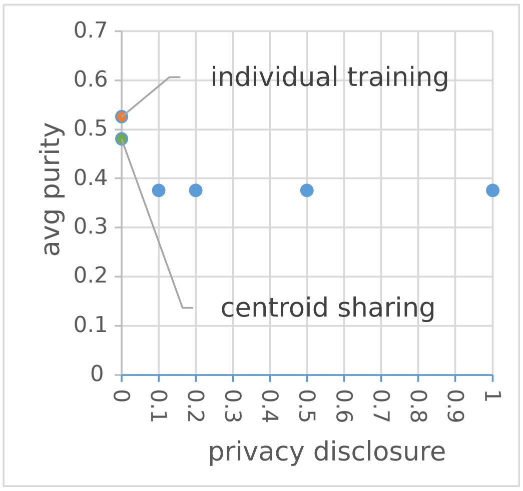
The presented scenarios demonstrate that the FedClust template can be provided to more than 30 clients and executed respectively. Once the template is developed, packaged, and published, it can be executed by any participating client concurrently. However, the FL interactions (i.e., FLFlows provided by the *IFL Core*) require a central server and some clients might drop out in real-world settings due to limited compute resources and connectivity [NY19]. An additional overhead for larger real-world setups is the matching of clients that want to collaborate with each other and have similar data distributions (see Chapter 4). Therefore, further evaluation needs to address these overhead aspects that could limit the scalabilty for larger setups.

5.4 Related Work

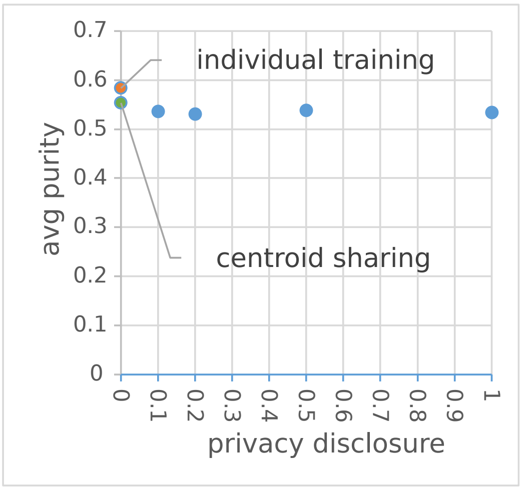
Lo et al. [LLPZ21] provided a reference architecture for FL systems, including patterns to address recurring design problems. The patterns have been surveyed based on a Systematic Literature Review (SLR) and consider significant parts of an FL lifecycle e.g., registration, client clustering, client selection, training, aggregation, evaluation, monitoring, and model deployment. However, the patterns in the lifecycle consider the model as the main managed entity, whereas our approach manages the template as model-generating entity in the *IFL Lifecycle*. For this, we support phases before clients register for FL-based model training, to support development, packaging, publishing,



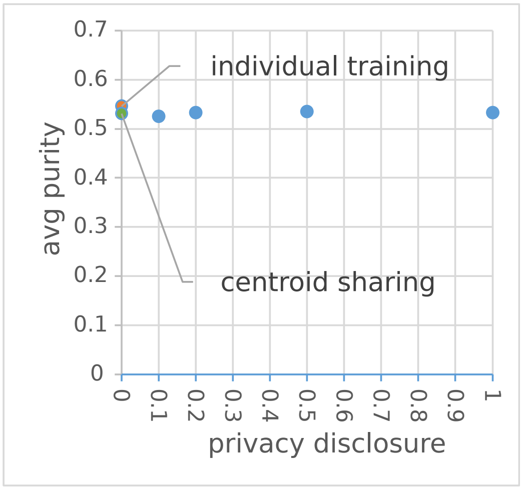
(a) unbalanced data



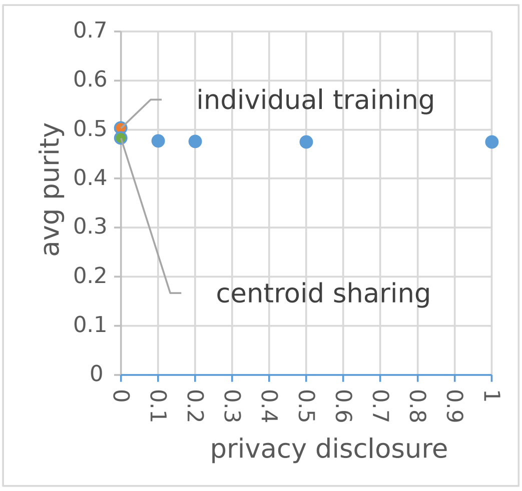
(b) unbalanced data and unseen labels



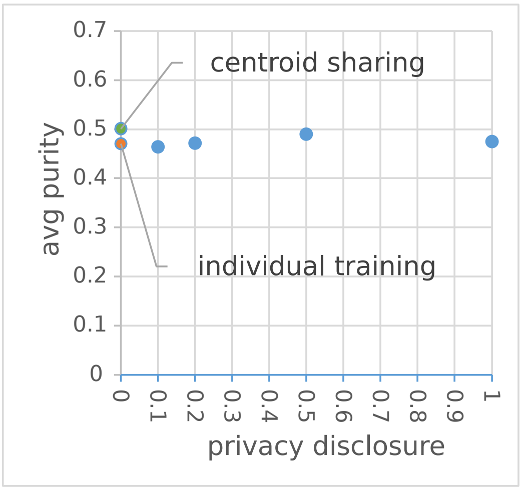
(c) unbalanced data and default k



(d) unbalanced data, unseen labels, default k



(e) balanced data, default k



(f) balanced data, unseen labels and default k

Figure 5.5: Scenario 2: Average purity results of FedClust with 33 clients

roll-out, and execution of the defined FL behavior, and the operations of FL models in the prediction stage.

Kairouz et al. [KMA+21a] propose an FL lifecycle with phases for problem identification and client instrumentation. While the former one is a purely manual task of the model engineer, the latter addresses the instrumentation of client devices with data processing and the collection of additional metadata (e.g., labels). With this, the authors consider lifecycle phases even before the model is instantiated and can be trained. However, this refers only to a high-level workflow that is not integrated into a lifecycle management environment as we demonstrated with the integration into the *IFL Lifecycle Cockpit* and the CM app in Section 5.3.4.

Zhuang et al. [ZGWZ22] introduce EasyFL, a low-code FL platform that addresses an FL lifecycle with two phases. First, an implemented FL algorithm is tested in the experimental phase. Second, the tested FL algorithm is deployed into production, where it trains and evaluates on distributed client data until the resulting model is served. Developers make use of a Python library with minimal coding effort to run pre-defined FL algorithms, while for expert users algorithms can be customized. EasyFL provides an API to register (containerized) server nodes, client nodes, models, and datasets. The platform executes the defined algorithm in the respective server and client training flows. Similar to our approach, EasyFL supports the execution of customizable FL code. However, the lifecycle followed by EasyFL does not address the management of FL artifacts, i.e., providing ways to share FL code and allow clients to pull these artifacts to make use of FLaaS. Furthermore, FL is executed by a central authority (e.g., data scientist), while individual clients can just be registered as worker nodes.

Beutel et al. [BTM+20] present Flower, an FL framework to implement customizable client and server behavior. With Flower, client training, validation, and server aggregation can be implemented using either pre-defined functions (i.e., *strategies*) provided by the tool or by customizing the functions to fit specific needs. The communication protocol for client-server interaction is based on synchronous remote procedure calls, which can also be updated with a custom protocol (e.g., MQTT). Similar to Flower, the *IFL Core* provides out-of-the-box *Trainer* functions (corresponding to Flower *strategies*) and customizable communication protocols. However, the *IFL Core* provides each client an interface for submitting tasks to individually define e.g., algorithms and collaboration criteria. With this, clients can individually impact the FL process given the local context. Hence, not all clients have to follow the same behavior (i.e., *Trainer* functions or *strategies*).

AWS SageMaker[4] inference pipelines enable the definition of ML training and corresponding data processing to deploy resulting models to AWS resources and to serve inference requests. The deployment is supposed to serve all client requests that follow a pre-defined schema for the payload that is used in the model inference. However, at this time, there is no integrated solution suitable for deploying inference pipelines to edge devices for

[4] `https://aws.amazon.com/de/sagemaker/`, accessed 2024-07-22

serving clients with FLaaS [DC21]. Hence, there is no client-specific customization for processing FL-based inference results.

This holds true for Microsoft Azure[5] (ML and Edge Stack) as well. It enables to download and (re-)train ML models and use them for inference on edge devices, however, FLaaS is not supported for client locations out-of-the-box.

The Katulu FL Suite[6] provides a Software Development Kit (SDK) integrating Flower to define and deploy containerized pipelines to distributed compute resources (e.g., edge devices). This enables to execute FL jobs on a server and multiple involved clients pushed by a central authority. In contrast, our approach further considers to pull templates with defined FL pipelines to participate in FLaaS.

Mashhadi et al. [MSM21] introduce an FL clustering approach based on DL models. For this, the authors pretrain a cluster model on the server based on an incomplete centralized dataset. Subsequently, the FedAvg algorithm is involving the clients for optimizing the DL. Although FedClust considers the option for privacy disclosure, our approach makes no initial assumption on a centralized dataset that needs to be on the server.

Based on that, we provide two novel aspects: 1) an approach for managing the lifecycle of a model-generating entity (template) instead of managing a model lifecycle, and 2) customizable and still scalable FLaaS.

Since 2023, when this chapter was originally published, there have been advancements in research and development of FL frameworks. Beltrán et al. [MBPS+23] surveyed FL frameworks with an emphasis on decentralized FL. This approach considers the aggregation of individual contributions to be carried out by the clients, rather than centrally by the server. For example, FedML[7] is an FL library that enables FL for a distributed fleet of edge devices [HLS+20]. In combination with the FedML-based platform named TensorOpera AI[8], FedML can be used in FL software artifacts that are deployed to registered clients (e.g., smartphones, browsers, data silos) for model training. The responsibility of aggregation within a certain topology can be defined in the algorithm that is deployed. In comparison to our template approach that also considers inference services for applications as part of the *Client Stages*, the integration of FedML artifacts into applications is only supported indirectly. For this, a separate MLOps pipeline (provided by TensorOpera AI) deploys and serves resulting models.

Another direction of research focuses on applying FL to Large Language Models (LLMs) to boost the performance of these models [FKM+23]. Therefore, finetuning (training only a subset of model parameters) is applied in a decentralized setup using multiple data silos and compute resources. In this setup, FL enables the use of data that is potentially only accessible for respective parties (enterprises), and FL utilizes more

[5]`https://azure.microsoft.com`, accessed 2024-07-22
[6]`https://katulu.io/`, accessed 2024-07-22
[7]`https://fedml.ai/home`, accessed 2024-10-01
[8]`https://tensoropera.ai/`, accessed 2024-10-01

compute resources in parallel to tackle the complex training task. For example, FATE-LLM is a system that applies FL on LLMs supporting multiple approaches [FKM+23]. First, an approach for federating homogenous LLMs over multiple clients that finetune their local models is provided. For heterogeneous local models in different sizes (different number of parameters), a second approach adds a common mentee model for every client that is used for FL. The mentee models transfer the learned knowledge to the respective heterogeneous local models by applying knowledge distillation. Knowledge distillation refers to the approach of transferring knowledge from a larger teacher model to a smaller student model (mentee model). In a third approach, FATE-LLM provides knowledge distillation on the server using a larger teacher LLM and a smaller student model that is then used for FL between all involved clients.

5.5 Summary

In this chapter, we presented the *IFL Lifecycle* for managing FL artifacts considering publishing, deployment, execution (training and prediction), and the integration into client applications. With this, clients can be served with FLaaS in a scalable and still customizable way. We presented an extensible library for FL, the *IFL Core*, which includes the proposed clustering algorithm FedClust.

In the evaluation, we have shown how FedClust is executed as template along the *IFL Lifecycle*. For this, the *IFL Lifecycle* has been integrated into a CM application for clustering conditions of industrial pumps. In the deployment and execution of the small-scale (Scenario 1) and the large-scale scenario (Scenario 2), we demonstrated the applicability of FedClust and the *IFL Lifecycle*. We have shown that disclosing client data to the server does not yield better performance than the privacy-preserving approach that only shares local cluster centroids.

This chapter was published as a paper at the 7th IEEE International Conference on Fog and Edge Computing (ICFEC) in 2023:

Hiessl, T., Lakani, S. R., Ungersboeck, M., Kemnitz, J., Schall, D., and Schulte, S. (2023). Lifecycle management of federated learning artifacts in industrial applications. In 2023 IEEE 7th International Conference on Fog and Edge Computing (ICFEC) (pp. 7-15). IEEE.

CHAPTER 6

FL Deployments of Industrial Applications on Cloud, Fog, and Edge

6.1 Introduction

One goal of FL is to improve model performance for all involved clients given an underlying ML task such as classification [MMR+17]. IoT-based applications utilize FL to improve the performance of ML models for industrial use cases like quality inspection [BHKS+23] or CM of production machines [HSKS20, HRK+22]. To realize use cases in IFL setups, collaboration between industrial partners requires resource efficiency [ASCF23, YL21] for multiple production lines, and a way do deal with heterogeneous data between partners [HPMG20].

The problem of resource efficiency of course also applies to the *IFL Lifecycle* that we have introduced in the previous chapter. So far, we have primarily focused on supporting the development and execution of IFL artifacts and their integration into applications (e.g., CM). Now, we turn our attention to optimal deployment and client selection. This includes research on different deployment styles of clients on available platforms (e.g., cloud, fog, or edge) and dynamic optimization of selected clients with the goal to minimize the overall used resources (e.g., energy) by still maximizing model performance. Specifically, we aim to support optimal decision-making for efficient operations, particularly for *IFL Templates*.

In this context, still many challenges for a practical deployment in industry exist to provide scalable solutions.

First, determining the optimal deployment of clients across various compute platforms like cloud, fog, and edge is an important question to address resource efficiency [EARMAA+18, YL21]. This deployment decision is crucial for achieving improved FL performance, reduced latency, and lower energy consumption. While larger companies may have diverse resources across all platforms, smaller businesses often rely on simpler solutions, as e.g., on-demand cloud instances or a few on-premise (edge) devices only. Understanding the implications of real-world FL deployments on different compute platforms is essential for planning suitable compute infrastructure [OÁP+23]. This enables companies to optimize resource utilization and ensure efficient FL operations within IIoT environments. Hence, it is necessary to identify appropriate deployment strategies considering cloud, fog, and edge resources or a suitable mix with respective benefits and limitations.

Second, selecting suitable clients that are deployed on heterogeneous resources is a known problem in FL [NY19]. Overall, clients should be selected to optimize performance metrics i.e., maximize overall model performance (e.g., accuracy), or minimize completion time and energy consumption of the FL process. Client selection on the server becomes more complex as the granularity of clients changes due to a re-design of the FL setup (e.g., creating multiple edge clients instead of a single one in a fog data center). For instance, this can be applied to an industrial setup, where the overall factory is represented with a single client that can be split to multiple production line clients. The resulting FL model can help to improve the production process e.g., with an enhanced anomaly detection accuracy. This is possible since there may exist other production lines with similar characteristics (e.g., data distribution) that benefit from model sharing through FL (see cohort concept introduced in Chapter 4). However, instead of connecting a single FL client for the given factory with other factories, the fine-grained multi-line approach might help to tailor the model to the specific process. Obviously, this increases the number of FL participants, which adds complexity to the client selection process. This is especially the case for a multi-platform deployment. Therefore, we identify the need for optimizing client selection given a fine-grained setup of clients and a multi-platform deployment with cloud, fog, and edge resources.

Third, the problem of personalization arises when collaboratively trained FL models need to be adapted to perform effectively on local data [MMRS20]. Multiple clients are involved with different data distributions generated by the underlying industrial processes. This is the case in deployments where multiple production lines have their own slightly different data distribution. So, engaging in collaborations with other clients may enhance the model performance, but it can also result in challenges such as misclassifications when applied to local data. This can be the occurrence of false positives in an anomaly detection application for identifying product errors in a complex production process [BHKS+23]. Hence, industrial clients need to address the challenge of personalization for their models to achieve best model quality.

To solve this, this work makes the following contributions:

- An IFL deployment architecture for executing FL in a multi-location setup using cloud, fog, and edge platforms for client training.

- An ILP-based optimization approach (*IFL Opt*) for client selection, considering energy consumption, time, and model performance.
- The FL algorithm *XMeanCohorting* that supports personalization by building cohorts (= groups of similar clients) to boost performance in the FL aggregation.
- Evaluation on a real-world industrial dataset from the electronics manufacturing industry comparing different FL client deployments on cloud, fog, and edge nodes.

In the last chapter, *IFL Templates* have been introduced which are facilitated by the *IFL Core* library. The *IFL Lifecycle* manages templates and supports the integration into applications. Based on that, this chapter extends the *IFL Core* with additional optimization for client selection and personalization. In addition, the *IFL Core* is integrated into IFL-based applications running on multiple platforms. Another evaluation dataset, with process data from assets of two real-world factories [BHKS$^+$23], is used in this chapter as well.

The remainder of this chapter is structured as follows: Section 6.2 presents the system design. Section 6.3 evaluates the system by discussing resulting performance metrics in three scenarios. Section 6.4 discusses related work, and Section 6.5 concludes the chapter with a summary of the findings.

6.2 System Design

We propose a system design that addresses the challenges of deploying FL clients in different granularities (e.g., factory or production line clients) to multiple platforms. The main building blocks include an FL library (*IFL Core*) with extensions for optimizing client selection (*IFL Opt*), and cohort building (*XMeanCohorting*). The building blocks are integrated and used in the proposed FL system to optimize performance metrics.

6.2.1 Architecture

IFL Core

To provide FL services for multiple clients, the *IFL Core* Python library is used. As compared to the architecture from the previous chapter depicted in Figure 5.1, we now highlight the extensions that are relevant for personalization and optimization as part of the overall FL procedure in Figure 6.1. For the sake of simplicity, we avoid presenting all previously introduced CMDs and flows from the *IFL Core*.

In the first stage, the *IFL Client API* is invoked for registering clients for FL. The server manages the process of FL by aggregating local model updates and sharing the result with involved clients. Furthermore, local model validation is iteratively enforced before passing the resulting FL model to the *IFL Client API* and further to the caller. To address client selection with efficient use of resources and optimal performance, the

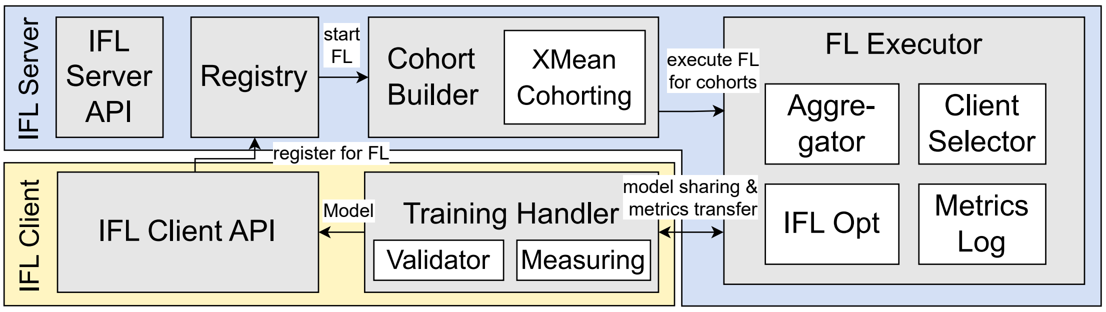


Figure 6.1: *IFL Core* with main components for client and server

IFL Opt module has been added and is invoked after each (communication) round. A round refers to one iteration of client training, model upload and server aggregation. *IFL Opt* supports client selection by using metrics recorded in the added *Metrics Log* that includes e.g., energy consumption, model performance, processing time and network delay. These are needed parameters for the ILP-based optimization model, which we present in Section 6.2.2.

To address model personalization, the *IFL Core* considers additional local training on the global model that results from the FL process. This final step adapts the model to the local data distribution [TYCY23]. To further boost model performance through advanced personalization, we added the *XMeanCohorting* algorithm which we describe in Section 6.2.3. It builds cohorts of clients with similar data distributions that result in multiple cohort models instead of a single global model. *XMeanCohorting* is invoked by the *Cohort Builder* that collects statistical metadata of the underlying data distribution.

IFL Deployment

The *IFL Core* can be used to connect clients from multiple locations (i.e., factories and underlying production lines) as the library is integrated into server and client applications and deployed to available resources. For this, Figure 6.2 depicts a multi-platform architecture, presenting deployment strategies for multiple locations like factories and production lines. The architecture considers four vertical layers, the *production process* with data-generating machines (assets), and the three independent compute platform layers *edge*, *fog*, and *cloud*. Process data (e.g., product features for quality inspection) is retrieved from process controllers by the collector edge device that pre-processes and stores the data in a factory-wide database. The collector device is deployed to the edge layer to filter for relevant training data therefore reducing the overall data load that is streamed to the database.

To execute FL, potentially multiple resource nodes are considered per location. For this, compute instances (e.g., VMs) can be used to host the *IFL Client*, a service using the IFL Core for FL and serving the resulting FL model to improve the production process (e.g., by detecting product quality issues). The multi-platform strategy is beneficial to

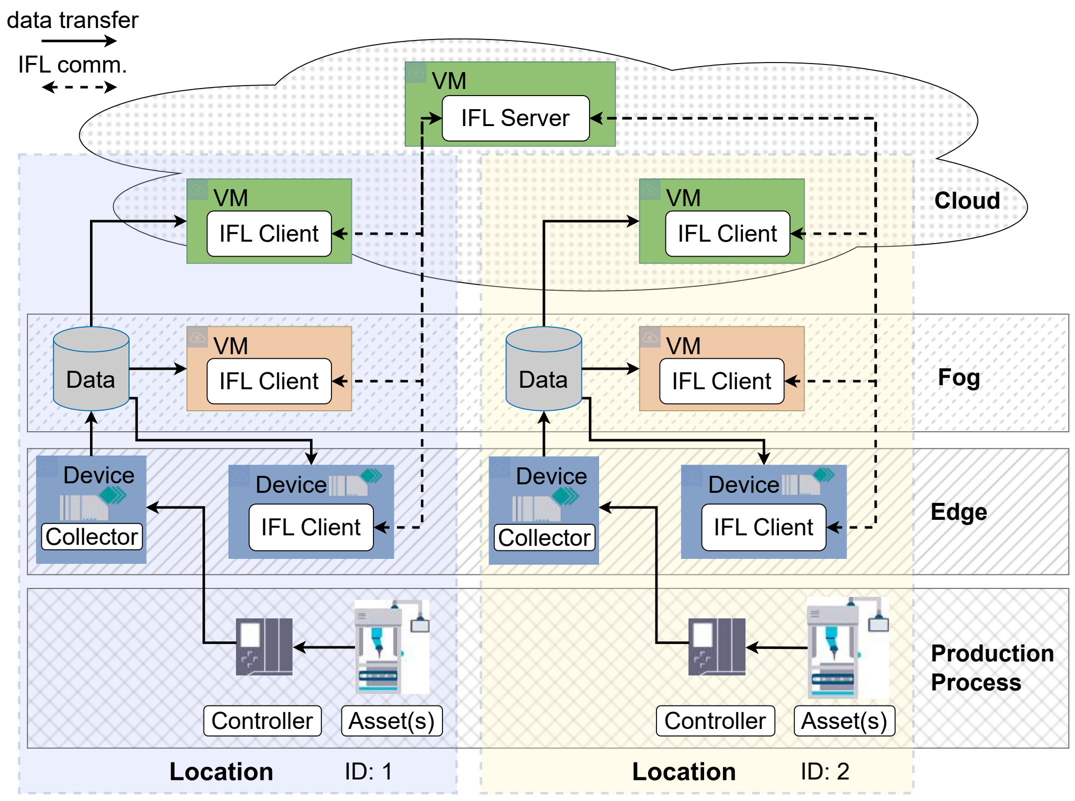


Figure 6.2: IFL deployment architecture for multiple locations

gain flexibility for executing peak loads like FL training. It allows for utilizing trade-offs between platforms with respect to energy consumption, network delay, and processing time.

For each location, the used *IFL Clients* identify themselves with a joint location ID at the server. The location ID refers to the data that is used by the client e.g., overall factory data, or production line data. This allows different granularities with respect to real-world FL collaborators (factories and production lines). A resource-efficient approach can be achieved by adopting a coarse-grained setup, where a single client with a single location ID is assigned per factory.

In order to optimize model performance, it can be beneficial to implement a fine-grained setup, where multiple clients per factory are instantiated, each with a unique location ID. Hence, more clients with similar data distributions can be identified. This is relevant in cases where a factory consists of multiple production lines that each have a similar process but generate slightly different data distributions (e.g., products have different components). Hence, only those clients (i.e., production lines) with similar data distributions are enabled to collaborate with FL. This reduces the risk for a poor FL model quality.

6.2.2 Optimization Model

The ILP-based optimization model *IFl Opt* uses the metrics log with the parameters defined in Table 6.1 to select clients for the upcoming round. Beside performance metrics, Table 6.1 contains the decision variable x_{ij} that determines which clients should be selected to minimize the objective function in Eq. 6.5. This function is a weighted sum of min-max-normalized values for the overall energy consumption e, overall response time r, and sum of model performances (e.g., accuracies) of all clients a from the most-recent round. The variables for the performance values for a given round j are defined in Eqs. 6.1 to 6.4. Since not all clients participate in each round, we introduce the *prev* function that returns the metrics from the most recent round where the clients delivered the metrics. The min and max values of e, r, a are computed in the first round, where *IFL Opt* is not invoked and all clients are selected.

Table 6.1: Notation of parameters and variables

Symbol	Description
k	Number of rounds in FL
D	Set of devices providing resources for clients
n	Number of devices in D
m	Number of locations collaborating in FL
L_p	Set of clients of the p-th location with $p \in 1..m$
e_{ij}	Energy consumption of client i in round $j \in 1..n$
a_{ij}	Model performance metric of client i in round $j \in 1..n$
d_{ij}	Network delay for transferring the trained model to the server for client i in round $j \in 1..n$
t_{ij}	Processing and training time of client i in round $j \in 1..n$
ϵ	Minimum increase in performance value between two rounds
x_{ij}	Decision variable for selection of client i in round $j \in 1..n$

$$r_j = \max_i r_{ij} \tag{6.1}$$

with

$$r_{ij} = t_{i,prev(i,j)}x_{ij} + d_{i,prev(i,j)}x_{ij} \tag{6.2}$$

$$e_j = \sum_{i=0}^{n} e_{i,prev(i,j)}x_{ij} \tag{6.3}$$

$$a_j = \sum_{i=0}^{n} a_{i,prev(i,j)}x_{ij} \tag{6.4}$$

Objective function:

$$L_j = w_e \frac{e_j - e_{min}}{e_{max} - e_{min}} + w_r \frac{r_j - r_{min}}{r_{max} - r_{min}} - w_a \frac{a_j - a_{min}}{a_{max} - a_{min}} \tag{6.5}$$

Constraints:

$$\sum_{i \in L_p} x_{ij} \geq 1 \qquad \text{for all } p \in 1..m \tag{6.6}$$

$$\delta_{ij} \geq \epsilon \, x_{ij} \qquad \text{for all } i \in 1..n \tag{6.7}$$

with

$$\delta_{ij} = (a_{i,prev(i,j)} - a_{i,prev(i,prev(i,j))}) x_{ij} \tag{6.8}$$

Two constraints limit the objective function. First, Eq. 6.6 requires that at least one client from a given location i is involved. This ensures that data from all the locations is considered in FL to increase the potential for high model performance values. If more than one client is involved, additional resources on the location are used.

The constraint in Eq. 6.7 ensures that only clients who have shown a minimum increase in model performance in previous rounds are selected. This constraint aims to increase the probability of further improving the overall model with their contribution. However, it is considered only if $a_{i,prev(i,j)}$ and $a_{i,prev(i,prev(i,j))}$ are defined. If the problem cannot be solved, another optimization run without this constraint is started. This can be the case in scenarios where constraints 6.6 and 6.7 do not match, since e.g., there is only one client x_{ij} in the location L_p, but δ_{ij} is lower than the required ϵ.

6.2.3 Cohort Building: XMeanCohorting

Model performance can be poor if FL is applied to clients with data from multiple underlying data distributions and sources, since no other similar client might match for collaboration. Therefore, it can be beneficial to split clients with respect to the original data source if possible (e.g., splitting data from factory clients into subsets for finer-grained production line clients). Based on this new client design, the model performance can be further improved through personalization. For this, we subset the clients into cohorts with similar data distribution as proposed in Algorithm 5. The inputs are the $mean(X_i^t)$ values of all the features X of the local datasets as computed by all clients. We create a new matrix F by concatenating all delivered means row-wise in line 2. To reduce the q dimensions to q^r dimensions (default=2), we apply Principal Component Analysis (PCA) [Jol02] to receive F^r in line 3. PCA provides lower dimensionality by still capturing the main variance of the data distribution. This method is needed in

Algorithm 5 `XMeanCohorting`

```
Input: $mean(X_i)$ for all $i \in D$, cluster dimensions $q^r$
1: Initialize $F$ as $n \times q$ matrix for $n$ clients and $q$ features
2: $F \leftarrow$ concatenate $mean(X_i)$ for all $i \in D$
3: $F^r \leftarrow PCA(F, q^r)$
4: $k \leftarrow elbow(F^r)$
5: $cohorts \leftarrow kMeans(F^r, k)$
6: return $cohorts$
```

line 4 and 5 to faciliate clustering solutions. For this, we apply k-means [KMN+02] and the elbow method [Tho53]. The k-means algorithm is a technique for partitioning data points into a predetermined number of clusters, aiming to minimize the variance within each cluster. The elbow method is employed to identify the optimal number of clusters by iteratively performing k-means with progressively larger values of k until further improvements become insignificant. Finally, the cluster result of k-means can be used as client-to-cohort assignment for subsequent client selection in the *IFL Core*. This algorithm extracts the key similarities between clients to facilitate collaboration among multiple locations that may have high-dimensional data.

6.3 Evaluation

In this section, we demonstrate how different deployment styles of an industrial FL setup perform on real-world data. We evaluate client deployments in multiple scenarios considering edge, fog, and cloud as well as multi-platform deployments with optimized client selection using *IFL Opt*. We further investigate the effect of using the *XMeanCohorting* algorithm in a production line FL setup.

6.3.1 Experimental Setup

The virtual machines in the cloud are Linux machines with a 3.0Ghz Intel Xeon Scalable Processor with 4 CPU cores and 16 GiB RAM hosted in Frankfurt (GER) using Amazon Web Services (AWS)[1]. The fog instances run as virtual Linux machines in a local data center in Vienna (AUT) based on Intel Xeon SP Gold 6230 20C/ 40T - 2,1GHz/ 3,9GHz. Each instance is provided with 8 CPU cores and 16 GiB RAM. For edge deployments, we use Siemens industrial PCs[2] including five microbox PCs (SIMATIC IPC427E) with Intel Core i5-6442EQ CPU @ 1.90GHz (three devices) and Intel Xeon CPU E3-1505L v5 @ 2.00GHz (two devices) all with 4 cores and 16 GiB RAM. Additionally, we use one rack PC (SIMATIC IPC847E) with Intel Xeon E-2278GE CPU @ 3.30GHz (8 cores and 128 GiB RAM) as edge device. These are located in two labs in Vienna (AUT).

[1]`https://aws.amazon.com/`, accessed 2024-07-22

[2]`https://www.siemens.com/global/en/products/automation/pc-based.html`, accessed 2024-07-22

6.3.2 Industrial Use Case and Data

The *IFL System* and its different deployment styles are evaluated on a real-world dataset from the electronics industry. For this, we address a multi-step manufacturing process for Printed Circuit Boards (PCBs) given a quality inspection task in two factories. The process starts with printing solder paste onto the PCB through a stencil, ensuring precise application, followed by component placement. Reflow soldering is conducted with specific temperature profiles, and Automated Optical Inspection (AOI) verifies component alignment and soldering integrity. However, this step is prone to error slipage, which motivates the use of an AI model to classify PCBs with respect to the three classes *OK*, *error*, and *pseudo-error* using process data from all steps. The process data include 16 numerical features, e.g., *height, area, offset, pos-x and pos-y, size-y, size-y* of the printed solder paste, and *image quality* and the *original label* of the AOI. The datasets of the two factories have a size of 975k and 610k rows respectively. The derived supervised learning problem is faciliated by class labels introduced by human inspectors in case of *error* classifications of the AOI. Possibly, the error is a false positive which is then re-labelled as *pseudo-error*. To reduce human inspection, the new classifier can be attached to the quality inspection process.

6.3.3 Scenarios and Experimental Design

Overall, we consider three scenarios that address the aforementioned quality inspection problem with different FL deployments and configurations. Each scenario has a set of seven deployments that are instantiated, executed and compared. The basic deployments are *edge, fog* and *cloud*, where all clients are deployed to the respective environment. Additionally, multi-platform deployments consider all three compute platforms with a subset of selected clients applying *IFL Opt* in four versions i.e., *eq opt* (equal weights: $w_a = w_r = w_e = \frac{1}{3}$), *time opt* ($w_r = 1$ and $w_a = w_e = 0$), *perf opt* ($w_a = 1$ and $w_r = w_e = 0$), *energy opt* ($w_e = 1$ and $w_r = w_e = 0$)

Scenario 1

In particular, Scenario 1 (*factory FL*) demonstrates the collaboration of two factories using two clients (one per factory) for the basic deployments and six clients (three per factory: cloud, fog, and edge) for the multi-platform deployments.

Scenario 2

Scenario 2 (*production line FL*) considers three subsets for each of the two factory datasets. Hence, six production lines (= clients) are instantiated overall, whereas each of them hold data from different types of PCBs. While the basic deployments consider six clients, the multi-platform ones use 18 clients (three per production line).

Scenario 3

Scenario 3 (*production line FL with cohorts*) applies FL similar to Scenario 2 but builds cohorts using *XMeanCohorting* to facilitate suitable collaboration between similar production lines.

Both scenarios 2 and 3 demonstrate FL collaboration on a multi-line level with more clients as compared to *factory FL* and investigate the impact of data heterogeneity by comparing cohort-based FL with FL using *FedAvg*.

Metrics

Deployments are benchmarked using six to nine metrics measured by *IFL Core*, including response time, energy consumption, and model performance. Response time metrics comprise the average network delay defined as $avg\ net\ delay = \frac{1}{k}\sum_{j=1}^{k}\left(\frac{1}{n}\sum_{i=1}^{n} d_{ij}\right)$ and the total network delay defined as $total\ net\ delay = \sum_{j=1}^{k} \max_{i=1}^{n} d_{ij}$. The average processing time ($avg\ proc\ time = \frac{1}{k}\sum_{j=1}^{k}\left(\frac{1}{n}\sum_{i=1}^{n} t_{ij}\right)$) is calculated as the mean processing time (i.e., data processing, training, metrics calculation) across all devices and rounds. The total processing time is determined as $total\ proc\ time = \sum_{j=1}^{k} \max_{i=1}^{n} t_{ij}$. The total energy consumption ($total\ energy = \sum_{j=1}^{k} (\sum_{i=1}^{n} e_{ij})$) is computed by summing up the energy consumption of all involved clients in each round. To compare the deployments with respect to model performance, the validation average balanced accuracy is defined as $val\ avg\ bal\ acc = \frac{1}{n}\sum_{i=1}^{n} a_i^{bal}$, where a_i^{bal} is the balanced accuracy used as model performance metric computed on the validation dataset of client i after all rounds using 40% of client data. Analogously, the validation average f1 score is defined as $val\ avg\ f1 = \frac{1}{n}\sum_{i=1}^{n} a_i^{f1}$). The maximum average f1 score ($max\ avg\ f1 = \max_{j=1}^{k}(\frac{1}{n}\sum_{i=1}^{n} a_{ij}^{f1})$) signifies the highest mean f1 score over all rounds. Finally, the maximum average balanced accuracy ($max\ avg\ bal\ acc = \max_{j=1}^{k}(\frac{1}{n}\sum_{i=1}^{n} a_{ij}^{bal})$) denotes the highest mean balanced accuracy among all rounds. Both $max\ avg\ f1$ and $max\ avg\ bal\ acc$ can be used to identify the maximum model performance considering model personalization.

6.3.4 Scenario Execution

We now demonstrate how the execution of the scenarios is performed using three compute platforms and a suitable FL solution to train the classifier for the PCB problem. For this, we create an *IFL Template* (see Chapter 5), which defines the code for ML training and data processing on the client side and the model aggregation (i.e., FL with or without cohorting) and the parameters for *IFL Opt* on the server side using Python, PyTorch[3] and the *IFL Core*. The used model is a feedforward neural network with four layers in total (incl. classification head) and max 256 neurons per layer. The network is trained in five epochs per round with a batch size of 64.

[3]`https://pytorch.org/`, accessed 2024-07-22

According to the scenario definition, the client instances are provisioned using the Infrastructure as Code (IaC) tool Terraform[4] for cloud and fog. Following that, the runtime is deployed to each client of the respective platform, which includes edge devices. Subsequently, the client's runtime installs and executes the template. After all clients are registered on the server, the FL procedure is repeated for ten rounds. The metrics are continously measured by the *IFL Core* and reported to the server. For energy consumption metrics, the *IFL Core* integrates Code Carbon[5]. This Python package is designed to estimate and monitor the energy consumption of code execution. Based on all delivered client metrics, *IFL Opt* is applied after every round and clients are selected from different resources. To ensure robustness, we execute all scenarios five times and collect mean and Standard Deviation (SD) values. These are presented in the dynamite plots (bar for mean and whiskers for SD) in Figs. 6.3 to 6.6.

6.3.5 Results of the PCB Quality Classification

The resulting metrics are now presented and discussed for all three scenarios. We compare the deployments in the respective scenario and reason about the impact of selected platforms, granularity of deployed clients (factory or line deployment), FL aggregation (with or without cohorting), and optimized client selection.

Factory FL Scenario: Two factories with 2-6 Clients

The network delay metrics in Figs. 6.3a and 6.3b show the highest values ($avg\ net\ delay \approx 0.3s$ and $total\ net\ delay \approx 4s$) for the edge deployment on average, closely followed by the energy opt deployment. This is due to the corresponding network distance between the edge clients and the server in the cloud. The fog deployment has improved network delay but is still inferior to the cloud, which benefits from a close connection to the server in the same data center.

The energy opt approach relies on edge devices, while the time opt approach considers cloud instances in this scenario, as the values are similar in their respective deployments. This is elucidated in the processing time metrics in Figs. 6.3c and 6.3d. However, here we observe that *fog* values are similar to *edge* values (*avg proc time* is between 4.3s and 4.7s and *total proc time* is between 45s and 49s), while cloud still outperforms both (1s and 12s respectively). One possible reason could be the selection of different hardware. However, the effect is not solely caused by hardware aspects, but also by the energy measurement process performed by clients during training. Code Carbon retrieves benchmark data from the Internet to estimate and monitor the running process, favoring approaches with minimal network delay, such as those using cloud clients. This limitation hinders direct comparisons of training times but emphasizes the significant impact of network distances within an FL scenario.

[4]`https://www.terraform.io/`, accessed 2024-07-22
[5]`https://codecarbon.io/`, accessed 2024-07-22

The total energy used for FL is depicted in Figure 6.3e. The best performance is achieved by *edge* and *energy opt* with less than 0.0002 kWh. On the contrary, *perf opt* might extensively utilize numerous resources (up to six clients) across all platforms, leading to more than three times the energy consumption. A medium level of energy is used by the instances from *fog* and *cloud*. The edge devices in this scenario are two microbox PCs with comparably limited resources in contrast to *fog* and *cloud*. So, for a target factory FL setup, energy can be saved by selecting compute resources without overprovisioning as long as the complexity of the underlying ML task is manageable.

The model performance values depicted in Figs. 6.3f to 6.3i show comparable results for the basic deployments, since no deployment-specific aspect could influence the model quality. However, for the multi-platform deployments *val avg bal acc* and *val avg* $f1$ perform inferior. This could be because the multi-platform deployment involves more clients (six) compared to the basic deployment (two) in the initial rounds, which is necessary to collect enough metrics for *IFL Opt*. The increased number of clients in the multi-platform deployment can lead to diverse local models, which may result in poorer metrics after aggregating and validating the global model. However, both *max avg* $f1$ and *max avg bal acc* show improved performance values, even in multi-platform deployments. This is because of the personalization before validation. Clients with slightly different data distribution benefit from this additional local training, which is relevant in the case of the two factories producing different types of PCBs.

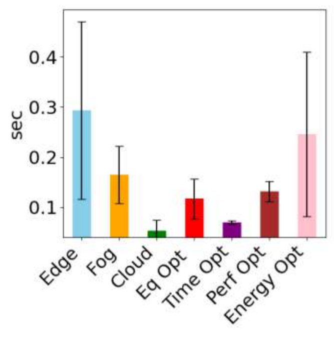
Production Line FL Scenario: 6-18 Clients

Figure 6.4 depicts six metrics for the production line FL scenario. The network delay metrics inherently highlight the outcome of the factory FL scenario, wherein the delay decreases as the deployments get closer to the server in the cloud. We observe higher standard deviations for all deployments. That is due to the increased number of clients in this scenario as network traffic and the probability for higher delays can increase.

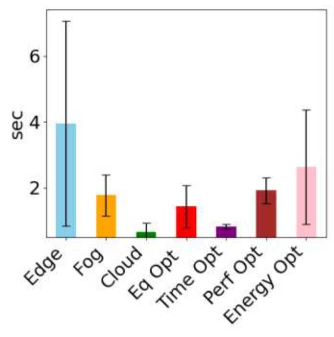
The *time opt* approach is close to the *cloud* deployment. This holds for the processing time metrics in Figs. 6.4c and 6.4d as well.

In terms of energy usage, *cloud* and *energy opt* have the lowest values. This is different from the factory FL scenario with two clients, where *edge* performed the best. In the current scenario, six edge devices are used, including a high-power rack PC and other CPU types. This suggests that utilizing on-demand resources in the fog or cloud, or other edge devices with lower energy consumption, can be beneficial. However, this is only true if high-power edge devices are not needed for other purposes and if fog or cloud deployments are supported for local production lines and factories.

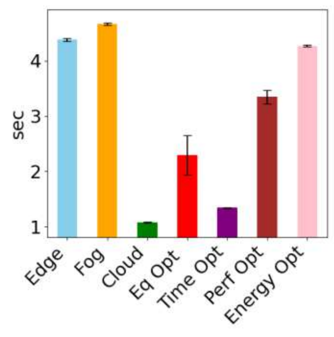
The *perf opt* approach again uses too many devices. Although the performance in terms of *val avg bal acc* is similar to other approaches, there is no notable advantage for resource-intensive operations. This is because the classifier cannot further be improved using *FedAvg*, when collaborating clients have heterogeneous data.



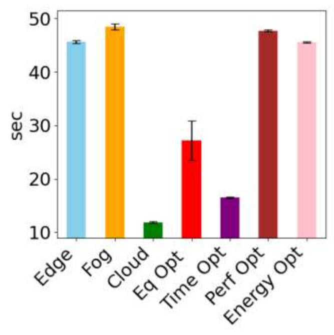
(a) avg net delay



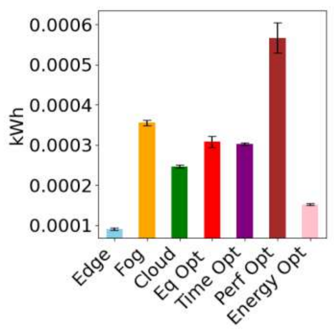
(b) total net delay



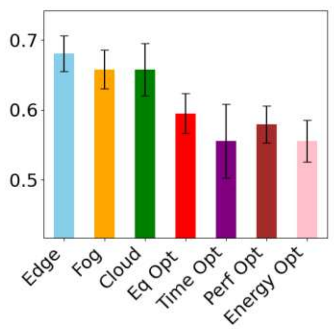
(c) avg proc time



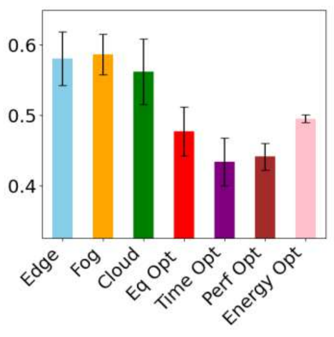
(d) total proc time



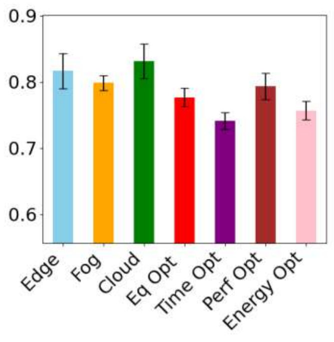
(e) total energy



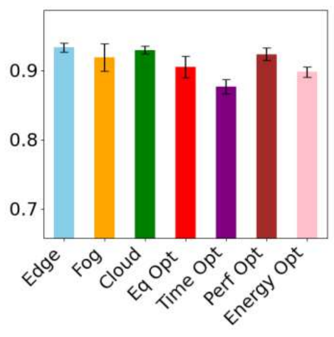
(f) val avg bal acc



(g) val avg f1



(h) max avg f1

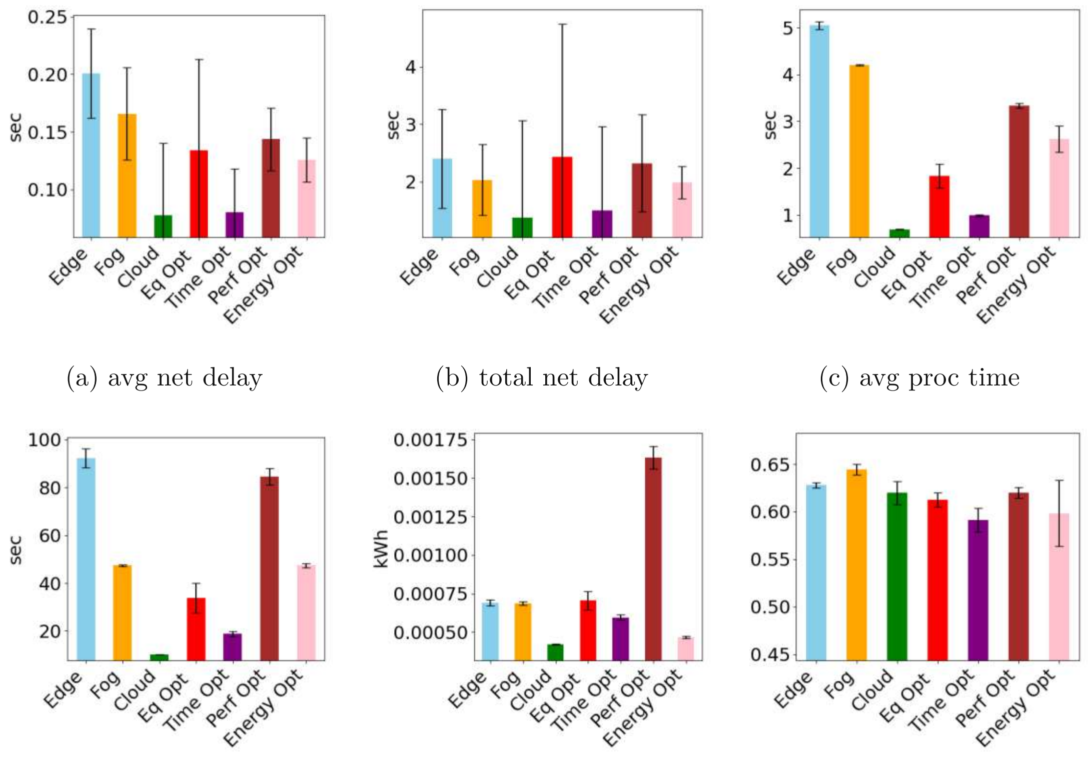


(i) max avg bal acc

Figure 6.3: Scenario 1 – Factory FL with 2-6 clients

Production Line FL Scenario with Cohorts: 6-18 Clients

Similar to the previous scenario, clients consider data for the respective production lines, but the *IFL Server* applies *XMeanCohorting* to facilitate collaboration only between similar production lines with results depicted in Figure 6.5. For network delay and processing time, the metrics are comparable to Scenario 2, since the only change in



(a) avg net delay (b) total net delay (c) avg proc time

(d) total proc time (e) total energy (f) val avg bal acc

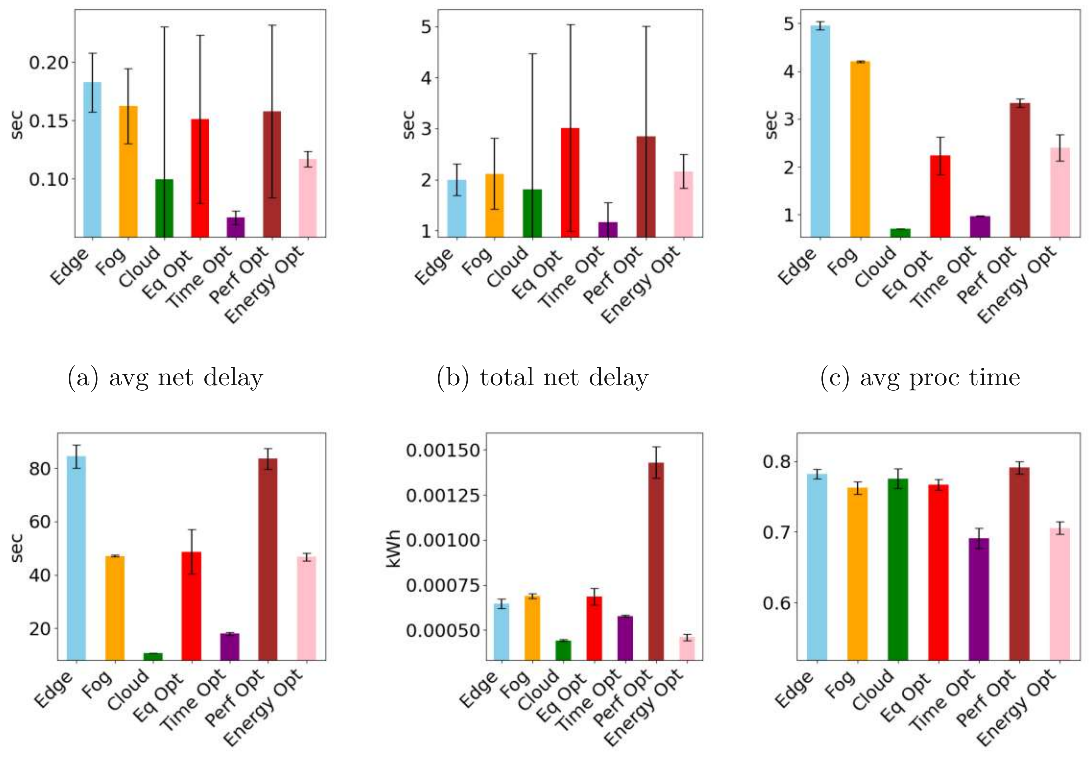
Figure 6.4: Scenario 2 – Production line FL: 6-18 clients

processing is the aggregation on the server.

However, we can observe that the energy-intensive *perf opt* slightly performs best with *val avg bal acc* ≈ 0.8. Compared to Scenario 2, only similar clients collaborate on respective cohort models instead of contributing to a single global model. *XMeanCohorting* enables that clients with similar PCBs join the same cohort, i.e., we observe three cohorts with three, two, and one client for the basic deployments and cohorts of nine, six and three clients for the multi-platform deployments respectively.

One potential reason for improved model performance using a multi-platform deployment is the parallel training, since more local epochs are trained on a given dataset in total. Although this can lead to relatively high accuracy values in a shorter duration, the energy consumption is increased. If time constraints are not a concern, individual clients utilizing the same computing platform may achieve similar model performance, though slightly delayed.

As seen in all scenarios, the multi-platform approaches using *IFL Opt* do not always provide best solutions as compared with the three basic deployments. This is due to initial overhead where all resources are used to identify the most suitable client selection



(a) avg net delay (b) total net delay (c) avg proc time

(d) total proc time (e) total energy (f) val avg bal acc

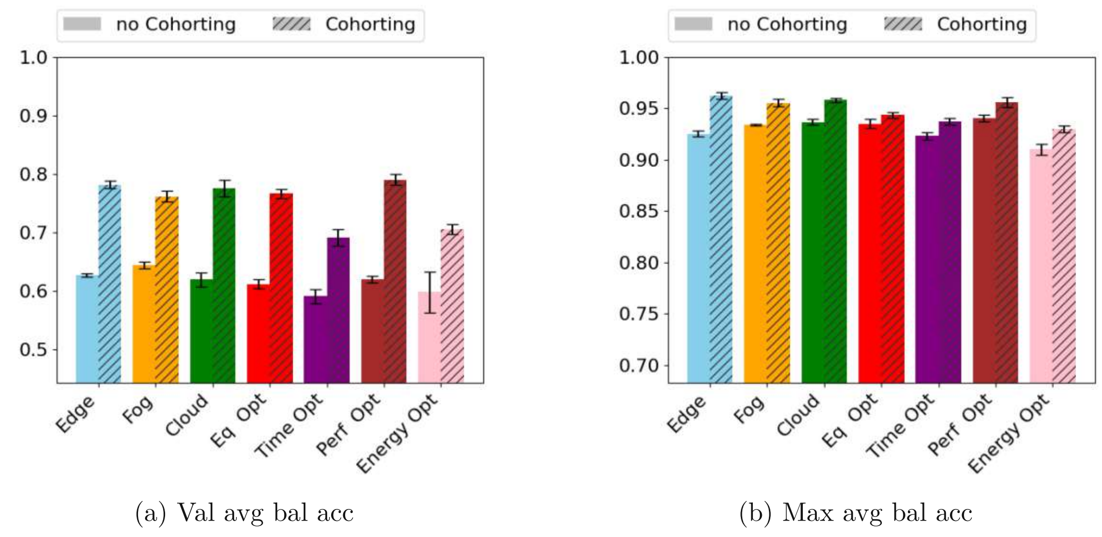
Figure 6.5: Scenario 3 – Production line FL with cohorts: 6-18 clients

for the optimization target based on the metrics needed as parameters. However, it can be seen that approaches with the respective preferences (weights) converge close to the best deployment. Furthermore, flexibility is ensured by changing selected clients and platforms after every round. Hence, robustness against clients with high network delay or processing time, named stragglers [PHCM21], is facilitated.

The impact of cohorts is underlined in Figure 6.6 that depicts grouped bar charts comparing production line FL with and without cohorts, i.e., *XMeanCohorting* is compared with *FedAvg*. For *val avg bal acc* in Figure 6.6a, *XMeanCohorting* outperforms *FedAvg* by 10-15% on average. Considering additional personalization, the performance increases by 15% up to 95% reflected by *max avg bal acc* in Figure 6.6b. Hence, combining cohort-based FL for the three production line clients with additional personalization facilitates optimal training.

6.4 Related Work

The FL research community experiences rapid development in implementation practices and approaches like client selection [FZG$^+$23]. We discuss the most important related



(a) Val avg bal acc (b) Max avg bal acc

Figure 6.6: Production line FL with and without cohorts

approaches.

Rajagopal et al. [RMB23] present deployments of the FL server on edge, fog, and cloud resources. The authors measure performance metrics such as energy consumption, network usage, cost, execution time as well as latency and conclude that edge deployments outperform fog and cloud. Different platforms for server deployments are considered by Nguyen et al. [NCOD22] as well. However, they consider a hierarchical deployment with clients on the edge and servers in the fog and cloud. They present an optimized selection of servers in the fog for a global aggregation on the cloud server to reduce the overall completion time and global loss. However, these approaches do not address client deployments on multiple platforms, and respective optimization of the client selection. In our work, we consider the impact of FL client deployments and selection, which is important as FL systems are implemented e.g., in factories, as multiple data center and edge device options are available for optimal resource usage.

When it comes to setup an FL deployment between multiple industry partners, Oldenhof et al. [OÁP+23] propose multiple cloud accounts for collaboration in the pharmaceutical industry. The authors emphasize the high demand in computation power given a drug discovery problem and FL runs that have been executed annually for three years. In our work, the FL deployment considers additional fog and edge resources that can provide more resource-efficient FL services (e.g., w.r.t energy consumption) given already existing devices (e.g., for production line data collection) with appropriate capacity for solving the underlying ML task. This is suitable in FL processes that are repeated more frequently on less demanding ML tasks with a subsequent model deployment for providing inference services on the same (training) device close to the data source. Hence, training and inference services can be integrated into the same application to reduce complexity for the user, as presented in Chapter 5.

Chahoud et al. [COM23] introduce dynamical placement of FL clients through container deployment addressing changing workloads and increasing the availability of clients in areas that lack training resources. While our approach does not dynamically deploy new FL clients, the IFL deployment architecture allows for registering many devices from the same location, as identified by the location id. Based on set (weight) parameters of *IFL Opt*, the optimization controls the number of used resources for a given location. The resources are selected according to the collected performance metrics.

Abdel et al. [ASCF23] propose to select clients so that overall resource wastage is minimized, i.e., training time of clients that do not contribute to the overall model performance is limited and stale updates of previous rounds are integrated also if they were sent late. However, client selection is considered as isolated task. On the contrary, our work integrates the *IFL Opt* approach into a system with preceding cohort-building of clients from multiple locations with a suitable data and client splitting. This faciliates model performance, energy consumption and completion time.

In 2024, after this chapter was originally published, de Souza et al. [dSMdC+24] also provided an approach combining resource optimization and model personalization. Specifically, they aim for reducing the communication and computation overhead by only sharing some layers of parameters and not the overall model. After sharing the respective layers, the local training is executed to personalize the model to optimize on the evaluation dataset of the client. To further reduce the computation overhead, the authors introduced a decay function in the client selection procedure. This function gradually limits the number of selected clients as the FL communication rounds progress and the model begins to converge.

Chahoud et al. [CSM+24] addressed the topic of on-demand deployment of containers (including FL client software) to mobile devices. Hence, edge devices are not even pre-provisioned with FL clients and are selected for contributing models, but they rather get containers deployed as they move to new areas. This is relevant if there is no or only limited data in certain areas so that contributions from respective edge devices are needed.

6.5 Summary

In this chapter, we presented an FL system that supports FL client deployments on multiple locations using cloud, fog, and edge resources. We integrated the presented cohort-building algorithm *XMeanCohorting* and the ILP-based client selection *IFL Opt* to improve model performance, energy consumption, and FL completion time.

In the evaluation, we have presented FL runs with different client deployment strategies using cloud, fog, and edge resources and a real-world industry use case addressing a quality inspection problem in a PCB manufacturing process. We executed three scenarios evaluating FL deployments on factory level with 2-6 clients and on production line level with 6-18 clients. In each scenario, we compared basic deployments with multi-platform deployments using the *IFL Opt* approach. We have shown that model performance is increased by 10-15% as *XMeanCohorting* is applied. Furthermore, *IFL Opt* facilitates convergence of the multi-platform approaches to the relatively best basic deployment while still ensuring flexibility for updating the client selection in each round.

This chapter was published as a paper at the 8th IEEE International Conference on Fog and Edge Computing (ICFEC) in 2024:

Blumauer-Hiessl, T., Schulte, S., Rezapour Lakani, S., Keusch, A., Pinter, E., Kaufmann, T., and Schall, D. (2024). Federated learning deployments of industrial applications on cloud, fog, and edge resources. In 2024 IEEE 8th International Conference on Fog and Edge Computing (ICFEC) (pp 19-26).

CHAPTER 7

FL Solution Blueprints for Use Cases Surveyed in Austrian Industries

7.1 Introduction

With the emergence of Industry 4.0, new opportunities for further optimization in production have been unfolded using connected, decentralized and decision-making systems [JAM+23]. In this regard, important enabling technologies are the IIoT and AI, which support the provision of insights based on data streamed from various sources in production.

FL arose as a collaborative approach for training AI models between multiple partners. Companies, for instance, Original Equipment Manufacturers (OEMs), can offer FLaaS to their industrial clients for the purpose of enhancing the performance of their ML models. These models are primarily utilized to analyze IIoT data generated from production machinery. The goal is to improve these models beyond what would be possible with only local ML training. Since data is often limited at one location (e.g., site, company), IFL can be used between networked industrial systems for autonomous or adaptive decision-making in production using e.g., robots, edge devices, and Programmable Logic Controllers (PLCs) [SNB+21]. However, in practice, AI and FL systems still face challenges to be adopted, i.e., to be considered for solving data-intensive problems.

In the previous chapters, we have addressed IFL system aspects and how these aspects can be optimized to provide high-quality models in a resource-efficient way. To better understand the status of FL in the industry and its practical adoption for real-world use cases, we conducted an interview-based study that is presented in this chapter. Given our observation that FL is rarely used in practice outside of research setups and Proof of Concepts (PoCs), our goal is to understand the requirements and derive blueprints for future adoption of collaborative AI and FL-based approaches.

Each blueprint is considered an FL-based solution design for identified business types to address their needs and AI use cases in a decentralized environment (e.g., a service company and its customers). For this, not only the learning part is relevant, but also the inference and the way data (model input and results) are processed. The concepts of *IFL Templates* and previously defined architectures are not explicitly mentioned in the blueprints. Nevertheless, the FL solutions (i.e., training) can be realized with templates and the *IFL Core*, as well as with a statically deployed *IFL System* providing FLaaS for the involved parties. This is facilitated with well-defined interfaces for ingesting data, executing FL, and retrieving the resulting FL model via API.

To adopt AI and ML in industrial production, certain prerequisites need to be addressed as e.g., strategic alignment, available resources and knowledge, innovation culture, and data accessibility [JWW21]. If these prerequisites are fulfilled and if there exists a certain potential for increasing efficiency with suitable models, the AI adoption process still needs to be implemented in practice. Therefore, respective use cases need to be transformed into actionable plans including concrete architectures for implementation. For this, the interview study, conducted with thirteen employees from eleven Austrian companies from different domains, should build the basis for the actionable plans. In detail, we want to report up-to-date industry use cases, related pain points, and attitudes towards the implementation of AI and FL. Furthermore, we identify business types including their characteristics and how companies can be supported in the adoption of AI and FL.

While the technical feasibility of FL has been demonstrated in the previous chapters, in industrial use cases [BHKS+23] and research such as drug discovery [OÁP+23], the actual adoption in the manufacturing industry is lagging behind. To address this issue, the respective FL-based solutions need to be tailored to identified business types and involved parties. In this context, we analyze the business types and investigate how the main collaborations of involved parties can be facilitated with FL. For this, we assume that these parties aim to use and improve ML models facing limited data or privacy restrictions. Additionally, there is a need to understand how the parties interact and which tasks need to be done as the FL solution is implemented.

To address the aforementioned needs, this chapter makes the following contributions:

- A collection of use cases, challenges, and requirements for industrial AI applications based on the interview results of eleven companies (thirteen persons) in multiple Austrian industries.
- Identification of three *industry personas* representing business types and their industrial AI application fields that can be facilitated with FL.
- Three FL blueprints addressing system architectures and implementation steps for identified personas.

Considering these contributions, we focus on the actual transfer from FL systems to different types of industrial companies with their respective use cases. We want to provide insights on the current state in industry, real-world use cases, and suitable IFL blueprints as solutions. Thus, we provide an outlook how concepts of this thesis can be transferred to industrial companies.

The remainder of this chapter is structured as follows: Section 7.2 discusses the related work. Section 7.3 presents the methodology used for the interviews and the subsequent identification of personas, which are summarized in Section 7.4. For each persona, we derive an FL blueprint as presented in Section 7.5 and discussed in Section 7.6. Section 7.7 concludes the chapter with the main findings and future work.

7.2 Related Work

FL research experiences accelerated growth in technical approaches [NDP+21]. In this work, we discuss the related approaches focusing on understanding industry requirements through interviews and the application in practice.

Jöhnk et al. [JWW21] have interviewed 25 AI experts in organizations and derived AI readiness factors in five categories (i.e., strategic alignment, resources, knowledge, data, and culture). Collaborative work between the departments is highlighted as a significant factor in successfully adopting AI. In contrast, our work centers on identifying industry personas, their prerequisites, and AI application fields to derive blueprints for intra- and inter-organizational collaboration using FL.

To decide upon a suitable FL architecture for own use cases, Lo et al. [LLPZ23] present decision models for architectural patterns for e.g., training, aggregation, model deployment, and client registration. For this, they analyze functional and non-functional requirements based on a systematic literature review. In this chapter, we do not only provide architectural solutions, but rather holistic blueprints for adoption in organizations. This is relevant since the implementation also needs a clear picture on how to collaborate between involved parties apart from technical aspects.

Deng et al. [DLLW23] present a platform for IFL, where individual parties submit tasks that can be evaluated for suitable collaboration. This requires matching data schemes and the same underlying ML problem (e.g., tool wear prediction in a machining process). If this is given, the partners can be selected by the submitting party and FL is started. Hence, this platform has to be very specific for selected use cases and needs to have a significant number of potential collaborators to enable FL runs. Similarly, a platform-based FLaaS approach is presented by Ungersboeck et al. [UHSM23], where the platform enables control and transparency on the underlying FL process managing multiple clients, their tasks and resulting models. Both approaches can be considered as a generic and technical FL solution applicable to many industries and use cases. However, there still exists the need for addressing the adoption of FL solutions with specific roles and responsibilities to ensure successful deployment and operations.

Interview-based studies for surveying requirements of FL systems have been conducted in Germany by Verlande et al. [VLR23, VRL23]. The goal is to understand how FL systems can be designed for improving IT security in Human Resource Management (HRM) for e.g., detecting malware in recruiting processes. Expert interviews result in requirement catalogues addressing user experience and service design. The main system qualities that are expected from the interviewees are easy installation and maintenance, low effort of operations, and a high reliability of results. Furthermore, system design elements have been proposed to integrate FL systems into a recruiting process. In our work, we survey the requirements in Austria's production industry with the goal of generalizing requirements to build three industry personas. In contrast to Verlande et al., we do not only focus on the identification of requirements of a given corporate function (i.e., HRM), but we rather aim for deriving FL solution blueprints that can be implemented in multiple industries.

7.3 Methodology

In this chapter, we explain how a qualitative research approach has been applied to derive insights from the interview data. The generation of qualitative data was realized by means of guideline-based (expert) interviews, which is a widespread, differentiated, and methodologically well-developed method [Hel11]. The qualitative interviews were conducted as expert interviews [BLM14, GL10], as these had to meet special selection criteria. The expert interview is a special form of the guided interview and is defined by selection and status of the interviewees [Hel11, Mar03]. The interview data were subsequently processed using qualitative content analysis and presented in form of industry personas that are a conceptual extension of the AI personas according to Holzinger et al. [HKK+22].

Data Collection

Before the interviews were conducted, a semi-structured interview guideline was prepared. Based on the research questions, the interview guideline consists of the following major topics: i) current pain points or use cases that could be solved by AI, ii) questions about data, data management and maturity levels including time series data and storage management of data such as on-premise computing vs. cloud computing vs. edge computing, iii) time series data analysis and AI solutions, and iv) the (potential) application or usage of FL. Additionally, a slide set was prepared to explain to the participants (where necessary) the concepts of i) time series data, ii) edge-computing vs. on-premise computing vs. cloud computing, and iii) FL. All interview partners were recruited through either the different company networks of the authors or through private relationships. Additionally, the authors used LinkedIn to promote the study on social media to recruit further participants. The interviews were conducted online using MS Teams. Except for one interview, all were recorded (audio and video) and automatically transcribed. All interviews were conducted in German, for presenting the results, the respective quotes were translated into English.

Analysis

The analysis of the collected primary data has been carried out in the form of qualitative content analysis according to Glaeser and Laudel [GL10]. The basic procedure consists of understanding and interpreting the collected data and texts (interview transcripts) in a systematic, rule-based manner. The aim of the analysis is to filter out AI-relevant aspects from the material according to previously defined classification criteria (structuring). The data analysis started with the definition of the structuring dimensions and the further differentiation into individual characteristics. The main chapters of the interview guidelines were selected as the central structuring dimensions. The results were processed by extracting, summarizing, and comparing relevant topics and aspects from the data material and then condensed into “industry personas” [HKK+22].

Participants

Finally, thirteen ($n = 13$) male people agreed to participate in the interviews. Eleven ($n = 11$) of them hold an MSc in a technical study (e.g., telematics, automation engineering, automotive engineering), one ($n = 1$) of them studied business administration and one ($n = 1$) of them holds an apprenticeship diploma. The participants reported to work in different industrial sectors (number of participants in brackets) such as automotive industry (3), metal industry (3), electronics (2), logistics (2), paper production industry (1), mechanical engineering (1), and waste management (1).

7.4 Industry Personas

During the interview analysis, we were able to cluster the interview results and derive three interview-overarching *industry personas* based on the method for persona identification and description by Holzinger et al. [HKK+22]. In this regard, we use the concept of personas for AI systems as archetypes that help designers to focus on the goals and needs throughout the AI product development process [HKK+22]. Based on that, we define an industry persona as a representative model of a specific business type within an industry, conceptualized based on their common traits, goals, and pain points in implementing AI applications (including collaborative AI approaches such as FL). The three identified industry personas are i) service business, ii) production optimization, and iii) complex product and project business. The personas are described with respect to five dimensions i.e., *goal* of the persona, *structural and behavioral aspects* of the industry or the business, *AI applications* planned to be used, *pain points* regarding the implementation, and the *attitude towards AI/FL*. Depending on the insights gained from the interviews, we need to mention that interview participants could be related to one or more industry persona, vice versa other participants could not be assigned to any of the industry personas (e.g., P7, P8), as shown in Table 7.1.

Table 7.1: Industry personas and assigned participants

Industry Persona (IP)	Participants
IP 1: Service Business	P1, P5, P6, P9
IP 2: Production Optimization	P2, P3, P5, P6, P11
IP 3: Complex Product and Project Business	P4, P10, P12

7.4.1 General Interview Results

The interview results show that there exists a plethora of different reasons for the application of AI solutions or the desire to introduce AI solutions into companies. These reasons are manifold and very diverse, like for example, *" . . . finding use cases to open up new business areas using AI"* (P1), *"We always want/need to be at the forefront of technology"* (P5), *" . . . addressing problems that can be solved with data-based approaches"* (P1) or to develop *"new project ideas"* (P2, P9), or apply new *"imaging technologies"* (P2, P9). However, as stated by P7 and P8 AI solutions are mostly used for small but very specific solutions. Answers regarding the improvement of services or processes were for example *"Optimisation of systems (machines) and production processes using AI"* (P6) or *"Predictive planning for the customer"* (P5).

7.4.2 Industry Persona 1: Service Business

Goal

The goal of this industry persona is related to improving services towards their customers. For example, P1 is an AI manager and data scientist working in logistics. Their goal is to develop recommender systems based upon time series data from deployed logistic systems at the customer's site to offer suggestions on how to improve the customer's businesses. Another example is related to P5, who is a production manager in a mechanical engineering company. Their goal is to create predictions, trends, or forecasts for long-time customers e.g., to predict a customer's order requirements. P9 is a product manager in logistics. They also need customer data to be able to develop additional services for e.g., maximizing transparency of a logistic system's state.

Structural and behavioral aspects

All interviewees have existing (P1, P6, P9) and also long-time (P5) relationships with their customers.

AI Applications

In all three examples of the interviews P1, P5 and P9 mentioned above, AI could help to provide and/or improve forecasting or recommender services (P1, P5, P9) or develop new services for their customers (P1, P9) based on the data provided by their customers.

Pain Points

However, data sharing was and still is a difficult topic. For example, P1 stated that it is *"hard to get customer data"* and P6 stated that *"... but the more complex the solution of the subcontractors is, the less is revealed"*. From the perspective of P5, their customers say that the order process has worked as it is now for decades, why should they change anything.

Attitude towards AI/FL

The attitudes towards AI and FL are ambiguous. For example, P1 stated that sharing AI models related to FL strongly depends on the use case and that the fear exists of sharing such a model with direct competitors. P5 refers to the fear of AI on the one hand of his own employees and on the other hand on the side of the customers towards "never change a running system". In contrast, P9 clearly stated that their customers are willing to share their data if the use of data is legally (contractually) secured and if the benefits of the service provided are comprehensible and recognizable.

7.4.3 Industry Persona 2: Production Optimization

Goal

The goal of this industry persona is related to production optimization within the companies of the interview partners. We uncovered several concrete examples, where existing processes could be improved when introducing an AI solution. For example, P5, who is a production manager in a mechanical engineering company explained that in his company there is one colleague in the office who programs all the production machines. This person has to extract the existing parameters used for programming i.e., which material is processed on which machine with which tools and how. The goal would be to have an AI that could take over the programming. AI could be used to create pre-generated programs where the same material is processed with the same tools on the same machine. An AI would have a lot of potential in this respect. Another example provided by P5 is about implementing Industry 4.0, meaning digitization and in this regard reducing the paper usage in the company. He stated that *"we have actually reached the point where 80-90% of production is paperless. We no longer use paper at all, whether it's delivery notes, invoices, etc. Everything is digital; the only place where we still have paper is in accounting because the accounting department is a bit stiff."* This is where the changeover is taking the longest. P6 is a data scientist and developer in the metal industry. From his perspective, there is much potential for AI regarding control systems to optimize systems and related production processes. Control systems are time-critical and have a cycle time of 100ms—no more and no less. P11 is the head of automation technology in the automotive industry. In their company, spot welding is used to assemble different parts of cars. Based on the continuously recorded process data (e.g., with the Microsoft Business Intelligence Tool), the plan is to use an AI solution to analyze the welding points for anomalies and then send notifications to maintenance or production if an anomaly is detected. A more general statement was given by P2, who is a senior data scientist working in production industry. He is aware that establishing new AI solutions or new data science solutions in the company is a very difficult job. So, their goal is to find ways to introduce new AI solutions together with potential target users.

Structural and behavioral aspects

All interviewees with this persona (P2, P3, P5, P6, P11) have found different types of processes that could be improved by using AI. P3 refers to improving the ordering process, while P5, P6 and P11 really point out AI solutions to improve manufacturing processes. Additionally, P2 and P5 see a lot of potential regarding different process improvements using AI, however, they clearly state that the human factor needs to be thoroughly considered.

AI Applications

Possible AI solutions that could help to improve the processes are object and anomaly detection (P11), forecasting (P3), generative AI for code and maybe text generation (P5) as well as parameter estimation for optimizing programming and control processes (P5, P6).

Pain Points

The pain point mentioned by P3 is as follows: the time and effort needed to put an order entry into their internal ERP system is especially for small orders very high and puts pressure on the balance sheet. So far, there exists no good solution for automatically recording small orders, but AI could help. The goal would be to automatically record the order data including all relevant information independent of the order size. P5 described his pain point as follows: The programming of the production machines works well as it is, but there is the *"human factor, who unfortunately sometimes makes sloppy mistakes, who doesn't remember certain empirical values, who then has to spend a lot of time searching for these empirical values"*. For the paperless office example, P5 reported that fear of new technologies or AI is a big issue. Although he thinks that AI could offer more opportunities than risks, the introduction must be well-planned. The whole implementation process must function as one uniform system, all data must be available in the right places so that everyone can work with it. P6 recognizes the potential of AI regarding their control systems, however, their control systems are currently lagging behind AI. In terms of software, the control systems would be ready (solutions with GPUs), but in terms of hardware, it is not yet fast enough i.e., real-time control by AI is not yet possible. Regarding analyzing the welding points, P11 mentioned two pain points: this use case has not yet been realized, first due to resource bottlenecks and second because of insufficient computing power. P2 stated that *"Production engineers are very busy and don't have time to deal with new data science methods."*, therefore, it is difficult to convince them to use newly developed AI solutions.

Attitude towards AI/FL

The attitude towards AI is there, however, there are several challenges that need to be overcome before being able to introduce AI in the respective companies. Especially P2 and P5 refer to the human factor that needs to be considered. P2 clearly states that in order to introduce a new AI solution, the management needs to be on board and the target groups need to be shown a clear benefit (e.g., reducing workload, increasing security). Similarly, P5 raises the issue of fear of AI, thus, his employees must be integrated into the introduction process, and this must be done very carefully. Additionally, P3 and P6 state that they would like to introduce AI for process improvement, however, P3 stated that for his use case, no AI solution exists that could solve his problem, while P6 states that the software needed for solving his problem is there, but the hardware is still too slow.

7.4.4 Industry Persona 3: Complex Product and Project Business

Goal

The goal of this industry persona is related to improving the quality of products in the manufacturing process. This means to establish a quality control process and better product management to continuously ensure the product functionality at the end-of-line. For example, P4 is a simulation engineer from the automotive industry. During test drives with their vehicles, they look at the physical variables that are recorded e.g., bus data that combines different components so-called "black-boxes" like the steering control of the vehicles including camera sensors. Therefore, they equip the vehicles with sensors to measure speeds, accelerations, any physical variables, wheel speed distances etc. They look at how a vehicle reacts when it gets into a certain situation i.e., whether it reacts correctly or incorrectly. If it is reacting incorrectly, they look at where the error comes from. Similarly, P12 is the head of development of high-voltage systems in the automotive industry. During vehicle testing, they continuously measure and collect data. As a driver, one can set triggers in the event of a fault, but the drivers do not know whether the fault has perhaps occurred already before the current event. In both cases, the goal is to use AI to support error detection in vehicles during test drives. Another example is provided by P10, who is a product owner for ML and quality inspection in the metal industry. They produce high-performance components for engines, transmissions, and industrial applications which need to fulfil very high quality standards. AI is mainly applied in in-line and end-of-line quality inspection.

Structural and behavioral aspects

P4 and P12 stated that AI could help to enhance the quality control processes e.g., during vehicle testing. In the case of P10, AI could help to improve the quality of their products (quality prediction through anomaly detection and classification).

AI Applications

Regarding existing AI applications or solutions, P4 stated that his company has taken first very small steps towards using AI. For example, they use AI during test drives, in their case the road is filmed from the vehicle using a built-in camera. They analyze the videos and try to automatically recognize traffic signs so that they do not have to label them manually. P12, for example, is aware that there exists an AI application to support the data analysis of the data collected through vehicle testing, however, this application is currently not used. In contrast, P10 mentioned that they already use different AI solutions for quality assurance. For example, they use in-line and end-of-line anomaly detection, where they simply recognize "good" and "bad" images through classification (DL). Regarding the optimization of production processes (production planning) like for example which orders are placed on which system at which time, their AI is used for bottleneck identification or to optimize the capacity utilization.

Pain Points

For example, P12 mentioned that they store all technical data from vehicle testing on huge on-premises, thus on servers/hardware located in the organisation's own facilities. However, this data or information is currently not used at all. In this regard, pattern analysis could be used to uncover the problems in the vehicles. This has potential, but they are not yet utilizing this potential. P12 stated that they are *"sitting on a treasure trove of data that they could/should exploit"*. One suggestion provided by P12 is that in their case AI could be used to automatically detect errors during vehicle tests. Upfront, the AI could be fed with the functional load specification and then the AI can monitor the measurement technology during driving to see whether the vehicle behaves as expected or whether an error occurs. P10 mentioned that their major barrier to implementing AI solutions is often related to the lack of resources. To be able to develop/train good models, it needs domain experts i.e., the specialists/experts in production or on the plant in question, however, these experts are often not available. Additionally, he stated that predictive quality and predictive maintenance seem to have great potential and that many use cases have been launched in this regard, however, they have almost never been established.

Attitude towards AI/FL

Here, the attitudes towards AI and FL are similar to those mentioned in the first industrial persona. For example, P4 stated that AI is generally rejected, or technicians or even car testers have existential fears, namely that AI could replace them.

7.5 FL Blueprints

To address the industry personas with their goals to realize AI-based applications in a complex business environment with multiple involved parties, we propose three solution blueprints making use of FL. Applying FL facilitates the collaboration between the involved parties (e.g., service companies with their customers) to jointly train the AI models that are integrated into the applications.

In this section, we introduce the fundamentals used in the FL blueprints, followed by each blueprint's system design, implementation steps, and collaboration mode.

7.5.1 Fundamentals

Collaboration Mode

To the best of our knowledge, there are currently no classification terms describing the awareness and activities of an FL party on how it collaborates in an FL solution. Therefore, we introduce two collaboration modes that are specified for each involved party in FL.

Implicit FL is given for a party if FL is performed in the background without the need for intervention or (re)configuration. This applies to parties that use e.g., applications for analytical purposes and directly or indirectly generating data and labels. Only minor initial setup (e.g., acknowledging that FL happens in the background) is tolerated in implicit FL.

Explicit FL is given if the party is aware that FL is applied and has the possibility to actively control the learning process by e.g., providing datasets, selecting collaborators, starting FL runs, and deploying models. In explicit FL, collaborators (e.g., data scientists of a consortia) need to align on the planned runs and the potential impact, while this is abstracted away by e.g., service providers in implicit FL.

Data Partitioning

According to the partitioning of samples and features, we recap the following terminologies from Chapter 2:

Horizontal Federated Learning (HFL) is given if involved parties have datasets with different data samples (e.g., products) but with the same features (e.g., name, color). Hence, each party is a client that trains models with the same architecture locally and forwards them to the server for aggregation into a global model to facilitate knowledge exchange.

Vertical Federated Learning (VFL) is given if involved parties have datasets with the same data samples but with different features. Hence, different model architectures are used and trained on the client (= passive party). Typically, these models are defined as *Split Models*, since the output (i.e., matrix of real values) is combined by the server (= active party) as input into another model. The active party provides the labels and initiates the global learning process with backpropagation.

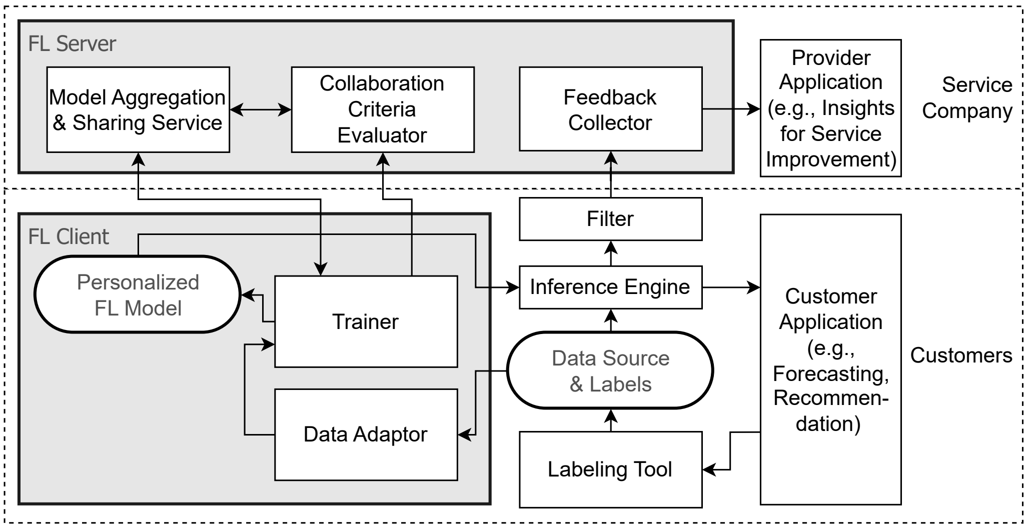

FL Server
Model Aggregation & Sharing Service
Collaboration Criteria Evaluator
Feedback Collector
Provider Application (e.g., Insights for Service Improvement)
Service Company
FL Client
Personalized FL Model
Trainer
Data Adaptor
Filter
Inference Engine
Data Source & Labels
Labeling Tool
Customer Application (e.g., Forecasting, Recommen-dation)
Customers

Figure 7.1: IFL blueprint for service businesses

7.5.2 FL Blueprint 1: Service Business

To provide value-adding services for sold products, service companies (industry persona 1) can make use of product-related usage insights collected from the customers. However, since customers are often hesitating to share (raw) data for ML-based services (e.g., product anomaly detection), a trustful infrastructure and collaboration mode has to be established. Additionally, the value of used applications need to be clear and recognizable for the customers supporting the product usage over the whole lifetime.

Figure 7.1 provides an architecture with two main parts, i.e., the server operated by the *Service Company* and the client installation that runs on the side of the *Customers*. Both sides run applications for e.g., forecasting product wear and making suitable recommendations. While the *Customer Application* is heavily based on raw data originating from data sources like IIoT sensors that come with the bought product, the *Provider Application* of the *Service Company* is based on non-privacy-disclosing feedback data from customers. Hence, the *Service Company* aims to provide suitable services (e.g., predictive maintenance) based on processed inference results (e.g., classifications of product issues) of multiple customers.

To support these applications, HFL is used to train a common model for all customers. This is relevant to increase the quality of the inference results which is the main input for the *Customer Application*. Therefore, the architecture in Figure 7.1 considers the *FL Client* component that runs the *Data Adaptor* to map data from data source into the schema used as input for the ML model. The *Trainer* runs the training procedure pre-defined by the *Service Company* to iteratively optimize the model. Based on that, FL is applied by using the *Model Aggregation & Sharing Service* that aggregates and distributes an FL model in each iteration. However, to ensure that models are shared only between trusted parties (avoiding knowledge transfer to competitors), the *Trainer* registers the *FL Client* with respective criteria for participation on the server using the *Collaboration Criteria Evaluator*. Since the data distributions of involved FL Clients are typically heterogeneous, the *Trainer* ensures that the model is adapted to the local data distribution (e.g., with a final local training after the last iteration) and stored as *Personalized FL Model*.

Furthermore, the customer side runs the *Inference Engine* to execute the model for each data tuple coming from the data source. The inference results are forwarded to the *Filter*, which enables to specify which results should be forwarded and shared with the *Feedback Collector* of the *Service Company*. With this, the data flow to the server and the respective privacy level can be controlled. To improve the model quality, human expertise is considered in respective supervised learning setups by adding labels to recorded data tuples using the *LabeLing Tool*. In this case, the tool allows for manual data labeling (e.g., adding class labels for observed product issues). Additionally, labels are extracted from the *Customer Application* to seamlessly incorporate feedback into the learning process. For this, the tool would act as a backend service for the application.

To implement this blueprint, the participating parties need to consider the following steps: First, the server infrastructure is deployed to the *Service Company* on suitable compute and storage resources as defined in the given IT strategy (e.g., cloud, on-premise data center). Second, the data ingest needs to be configured for each *FL Client* to specify which local data points are used for FL. If the setup considers a *Customer Application*, this can be pre-configured in a suitable user interface during the installation process. Third, the *Customer Application* can be used and labels are generated as feedback is provided by the customer by e.g., documenting product-related issues. Fourth, the collected dataset is used by the deployed system to initiate HFL to increase model performance. The updated and personalized model is then used in the *Customer Application* to e.g., improve product anomaly detection. Finally, local insights resulting from the model inference are uploaded to the *Service Company* according to the configurable filter.

The collaboration mode can be described as implicit FL with initial configuration effort since FL is executed in the background as the *Customer Application* is used. The configuration effort is limited to the data mapping using the *Data Adaptor* and the definition of collaboration criteria for FL. Hence, the collaboration is suitable for setups that run on a long-term basis (e.g., product lifetime with service support) without explicitly intervening in the FL process.

7.5.3 FL Blueprint 2: Production Optimization

To optimize complex production processes and individual production steps, AI models can be used for e.g., anomaly detection and root cause analysis purposes, which allows timely intervention and therefore reduction of costs. Based on the resulting insights, the process automation system can be updated with suitable parameters. This addresses the

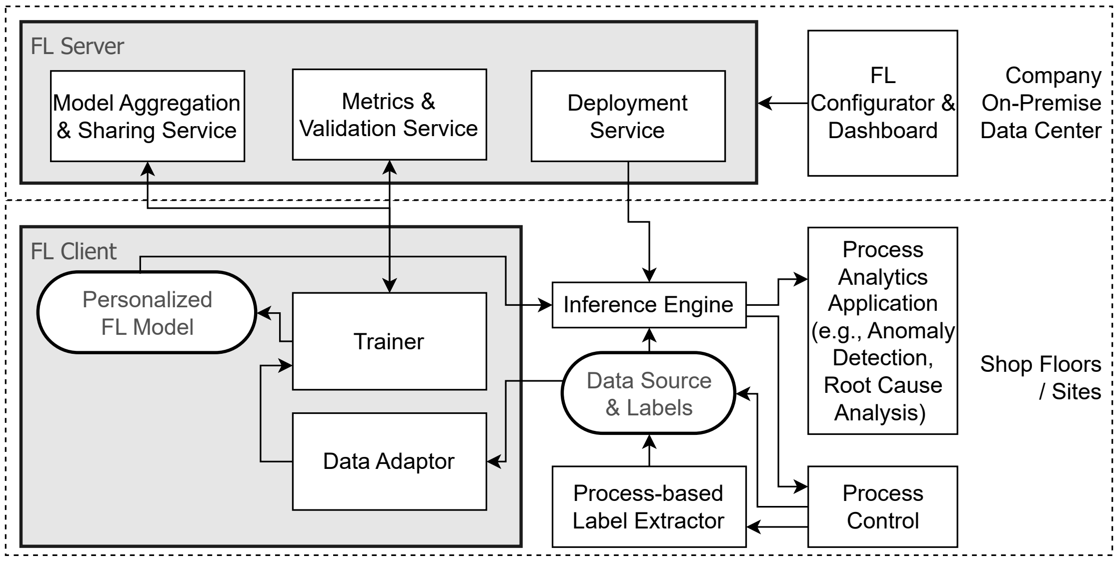

FL Server
Model Aggregation & Sharing Service
Metrics & Validation Service
Deployment Service
FL Configurator & Dashboard
Company On-Premise Data Center
FL Client
Personalized FL Model
Trainer
Data Adaptor
Inference Engine
Data Source & Labels
Process-based Label Extractor
Process Analytics Application (e.g., Anomaly Detection, Root Cause Analysis)
Process Control
Shop Floors / Sites

Figure 7.2: IFL blueprint for production optimization between multiple sites

need of the production optimization persona to tackle the lack of skills and error-prone production steps.

In this regard, we propose the architecture in Figure 7.2 supporting *Process Analytics Application(s)*. Multiple sites and shop floors of a company are involved to share knowledge and therefore boost performance for the model(s) used in the backend of the respective app deployments. To this end, the FL-based setup considers two main parts.

First, the *FL Server* is deployed to a data center with access to all involved sites (e.g., on-premise deployment). In addition, the *FL Configurator & Dashboard* is connected to the server for managing and monitoring the FL process. With this, responsible experts from the sites can initiate a new FL run with multiple involved clients, according to the underlying model and data distribution that need to be suitable for significant model improvements. Therefore, the *Model Aggregation & Sharing Service* provides FLaaS for the sites considering HFL for comparable production steps and corresponding ML tasks. The metrics (e.g., classification accuracies) resulting from local validation of the FL model are collected by the *Metrics & Validation Service*, which provides on-demand validation runs for the experts using the *FL Configurator & Dashboard*. Based on presented metrics from different clients and the respective development over time using the dashboard functionality, the experts can make use of the *Deployment Service*. It supports deploying the model to sites that likely benefit from the knowledge transfer with e.g., a better anomaly detection rate in the underlying process.

Second, the *FL Client* is considered at every site for local training and to support the execution of the *Process Analytics Application*. The process data is retrieved from the *Process Control* via standards like OPC and is then stored in the *Data & Label Storage* for subsequent training. The *Process-based Label Extractor* analyzes the process data by identifying e.g., manual interventions and events such as process interruptions that are considered unusual to derive class labels and store them along with the training data. This implicit way of extracting labels makes use of an initial configuration (e.g., using thresholds and rules) and complements explicit labeling, where data is manually labeled in the storage by experts on-demand. Both styles are relevant to collecting as many labels as possible to enable improved model performance. Based on data and labels, the *Data Adaptor* and *Trainer* are used the same as for the service business to create a *Personalized FL Model*. However, in this case, it is configured by the site experts that finally make use of the FL Configurator to load the (personalized) model into the site's *Inference Engine*. After that, the data tuples received from the underlying process are continuously ingested into the *Inference Engine* and the result is forwarded to the *Process Analytics Application* to optimize the site's production. Furthermore, the inference results can be used for updating parameters and set points in the process control. This enables a closed-loop automation cycle enhanced with FL.

The implementation steps for this blueprint consider an initial setup on every site selecting hardware resources for FL and process analytics and the installation of the presented services. Based on that, the FL initiatives are planned by data scientists and experts of the sites to figure out which sites can collaborate on a joint model. The HFL process is then initiated with the *FL Configurator* and the results are validated before the deployment decision is made by the experts. For this, the impact of the updated model for the process analytics needs to be discussed, and potentially be executed in a test environment before the model inference is used in production.

The collaboration mode is explicit FL as the responsible experts from the sites are actively starting and controlling the FL process using the *FL Configurator & Dashboard*. This is relevant since the impact on production is given for every site by updating values for the automation layer using the process control. While the decision for joint FL runs involves multiple site experts, the deployment decision of finished FL models is made by single sites. To address the limited availability of site experts (e.g., process and production engineers), the main effort of model building, training and initial validation can be decoupled and solved by central data scientists. The remaining work comprises the initial setup (e.g., data mapping and ML task definition) and the final validation before models are deployed to production environments.

7.5.4 FL Blueprint 3: Complex Product and Project Business

Product or project development between multiple *Subcontractors* and a *Prime Contractor* can be a complex task especially when it comes to system integration [Hob98]. Multi-firm alliances are maintained to coordinate the production process with close interactions between partners. In particular, the quality inspection process is an important use case that can be supported with an FL solution.

For this, Figure 7.3 provides an architecture for VFL between a *Prime Contractor* as *Active Vertical FL Party* and involved *Subcontractors* as *Passive Vertical FL Parties*.

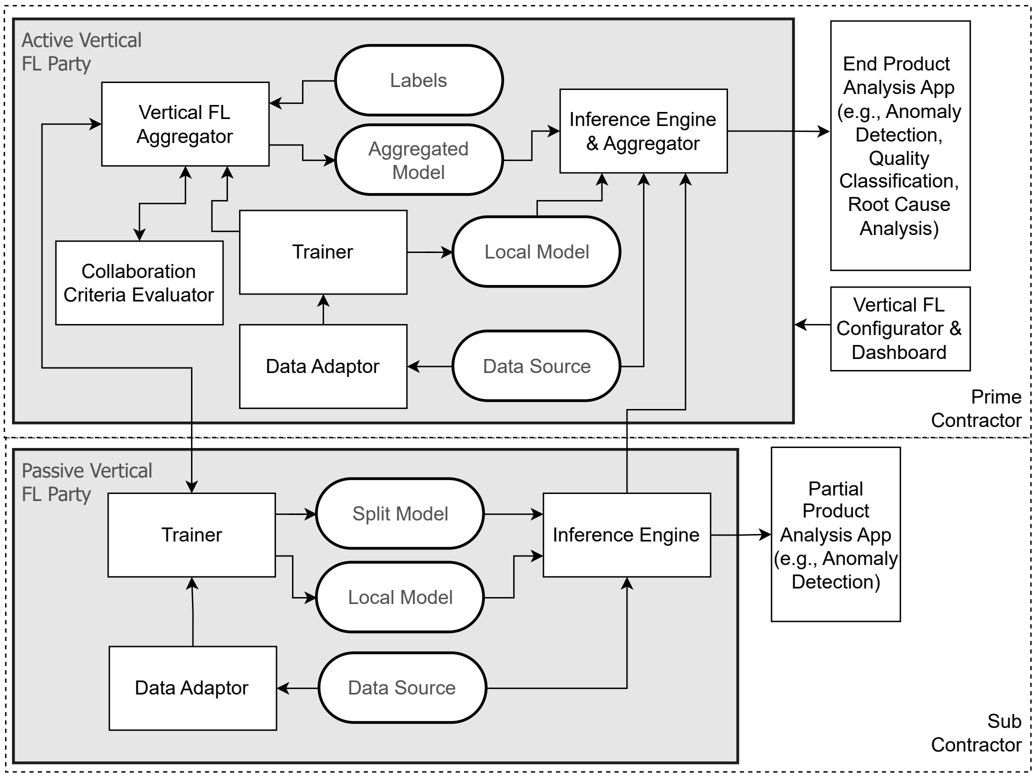

Active Vertical FL Party
Labels
Vertical FL Aggregator
Aggregated Model
Inference Engine & Aggregator
End Product Analysis App (e.g., Anomaly Detection, Quality Classification, Root Cause Analysis)
Collaboration Criteria Evaluator
Trainer
Local Model
Data Adaptor
Data Source
Vertical FL Configurator & Dashboard
Prime Contractor
Passive Vertical FL Party
Trainer
Split Model
Inference Engine
Partial Product Analysis App (e.g., Anomaly Detection)
Local Model
Data Adaptor
Data Source
Sub Contractor

Figure 7.3: IFL blueprint for complex product and project businesses

The overall goal of the *Prime Contractor* is to improve the end product by e.g., gaining insights into the product quality using the *End Product Analysis App* that is based on the trained FL model and collaborative inference of all contractors. This application supports e.g., quality classification, anomaly detection, or root cause analysis as the product is assembled. Similarly, the *Subcontractor* is interested in product quality of the respective part using the *Partial Product Analysis App*. To supply these apps with sufficiently trained FL models, the *Vertical FL Configurator Dashboard* is used by the *Prime Contractor* to initiate the FL process and control it accordingly by monitoring validation results. On both sides, VFL is applied.

For this, a *Subcontractor's Trainer* trains models on their local *Data Source* (e.g., tabular, image-based or time series-based quality data resulting from product inspection) after applying the *Data Adaptor* to map the local data as input features. A *Subcontractor* has two options, training of a *Local Model* without sharing insights with the server, and training a *Split Model*. The latter one is trained in the context of VFL as the output (e.g., values in the last layer in a neural network) is forwarded to the active VFL party (i.e., the *Prime Contractor*). With this, the active party can combine the outputs of multiple passive parties using the *Vertical FL Aggregator*, and plugging the outputs into the model of the active party. Hence the combined output is compared with the *Labels* (e.g., product quality classification after assembly), and using a suitable loss function, the errors are backpropagated to all involved parties. The repeated execution of this training procedure enables to train all *Split Models* and the combined global one. This approach is useful to enable model inference for the *End Product Analysis App* in a holistic way considering features of partial products that have been assembled into the end product. For this, the inference results of the *Subcontractors* (using the respective *Inference Engine*) need to be combined as well by the *Prime Contractor* using the *Inference Engine & Aggregator*. The collective inference output is displayed in the analysis app and allows for e.g., root cause analysis and anomaly detection in a detailed way to provide feedback to each contractor.

Although there is no model exchange between the *Subcontractors*, the *Collaboration Criteria Evaluator* is considered in this blueprint to configure under which conditions VFL is executed. The respective criteria can be configured on the passive party as the *Trainer* registers for FL at the active party. Furthermore, the *Prime Contractor* can use the *VFL Configurator & Dashboard* as a frontend for the *Collaboration Criteria Evaluator* to centrally control which *Subcontractors* (i.e., input features) are considered in the overall model (or multiple models in case of product variants). Additionally, the *Inference Engine* of the *Subcontractors* can be configured to only forward inference results to the *Prime Contractor* if specified accordingly. These settings address the hesitancy of *Subcontractors* to share data and collaborate accordingly since the configurability allows the involved parties to control what is shared while supporting use cases as root cause analysis and therefore immediate problem-solving.

The implementation of this blueprint considers the initial setup to install the FL services for the active and passive parties as well as the analysis applications. Afterwards, the training and the inference data flow need to be configured in a rule-based style to ensure which output is shared. In the meantime, the *Prime Contractor* collects and provides the *Labels* for an initial set of training data. Next, the VFL training is started and monitored using the *VFL Configurator & Dashboard*. As the model quality increases to an acceptable level, each party needs to deploy the model to the respective *Inference Engine*. Hence, the collective inference process is initiated and runs continuously in the background providing insights to the connected apps.

The collaboration mode of this blueprint is explicit FL for the active party since the FL runs need to be started and controlled by the *Prime Contractor* using the *VFL Configurator & Dashboard*. For the passive party, implicit FL is given since FL is applied seamlessly in the background after the rules for inference and model output sharing are declared in the setup phase. In case if no collaboration is required or acceptable for certain data instances, the *Local Model* can be used for isolated inference.

7.6 Discussion

In this section, we discuss the FL blueprints for the identified personas based on the five dimensions presented in Table 7.2. We have defined these dimensions to provide distinguishing characteristics and comparability of the blueprints.

Table 7.2: Properties of the proposed FL solution blueprints

	Service Business	Production Optimization	Complex Product and Project Business
FL paradigm	HFL	HFL	VFL
Collaboration mode	implicit FL	explicit FL	implicit FL (passive party) and explicit FL (active party)
Key benefit	privacy-preserving product usage insights for additional services	improved processes through FL models and interaction with process control	usage of *Subcontractor Split Models* for detailed feedback for the end product
Main addressed needs	integration of quality and usage feedback	optimizing (manual) error-prone tasks	privacy-preserving and configurable interaction between contractors
Main implementation challenge	rollout of solution and initial data collection	initial setup effort and achieving production-readiness	master operations and collecting inference results from all *Subcontractors*

First, the *FL paradigm* is HFL for the service business and the production optimization blueprints. This paradigm is used in the majority of approaches in literature [NDP+21], since it is less complex than VFL due to the same ML model architecture used by all clients. For the mentioned personas, HFL is applied since all involved clients collect the same data scheme from a similar product or production process. The resulting (personalized) FL model can be used by everyone without further dependencies on other parties when it comes to inference. This is different for complex product and project businesses as the *Subcontractors* view different data schemes and therefore apply VFL. A *Split Model* architecture needs additional alignment between the involved parties in the setup process as well as for the joint inference. However, a *Split Model* can be used individually for the local dataset focusing on analytics tasks for the partial product only.

Second, the *collaboration mode* is implicit FL for the service business blueprint and the passive party in the complex product and project business blueprint. This setup supports parties that do not want to be in the loop of the FL process and rather focus on the application that is based on the model in the backend. For this, the system needs to be installed by the provider or main contractor with sufficient technical resources. While the explicit FL mode (applied in the production optimization blueprint and for the active party in complex product and project business) provides certain controllability of the learning and deployment process, implicit FL reduces obstacles in the adoption of an

FL-based system especially for customers and *Subcontractors* that need to be convinced. Although *Subcontractors* are not explicitly involved in the FL process, the model still needs to be deployed to validate the suitability in local production.

Third, the *key benefit* of the service business blueprint is to provide product usage insights from customers in a privacy-preserving and agreed way. This lowers the barrier to connecting with customers and retrieving information that can be integrated into additional services like predictive maintenance. This can be used on the server side (Service Company) or directly on the client side (customer). For the production optimization blueprint, multiple sites connect to improve their processes with FL models for e.g., reduction of rejected production batches [KHJ+23]. The direct integration of the inference results into the process control using e.g., edge devices provides a low-latency approach for near real-time automation systems. However, based on the latency requirements of the use case, the FL model needs to be optimized and executed on a suitable hardware architecture [WYY+19]. The *Prime Contractor* in the complex product and project business blueprint can use the detailed insights from all *Split Models* of involved *Subcontractors* to e.g., identify errors in partial products or certain combinations of them in the end product. However, this needs input from the *Inference Engines* of *Subcontractors*, which can control what is forwarded to the *Prime Contractor's* inference aggregator. While this addresses the *Subcontractor's* concerns regarding data sharing, the overall inference result of the combined FL model cannot be derived sometimes. It can be concluded that further agreements (i.e., service levels) need to be in place to successfully collaborate on a long-term basis.

Fourth, the *main addressed needs* are the integration of quality and usage feedback to provide improved services, the optimization of (manual) error-prone tasks, and the privacy-preserving and configurable interaction between contractors to e.g., improve the product (quality). While these needs have been identified from different personas and addressed by corresponding FL blueprints, there is a certain overlap of architectural core components (i.e., *Collaboration Criteria Evaluator*, *Trainer*, *Data Adaptor*, *Model Aggregation & Sharing Service*). These components ensure the general attributes of IFL, to ensure privacy, heterogeneous data schemes and data distributions, and collaborations with selected partners. However, the blueprints are not limited to the respective personas and can rather be applied in different use cases.

Fifth, the *main implementation challenge* for the service blueprint is the rollout of the solution and the initial data collection. In particular, the customers need to use the application for a significant period to provide enough labels for FL. The service provider needs to identify the requirements of the customers, to e.g., configure the collaboration criteria in the client setup. Similarly, the initial setup effort is relevant in the implementation of the production optimization blueprint as well. In addition, to make the FL setup production-ready, multiple runs with involved sites might be necessary until the FL model can be deployed to production. Furthermore, general data availability (i.e., data recording in processes through either manual input or IIoT sensors) need to be given for FL, which cannot be taken as granted as P5 mentions that a number of departments have hardly digitalized their processes. For the complex product and project business blueprint, the main challenge is to master the operations that are needed to not only apply VFL but also to integrate inference results. To derive an aggregated inference result, all inputs are relevant and need to be collected from potential stragglers. This process lasts until all sub-products are produced and assembled and the model inference is applied on all *Split Models* as well as the combined model.

7.7 Summary

In this chapter, we presented the results of thirteen interviews in several Austrian industries to identify, e.g., AI applications, pain points, and the attitude towards AI and FL. The interviews have been conducted in 2024, to analyze the prerequisites for collaborative AI solutions (i.e., FL) because the industry has not yet adopted these technologies significantly. Based on that, we derived industry personas, each aiming to improve their underlying business or production activities (i.e., service business, production optimization and complex product and project business). To provide suitable AI solutions, we present three collaborative implementation blueprints (one per industry persona) using FL. Each blueprint consists of a system architecture, implementation steps, and a collaboration mode for the involved parties. For this, we introduced two novel definitions i.e., implicit FL and explicit FL. The FL blueprints have been discussed with respect to five dimensions (FL paradigm, collaboration mode, key benefit, main addressed needs, and main challenge) and the related work addresses other interview-based FL studies and system designs for industrial use.

To the best of our knowledge, this is the first work to offer practical FL designs, implementation guidelines for industry personas, and an outline for inter-party collaboration.

This chapter was published as a paper at the 26th International Conference on Business Informatics in 2024:

Blumauer-Hiessl, T., Fessl, A., Breitfuss, G., Schall, D., and Schulte, S. (2024). Federated learning solution blueprints for use cases surveyed in Austrian industries. In 2024 IEEE 26th Conference on Business Informatics (CBI) (pp 80-89).

CHAPTER 8

Conclusion and Future Work

Finally, we provide a summary of the results that have been presented in this thesis. We revisit the research questions from Section 1.2 and provide a critical analysis how we addressed these questions with the contributions in Section 8.1. In Section 8.2, we give an overview of the contributions and how they advanced the current state of the art. Section 8.3 concludes this thesis with an outlook on the future work.

8.1 Research Questions Revisited

RQ1 *How can systems and algorithms improve the operation of FL clients in real-world environments with skewed data and independent edge devices?*

To address this research question, we have proposed the *IFL System* (Chapter 3) with service interfaces for heterogeneous (edge) clients. The concept of cohorts has been introduced in the context of ML on IIoT-based asset data. To provide a solution for skewed asset data, we introduced the *IFL Process* (Chapter 4) enabling FL for cohorts of clients with similar data distributions. We have shown that the cohort-based FL model performance (accuracy) outperforms FL scenarios without cohorts or individual training. The proposed architecture supports individual and independent collaboration on shared ML models for every client by considering respective tasks with client requirements (i.e., federation criteria).

RQ2 *How should ML/FL lifecycles (individual steps and overall composition) be updated, enhanced, or managed by FL systems, to ensure high-quality models, customizability for individual clients, and integrability of FL into applications?*

With the *IFL Lifecycle* presented in Chapter 5, we enable to manage FL artifacts (= *IFL Templates*) considering steps including development, publishing, deployment, execution (training and prediction), and the integration into client applications. Using

IFL Templates for given use cases, clients can be served with FLaaS in a scalable and still customizable way. We presented an extensible library for FL, the *IFL Core*, which can be used to create an *IFL Template* for further integration into client applications (e.g., CM). In combination, the *IFL Core* and *IFL Templates* provide functionality for quality validation in a decentralized way as well as pre- and post-processing steps to facilitate the creation of high-quality and customized results for individual clients.

RQ3 *How can FL deployments be optimized on heterogeneous compute resources with respect to response times, energy consumption and model performance?*

In Chapter 6, we have presented an architecture for an FL system that supports FL client deployments on multiple locations using cloud, fog, and edge resources. The cohort-building algorithm *XMeanCohorting* facilitates the improvement of the model performance through personalization based on the cohorts concept. We have shown that model performance is increased by 10-15% as *XMeanCohorting* is applied. To further address the optimization of model performance and especially response times and energy consumption, the ILP-based client selection *IFL Opt* approach has been introduced. *IFL Opt* allows for the convergence of the multi-platform approaches to the relatively best basic deployment while also providing the flexibility to modify the selection of clients in each iteration.

RQ4 *What are the current pre-requisites for FL in industry and how can FL solutions be provided to overcome existing challenges?*

In Chapter 7, we presented interview results of 13 interviewees from Austrian industries. They have been asked to provide information with respect to AI applications, pain points in the underlying processses or in the implementation, and the attitude towards AI and FL. This can be considered as pre-requisites for future FL implementations, as there are no FL implementations in place so far. We have derived industry personas, and provide FL blueprints as a basis for a solution for each of them. The blueprints comprise architectures, implementation steps and collaboration modes to address the collaboration between involved parties. The proposed solution i.e., the collaboration models facilitate 1) performance improvements of commonly used ML models (e.g., for production optimization on different sites), 2) privacy-preserving training on data that is not accessible for a remote party (e.g., service company), and 3) distributed training and inference on product (parts) for a complex product integration using vertical FL. The proposed approaches address the surveyed challenges of the personas and can therefore be used to reduce the barrier for implementation in industry.

8.2 Summary of Contributions

In this thesis, we have shown how FL systems can be used to address needs of privacy-preserving learning that comes with improvements for underlying ML models in distributed setups with limited data and labels. Several algorithms have been integrated in the proposed FLaaS-based systems to tackle the challenges of non-IID data, client selection optimization, and personalization of models. An holistic lifecycle management supports the process of developing FL solutions until eventual deployment and integration into applications. To support the implementation in industry, FL blueprints are provided for three identified industry personas. The presented approaches are connected to several industrial scenarios and use cases, either as basis for requirements or as evaluation environment.

In Chapter 3, we have designed an FL system (*IFL System*) for industrial requirements and setups. The motivational scenario considers IIoT input data from industrial assets operating in heterogeneous environments. We have addressed the non-IID data problem by using the concept of population and cohort building based on the input metadata of clients. We have formulated workflows that need to be considered in *IFL System*s like e.g., individual FL client registration, matching of clients through joint metadata and federation criteria for collaboration with other partners.

We provide a detailed design of the *IFL System* in Chapter 4. The *IFL Process* has been introduced, providing FLaaS for individual and independent clients. A key part is the proposed *PopulationCohortBuilding* (Algorithm 2), that is integrated into the *IFL Process*. In the evaluation scenarios, we used vibration data from sensors mounted on centrifugal pumps to classify several conditions of the pump. We have shown that the cohort building algorithm outperforms individual FL and central learning. The results have been close to the optimum where all data has been centralized for a given cohort.

In Chapter 5, we introduced the *IFL Lifecycle*, which manages FL artifacts (= templates) throughout publishing, deployment, execution, and integration into client applications. The *IFL Lifecycle* is a process that is supported by the presented components, i.e., 1) the *IFL Core* library for developing FL training and prediction functionality, 2) the server for executing the server stage of the defined template, 3) and client applications (e.g., CM applications) for running the client stages for training and prediction (incl. pre- and post-processing). The template approach enables clients to develop scalable and customizable FL solutions. The templates can be used for tackling e.g., supervised or unsupervised tasks in an FL context. To address unsupervised FL (i.e., distributed and privacy clustering), we have proposed the FedClust algorithm with configurable levels of privacy disclosure. In the evaluation, we have demonstrated how FedClust is executed as a template along the *IFL Lifecycle*. To do this, we have integrated the *IFL Lifecycle* into a CM application for clustering industrial pump conditions. Through the deployment and execution of both a four-client and a 33-client scenario, we showcased the applicability of FedClust and the *IFL Lifecycle*. Our findings revealed that sharing only local cluster centroids using a privacy-preserving approach achieves similar performance to disclosing client data to the server.

In Chapter 6, we provided a deployment architecture for FL, enabling client deployments across multiple locations, utilizing cloud, fog, and edge resources. To enhance model performance, reduce energy consumption, and improve FL completion time, we incorporated the cohort-building algorithm *XMeanCohorting* and the ILP-based client selection *IFL*

Opt into the FL system. In the evaluation, we presented FL runs that utilized different client deployment strategies, including cloud, fog, and edge resources. We also employed a real-world industry use case, which addressed a quality inspection problem in a PCB manufacturing process. We conducted three scenarios to assess FL deployments at both the factory and production line levels, involving 2-6 clients and 6-18 clients, respectively. Within each scenario, we compared basic deployments with multi-platform deployments using the *IFL Opt* approach. Our findings demonstrated that applying *XMeanCohorting* led to a 10-15% improvement in model performance. Additionally, *IFL Opt* facilitated the convergence of multi-platform approaches towards the best-performing basic deployment, while still allowing for flexibility in updating the client selection in each round. We have also found that fine-grained client splitting (e.g., one client per site and multiple clients per factory) facilitated better cohort building and therefore provided improved model performance.

Chapter 7 presented the results of 13 interviews conducted in various Austrian industries, aiming to identify AI applications, pain points, and the attitudes towards AI and FL. Based on these findings, industry personas were derived, each focusing on improving specific business or production activities such as service business, production optimization, and complex product and project business. To offer suitable AI solutions, three collaborative implementation blueprints utilizing FL have been presented, with each blueprint tailored to a corresponding industry persona. These blueprints encompass system architectures, implementation steps, and collaboration modes for all involved parties. To support this, two novel definitions were introduced, namely implicit FL and explicit FL.

8.3 Future Work

This chapter concludes the thesis by providing an outlook to future work that may extend or complement the presented contributions.

First, in the context of cohort-based IFL systems (Chapters 3 and 4), efficient asynchronous FL for industrial edge devices without involving a server as central authority is an interesting future research direction. In particular, long-running (industrial) processes with infrequent data and label updates might benefit from continuous model updates shared with appropriate partners in the cohort. Hence, it can be worth investigating the effect of asynchronous updates that are shared continuously or at relatively short intervals in contrast to synchronous updates executed after relatively large intervals. This may enable fine-grained improvements of model quality and may avoid rollbacks of model updates that caused a negative knowledge transfer (= decreasing model quality). Furthermore, forecasting of potentially negative knowledge transfer could complement the idea of dynamically reorganizing cohorts over time. This can be relevant as the underlying data distribution of a client can change so that either some clients are excluded from a cohort, or new ones join. It can also be relevant to partially exchanging model updates between neighboring (similar) cohorts to further optimize model performance.

Furthermore, cohort reorganization between two communication rounds may reduce cohort building problems (e.g., random initialization in clustering) that can lead to a poor cohort model quality. Potentially, it is interesting to derive an individual FL model for a given client that has collaborated with different other clients over multiple communication rounds. Overall, developing innovative strategies to guarantee efficient FL models across diverse client cohorts can yield significant benefits.

Second, we plan to evaluate the *IFL Lifecycle* (Chapter 5) as part of the CM system in a real production environment based on heterogeneous edge devices. Investigating how users of CM applications apply concrete FL templates and interact within a group of collaborators would be interesting to reveal potential shortcomings of the proposed *IFL System* in the field. When it comes to apply globally trained cluster models as resulting from FedClust, it is relevant to answer the question to which extent the collaborators can help an individual client to correctly cluster local data. In particular, new cluster labels should be correctly used in inference tasks, even though these labels have not been utilized during the training of an individual client. Therefore, as future work, it could be interesting to evaluate the performance of FedClust on additional industrial datasets, especially considering variations in the number of unseen labels in the test data set.

Third, as future work based on Chapter 6, we plan to support the automated splitting of clients (i.e., their respective datasets) into finer-grained clients with a single data distribution and support automated provisioning of these. This can help to improve model performance through more homogenous cohorts of finer-grained clients. The automated provisioning of finer-grained clients may optimize the used resources and potentially reduces the manual effort of organizing clients by e.g., data scientists. Additionally, optimizing the deployment of resulting FL models for inference services could be of interest. Han et al. [HWJH24] (see Section 4.4) have already optimized the interplay between inference and FL training on the same device to provide high-quality inference results. However, the deployment of inference services to optimal locations, potentially decoupled from the locations of the dependent training clients, has not been addressed in FL so far.

Fourth, in Chapter 7, we proposed blueprints for solving AI problems with FL. As future work, the three blueprints need to be implemented in companies representing the different industry personas. Subsequently, another interview-based study should be considered to survey feedback from the implementation and ongoing operations. This can be of interest to investigate how these blueprints can be extended to provide optimal results (e.g., high model performance, low energy consumption, short response times) for the underlying problems. Overall, practical FL systems and solutions should be validated in the field with respect to the resulting model improvements, the impact on the underlying use case, the ease of use in a group of collaborators, customizability, and scalability.

APPENDIX A

IFL Terms

Table A.1: IFL terms used in this thesis

IFL Term	**Description**
Client Stages	Code that defines the behavior for training and inference on the client.
FL Cohort	A group of clients (or interchangeably a group of *FL Tasks*) with similar data distribution. An *FL Cohort* is a subset of an *FL Population*.
FL Plan	Execution instruction(s) for an *FL Task* (e.g., training and validating the model on the client).
FL Population	A group of *FL Tasks* created by clients which train models on asset data with the same data scheme. In this thesis, an *FL Population* is interchangeably considered as a group of clients, since an *FL Task* identifies the client and contains respective metadata for grouping.
FL Task	Representation of client requests for participating in FL.
IFL	Applying the concept of FL to industrial machines distributed across multiple factories, facing heterogeneous environmental and operational conditions.
IFL Client	FL clients (software) addressing industrial requirements and considering data of industrial assets.

Table A.2: IFL terms used in this thesis (ctd.)

IFL Term	**Description**
IFL Core	A Python library for FL that supports FLaaS and can be extended with custom behavior.
IFL Execution	Final stage of the *IFL Process*, which executes FL for identified cohorts, validates results, and provides the model to involved clients.
IFL Lifecycle	An FL lifecycle supporting the software development, publishing, deployment, and execution of FL solutions (e.g., *IFL Templates*). The *IFL Lifecycle* includes the *IFL Process*.
IFL Process	A multi-step approach that is provided as a service which encompasses algorithms for on-demand FL requests, the subsequent building of groups of clients (= cohorts) with similar data to address the non-IID data problem, and the FL training procedure.
IFL Server	Server that runs the *IFL Services*.
IFL Services	Software services providing an API to the *IFL Clients* providing knowledge aggregation and distribution on a central server.
IFL System	A system consisting of *IFL Clients* and *IFL Services* provided by the *IFL Server*.
IFL Template	Customizable software artifact with stages for defining server and client code (i.e., *Server Stage* and *Client Stages*) making use of the *IFL Core*.
Industry Application	Application that can make use of IFL.
ML Model	Representation of an ML model referenced in a *FL Task* considering multiple parameters and a training dataset (see domain model in Figure 3.2).
Server Stage	Code that defines the behavior of the FL server.

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

AI Artificial Intelligence. xii, 5, 7, 31, 105

ANN Artificial Neural Network. 52, 54, 55

AOI Automated Optical Inspection. 95

API Application Programming Interface. 4, 71

AWS Amazon Web Services. 83, 94

CESP Collaborative Edge Service Placement. 21

CM Condition Monitoring. 1, 7, 16, 25, 48, 67–69, 74, 77, 79, 83, 85, 87, 126, 127

CMD Command. 71

CoAP Constrained Application Protocol. 17

DL Deep Learning. 11, 49, 51, 84, 114

FedAvg Federated Averaging. 9, 44

FedClust Federated Clustering. 7, 69, 73–82, 84, 85, 127, 129, 134

FFT Fast Fourier Transform. 54

FL Federated Learning. ix, xi, 1, 2, 25–31, 33–37, 46, 56–63, 67, 105, 108, 128, 131, 133

FLaaS FL-as-a-Service. 2, 4, 7, 15, 16, 40, 41, 48–50, 64, 67–69, 72, 83–85, 105, 106, 126, 127

HFL Horizontal Federated Learning. 11, 12, 115

HRM Human Resource Management. 108

IaaS Infrastructure-as-a-Service. 20

IaC Infrastructure as Code. 97

IFL Industrial Federated Learning. 2, 27, 29, 31, 34, 35, 37, 56, 58–60, 68, 132

IID Independent and Identically Distributed. 3, 13, 51, 52, 54, 56, 57, 59

IIoT Industrial Internet of Things. ix, xi, 18, 19, 23, 30, 51, 77, 88, 105, 116, 123, 125, 127

ILP Integer Linear Programming. 7, 89, 90, 92

IoT Internet of Things. 1, 9, 13, 16, 20, 67, 87

JSON JavaScript Object Notation. 54, 55

LLM Large Language Model. 84

MAN Metropolitan Area Network. 22

MFCC Mel-frequency Cepstral Coefficients. 52

ML Machine Learning. xi, 1, 11, 25–27, 35, 51, 67, 105, 113

OEM Original Equipment Manufacturer. 105

OPC Open Platform Communications. 32, 118

PaaS Platform-as-a-Service. 20

PCA Principal Component Analysis. 93

PCB Printed Circuit Board. 95

PLC Programmable Logic Controller. 105

PoC Proof of Concept. 106

QoI Quality of Information. 27, 30, 32, 35, 37

QoS Quality of Service. 21, 35

ReLu Rectified Linear Units. 52, 54

RFID Radio Frequency Identification. 16

SD Standard Deviation. 97

SDK Software Development Kit. 84

SLR Systematic Literature Review. 81

TCP Transmission Control Protocol. 17

VFL Vertical Federated Learning. 11, 12, 115

VM Virtual Machine. 17, 20, 22, 90

WF workflow. 27, 35, 37

Bibliography

[AASC19] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. Federated learning with personalization layers. *arXiv preprint arXiv:1912.00818*, 2019.

[AATA+21] Abdulmalik Alwarafy, Khaled A. Al-Thelaya, Mohamed Abdallah, Jens Schneider, and Mounir Hamdi. A survey on security and privacy issues in edge-computing-assisted internet of things. *IEEE Internet of Things Journal*, 8(6):4004–4022, 2021.

[AB14] Sebastian Abt and Harald Baier. Are we missing labels? a study of the availability of ground-truth in network security research. In *2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, pages 40–55. IEEE, 2014.

[ARPS20] Mohammed Aledhari, Rehma Razzak, Reza M. Parizi, and Fahad Saeed. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access*, 8:140699–140725, 2020.

[ASCF23] Ahmed M. Abdelmoniem, Atal Narayan Sahu, Marco Canini, and Suhaib A. Fahmy. Refl: Resource-efficient federated learning. In *Eighteenth European Conference on Computer Systems*, EuroSys '23, page 215–232. ACM, 2023.

[ATMT21] Sawsan Abdulrahman, Hanine Tout, Azzam Mourad, and Chamseddine Talhi. Fedmccs: Multicriteria client selection model for optimal iot federated learning. *IEEE Internet of Things Journal*, 8(6):4723–4735, 2021.

[ATOS+21] Sawsan AbdulRahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. A survey on federated learning: The journey from centralized to distributed on-site learning and beyond. *IEEE Internet of Things Journal*, 8(7):5476–5497, 2021.

[BB86] Ronald Newbold Bracewell and Ronald N Bracewell. *The Fourier transform and its applications*. McGraw-Hill New York, 1986.

[BCM$^+$18] Theodora S Brisimi, Ruidi Chen, Theofanie Mela, Alex Olshevsky, Ioannis Ch Paschalidis, and Wei Shi. Federated learning of predictive models from federated electronic health records. *International Journal of Medical Informatics*, 112:59–67, 2018.

[BEG$^+$19] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. *arXiv preprint arXiv: 1902.01046*, 2019.

[BGvDH19] Thomas Bierweiler, Herbert Grieb, Stefan von Dosky, and Michael Hartl. Smart Sensing Environment – Use Cases and System for Plant Specific Monitoring and Optimization. In *Automation 2019*, pages 155–158. 2019.

[BHCW18] Hugh Boyes, Bil Hallaq, Joe Cunningham, and Tim Watson. The industrial internet of things (iiot): An analysis framework. *Computers in Industry*, 101:1–12, 2018.

[BHFB$^+$24] Thomas Blumauer-Hiessl, Angela Fessl, Gert Breitfuss, Daniel Schall, and Stefan Schulte. Federated learning solution blueprints for use cases surveyed in austrian industries. In *26th International Conference on Business Informatics (CBI 2024)*, pages 80–89. IEEE, 2024.

[BHKS$^+$23] Thomas Blumauer-Hiessl, Thomas Kaufmann, Erik Schwulera, Michael Kühne-Schlinkert, Konstantin Schmidt, and Boris Scharinger. Federated learning in industry. `https://www.siemens.com/global/en/products/automation/topic-areas/artificial-intelligence-in-industry/whitepaper-federated-learning-in-the-industry.html`, 2023.

[BHSRL$^+$24] Thomas Blumauer-Hiessl, Stefan Schulte, Safoura Rezapour Lakani, Alexander Keusch, Elias Pinter, Thomas Kaufmann, and Daniel Schall. Federated learning deployments of industrial applications on cloud, fog, and edge resources. In *2024 IEEE 8th International Conference on Fog and Edge Computing (ICFEC)*, pages 19–26. IEEE, 2024.

[BIK$^+$17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *2017 ACM SIGSAC Conference on Computer and Communications Security*, page 1175–1191. ACM, 2017.

[Bis07] Christopher M Bishop. *Pattern recognition and machine learning*. Springer New York, NY, USA, 2007.

[BKKZ22] Alexander Brecko, Erik Kajati, Jiri Koziorek, and Iveta Zolotova. Federated learning for edge computing: A survey. *Applied Sciences*, 12(18):9124, 2022.

[BLM14] Alexander Bogner, Beate Littig, and Wolfgang Menz. *Interviews mit Experten: eine praxisorientierte Einführung*. Springer-Verlag, 2014.

[BLS+13] Subramaniam Bagavathiappan, Barid Baran Lahiri, Thangavelu Saravanan, John Philip, and T Jayakumar. Infrared thermography for condition monitoring–a review. *Infrared Physics & Technology*, 60:35–55, 2013.

[BMZA12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *First Edition of the MCC Workshop on Mobile Cloud Computing*, pages 13–16, 2012.

[BTM+20] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D Lane. Flower: A Friendly Federated Learning Research Framework. *arXiv preprint arXiv:2007.14390*, 2020.

[CAZ+20] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. In *International Symposium on High-Performance Parallel and Distributed Computing (HPDC '20)*, pages 125–136. ACM, 2020.

[CBL+18] Ken Chang, Niranjan Balachandar, Carson Lam, Darvin Yi, James Brown, Andrew Beers, Bruce Rosen, Daniel Rubin, and Jayashree Kalpathy-Cramer. Distributed deep learning networks among institutions for medical imaging. *Journal of the American Medical Informatics Association*, 25, 03 2018.

[CBZ20] Michele Compare, Piero Baraldi, and Enrico Zio. Challenges to iot-enabled predictive maintenance for industry 4.0. *IEEE Internet of Things Journal*, 7(5):4585–4597, 2020.

[CH13] Lothar Cremer and Manfred Heckl. *Structure-borne sound: structural vibrations and sound radiation at audio frequencies*. Springer Science & Business Media, 2013.

[CKD+04] Chris Clifton, Murat Kantarcioundefinedlu, AnHai Doan, Gunther Schadow, Jaideep Vaidya, Ahmed Elmagarmid, and Dan Suciu. Privacy-preserving data integration and sharing. In *9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, page 19–26. ACM, 2004.

[CMF20] Andrew A. Cook, Göksel Mısırlı, and Zhong Fan. Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal*, 7(7):6481–6494, 2020.

[CNG+18] Inés Sittón Candanedo, Elena Hernández Nieves, Sara Rodríguez González, M. Teresa Santos Martín, and Alfonso González Briones. Machine learning predictive model for industry 4.0. In Lorna Uden, Branislav Hadzima, and I-Hsien Ting, editors, *Knowledge Management in Organizations*, pages 501–510. Springer International Publishing, 2018.

[CNSR20] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24, 2020.

[COM23] Mario Chahoud, Safa Otoum, and Azzam Mourad. On the feasibility of federated learning towards on-demand client deployment at the edge. *Information Processing & Management*, 60(1):103150, 2023.

[CSM+24] Mario Chahoud, Hani Sami, Azzam Mourad, Hadi Otrok, Jamal Bentahar, and Mohsen Guizani. On-demand model and client deployment in federated learning with deep reinforcement learning. *arXiv preprint arXiv:2405.07175*, 2024.

[DC21] Randy DeFauw and Collin Cudd. Applying Federated Learning for ML at the Edge. AWS architecture blog, 12 2021. `https://aws.amazon.com/blogs/architecture/applying-federated-learning-for-ml-at-the-edge/`.

[DCM+12] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *25th International Conference on Neural Information Processing Systems - Volume 1*, page 1223–1231. Curran Associates Inc., 2012.

[DLLW23] Tianchi Deng, Yingguang Li, Xu Liu, and Lihui Wang. Federated learning-based collaborative manufacturing for complex parts. *Journal of Intelligent Manufacturing*, 34(7):3025–3038, 2023.

[DODGS19] Alberto Diez-Olivan, Javier Del Ser, Diego Galar, and Basilio Sierra. Data fusion and machine learning for industrial prognosis: Trends and perspectives towards industry 4.0. *Information Fusion*, 50:92–111, 2019.

[DPM+22] Aaron Yi Ding, Ella Peltonen, Tobias Meuser, Atakan Aral, Christian Becker, Schahram Dustdar, Thomas Hiessl, Dieter Kranzlmüller, Madhusanka Liyanage, Setareh Maghsudi, Nitinder Mohan, Jörg Ott, Jan S.

Rellermeyer, Stefan Schulte, Henning Schulzrinne, Gürkan Solmaz, Sasu Tarkoma, Blesson Varghese, and Lars Wolf. Roadmap for edge ai: a dagstuhl perspective. *SIGCOMM Comput. Commun. Rev.*, 52(1):28–33, March 2022.

[DR+14] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[dSMdC+24] Allan M de Souza, Filipe Maciel, Joahannes BD da Costa, Luiz F Bittencourt, Eduardo Cerqueira, Antonio AF Loureiro, and Leandro A Villas. Adaptive client selection with personalization for communication efficient federated learning. *Ad Hoc Networks*, 157:103462, 2024.

[EARMAA+18] Ponciano Jorge Escamilla-Ambrosio, Abraham Rodríguez-Mota, Eleazar Aguirre-Anaya, Raul Acosta-Bermejo, and Moisés Salinas-Rosales. Distributing computing in the internet of things: Cloud, fog and edge computing overview. In Yazmin Maldonado, Leonardo Trujillo, Oliver Schütze, Annalisa Riccardi, and Massimiliano Vasile, editors, *NEO 2016: Results of the Numerical and Evolutionary Optimization Workshop NEO 2016 and the NEO Cities 2016 Workshop*, pages 87–115. Springer International Publishing, 2018.

[FKM+23] Tao Fan, Yan Kang, Guoqiang Ma, Weijing Chen, Wenbin Wei, Lixin Fan, and Qiang Yang. Fate-llm: A industrial grade federated learning framework for large language models. *arXiv preprint arXiv:2310.10049*, 2023.

[FVH19] Ferdinando Fioretto and Pascal Van Hentenryck. Privacy-preserving federated data sharing. In *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 638–646. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[FYS+20] Angelo Feraudo, Poonam Yadav, Vadim Safronov, Diana Andreea Popescu, Richard Mortier, Shiqiang Wang, Paolo Bellavista, and Jon Crowcroft. Colearn: Enabling federated learning in mud-compliant iot edge networks. In *Third ACM International Workshop on Edge Systems, Analytics and Networking (EdgeSys '20)*, pages 25–30. ACM, 2020.

[FZG+23] Lei Fu, Huanle Zhang, Ge Gao, Mi Zhang, and Xin Liu. Client selection in federated learning: Principles, challenges, and opportunities. *IEEE Internet of Things Journal*, 10(24):21811–21819, 2023.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[GCYR22] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. *IEEE Transactions on Information Theory*, 68(12):8076–8091, 2022.

[GHYR19] Avishek Ghosh, Justin Hong, Dong Yin, and Kannan Ramchandran. Robust federated learning in a heterogeneous environment. *arXiv preprint arXiv: 1906.06629*, 2019.

[GL10] Jochen Gläser and Grit Laudel. *Experteninterviews und qualitative Inhaltsanalyse*. Springer-Verlag, 2010.

[GSDH17] Zhiqiang Ge, Zhihuan Song, Steven X. Ding, and Biao Huang. Data mining and analytics in the process industry: The role of machine learning. *IEEE Access*, 5:20590–20616, 2017.

[Hel11] Cornelia Helfferich. *Die Qualität qualitativer Daten*, volume 4. Springer, 2011.

[HKH$^+$19] Thomas Hiessl, Vasileios Karagiannis, Christoph Hochreiner, Stefan Schulte, and Matteo Nardelli. Optimal placement of stream processing operators in the fog. In *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10, 2019.

[HKK$^+$22] Andreas Holzinger, Michaela Kargl, Bettina Kipperer, Peter Regitnig, Markus Plass, and Heimo Müller. Personas for artificial intelligence (ai) an open source toolbox. *IEEE Access*, 10:23732–23747, 2022.

[HLS$^+$20] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, et al. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518*, 2020.

[HMR$^+$19] Waldemar Hummer, Vinod Muthusamy, Thomas Rausch, Parijat Dube, Kaoutar El Maghraoui, Anupama Murthi, and Punleuk Oum. ModelOps: Cloud-Based Lifecycle Management for Reliable and Trusted AI. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pages 113–120, 2019.

[Hob98] Mike Hobday. Product complexity, innovation and industrial organisation. *Research Policy*, 26(6):689–710, 1998.

[HPH18] Adnan Husaković, Eugen Pfann, and Mario Huemer. Robust machine learning based acoustic classification of a material transport process. In *2018 14th Symposium on Neural Networks and Applications (NEUREL)*, pages 1–4, 2018.

[HPMG20] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. The non-IID data quagmire of decentralized machine learning. In *37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4387–4398. PMLR, 13–18 Jul 2020.

[HRK$^+$22] Thomas Hiessl, Safoura Rezapour Lakani, Jana Kemnitz, Daniel Schall, and Stefan Schulte. Cohort-based federated learning services for industrial collaboration on the edge. *Journal of Parallel and Distributed Computing*, 167:64–76, 2022.

[HRLU$^+$23] Thomas Hiessl, Safoura Rezapour Lakani, Michael Ungersboeck, Jana Kemnitz, Daniel Schall, and Stefan Schulte. Lifecycle management of federated learning artifacts in industrial applications. In *2023 IEEE 7th International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 2023.

[HRM$^+$18] Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *CoRR*, abs/1811.03604, 2018.

[HSKS20] Thomas Hiessl, Daniel Schall, Jana Kemnitz, and Stefan Schulte. Industrial Federated Learning – Requirements and System Design. In *Highlights in Practical Applications of Agents, Multi-Agent Systems, and Trust-worthiness. The PAAMS Collection*, pages 42–53. Springer International Publishing, 2020.

[HWJH24] Pengchao Han, Shiqiang Wang, Yang Jiao, and Jianwei Huang. Federated learning while providing model as a service: Joint training and inference optimization. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 631–640. IEEE, 2024.

[JAM$^+$23] Zohaib Jan, Farhad Ahamed, Wolfgang Mayer, Niki Patel, Georg Grossmann, Markus Stumptner, and Ana Kuusk. Artificial intelligence for industry 4.0: Systematic review of applications, challenges, and opportunities. *Expert Systems with Applications*, 216:119456, 2023.

[Jen94] Nick R. Jennings. The archon system and its applications. In *2nd International Conference on Cooperating Knowledge Based Systems (CKBS-94)*, pages 13–29, 1994.

[Jol02] Ian T Jolliffe. *Principal component analysis for special types of data.* Springer, 2002.

[JWW21] Jan Jöhnk, Malte Weißert, and Katrin Wyrtki. Ready or not, ai comes—an interview study of organizational ai readiness factors. *Business & Information Systems Engineering*, 63:5–20, 2021.

[KAH+19] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KBG+22] Jana Kemnitz, Thomas Bierweiler, Herbert Grieb, Stefan von Dosky, and Daniel Schall. Towards Robust and Transferable IIoT Sensor based Anomaly Classification using Artificial Intelligence. In *Data Science–Analytics and Applications*, pages 14–19. Springer, 2022.

[KHJ+23] Alexander Keusch, Thomas Hiessl, Martin Joksch, Axel Sündermann, Daniel Schall, and Stefan Schulte. Edge intelligence for detecting deviations in batch-based industrial processes. In *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*, pages 1–8. IEEE, 2023.

[KHY+19] Mareike Kritzler, Jack Hodges, Dan Yu, Kimberly Garcia, Hemant Shukla, and Florian Michahelles. Digital companion for industry. In *Companion Proceedings of The 2019 World Wide Web Conference*, page 663–667. ACM, 2019.

[KKG+20] Rajalakshmi Krishnamurthi, Adarsh Kumar, Dhanalekshmi Gopinathan, Anand Nayyar, and Basit Qureshi. An overview of iot sensor data processing, fusion, and analysis techniques. *Sensors*, 20(21), 2020.

[KKP20] Nicolas Kourtellis, Kleomenis Katevas, and Diego Perino. FLaaS: Federated Learning as a Service. In *1st Workshop on Distributed Machine Learning*, pages 7–13, 2020.

[Kle06] Achim Klenke. *Wahrscheinlichkeitstheorie*, volume 1. Springer, 2006.

[KMA+21a] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[KMA+21b] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and Open Problems in Federated Learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[KMN$^+$02] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[KS20] Vasileios Karagiannis and Stefan Schulte. Comparison of alternative architectures in fog computing. In *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*, pages 19–28. IEEE, 2020.

[KS22] Hermann Kopetz and Wilfried Steiner. Internet of things. In *Real-Time Systems: Design Principles for Distributed Embedded Applications*, pages 325–341. Springer International Publishing, 2022.

[KSLP19] Vasileios Karagiannis, Stefan Schulte, Joao Leitao, and Nuno Preguiça. Enabling fog computing using self-organizing compute nodes. In *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10. IEEE, 2019.

[KZ01] Gary King and Langche Zeng. Logistic regression in rare events data. *Political Analysis*, 9(2):137–163, 2001.

[KZDB16] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. The emerging role of data scientists on software development teams. In *38th International Conference on Software Engineering*, pages 96–107. ACM, 2016.

[LDSP18] Jay Lee, Hossein Davari, Jaskaran Singh, and Vibhor Pandhare. Industrial artificial intelligence for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 18:20–23, 2018.

[LGN$^+$21] Yi Liu, Sahil Garg, Jiangtian Nie, Yang Zhang, Zehui Xiong, Jiawen Kang, and M. Shamim Hossain. Deep anomaly detection for time-series data in industrial iot: A communication-efficient on-device federated learning approach. *IEEE Internet of Things Journal*, 8(8):6348–6358, 2021.

[LHL$^+$21] Quyuan Luo, Shihong Hu, Changle Li, Guanghui Li, and Weisong Shi. Resource scheduling in edge computing: A survey. *IEEE Communications Surveys & Tutorials*, 23(4):2131–2165, 2021.

[LK18] Kelvin Lui and Jeff Karmiol. AI infrastructure reference architecture. `https://www.ibm.com/downloads/cas/W1JQBNJV`, 2018.

[LLGL24] Bingyan Liu, Nuoyan Lv, Yuanchun Guo, and Yawen Li. Recent advances on federated learning: A systematic survey. *Neurocomputing*, 597:128019, 2024.

[LLPZ21] Sin Kit Lo, Qinghua Lu, Hye-Young Paik, and Liming Zhu. FLRA: A Reference Architecture for Federated Learning Systems. In *European Conference on Software Architecture*, pages 83–98. Springer International Publishing, 2021.

[LLPZ23] Sin Kit Lo, Qinghua Lu, Hye-Young Paik, and Liming Zhu. Decision models for selecting architectural patterns for federated machine learning systems. *Available at SSRN 4474488*, 2023.

[LLZ+22] Sin Kit Lo, Qinghua Lu, Liming Zhu, Hye-Young Paik, Xiwei Xu, and Chen Wang. Architectural patterns for the design of federated learning systems. *Journal of Systems and Software*, 191:111357, 2022.

[LSKW02] Yang Lee, Diane Strong, Beverly Kahn, and Richard Wang. Aimq: A methodology for information quality assessment. *Information & Management*, 40:133–146, 2002.

[LWLX19] Boyi Liu, Lujia Wang, Ming Liu, and Chengzhong Xu. Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems. *CoRR*, abs/1901.06455, 2019.

[LXZ15] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information Systems Frontiers*, 17:243–259, 2015.

[Man08] Christopher D Manning. *Introduction to information retrieval.* Syngress Publishing, 2008.

[Mar03] Winfried Marotzki. Leitfadeninterview. In *Hauptbegriffe Qualitativer Sozialforschung: Ein Wörterbuch*, page 114. Leske + Budrich, 2003.

[MBPS+23] Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Gérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges. *IEEE Communications Surveys & Tutorials*, 25(4):2983–3013, 2023.

[MG+11] Peter Mell, Tim Grance, et al. The NIST definition of cloud computing. *NIST Special Publication*, 2011.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *20th International Conference on Artificial Intelligence and Statistics*, volume 54, pages 1273–1282. PMLR, 2017.

[MMRS20] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv: 2002.10619*, 2020.

[MPP+21] Viraaji Mothukuri, Reza M. Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021.

[MRM+23] Carlo Mazzocca, Nicolò Romandini, Matteo Mendula, Rebecca Montanari, and Paolo Bellavista. Truflaas: Trustworthy federated learning as a service. *IEEE Internet of Things Journal*, 10(24):21266–21281, 2023.

[MSM21] Afra Mashhadi, Joshua Sterner, and Jeffrey Murray. Deep Embedded Clustering of Urban Communities Using Federated Learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[MVM16] Dimitris Mourtzis, Ekaterini Vlachou, and Nikolaos Milas. Industrial Big Data as a Result of IoT Adoption in Manufacturing. *Procedia CIRP*, 55:290–295, 2016.

[NCOD22] Van-Dinh Nguyen, Symeon Chatzinotas, Björn Ottersten, and Trung Q Duong. Fedfog: Network-aware optimization of federated learning over wireless fog-cloud systems. *IEEE Transactions on Wireless Communications*, 21(10):8581–8599, 2022.

[NDP+21] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658, 2021.

[NY19] Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[OÁP+23] Martijn Oldenhof, Gergely Ács, Balázs Pejó, Ansgar Schuffenhauer, Nicholas Holway, Noé Sturm, Arne Dieckmann, Oliver Fortmeier, Eric Boniface, Clément Mayer, et al. Industry-scale orchestrated federated learning for drug discovery. In *AAAI Conference on Artificial Intelligence*, volume 37, pages 15576–15584, 2023.

[PF13] Foster Provost and Tom Fawcett. *Data Science for Business: What you need to know about data mining and data-analytic thinking*. O'Reilly Media, Inc., 2013.

[PGM^{+}19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: an imperative style, high-performance deep learning library. In *33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2019.

[PHCM21] Jungwuk Park, Dong-Jun Han, Minseok Choi, and Jaekyun Moon. Sageflow: Robust federated learning against both stragglers and adversaries. *Advances in Neural Information Processing Systems*, 34:840–851, 2021.

[PKH19] Krishna Pillutla, Sham M. Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *arXiv preprint arXiv:1912.13445*, 2019.

[PSJ^{+}23] Sharnil Pandya, Gautam Srivastava, Rutvij Jhaveri, M. Rajasekhara Babu, Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, Spyridon Mastorakis, Md. Jalil Piran, and Thippa Reddy Gadekallu. Federated learning for smart cities: A comprehensive survey. *Sustainable Energy Technologies and Assessments*, 55:102987, 2023.

[QLDG09] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. Cloud computing: An overview. In *1st International Conference on Cloud Computing*, pages 626–631. Springer, 2009.

[RBP^{+}22] Swarna Priya Ramu, Parimala Boopalan, Quoc-Viet Pham, Praveen Kumar Reddy Maddikunta, Thien Huynh-The, Mamoun Alazab, Thanh Thi Nguyen, and Thippa Reddy Gadekallu. Federated learning enabled digital twins for smart cities: Concepts, recent advances, and future directions. *Sustainable Cities and Society*, 79:103663, 2022.

[RBWA21] Emmanuel Raj, David Buffoni, Magnus Westerlund, and Kimmo Ahola. Edge mlops: An automation framework for aiot applications. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 191–200, 2021.

[RMB23] Shinu M. Rajagopal, Supriya M., and Rajkumar Buyya. Fedsdm: Federated learning based smart decision making module for ecg data in iot integrated edge–fog–cloud computing environments. *Internet of Things*, 22:100784, 2023.

[Rou87] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[SAIR11] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171. IEEE, 2011.

[Sat17] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[SD16] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.

[SDO18] Omer Berat Sezer, Erdogan Dogdu, and Ahmet Murat Ozbayoglu. Context-aware computing, learning, and big data in internet of things: A survey. *IEEE Internet of Things Journal*, 5(1):1–27, 2018.

[SMD21] Rituparna Saha, Sudip Misra, and Pallav Kumar Deb. Fogfl: Fog-assisted federated learning for resource-constrained iot devices. *IEEE Internet of Things Journal*, 8(10):8456–8463, 2021.

[SMS21] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE Transactions on Neural Networks and Learning Systems*, 32(8):3710–3722, 2021.

[SNB$^+$21] Stefano Savazzi, Monica Nicoli, Mehdi Bennis, Sanaz Kianoush, and Luca Barbieri. Opportunities of federated learning in connected, cooperative, and automated industrial systems. *IEEE Communications Magazine*, 59(2):16–21, 2021.

[SNS$^+$17] Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Optimized iot service placement in the fog. *Service Oriented Computing and Applications*, 11(4):427–443, 2017.

[SRE$^+$18] Micah J Sheller, G Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation. In *International MICCAI Brainlesion Workshop*, pages 92–104. Springer, 2018.

[SS15] Reza Shokri and Vitaly Shmatikov. Privacy-Preserving Deep Learning. In *22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, page 1310–1321. ACM, 2015.

[SSA+21] Niclas Simmler, Pascal Sager, Philipp Andermatt, Ricardo Chavarriaga, Frank-Peter Schilling, Matthias Rosenthal, and Thilo Stadelmann. A Survey of Un-, Weakly-, and Semi-Supervised Learning Methods for Noisy, Missing and Partial Labels in Industrial Vision Applications. In *2021 8th Swiss Conference on Data Science (SDS)*, pages 26–31. IEEE, 2021.

[SYL21] Yuxin Shi, Han Yu, and Cyril Leung. Towards fairness-aware federated learning. *arXiv preprint arXiv:2111.01872*, 2021.

[TBZ+19] Nguyen H. Tran, Wei Bao, Albert Zomaya, Minh N. H. Nguyen, and Choong Seon Hong. Federated learning over wireless networks: Optimization model design and analysis. In *IEEE Conference on Computer Communications*, pages 1387–1395. IEEE, 2019.

[TDZD23] Javid Taheri, Schahram Dustdar, Albert Zomaya, and Shuiguang Deng. AI/ML on edge. In *Edge Intelligence: From Theory to Practice*, pages 183–211. Springer International Publishing, 2023.

[Tho53] Robert L Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, 1953.

[TS10] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI Global, 2010.

[TYCY23] Alysa Ziying Tan, Han Yu, Lizhen Cui, and Qiang Yang. Towards personalized federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12):9587–9603, 2023.

[UHSM23] Michael Ungersböck, Thomas Hiessl, Daniel Schall, and Florian Michahelles. Explainable federated learning: A lifecycle dashboard for industrial settings. *IEEE Pervasive Computing*, 22(1):19–28, 2023.

[VDP96] H. Van Dyke Parunak. Applications of distributed artificial intelligence in industry. In Gregory M. P. O'Hare and Nick R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 139–164. Wiley Interscience, 1996.

[VLR23] Lisa Verlande, Ulrike Lechner, and Steffi Rudel. Design of a federated learning system for it security: Towards secure human resource management. In *11th Latin-American Symposium on Dependable Computing*, page 131–136. ACM, 2023.

[VRL23] Lisa Verlande, Steffi Rudel, and Ulrike Lechner. Requirements for a federated learning system to strengthen IT security in human resource management. In *Wirtschaftsinformatik 2023 Proceedings*, 2023.

[VSP+23] Riccardo Venanzi, Michele Solimando, Marina Patrali, Luca Foschini, and Periklis Chatzimisios. Siemens and edgex iiot platforms: A functional and performance evaluation. In *IEEE International Conference on Communications*, pages 834–839. IEEE, 2023.

[WJH+21] Lin Wang, Lei Jiao, Ting He, Jun Li, and Henri Bal. Service placement for collaborative edge applications. *IEEE/ACM Transactions on Networking*, 29(1):34–47, 2021.

[WLD+20] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

[WYY+19] Su Weibin, Liu Yun, Du Yi, Dong Yingguo, Pan Mingbo, and Xu Gang. Three-real-time architecture of industrial automation based on edge computing. In *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pages 372–377. IEEE, 2019.

[YCKB18] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *35th International Conference on Machine Learning (ICML)*, volume 80, pages 5650–5659. PMLR, 2018.

[YFN+19] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[YHQL15] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78. IEEE, 2015.

[YL21] Rong Yu and Peichun Li. Toward resource-efficient federated learning in mobile edge computing. *IEEE Network*, 35(1):148–155, 2021.

[YLCT19] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology*, 10(2), 2019.

[YWL20] Pengqian Yu, Laura Wynter, and Shiau Hong Lim. Fed+: A family of fusion algorithms for federated learning. *arXiv preprint arXiv:2009.06303*, 2020.

[ZGWZ22] Weiming Zhuang, Xin Gan, Yonggang Wen, and Shuai Zhang. Easyfl: A low-code federated learning platform for dummies. *IEEE Internet of Things Journal*, 9(15):13740–13754, 2022.

[ZHLZ10] XM Zhao, Qinghua Hu, Yaguo Lei, and Ming J. Zuo. Vibration-based fault diagnosis of slurry pump impellers using neighbourhood rough set models. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 224(4):995–1006, 2010.

[ZLL$^+$23] Yifeng Zheng, Shangqi Lai, Yi Liu, Xingliang Yuan, Xun Yi, and Cong Wang. Aggregation service for federated learning: An efficient, secure, and more resilient realization. *IEEE Transactions on Dependable and Secure Computing*, 20(2):988–1001, 2023.

[ZXB$^+$21] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

[ZXLJ21] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. Federated learning on non-iid data: A survey. *Neurocomputing*, 465:371–390, 2021.