# Federated Learning on Medical Images: Closing Gaps to the Real-World

Master Thesis

by


## Marius Kempf

Matriculation number: 1843925

At the Department of Economics and Management

Digital Service Innovation (DSI)

Karlsruhe Service Research Institute (KSRI) &
Institute of Information Systems and Marketing (IISM)


| | |
|---|---|
| Advisor: | Prof. Dr. Gerhard Satzger |
| Second Advisor: | Prof. Dr. Hansjörg Fromm |
| Advisor DKFZ: | Prof. Dr. Klaus H. Maier-Hein |
| Supervisor: | Dr. Niklas Kühl |
| Supervisors DKFZ: | Klaus Kades, Maximilian Zenk |
| Date of Submission: | August 2, 2021 |

# Declaration of Academic Integrity

I hereby confirm that the present thesis is solely my own work and that if any text passages or diagrams from books, papers, the Web or other sources have been copied or in any other way used, all references—including those found in electronic media—have been acknowledged and fully cited.

*Karlsruhe, 02 August 2021* *Marius Kempf*

# Abstract

Data science offers many opportunities to gain new and valuable insights from medical (imaging) data to achieve enhanced patient care. The potential is immense, but medical data is sensitive and no public resource. This often conflicts with the demands of data hungry machine learning techniques. Federated learning is a paradigm that promises to leverage geographically distributed data while maintaining privacy constraints. However, while the principle methodology for federated training mechanisms is readily established on basis of simulated scenarios, real-world applications in the health care sector are still rare and come with plenty of remaining challenges. We, therefore, ask the question, what is needed to bring a federated learning solution into clinics. To bridge this gap in the medical imaging domain, we gathered requirements for respective technical solutions for real-world federated learning. We propose an integration of PySyft into the Joint Imaging Platform and a solution purely based on the Joint Imaging Platform: JIP Federated. The proposed as well as several existing federated learning solutions are evaluated using the identified requirements. We demonstrate the applicability of our solution on basis of a federated segmentation experiment on distributed brain MRI scans. JIP Federated is flexible and able to address some of the challenges that the community currently faces. It is available as an open-source project. The proposed solution might contribute to facilitating applied federated learning projects that truly go across medical institutions.

# Contents

# Acronyms

| | |
|---|---|
| API | Application Programming Interface |
| AUROC | Area Under the Receiver Operator Characteristic |
| BraTS | Brain Tumor Segmentation |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| CV | Computer Vision |
| DAG | Directed Acyclic Graph |
| DC | Dice Score |
| DKTK | German Cancer Consortium |
| DL | Deep Learning |
| ET | Enhancing Tumor |
| FATE | Federated AI Technology Enabler |
| FedAvg | Federated Averaging |
| FL | Federated Learning |
| FN | False Negative |
| FP | False Positive |
| FPR | False Positive Rate |
| GDPR | General Data Protection Regulation |
| GPU | Graphical Processing Unit |
| GUI | Graphical User Interface |
| HIPPA | Health Insurance Portability and Accountability Act |
| JIP | Joint Imaging Platform |
| ML | Machine Learning |
| MLP | Multi Layer Perceptron |
| MNIST | Modified National Insitute of Standards and Technology |
| MONAI | Medical Open Network for AI |
| MRI | Magnetic Resonance Imaging |
| PFL | PaddleFL |
| PP | PaddlePaddle |
| ROC | Receiver Operator Characterisitc |
| SGD | Stochastic Gradient Descent |
| SVM | Support Vector Machine |
| TC | Tumor Core |
| TFF | TensorFlow Federated |

| | |
|---|---|
| TN | True Negative |
| TP | True Positive |
| TPR | True Positive Rate |
| URL | Uniform Resource Locator |
| WT | Whole Tumor |

# List of Figures

# List of Tables

# 1 Introduction

From the moment it was possible to load and store medical images into a computer, researchers have begun to develop solutions for automated image analysis. Initially, analysis was done by, from today's perspective, simple rule-based systems. Later on, also machine learning (ML) techniques were applied. For the first applications of machine learning, features were extracted manually. Sample data was collected, and *handcrafted* feature vectors containing information about characteristics such as shapes, sizes or edges were derived from the images. Then, the features were entered as input to an ML model such as a multi-layer perceptron (MLP) (Rumelhart et al., 1986) or a support vector machine (SVM) (Vapnik, 1995). During training, the models search for the most optimal decision boundary, which is then used to separate the classes (in case of a classification).

The next step was to leave it to the computers, respectively models, to decide what information they need from an image to serve as a good representation. The concept lets layered networks extract the features themselves by starting with high-level features and going down to very specific low-level features. To this day, the most successful type of such models for image analysis are convolutional neural networks (CNNs). The architectures contain a multitude of layers which transform their input by applying convolutional filters. This first part of the model can be referred to as *feature-extractor*, because it generates meaningful features based on the given input by applying convolutions. It is followed by one or more classification layers, using the extracted features for its decision-making. In contrast to previous approaches, convolutional neural networks are an end-to-end processing architecture using the raw input images. For computer vision (CV), convolutional neural networks are the technology of choice to this day.

Back in the year 1998, LeCun et al. (1998) published first pioneering work using such CNNs in the real-world application of hand-written digit recognition. Over time, convolutional neural networks were used for increasingly complex applications and thus the models became also more complex, having millions of trainable parameters. This was only possible because the available computing resources and the available image data grew at the same. The size of the models and the architecture of consecutively stacked layers led to the term deep learning (DL) (Lecun et al., 2015). A major breakthrough was the publication of Krizhevsky et al. (2012) within the context of the ImageNet challenge[1] in 2012. The authors introduced a convolutional neural network referred to as AlexNet and won the challenge by a large margin. Following this performance leap, related architectures were developed, that became even deeper.

[1] `https://www.image-net.org/index.php`

The recent years have proved that deep learning is able to achieve state-of-the-art performance, even surpassing human-level performance, on a multitude of different tasks not only in computer vision but also in speech recognition and natural language processing (He et al., 2015; Hinton et al., 2012; Devlin et al., 2019). Also, within the medical domain, deep learning has gained attention, and models with expert-level performance have been reported (Esteva et al., 2017).

The availability of huge amounts of annotated data is one of the key factors for this success, namely datasets such as ImageNet (Deng et al., 2010). Deep learning in the medical domain is not different in that regard: To achieve human-level or clinical-grade performance, the amount and quality of annotated data available for model development are crucial. This applies in particular to very rare diseases or diseases with highly heterogeneous disorders (Esteva et al., 2017; De Fauw et al., 2018; Dluhoš et al., 2017).

When privacy is not a concern, the available data is usually centralized at one location and is then used for model training. Often, the datasets are even made publicly available as a contribution to the scientific community. In the medical domain, however, data is highly personal and, therefore, sensitive in terms of patients' privacy. Conclusively, its usage is strictly regulated and comes with a multitude of barriers and regulations, such as the United States *Health Insurance Portability and Accountability Act* (HIPPA) (US Department of Health and Human Services, 2020) or the European *General Data Protection Regulation* (GDPR) (European Commission, 2018) (Van Panhuis et al., 2014).

A paradigm to address and overcome the challenges of data volume and privacy while still allowing to learn over multiple, distributed instances, is federated learning (FL). Federated learning decouples the model training from the need for direct access to the raw training data of each participant. Its collaborative approach of aggregating knowledge from models trained in a decentralized manner works without accessing or exchanging the underlying data (Brendan McMahan et al., 2017).

Initially developed for learning from private data on mobile devices, such as written text, it is also applicable within the medical domain. One can think of a multitude of different applications of federated learning in healthcare (Xu et al., 2021). For instance, by applying federated learning on distributed electronic health records, a support vector machine, multi-layer perceptron, and logistic regression model have been trained to predict adverse drug reactions of patients and have achieved comparable performance metrics to models trained on centralized data (Choudhury et al., 2019). It has also been shown that federated learning can be used to improve models predicting the in-hospital mortality of patients admitted to the intensive care unit (Sharma et al., 2019).

In recent times, an increasing number of publications address federated learning in the context of medical image data and therefore combine federated learning with deep learning. For instance, Kaissis et al. (2021) applied federated deep learning on paediatric chest x-ray images, showing classification performance similar to locally trained models while maintaining patient privacy. Sheller et al. (2020) trained a model for tumor segmentation with federated learning, simulating a real distribution across institutions by separating magnetic resonance image (MRI) scans of subjects with brain tumors according to the meta-information about the origin of the images. The authors showed that 99% of the model performance compared to training on centralized data is reached. Roth et al. (2020) showed that models trained in a federated manner perform on average 6.3% better than their counterparts trained on the locally available data only. The objective was to classify breast density (being an indicator for a patient's risk to develop breast cancer) based on mammographic images. Their work has a distinctive characteristic: federated learning was applied in a real-world scenario. While many publications report federated learning experiments in a simulated setting, only few performed experiments across geographically distributed institutions (see Section 2.3 on page 7).

To fully profit from federated learning in healthcare, technical solutions for its real-world application need to be identified. Currently, there is a lot of activity resulting in promising projects which address this topic. Frameworks such as PySyft (Ryffel et al., 2018) or platform solutions like NVIDIA Clara Train[2] have attracted attention in the research community and are contributing to make federated learning real-world. While these federated learning solutions are tackling the challenges with available but private data, the healthcare sector is a very specific environment. Therefore, a possible application in real clinics comes with specific requirements to the eligible solutions.

This work investigates which functionalities are provided by current federated learning solutions and what technical requirements are necessary to bridge the gap between federated learning solutions and their application in medical institutions. Further, we demonstrate how the Joint Imaging Platform[3] (JIP) (Scherer et al., 2020) can be used for federated learning in two ways. Due to the Joint Imaging Platform's openness and flexibility, it can be easily combined with other solutions. Therefore, one option is to integrate another federated learning solution into the Joint Imaging Platform, as we demonstrate with the integration of PySyft. Further, the already existing components of the Joint Imaging Platform can be used to realize federated learning - this is what we introduce as JIP Federated. For both options, the implementations are described in detail. In addition, we conducted experiments to

[2] https://developer.nvidia.com/clara

[3] https://jip.dktk.dkfz.de/jiphomepage/

compare the runtime of the JIP-based solutions, as well as the performance of JIP Federated when segmenting gliomas in human brain MRI scans.

In summary, we make the following contributions with our work:

- We compare existing federated learning solutions based on requirements derived from the healthcare and medical image domain. To the best of our knowledge, there is no comprehensive comparison of existing solutions regarding our domain. Thus, we provide researchers and practitioners with an overview of their options in the area of federated learning in healthcare.
- We integrate PySyft into the Joint Imaging Platform to serve as a federated learning solution. Existing solutions mostly do not provide the required features for a medical environment. Hence, we demonstrate the benefit of the Joint Imaging Platform being a flexible open-source platform by combining its advantages with an existing federated learning solutions.
- We implement JIP Federated and demonstrate its capability in a multi-node setup. Thus, we provide an open-source solution, based on the Joint Imaging Platform, to conduct real-world federated learning on medical images in a clinical environment.

The remainder of this work is organized as follows. Section 2 reviews the related work in terms of federated learning and its application on medical images. In Section 3 the applied methods to compare federated learning solutions are provided. Further, it describes the implementations that have been developed and the setup of our experiments. Results are provided in Section 4. Section 5 discusses the results and points out limitations. Section 6 concludes our article with remarks on possible future work.

# 2 Background & Related Work

## 2.1 Federated Learning

In the traditional data-sharing paradigm, multiple institutions collaborate with each other by exchanging or aggregating their data into a central data storage. In contrast, no raw data is exchanged when applying FL. Instead, the data governance stays with each institution, and only models are shared with others. To realize such a collaborative effort, different design choices for FL can be implemented. A distinction can be made on the basis of the topology and the FL *compute plan*. The topology can be implemented as either centralized or decentralized, meaning the communication is routed via a central server or happens directly between the participants. Popular compute plans are *sequential training*, *peer-to-peer training*, and the usage of an *aggregation server* (Rieke et al., 2020).

*Sequential training* refers to a training process in which the model is continuously passed on from participant to participant, so each of them waits until the predecessor is finished. The participants can train the model locally for several steps or epochs, depending on the configuration of the training (Chang et al., 2018). One challenge when applying this compute plan is that it is negatively affected by a phenomenon known as catastrophic forgetting (French, 1999), which results in a model highly favoring the data it was trained on most recently.

The compute plan using an *aggregation server* can be described as follows. At the start of the collaborative training, a machine learning model is sent to all participants. Each replica is individually trained on the locally available data, and then the updated models are sent back to the central server. After receiving the models with their new states, they are aggregated using an aggregation algorithm. The initially proposed is *Federated Averaging* (FedAvg), which is still most widely used today. It computes a weighted average of the received model's parameters, and the resulting consensus model is then again shared with each participant for further training. This process is carried out repeatedly, with each iteration being referred to as a federated round (Brendan McMahan et al., 2017).

*Peer-to-peer training* can be described as training across multiple nodes without centralization. No central instance exists, and therefore participants communicate directly with each other. This implies that model aggregation itself also takes place on the individual participants. *Peer-to-peer training* generalizes the approaches of *aggregation server* and *sequential training*. The implementation of such a compute plan can be formed in arbitrary configurations (Chang et al., 2018; Lalitha et al., 2019; Roy et al., 2019).

Although FL provides a basic notion of privacy by keeping the data distributed, additional technical solutions are often required to ensure the privacy and security needed for the clinical application of FL. Kaissis et al. (2020) provide an overview of methods for secure and privacy-preserving FL and also possible threats for such learning settings in the field of medical imaging.

## 2.2 Federated Learning Solutions

In order to realize FL, technical solutions are required. One can either use a customized approach for each individual scenario or an existing FL solution. Such FL solutions are presented in the following.

**NVIDIA Clara Train** extends the NVIDIA Clara application framework optimized for the healthcare domain. It comes with useful features for medical images and enables developers to train models across institutions. In addition, it has already found some application in academia, as pointed out in the literature review in Section 2.3 (see page 7). In contrast to other solutions, the source code is not publicly available.

**Federated AI Technology Enabler**[4] (FATE) is a framework developed as open-source project by FedAI[5], a group of developers of the company WeBank. It provides a secure computing framework to strengthen the federated artificial intelligence ecosystem. Its source code is publicly available. FATE has not yet been applied on medical images in scientific articles.

**FedML**[6] is an open research library aiming to facilitate the development of new FL algorithms and fair performance comparison. The solution addresses multiple challenges which occur during scientific research in the field of FL. It is also applicable on edge devices such as smartphones (He et al., 2020). The source code is made publicly available. So far, FedML has not been used in academic articles on FL using medical images.

**PySyft**[7] is a privacy-preserving FL solution which is originally built over PyTorch but also supports implementations in TensorFlow. It aims to enable developers to write software which can compute over data which is not locally available but on remote machines they do not control. It comes with a large and active open-source community named Openmined[8] which is continuously enhancing the solution (Ryffel et al., 2018). As shown in Section 2.3 (see page 7), PySyft has already been used in

[4] `https://fate.fedai.org/`
[5] `https://www.fedai.org/`
[6] `https://fedml.ai/`
[7] `https://github.com/OpenMined/PySyft`
[8] `https://www.openmined.org/`

relevant research articles.

**PaddleFL**[9] (PFL) is a solution based on PaddlePaddle (PP), a DL framework developed by the company Baidu. PFL benefits from the underlying PP being a distributed and scalable platform with advanced functionalities. Its source code is publicly available. PFL has not yet been used in published research on FL with medical images.

**TensorFlow Federated**[10] (TFF) extends TensorFlow to facilitate open research and experimentation with FL in a simulated environment. It comes with two application programming interfaces (APIs): FL API and Federated Core API. Both represent different layers that can be adapted and extended by developers. By providing a multi-machine simulation runtime, TFF can be used for FL experiments applying different algorithms. As TensorFlow itself, TFF is an open-source project. So far, TFF has not been used in academic articles on FL using medical images.

Please note that further solutions may exist. In our understanding, the abovementioned are the most promising and well-known solutions and have already been examined in other publications (Li et al., 2019a; He et al., 2020).

## 2.3 Federated Learning with Medical Image Data

We systematically performed a literature search to identify publications applying FL on medical images. The basis for our search is the work of Kirienko et al. (2021). The authors applied the search term *"distributed learning" or "federated learning"* on the EMBASE and PubMed/MEDLINE databases. Out of the 26 articles identified, six applied FL using medical image data and are therefore relevant for our research context. Contributions up to July 21, 2020 were taken into consideration. We adopted the applied search procedure and extended the time period until May 31, 2021, resulting in 146 initial hits. After removing duplicates and initial screening of titles and abstracts, eight articles remained. Through forward and backward search, four additional articles were identified. Together with two previously known articles (Wang et al., 2020; Roth et al., 2020) and the results selected from Kirienko et al. (2021), this adds up to 20 articles applying FL on medical images.

The articles were analyzed regarding the following characteristics: used FL setting, FL solution, number of participants, compute plan, used algorithm, and given task. We differentiate between three types of FL environments: *real-world*, *synthetic*, and *simulated*. A setting that trains a model across geographically distributed institutions is referred to as *real-world*. Articles were classified as *synthetic* if a FL solution

[9] `https://github.com/PaddlePaddle/PaddleFL`

[10] `https://www.tensorflow.org/federated`

| Reference | FL Setting | FL Solution | Participants | Compute Plan | Algorithm | Task |
|---|---|---|---|---|---|---|
| Xu et al. (2020) | real-world | custom solution | 4 | FedAvg | 3D-Densenet | Classification |
| Balachandar et al. (2020) | simulated | - | 4 | Sequential | GoogleLeNet | Classification |
| Wang et al. (2020) | real-world | NVIDIA Clara Train | 2 | FedAvg | C2FNAS | Segmentation |
| Remedios et al. (2020) | real-world | custom solution | 2 | Sequential | U-Net | Segmentation |
| Remedios et al. (2019) | real-world | custom solution | 2 | Sequential | CNN (Incept.) | Segmentation |
| Chang et al. (2018) | simulated | - | 4 | Sequential & Ensembling | ResNet34 | Classification |
| Kaissis et al. (2021) | synthetic | PriMIA (based on PySyft) | 3 | FedAvg | ResNet18 | Classification |
| Dou et al. (2021) | synthetic | custom solution | 3 | FedAvg | CNN | Segmentation |
| Roth et al. (2020) | real-world | NVIDIA Clara Train | 7 | FedAvg | DenseNet-121 | Classification |
| Feki et al. (2021) | simulated | - | 4 | FedAvg | VGG16 & ResNet50 | Classification |
| Sarma et al. (2021) | real-world | NVIDIA Clara Train | 3 | FedAvg | 3D AH Net | Segmentation |
| Sheller et al. (2020) | simulated | - | 10 | FedAvg & Sequential | U-Net | Segmentation |
| Baheti et al. (2020) | simulated | - | 3 | FedAvg | V-Net | Classification |
| Yang et al. (2021) | synthetic | NVIDIA Clara Train | 3 | FedAvg | 3D U-Net | Segmentation |
| Sheller et al. (2019) | simulated | - | 4, 6, 8, 32 | FedAvg | U-Net | Segmentation |
| Li et al. (2019b) | synthetic | NVIDIA Clara Train | 13 | FedAvg | CNN | Segmentation |
| Andreux et al. (2020) | simulated | - | 2, 5 | FedAvg | CNN | Segmentation |
| Yan et al. (2020) | simulated | - | 2, 4, 8 | FedAvg | CNN | Classification |
| Lee et al. (2021) | real-world | PySyft | 6 | FedAvg | multiple | Classification |
| Flores et al. (2021)* | real-world | NVIDIA Clara Train | 20 | FedAvg | ResNet34 | Classification |

Table 1: List of identified articles applying FL using medical images in a real-world, synthetic, or simulated setting (*Preprint)

imitates a distributed system technically, by using multiple workers or computation units, such as individual servers, without having an actual cross-institutional setting. *Simulated* implies that no distributed network was used, but it was ensured programmatically that a predefined data partitioning was obeyed during the FL simulation.

Table 1 displays the identified articles categorized as either *real-world*, *synthetic*, or *simulated*. The results contain eight articles that used *real-world* settings, while four are classified to be *synthetic* and the remaining eight articles as *simulated*. Two FL solutions can be identified: The FL solutions used by six articles, and thus by most, is the proprietary solution NVIDIA Clara Train (Wang et al., 2020; Roth et al., 2020; Sarma et al., 2021; Yang et al., 2021; Li et al., 2019b; Flores et al., 2021). The open-source solution PySyft is used in two articles (Kaissis et al., 2021; Lee et al., 2021). Note that PriMIA[11] extends PySyft for medical image data. Four solutions do not use a specific framework, but customized solutions for the given setting (Xu et al., 2020; Remedios et al., 2020, 2019; Dou et al., 2021).

## 2.4 Joint Imaging Platform

Our developed solution for FL in medical environments is based on the Joint Imaging Platform for Federated Clinical Data Analytics (JIP) (Scherer et al., 2020). The JIP provides a distributed infrastructure for image analysis and ML across the sites of the strategic initiative German Cancer Consortium[12] (DKTK). The platform was developed and is maintained by the German Cancer Research Center. In addition to the DKTK sites, the JIP is also up and running in the majority of university clinics across Germany. The standardized infrastructure offers great potential for federated collaboration. It builds the foundation for our FL solutions presented in Section 3 (see page 11).

The technological backbone consists of five components that build the general architecture: JIP SYSTEM, JIP BASE, JIP STORE, JIP META, and JIP FLOW. All components are based on open-source technologies, which makes the JIP a publicly available open-source project itself[13]. The system was designed in such a way that it is easy to run in a protected hospital IT infrastructure. The platform is operated from a graphical user interface (GUI) in the browser and protected by a single sign-on. Figure 1 provides an overview of the architecture and used technology stack of the JIP.

[11] https://github.com/gkaissis/PriMIA
[12] https://dktk.dkfz.de/en
[13] https://github.com/kaapana/kaapana

The key components for implementing FL are MinIO[14], Apache Airflow[15] in combination with Kubernetes[16], and Helm[17]. MinIO is a broadly used high performance object storage which comes with a powerful API. To organize the hosted objects, MinIO uses buckets, which are similar to folders or directories of a file system. Airflow allows automatic parallel execution of workflows across the underlying computing cluster. These workflows in Airflow are formally defined as *Directed Acyclic Graphs* (DAGs). Each node of a DAG represents an operator, carrying out one specific task of the workflow. To do this, either a python script or a Docker[18] container is launched and executed. The successive tasks represent the logical sequence for FL with the JIP. Helm is a package manager for Kubernetes, which organizes packages containing everything needed to run an application, tool, or service inside a Kubernetes cluster.

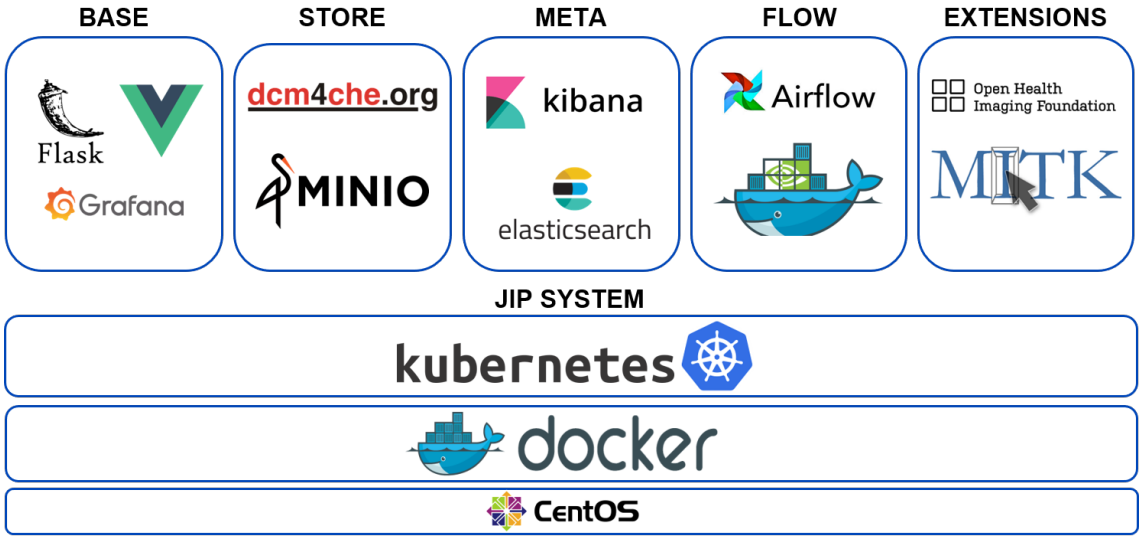


Figure 1: Architecture and technology stack of the JIP, replicated from Scherer et al. (2020)

[14] https://min.io/
[15] https://airflow.apache.org/
[16] https://kubernetes.io/
[17] https://helm.sh/
[18] https://www.docker.com/

# 3 Methods

## 3.1 Requirements & Comparison of FL Solutions

Implementations of FL solutions in a real-world medical environment need to fulfill certain requirements. These are needed in order to utilize FL solutions in practical scenarios, meaning training a model across actual clinics. In this work, we derived these requirements by reviewing the related work, in particular synthetic and real-world implementations.

In addition, we add further requirements assessed from the experience of three domain experts in the field of FL and DL on medical image data. These requirements are considered by the experts to be relevant for the successful implementation of FL in a medical environment.

Based on the identified requirements, we compare multiple FL solutions, including our own proposed solution based on the JIP. Besides NVIDIA Clara Train and PySyft, which have already been used in scientific publications (see Section 2.3 on page 7), we also include the other FL solutions introduced in the related work (see Section 2.2 on page 6). These solutions were already examined in earlier scientific publications (Li et al., 2019a; He et al., 2020), have a significant number of GitHub stars, and have an active repository with current contributions. Custom implementations are not considered further for comparison due to their lack of scalability. In order to gather the relevant information, we have intensively studied the provided documentation as well as tutorials and examples, if provided.

## 3.2 PySyft Integration into the JIP

FL experiments can be conducted in various constellations. In order to realize our implementations, we had to make assumptions first. We assume that there are participating clinics and a central instance, each of which hosts its own JIP installation. The communication always takes place via the central instance and not directly between the participants. This is, in particular, also the case with sequential compute plans. We assume that the data scientist who wants to conduct a federated experiment operates from the central instance.

The integration of PySyft is limited to its release version 0.2.9 because more recent versions do not yet support the local deployment of the computing instance. This means that they can only be deployed via a public cloud service, which excludes the more recent PySyft versions from the usage in a secured medical environment. PySyft comes with three components. *PySyft-Notebook* is a jupyter-notebook environment with all required packages installed. It runs on the central instance and

serves the data scientist as the development environment for his experiment. *PySyft-Grid* serves as the network to manage the communication. It can be accessed by the data scientist and searched for available data. *PySyft-Node* is the component that hosts the data, provides references to it in the *PySyft-Grid*, and performs the training computations. A node can be registered in a *PySyft-Grid*, which then enables the data scientist to search and work with the provided references to the data hosted by the node. The hosted data can be marked with identification tags.

All three components are provided by Openmined as ready-to-use container images and are easy to be integrated into the JIP without major adjustments due to the underlying Kubernetes cluster. To open the communication between the different nodes specific ports were excluded from the single sign-on.

For the experiments in this work, the *PySyft-Node* was integrated into an Airflow DAG as an operator. Together with an operator to load the data into Airflow and another operator to provide the data, they form the DAG in the participating clinics. The data-providing operator performs the image transformations and then sends them all to the *PySyft-Node*. After a given time, the operators stop, and the data is not provided in the *PySyft-Grid* any further. The described workflow is illustrated as DAG in Figure 2 C.

*PySyft-Grid* and *PySyft-Notebook* are hosted on the central instance. From the notebook, it is then possible to search for the relevant data within *PySyft-Grid* using the assigned identification tags. The received references to the data on the participating clinics can then be used as usual in PyTorch to implement the training logic.

## 3.3 JIP Federated

As with the JIP's PySyft integration, we assume a setting of multiple participating clinics and one central instance, all having their individual JIP installed. The data scientist implements and runs the experiment from the central instance. The DAG on the central instance is illustrated in Figure 2 A, the DAG for local training on the participants in Figure 2 B.

We implemented JIP Federated in such way that it is primarily based on the two JIP components Apache Airflow and the object store MinIO as well as their corresponding APIs. Also here, some ports and URLs were whitelisted to enable a communication between the nodes. Different from the PySyft integration, the federated training is scheduled by a DAG on the central instance instead of a jupyter-notebook. An API call to Airflow is used to start the central DAG and thus the federated experiment. With the API call, relevant parameters such as the number of

federated rounds, corresponding epochs per round, and learning rate are provided.

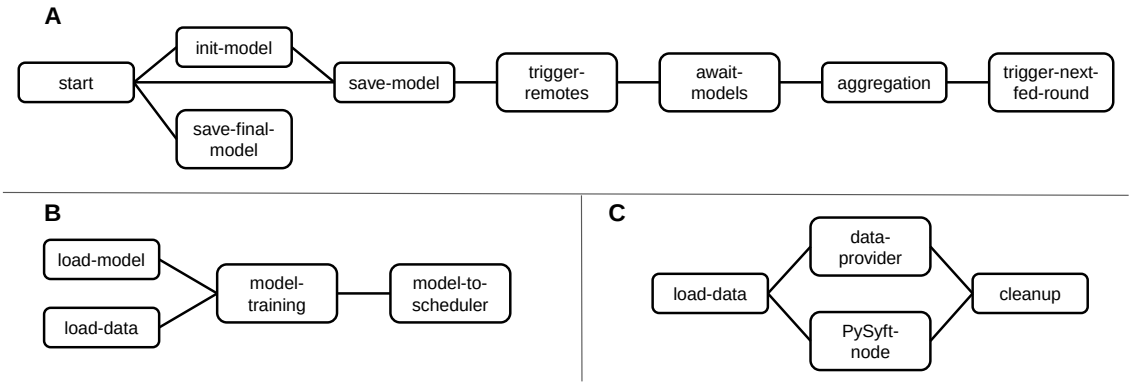


Figure 2: DAGs for JIP Federated (A & B) and the JIP's PySyft integration (C). (A) DAG on central instance, which schedules the federated experiment and its individual process steps. (B) DAG on the participating institutions, which loads the local data and the consensus model from the central instance to execute the model training. When training is completed, the updated model is saved back to the central instance. (C) DAG on a participating instance loading and providing its locally available data to the PySyft-Node, which then hosts references to the data into the grid and executes the computations during training. Note that the central instance of the PySyft integration hosts its PySyft-Notebook and PySyft-Grid without being integrated into a DAG.

The DAG on the central instance controls the processes of the experiment and can be described as follows (see Figure 2 A). First, depending on the current round, either a new model is initialized or the model of the previous round is directly stored into the object store. In case the last round has finished, the final model is saved, and the experiment is completed. Otherwise, it continues with the next operator. After saving the model, API calls are triggered on the participants' sites in order to start the corresponding training DAGs there. The relevant training information is also passed on to the participants via the API call. Subsequently, the workflow waits for the participants to send the models back to the central instance's object store. A regular check is made via the object store's API to see whether all awaited models have arrived. When this is the case, the following operator performs the model aggregation and constructs the consensus model. Before completing the current workflow, the DAG triggers itself for another run. Required data and information, such as the current federated round, are passed on to the next DAG. Thus, one federated round is represented by one run of the DAG on the central instance.

At each participating clinic, identical training DAGs are implemented (see Figure 2 B). The DAG starts with loading the data into Airflow. In parallel, a second operator accesses the object store of the central instance and loads the model from the previous round directly from there. The operator to train the model brings the

loaded data and model together and executes the training. After the local training is completed, the last operator saves the model back into the object store on the central instance.

Due to the modularity of the DAGs, the individual operators can be reused for other experiments. In most cases, a developer only needs to customize the experiment-specific operators for model aggregation and model training on the nodes to implement a new experiment.

## 3.4 Development

In the following, the implementations of the JIP's PySyft integration and JIP Federated are described in detail. The implementations are publicly available on GitHub embedded in the JIP project—the JIP's PySyft integration can be found on the branch *feature/openmined*[19], JIP Federated on the branch *feature/federated-training*[20].

A central operator is the *LocalMinioOperator*[21]. It serves as an interface between the Airflow environment and the object-store MinIO. The operator can be used for applying the operations *get*, *remove*, and *put* on the buckets and available files. Across both implementations and all experiments, it is used, for instance, to load data into the Airflow environment or to load and store models.

### 3.4.1 PySyft Integration into the JIP

The three containerized PySyft components (PySyft-Notebook, PySyft-Grid, PySyft-Notebook) are embedded into the JIP as Helm charts[22]. This allows to install them as JIP extension via the JIP's GUI. *PySyft-Grid* and *PySyft-Notebook* are installed as such extensions on the central instance. On the participants, the *PySyft-Node* is integrated into a DAG[23] which is depicted in Figure 3. In addition to an operator to load the data into the Airflow environment, the following three operators are implemented for this DAG.

[19] `https://github.com/kaapana/kaapana/tree/feature/openmined`
commit hash: aab58dac046f22d1b67ee2b49d9f4c72f7b0b69d

[20] `https://github.com/kaapana/kaapana/tree/feature/federated-training`
commit hash: 1da26c0dfbe53d3a092762bd8e8a9e2e564148ef

[21] `https://github.com/kaapana/kaapana/blob/develop/workflows/airflow-components/plugins/kaapana/operators/LocalMinioOperator.py`

[22] `https://github.com/kaapana/kaapana/tree/feature/openmined/services/applications/openmined`

[23] `https://github.com/kaapana/kaapana/blob/feature/openmined/workflows/airflow-components/dags/dag_openmined_provide_data.py`

*RunNodeOperator* is implemented as *Application-Operator*. These operators can launch applications via the Helm API, similar to manually installing an extension. By running this operator, the *PySyft-Node* is started as a temporary application in the JIP. This is done in parallel with the operator described next.

*ProvideDataOperator* starts a Docker container[24], mounting the directory containing the previously loaded data. When the container is launched, it executes its process script. Meaning that the available data is loaded, transformed, tagged, and then sent to the *PySyft-Node*. A parameter can be set for how long the data should be provided into the node.

When the *ProvideDataOperator* is completed, the subsequent *ShutdownNodeOperator* is launched to call the Helm API to uninstall *PySyft-Node* chart. From this point on, the node is unsubscribed from the *PySyft-Grid* and its data is not hosted any further. As a last step, the used Airflow directories are emptied.

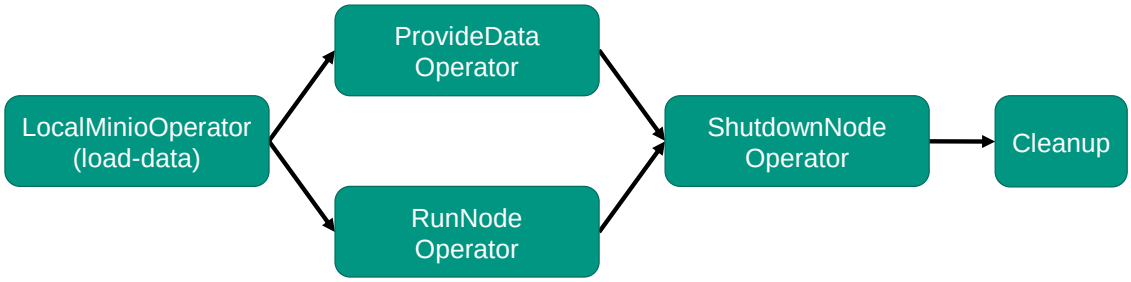


Figure 3: Detailed DAG of the JIP's PySyft integration for providing locally available data of participating institutes

The operators used for the JIP's PySyft integration are can be found in the corresponding repository[25]. Exemplary experiment-notebooks are made available together with the Docker image of the *PySyft-Notebook* component[26].

### 3.4.2 JIP Federated

As described in Section 3.3 (see page 12), each experiment using JIP Federated is based on two DAGs[27]—one implemented on the central instance and the second on the participating institutions. Figure 4 provides a detailed illustration of the

[24] https://github.com/kaapana/kaapana/tree/feature/openmined/workflows/processing-container/openmined/provide-data

[25] https://github.com/kaapana/kaapana/tree/feature/openmined/workflows/airflow-components/dags/openmined

[26] https://github.com/kaapana/kaapana/tree/feature/openmined/workflows/processing-container/openmined/provide-data

[27] https://github.com/kaapana/kaapana/tree/feature/federated-training/workflows/airflow-components/dags

two DAGs. In the following, the implemented operators are described in detail—the source-code is publicly available[28]. The same applies to the corresponding processing containers[29]. Note that depending on whether *aggregation* or *sequential training* is chosen as compute plan, the behavior of the operators slightly differ.

*EntrypointOperator* is the first operator being executed when the DAG is started. Given the parameter provided by the API call of the data scientist or the previous DAG-run, it determines the next step. For the first federated round, the model is initialized. In case the training is completed, the next step is to save the final model and finalize the workflow. Otherwise, the training continues and the model from the previous round is saved to MinIO. The routing is realized by the *KaapanaBranchPythonBaseOperator* which branches the workflow depending on the information received from the *EntrypointOperator*.

*ExperimentOperator* is used at two points in the DAG. First, it initializes the model in the first round. Further, it is used for aggregation of models received from the participants. When a *sequential* compute plan is applied, the model is simply loaded and directly saved again into the target directory. To decide between the tasks to be executed, corresponding parameters are passed when starting the operator. While all other operators in this DAG are generic, the *ExperimentOperator* is experiment-specific. The corresponding operators for each of the conducted experiments are also available in the repository and can be identified by their names (i.e. `ExperimentsBraTSOperator.py`).

*TriggerRemoteWorkersOperator* triggers the DAGs on the participating institutes. For each participant, an API call is executed which contains the required information and parameters, such as current federated round, number of epochs to train locally, and learning rate to use. In addition, information about the central instance are provided so that an allocation can be done. Note that no model is actively sent by the central instance itself. In parallel, a *LocalMinioOperator* empties the model cache in MinIO, because the new models of the participants are expected there by the next operator.

*AwaitingModelsOperator* is used to keep the workflow waiting for the models to be received back. It uses the API of MinIO to check every three seconds whether the models of the participants have already arrived in the model cache (which has been emptied before to guarantee that no models from earlier rounds are used). As soon as all newly updated models are available, the loop ends and the operator completed its task.

[28] `https://github.com/kaapana/kaapana/tree/feature/federated-training/workflows/airflow-components/dags/federated_training`

[29] `https://github.com/kaapana/kaapana/tree/feature/federated-training/workflows/processing-container/federated-training`

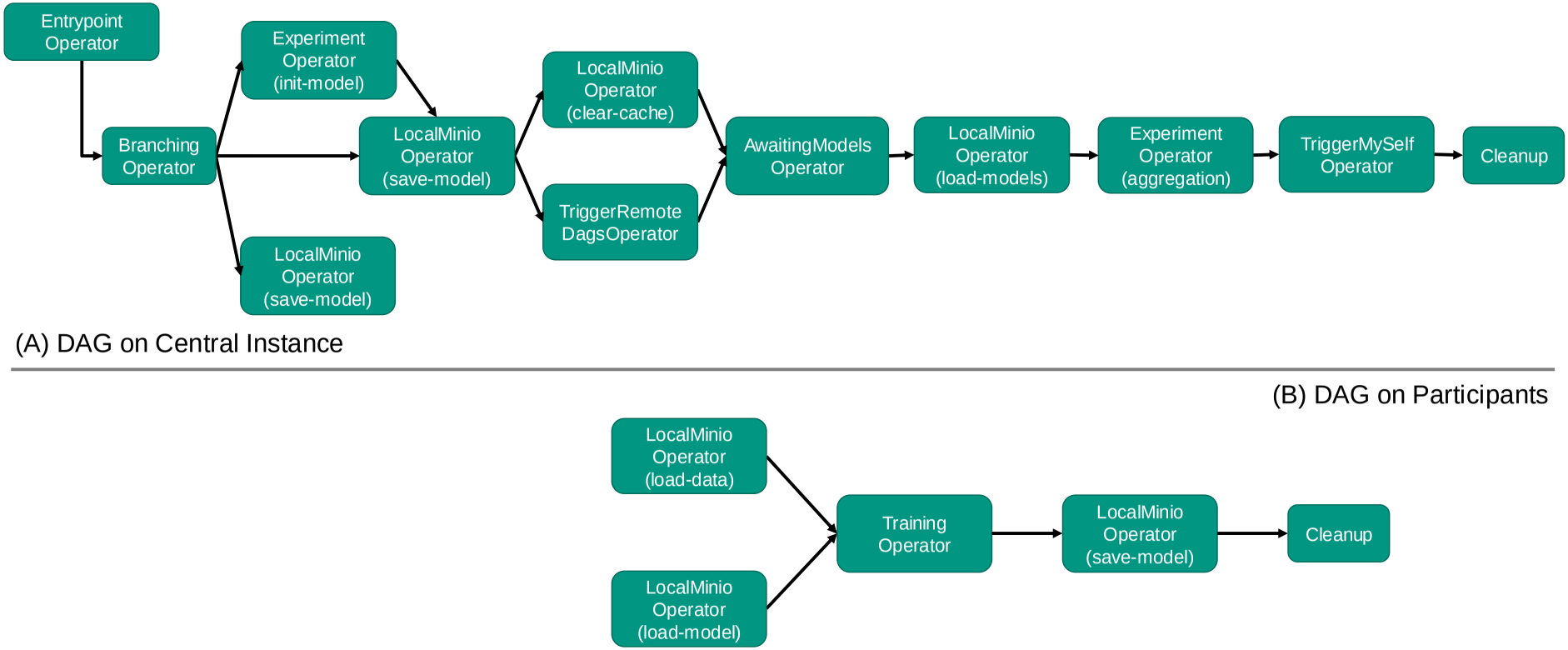


Figure 4: DAGs for JIP Federated. (A) DAG on the central instance, scheduling the overall workflow. (B) DAG implemented on the participating instances to perform the model training on locally available data.

*TriggerMyselfOperator* is the final operator of the DAG. It serves for launching the next DAG-run. For this, it transfers the required data, such as the current model checkpoint, into a newly created folder for the following DAG run. The operator also applies a new API call based on the previous call and therefore triggers the next round. In case of a *sequential* compute plan, this operator determines which participant is next to receive the model for training. In addition, the operator manages the incrementing of the performed federated rounds depending on the applied compute plan.

For the model training on the participating institutes, the following operators are used for the corresponding DAG. When the DAG is externally triggered by the central instance' *TriggerRemoteWorkersOperator*, two *LocalMinioOperator* are started to load the locally available data and the model into the Airflow environment. Note that the model is loaded directly from the object store of the central instance—this actually implies that the model is not sent to but pulled by the participants.

*TrainingOperator* launches a Docker container which accesses the loaded data and model. The container executes a processing script which is similar to a regular PyTorch experiment. Data transformations are applied, and the model is trained for the given number of epochs. As before, these experiment-specific operators can be identified by their name (i.e. `TrainingBraTSOperator.py`).

When the training is completed, another *LocalMinioOperator* is used to store the updated model again directly into the object store of the central instance. This completes the DAG on the participant. Depending on the configuration, checkpoints of the model can also be saved locally on the participant.

## 3.5 Experiments, Algorithms & Datasets

Two classification experiments were carried out to compare the JIP's PySyft integration and JIP Federated. These experiments aim to compare the runtime and thus the efficiency of the two solutions while achieving similar learning behavior and performance. Additionally, a segmentation experiment on complex multi-modal medical images was conducted to demonstrate the capabilities JIP Federated. We demonstrate that it is feasible to perform FL with the JIP on a standard benchmark dataset of medical brain tumor scans. For all experiments, we assume that three participating institutes provide their local data. Furthermore, in every federated round one epoch is trained on the available data.

The runtime experiments were conducted on two different publicly available datasets: the widely known Modified National Institute of Standards and Technology[30] (MNIST)

[30] `http://yann.lecun.com/exdb/mnist/`

(LeCun et al., 2010) dataset, consisting of 70,000 digit handwriting images (60,000 for training and 10,000 for testing), and the paediatric pneumonia dataset[31] originally proposed by Kermany et al. (2018). It contains paediatric chest radiographs split into 5,232 samples for training and 624 for testing. The data is classified as either normal (with no signs of infection, 1,349 samples) or infected (bacterial or viral pneumonia, 3.883 samples). The test data contains 234 normal samples and 390 samples classified as infected. In order to use the datasets in a federated setting, non-overlapping subsets were formed. Each class in the training data was randomly divided into three parts and distributed among the participants. The test data was held out for final global testing after training. This process was applied on both datasets.

Before the samples of the two datasets were used for training, image transformations were applied. While the MNIST images were only normalized, the following transformations were performed on the pneumonia images. Image resizing to $256 \times 256$ pixels, random horizontal flipping, random vertical flipping, and image normalization.

For the MNIST experiments, the algorithm and training parameters were adopted from the basic MNIST example of PyTorch[32]. Training was performed for 14 federated rounds using a batch size of 64 and a learning rate of $10^{-2}$. For the experiment on pneumonia radiographs, the ResNet18 (He et al., 2016) architecture pre-trained on ImageNet was used, following Kaissis et al. (2021). The model was trained for 40 federated rounds with a learning rate of $10^{-4}$ and a batch size of 8 (also used by the baseline for the processing time experiments in Kaissis et al. (2021)). Due to PySyft's limitation of not supporting momentum terms in the integrated release version, Stochastic Gradient Descent (SGD) was used as optimizer.

To measure the performance of the trained models on the test data, the metric *Accuracy* was used. Its value represents the proportion of the correctly classified samples over all samples. The metric is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

where TP = True Positive, FP = False Positive, TN = True Negative, FN = False Negative. In addition, the area-under-the-receiver-operator-characteristic (AUROC) is measured for the experiments on the slightly imbalanced pneunomia dataset. The metric is a performance metric based on the receiver-operator-characteristic (ROC) curve, which represents possible combinations of true-positive-rate (TPR) and the false-postive-rate (FPR). By mapping the FPR on the x-axis and the TPR on the

[31] `https://data.mendeley.com/datasets/rscbjbr9sj/3`

[32] `https://github.com/pytorch/examples/blob/master/mnist/main.py`; commit hash: 0f0c9131ca5c79d1332dce1f4c06fe942fbdc665

y-axis, a curve is formed. The area underneath this curve is the AUROC and its value ranges between 0 and 1 (higher is better).

Furthermore, all JIP instances for the experiments were set up identically, and training was carried out on the central processing unit (CPU). Computing resources of 10,000 Mi[33] were allocated to each node, respectively training container, by the underlying Kubernetes.

In the federated segmentation experiment with JIP Federated, a U-Net (Ronneberger et al., 2015) was trained to segment diagnosed gliomas in MRI scans of human brains. The U-Net architecture was set up with a channel sequence of $(16, 32, 64, 128, 256)$ and convolution strides of size two. The number of residual units was set to two. The results and training behavior is compared to an experiment where an identical U-Net architecture was trained on the centralized data using the same hyperparameter configuration. In both cases, the models were trained for 500 epochs, respectively federated rounds. Each federated round consisted of one local epoch on each of the participants. The model training computations were performed on graphical processing units (GPUs). The brain dataset from the *Medical Segmentation Decathlon*[34] was used. The dataset consists of a subset of the Brain Tumor Segmentation (BraTS) (Menze et al., 2015) Challenge 2016 and 2017 data (Antonelli et al., 2021). Besides unlabeled test samples, it contains 484 multi-modal MRI scans and their corresponding labels. 96 of the labeled samples were separated to serve as test data for measuring the model performance. For the experiment, three participating sites were established on individual server instances. Accordingly, the data was randomly sampled and divided into three non-overlapping subsets. Each instance was provided with 129 samples for training and 32 for testing. For centralized training, the identical data was merged into central datasets and then used for training and testing. In the remainder of this work, this data will be referred to as BraTS dataset.

To train the model with this data, we used the Adam optimizer (Kingma and Ba, 2015) with a learning rate of $10^{-4}$ and weight decay of $10^{-5}$. For the model training, the brain scans undergo the following transformations. Random vertical flipping, random spacial cropping (with roi_size $= 128 \times 128 \times 64$), intensity normalization, random intensity shifting, and random intensity scaling. The parameters of the optimizer, the applied transformations, and the U-Net architecture were adopted from the 3D brain segmentation tutorial[35] of the Medical Open Network for AI (MONAI) Project[36]. After each federated round, the models are aggregated by applying *Federated Averaging*.

[33] 1 *Mi* (*mebibyte*) $= 1024 \times 1024$ *bytes*

[34] `http://medicaldecathlon.com/`

[35] `https://github.com/Project-MONAI/tutorials/blob/master/3d_segmentation/brats_segmentation_3d.ipynb`; commit hash: d340613cea8b0f10b780694c636c1d7ff8f7658d

[36] `https://monai.io/`

When models are aggregated into a consensus model, the question of how to handle the used optimizer arises. For this *optimizer state treatment*, three methods are apparent. As with the model, a consensus optimizer state could be derived from averaging the states of the participants. Alternatively, the optimizer state could be kept on the local instance only or reset for each new federated round. Based on the results shown in the work of Li et al. (2019b), we decided to also reset the optimizer state before continuing a new federated round.

As in all BraTS challenges, the *Dice Coefficient* (DC) was used for the quantitative performance evaluation of the model. This metric measures the overlap of two binary segmentation masks and ranges from 0 to 1 (higher is better). DC is defined as:

$$DC = \frac{2|P \cap T|}{|P| + |T|} \tag{3.2}$$

where P is the prediction and T the ground truth mask. The DC is measured for all three (overlapping) tumor regions[37]:

- Enhancing tumor (ET): label 2
- Tumor core (TC): labels 2 and 3
- Whole tumor (WT): labels 1, 2 and 3

Table 2 summarizes the chosen parameter configurations for all conducted experiments. Note that apart from the listed parameters, the Adam optimizer was used with its PyTorch default values for the segmentation experiment.

| Experiment | Federated Rounds | Epochs per Round | Optimizer | Batch Size | Learning Rate | Weight Decay |
|---|---|---|---|---|---|---|
| MNIST | 14 | 1 | SGD | 64 | $10^{-2}$ | - |
| Pneunomia | 40 | 1 | SGD | 8 | $10^{-4}$ | - |
| BraTS | 500 | 1 | Adam | 2 | $10^{-4}$ | $10^{-5}$ |

Table 2: Training parameters for conducted FL experiments on MNIST, pneumonia and BraTS data

[37] In the medical segmentation decathlon, the label convention slightly differs from BraTS.

# 4 Results

## 4.1 Requirements & Comparison of FL Solutions

Table 3 summarizes the results of our requirement analysis and the comparison of the FL solutions considered in this work. The identified requirements are described in more detail in the following. Please note that entries marked with a hyphen could not be determined with full confidence by the author team[38]. The two entries marked by additional brackets are explained in Section 5.1 (see page 27).

From an FL perspective, the following requirements are derived. While applying *model aggregation* is the most common compute plan, others such as *sequential training* should be realizable in the solution (Li et al., 2019b; Chang et al., 2018). We therefore consider the ability to implement at least these two compute plans as a proxy for the flexibility of a solution, and include it as a requirement.

To maintain the data privacy and security of communication and thus the motivation behind FL, the solutions should provide *privacy mechanisms* to address potential attacks and information leakage. While there might be further ones, we include the following mechanisms retrieved from Kaissis et al. (2020) in our comparison: differential privacy (**?**), homomorphic encryption (Acar et al., 2018), and secure multi-party computation (Zhao et al., 2019).

Further, there are technical requirements that are practically important for FL solutions in the context of medical imaging. Although it is possible to run computationally expensive experiments on the CPU, *GPU support* is essential. This is particularly the case when dealing with image data, and even more when object segmentation is the given task (Kaissis et al., 2021; Lee et al., 2021). Another important requirement in a clinical setting is the compatibility with medical *image data formats*, such as *Digital Imaging and Communication in Medicine* (DICOM) (Kaissis et al., 2021). In addition to the support of DICOM, it is essential for the solutions' application in clinics that they include an adapter to the *Picture Archiving and Communication System* (PACS). PACS is the system for managing and archiving medical images and is connected to the clinic's imaging hardware. Because of the healthcare environment being highly sensitive, as well as individual for each site, solutions are required that are as flexible and adaptable as possible. In addition, the lock-in effect needs to be prevented. Therefore, it is relevant whether the solution is *open-source* or proprietary from one provider. This is accompanied by an interest in the possibility of *offline installation*, meaning that the solution is installable and runnable in an environment disconnected from the internet. From a developer's perspective, it is also relevant to have *information* about the solution's implementation

[38] Not all the available resources are detailed enough to allow a reliable classification.

| Requirement | | TFF | FedML | FATE | PaddleFL | PySyft | NVIDIA Clara Train | JIP Federated |
|---|---|---|---|---|---|---|---|---|
| Open-source | | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ |
| Offline installation | | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ |
| Type of solution | Platform Solution | | | | | | ✓ | ✓ |
| | Programming Framework | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| GPU support | Single GPU | ✓ | ✓ | - | ✓ | ✗ | ✓ | ✓ |
| | Multiple GPUs | ✓ | ✓ | - | ✓ | ✗ | ✓ | ✓ |
| Data handling | Non-DICOM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | DICOM | ✗ | ✗ | ✗ | ✗ | (✓) | ✓ | ✓ |
| | PACS Connectivity | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ |
| Privacy mechanisms | Differential Privacy | (✓) | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ |
| | Homomorphic Encryption | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ | ✗ |
| | Secure Multi-Party Computation | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ |
| DL frameworks | PyTorch | ✗ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |
| | TensorFlow | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ |
| | PaddlePaddle | ✗ | ✗ | ✗ | ✓ | ✗ | - | ✓ |
| Compute plans | Aggregation | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Sequential | - | - | ✓ | ✗ | ✓ | ✓ | ✓ |
| Information | Documentation, Tutorials, & Examples | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |
| | Community | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |

Table 3: Comparison of different FL solutions based on derived requirements for real-world FL with medical image data. Note that entries marked with a hyphen could not be determined with full confidence. ✓ indicates that a requirement is fulfilled, ✗ indicates that it is not.

and usage. Accordingly, we also include the availability of detailed documentation, tutorials, and an active user and developer community into our comparison. Furthermore, we distinguish between two different *types of solutions*. It can either be a programming framework, which comes with no or few surrounding features, or an extensive platform solution.

## 4.2 Comparison of the PySyft Integration & JIP Federated

Based on the two classification experiments, we compare the JIP's PySyft integration and JIP Federated with regard to training duration. For a fair comparison, it must be ensured that both implementations result in similar training behavior and final performance.

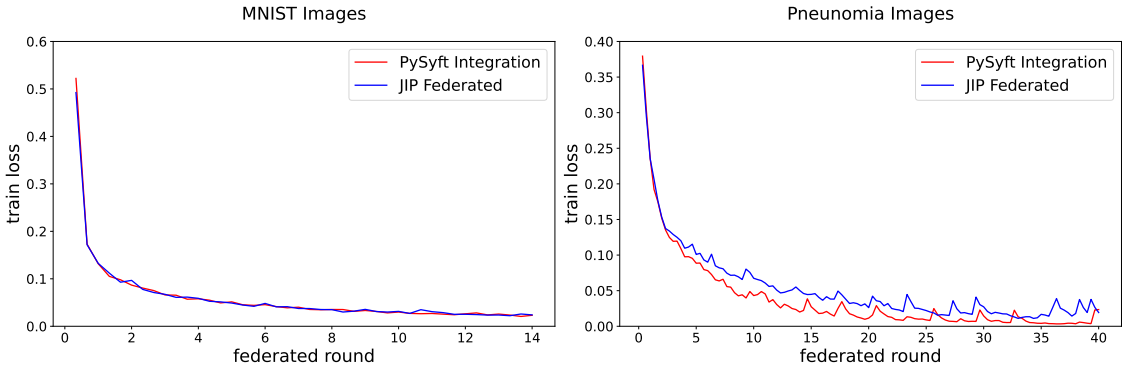


Figure 5: Loss curves during model training on MNIST and pneumonia images using the JIP's PySyft integration (red) and JIP Federated (blue)

Figure 5 illustrates the learning curves using the JIP's PySyft integration and JIP Federated when training on both datasets, MNIST and the pneumonia images. It shows the convergence when applying a *sequential compute plan*. The depicted loss values represent the average values over the batch losses of one local epoch each.

| Experiment | Accuracy | AUROC |
|---|---|---|
| PySyft on MNIST | 99.02 % | - |
| JIP Federated on MNIST | 99.20 % | - |
| PySyft on pneumonia | 91.35 % | 0.887 |
| JIP Federated on pneumonia | 92.79 % | 0.906 |

Table 4: Performance scores on MNIST and pneumonia test data (one run)

The performance of the final model checkpoints is compared using the accuracy on the held-out test data. For the experiments on the slightly unbalanced pneumonia dataset, the AUROC is considered additionally. The achieved scores are listed in Table 4.

Table 5 lists the runtime differences between starting the first federated round and saving the final model of the conducted experiment.

| Implementation | MNIST | Pneumonia Images |
|---|---|---|
| PySyft | 01:46:32 | 20:51:21 |
| JIP Federated | 01:38:50 | 08:55:03 |

Table 5: Runtime of conducted experiments on MNIST and pneunomia data (hh:mm:ss; one run)

## 4.3 JIP Federated Segmentation Experiment

The segmentation experiment on the BraTS data serves to demonstrate that JIP Federated is capable of performing complex deep learning tasks. Figure 6 shows the training loss together with the dice scores on the three available segmentation masks (ET, TC, WT) described in Section 3.5 (see page 18). In addition, the mean dice score built from the scores on the three segmentation masks is displayed. The dice scores show the performance on the test data for each federated round, respectively epoch. Each data point represents the mean value over one iteration on the corresponding data.

In order to imitate a realistic FL scenario, the test performance was computed locally on each participant using the test data available there. This results in three curves, depicted in gray, showing the local behavior. The local training loss and testing performance scores of the participating clinics were aggregated to averaged values depicted in red. As upper baseline, the curves of the model training on the centralized data are displayed in blue.

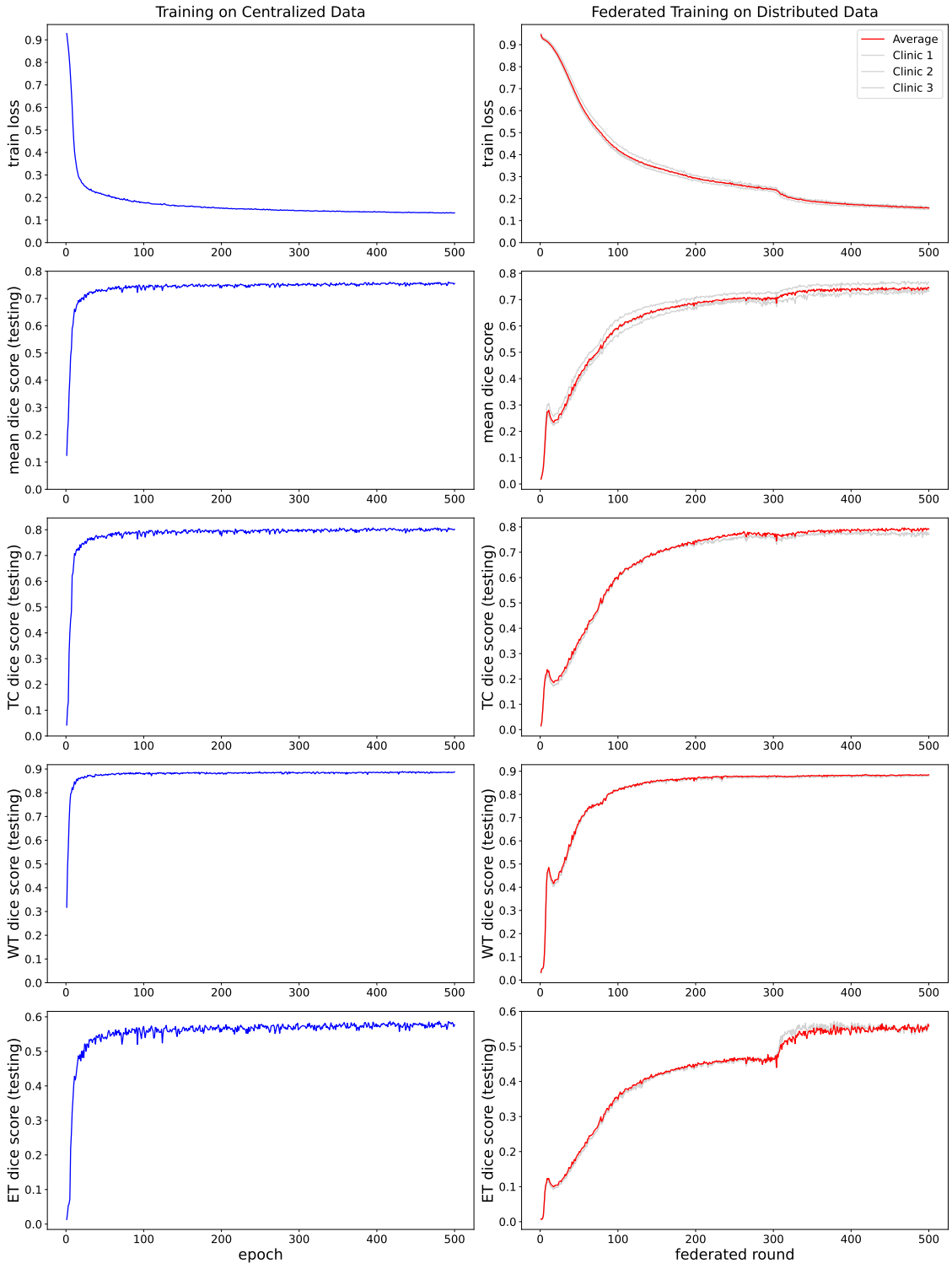


Figure 6: Training loss and testing performance of models trained on centralized data (blue) and in a federated scenario across three participating clinics (red). Curves are displayed for the three segmentation masks tumor core (TC), whole tumor (WT), enhancing tumor (ET) and their mean scores.

# 5 Discussion

## 5.1 Requirements & Comparison of FL Solutions

When examining the previously presented comparison (see Section 4.1 on page 22) of the various FL Solutions, the following can be noted. Apart from PySyft all FL solutions support GPU. This also applies to the current version of the PriMIA extension, which has been adapted for the field of medical image data. Both, NVIDIA Clara Train and JIP Federated provide an adapter for the clinical PACS, which is particularly advantageous in the field of medical image processing, while all others do not. The same applies to the support of the DICOM format, which is equally essential. Note that PriMIA extends PySyft in such a way that it supports DICOM while PySyft itself does not. Except for PySyft and PaddleFL all solutions come with extensive information. While PySyft does not yet provide a complete and detailed documentation, the resources regarding PaddleFL are not fully provided in English language. In addition, the community of PaddleFL communicates mainly in Chinese.

When comparing the privacy mechanisms, a weakness of JIP Federated becomes apparent. All other solutions support at least one of the considered mechanisms, whereas JIP Federated has not implemented any mechanism yet. Note that TFF does not natively come with differential privacy. However, it can be seamlessly added by using TensorFlow Privacy[39], which is completely interoperable with TFF. While the implementation of such mechanisms is beyond the scope of this work, it is foreseeable that this will be one of the next steps in further development. Until then, this is an obvious disadvantage of JIP Federated.

NVIDIA Clara Train and JIP Federated are both extensive platform solutions with features for the healthcare environment, and in particular for working with medical images. The differences between the two solutions are the following. While the codebase of NIVDIA Clara Federated is not publicly available, JIP Federated is fully open-source. This allows developers to extend the solution according to their needs, to understand the functionalities, and therefore trust the solution. From our perspective, all this can be seen as advantageous. Further, NVIDIA Clara Train provides the privacy mechanisms differential privacy and homomorphic encryption, which is a major advantage compared to JIP Federated, which does not support such mechanisms yet.

Please note that this categorization was done to the best of our knowledge and based on the available information. As mentioned before, it was not always straightforward to identify the relevant details in the given information.

[39] `https://github.com/tensorflow/privacy`

## 5.2 Implementation of the PySyft Integration & JIP Federated

As the PySyft components are available as containers, the integration into the JIP is seamlessly possible. This also applies to the integration as container-based operators into a DAG. Additionally, the usage of the PySyft-Notebook is not mandatory. The software library can also be installed independently of the provided container and used within a python script, for example.

The integration of PySyft clearly shows the advantages of the JIP. It highlights the flexibility and expandability of the platform, not only for PySyft but also for other potential solutions and frameworks. We showed that due to the open codebase, it is possible to combine other FL solutions with the JIP. In particular, solutions that are provided as containers can be deployed seamlessly in the JIP's Kubernetes cluster.

For JIP Federated, the key component to orchestrate the workflows within and across the central instance and the participants is Airflow. Because Airflow operators are the essential components for the workflow logic, they should be as generic and reusable as possible to minimize the development effort for a new experiment. We have managed to implement all operators, except the actual experiment operators, in such a way that they are completely transferable as they are. Therefore, a data scientist who wants to conduct his experiment in the JIP only needs to transfer his code into the two containers for model aggregation and model training on the participants. We see this as advantageous in terms of development effort.

A drawback of the JIP federated implementation is the large number of operators needed for the individual tasks. Especially, the scheduling DAG on the central instance requires numerous operators. This is necessary to implement the different steps of the workflow logic but brings an unfortunate side effect: Due to the dependencies between the individual tasks, the operators can only work in parallel to a minor extent. Since each operator needs time to be started up and shut down again, even tasks that are actually fast require more time. This idle time accumulates and adds up to the actual processing time. This disadvantage is reduced in its impact for experiments in which the actual training on the participants takes a larger share of the overall workflow time than the process on the central instance. This is particularly evident in the results of the runtime experiments listed in Table 5 (see page 25). The implementations' runtime for model training does differ only by a few minutes when using MNIST data. In contrast, PySyft requires more than twice as long as JIP Federated for the more complex task on the pneumonia data.

Further, there is a difference between the PySyft Integration and JIP Federated in terms of data handling and pre-processing. With PySyft the random transformations

are applied to the images only once. The data-providing operator then serves the pre-processed images in this form for the whole training via the PySyft node on the CPU. In contrast, JIP Federated applies new random transformations in each federated round. This results in a higher variability within the images used for training, which then becomes apparent during model training and final testing.

## 5.3 Experiments

For the runtime experiments on the MNIST and pneumonia datasets, the shape of the training curves and the performance of the resulting models are comparable. The models trained with JIP Federated perform only slighly better according to accuracy and ROC-AUC. Thus, the implementations lead to similar training behavior, and the runtime can be compared with each other. The reason for the slightly different training curves on the pneumonia images can be found in the differences in the implementations mentioned before. Applying the image transformations only once leads to a smaller variance in the training data of the PySyft experiment and, therefore, to a lower training loss. However, the resulting model generalizes slightly poorer, which explains the weaker test performance of the model trained with PySyft. The noticeable peaks during training on the pneumonia data might be due to outliers or poor data labelling.

Although the runtime experiments themselves are relatively basic, both solutions require a rather long time. However, the difference of more than seven hours when training on the pneumonia dataset demonstrates the shortcomings regarding the computing efficiency of PySyft. The comparatively long training time has already been noted by the developers and explained by the implemented privacy mechanisms (Ryffel et al., 2018). JIP Federated also requires rather a long time for the experiments, which can be explained by the already mentioned large number of required Airflow operators.

Examining the training behavior of the federated segmentation experiment, we find that the curves are very similar for all three participants. The apparent reason for this is that the BraTS data was distributed randomly and not according to its origin or some criteria that influence the imaging, such as the used hardware. Further, the curves of the dice scores have notable buckles at two points, which also can be identified in the shape of the loss curve during training. The first occurs after about ten federated rounds and the second after 300. The buckles can also be seen in the ET and TC curves, in the latter in particular the second one is very significant. The first buckle occurs strongly in all dice score curves. Furthermore, the curve of the dice score for the ET segmentation mask converges later and less stable compared to the curves for the TC and WT segmentation masks.

When comparing the federated experiment with the centralized setting, some differences can be observed as well. First, the buckles seen during the federated training do not occur in the traditional setting of training on centralized data. We suspect that this could be due to the chosen *optimizer state treatment*, but have not conducted experiments with other treatment options. Further, the model converges more slowly in a federated setting. One reason for this is probably the aggregation of the models: a simple average of model parameters cannot adequately combine the knowledge learned at the different sites. This is the usual trade-off FL faces: the convergence could be accelerated by performing fewer local training steps (moving closer to mini-batch gradient descent), but that would increase the communication frequency. What can also be noted is that in both experiments, the convergence on the TC and ET segmentation masks is less stable than on the WT segmentation masks.

Although the convergence is slower, the model trained with FL achieves a mean dice score of 0.745 after 500 epochs. This is equivalent to 98.58 % of the performance by the model trained on centralized data (mean dice score of 0.7554). Thus, it can be said that the federated trained model converges slower, but nearly reaches the upper performance baseline. In general, it can be noted that it is more difficult for the models to segment the ET region. Compared to segmenting the TC and WT regions, the dice score is significantly lower. This, of course, also considerably lowers the overall performance measured as mean dice score. In both, the federated and centralized experiment, the highest values are obtained on the WT segmentation masks. Note also that we did not perform any hyperparameter tuning for FL and just adopted the parameters that were provided in the MONAI tutorial[40], which was developed on centralized data.

[40] `https://github.com/Project-MONAI/tutorials/blob/master/3d_segmentation/brats_segmentation_3d.ipynb`; commit hash: d340613cea8b0f10b780694c636c1d7ff8f7658d

# 6 Conclusion & Outlook

## 6.1 Conclusion

In order to use deep learning models in clinics, they must be robust and generalize sufficiently. One way to achieve this is to train these models on very large datasets. However, annotated image data in the healthcare domain is rare and expensive—and is subject to privacy aspects. Federated learning is a promising approach to address these challenges. But first, technical solutions are needed to enable the application of federated learning in the healthcare environment.

We showed which federated learning solutions exist and what is needed to bring federated learning into medical institutions. There is not yet one solution that meets all requirements, but many promising projects could be identified. The platform solutions NVIDIA Clara Train and JIP Federated, the latter of which is presented within this article, provide features that are advantageous for the application in the healthcare environment, in particular for the usage of medical images.

We demonstrated that the Joint Imaging Platform is a promising solution, either in combination with an existing federated learning framework or as a stand-alone solution itself. With the seamless integration of PySyft, the high flexibility of the platform was demonstrated, so the possibility to combine other solutions with the Joint Imaging Platform's infrastructure and the resulting benefits are given. Further, we designed a JIP-only federated learning solution—JIP Federated. By using JIP Federated for the segmentation of gliomas in brain MRI scans, we demonstrated that it is a capable federated learning solution for complex medical task. Its implementation is highly flexible, easy to apply on other federated learning experiments, and brings further advantages through its platform features. Thus, we provide a comprehensive open-source solution to conduct real-world federated learning in the domain of medical image computing.

The Joint Imaging Platform is an established and trusted tool in the healthcare community and is already being used in a multitude of German clinics and beyond. As proved by the experiments, JIP Federated can be a solution to make federated learning real-world. Future work should focus on using JIP Federated for experiments across geographically distributed clinics on large datasets that cannot be shared directly. A further aspect needs to be the extension of the Joint Imaging Platform to provide privacy mechanisms such as Differential Privacy, Homomorphic Encryption, and Secure Multi-Party Computing as additional security layers for the communication between instances. With this work on federated learning with the Joint Imaging Platform, we also hope to give an incentive for further federated experiments using JIP Federated.

## 6.2 Outlook

Medical image analysis has come a long way from rule-based systems and hand-crafted features for early computer vision to modern techniques. Nowadays, deep learning is the technology of choice in computer vision and therefore also for medical images. In recent time, it has proved its capabilities and potential on a multitude of tasks in the domain of medical images. Deep learning is currently and will be in the future the technology for processing and analyzing medical images. However, its full potential can only be unlocked if the appropriate volume of data is available to train such deep architectures.

Federated learning provides a solution to overcome the challenge of not having a sufficient amount of data. If the gap between simulation and real application can be closed by developing comprehensive technical solutions, enormous volumes of data will be available for usage. All around the world, respective data is collected on a daily basis, but until now it has not been available for traditional machine learning settings, also for reasons of patients privacy and regulations. The data would be accessible and usable by means of technical solutions for federated learning across geographically distributed institutions.

Using this data would result in among others two consequences. The developed models can achieve significantly higher performance on numerous tasks, which is in particular important in a critical field like healthcare where life and death can be at stake. Further, these models will be able to generalize better because of their extensive knowledge base. That is, they are more capable to adapt to previously unseen samples which do not fall into the distribution of the samples used for training. All of this would be possible, because systems are available that have gained experience from the data of millions of patients and the knowledge of numerous medical experts who labeled the data. Such comprehensive systems can contribute to improve healthcare, individualize patient treatment, and to better understand or even cure severe and rare diseases. Further, models with high performance and the ability to generalize sufficiently could even be used as a form of decision support in regions of the world with poor or hardly any advanced healthcare services.

Certainly, many things remain to be done to bring the technical solutions to the point where federated learning projects truly go across medical institutions, but first steps are being taken. The healthcare sector remains very sensitive, highly regulated and also political, but there are more than just a few researches and practitioners who see federated learning as the future of digital health (Rieke et al., 2020).

# References

Acar, A., Aksu, H., Uluagac, A. S., and Conti, M. (2018). A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys*, 51(4):1–35.

Andreux, M., Ogier Du Terrail, J., Beguier, C., and Tramel, E. W. (2020). Siloed Federated Learning for Multi-Centric Histopathology Datasets. In *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*, pages 129–139.

Antonelli, M., Reinke, A., Bakas, S., Farahani, K., AnnetteKopp-Schneider, Landman, B. A., Litjens, G., Menze, B., Ronneberger, O., Summers, R. M., van Ginneken, B., Bilello, M., Bilic, P., Christ, P. F., Do, R. K. G., Gollub, M. J., Heckers, S. H., Huisman, H., Jarnagin, W. R., McHugo, M. K., Napel, S., Pernicka, J. S. G., Rhode, K., Tobon-Gomez, C., Vorontsov, E., Huisman, H., Meakin, J. A., Ourselin, S., Wiesenfarth, M., Arbelaez, P., Bae, B., Chen, S., Daza, L., Feng, J., He, B., Isensee, F., Ji, Y., Jia, F., Kim, N., Kim, I., Merhof, D., Pai, A., Park, B., Perslev, M., Rezaiifar, R., Rippel, O., Sarasua, I., Shen, W., Son, J., Wachinger, C., Wang, L., Wang, Y., Xia, Y., Xu, D., Xu, Z., Zheng, Y., Simpson, A. L., Maier-Hein, L., and Cardoso, M. J. (2021). The Medical Segmentation Decathlon. *arXiv preprint arXiv:2106.05735*.

Baheti, P., Sikka, M., Arya, K. V., and Rajesh, R. (2020). Federated learning on distributed medical records for detection of lung nodules. In *VISIGRAPP 2020 - Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, volume 4, pages 445–451.

Balachandar, N., Chang, K., Kalpathy-Cramer, J., and Rubin, D. L. (2020). Accounting for data variability in multi-institutional distributed deep learning for medical imaging. *Journal of the American Medical Informatics Association*, 27(5):700–708.

Brendan McMahan, H., Moore, E., Ramage, D., Hampson, S., and Agüera y Arcas, B. (2017). Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pages 1273–1282.

Chang, K., Balachandar, N., Lam, C., Yi, D., Brown, J., Beers, A., Rosen, B., Rubin, D. L., and Kalpathy-Cramer, J. (2018). Distributed deep learning networks among institutions for medical imaging. *Journal of the American Medical Informatics Association*, 25(8):945–954.

Choudhury, O., Park, Y., Salonidis, T., Gkoulalas-Divanis, A., Sylla, I., and Das, A. K. (2019). Predicting Adverse Drug Reactions on Distributed Health Data using Federated Learning. In *AMIA ... Annual Symposium proceedings. AMIA Symposium*, pages 313–322.

De Fauw, J., Ledsam, J. R., Romera-Paredes, B., Nikolov, S., Tomasev, N., Blackwell, S., Askham, H., Glorot, X., ODonoghue, B., Visentin, D., van den Driessche, G., Lakshminarayanan, B., Meyer, C., Mackinder, F., Bouton, S., Ayoub, K., Chopra, R., King, D., Karthikesalingam, A., Hughes, C. O., Raine, R., Hughes, J., Sim, D. A., Egan, C., Tufail, A., Montgomery, H., Hassabis, D., Rees, G., Back, T., Khaw, P. T., Suleyman, M., Cornebise, J., Keane, P. A., and Ronneberger, O. (2018). Clinically applicable deep learning for diagnosis and referral in retinal disease. *Nature Medicine*, 24(9):1342–1350.

Deng, J., Dong, W., Socher, R., Li, L.-J., Kai Li, and Li Fei-Fei (2010). ImageNet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255.

Devlin, J., Chang, M. W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dluhoš, P., Schwarz, D., Cahn, W., van Haren, N., Kahn, R., Španiel, F., Horáček, J., Kašpárek, T., and Schnack, H. (2017). Multi-center machine learning in imaging psychiatry: A meta-model approach. *NeuroImage*, 155:10–24.

Dou, Q., So, T. Y., Jiang, M., Liu, Q., Vardhanabhuti, V., Kaissis, G., Li, Z., Si, W., Lee, H. H., Yu, K., Feng, Z., Dong, L., Burian, E., Jungmann, F., Braren, R., Makowski, M., Kainz, B., Rueckert, D., Glocker, B., Yu, S. C., and Heng, P. A. (2021). Federated deep learning for detecting COVID-19 lung abnormalities in CT: a privacy-preserving multinational validation study. *npj Digital Medicine*, 4(60).

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118.

European Commission (2018). General Data Protection Regulation (GDPR).

Feki, I., Ammar, S., Kessentini, Y., and Muhammad, K. (2021). Federated learning for COVID-19 screening from Chest X-ray images. *Applied Soft Computing*, 106.

Flores, M., Dayan, I., Roth, H., Zhong, A., Harouni, A., Gentili, A., Abidin, A., Liu, A., Mount, A. C., Health, S., Wood, S. B., Tsai, C.-S., and Hsu, C.-N. (2021). Fed-

erated Learning used for predicting outcomes in SARS-COV-2 patients. *PubMed preprint PMID: 33442676*.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135.

He, C., Li, S., So, J., Zeng, X., Zhang, M., Wang, H., Wang, X., Vepakomma, P., Singh, A., Qiu, H., Zhu, X., Wang, J., Shen, L., Zhao, P., Kang, Y., Liu, Y., Raskar, R., Yang, Q., Annavaram, M., and Avestimehr, S. (2020). FedML: A Research Library and Benchmark for Federated Machine Learning. *arXiv preprint arXiv:2007.13518*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 770–778.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A. R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

Kaissis, G., Ziller, A., Passerat-Palmbach, J., Ryffel, T., Usynin, D., Trask, A., Lima, I., Mancuso, J., Jungmann, F., Steinborn, M.-M., Saleh, A., Makowski, M., Rueckert, D., and Braren, R. (2021). End-to-end privacy preserving deep learning on multi-institutional medical imaging. *Nature Machine Intelligence*, 3:473–484.

Kaissis, G. A., Makowski, M. R., Rückert, D., and Braren, R. F. (2020). Secure, privacy-preserving and federated machine learning in medical imaging. *Nature Machine Intelligence*, 2(6):305–311.

Kermany, D. S., Goldbaum, M., Cai, W., Valentim, C. C., Liang, H., Baxter, S. L., McKeown, A., Yang, G., Wu, X., Yan, F., Dong, J., Prasadha, M. K., Pei, J., Ting, M., Zhu, J., Li, C., Hewett, S., Dong, J., Ziyar, I., Shi, A., Zhang, R., Zheng, L., Hou, R., Shi, W., Fu, X., Duan, Y., Huu, V. A., Wen, C., Zhang, E. D., Zhang, C. L., Li, O., Wang, X., Singer, M. A., Sun, X., Xu, J., Tafreshi, A., Lewis, M. A., Xia, H., and Zhang, K. (2018). Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning. *Cell*, 172(5):1122–1131.

Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR.

Kirienko, M., Sollini, M., Ninatti, G., Loiacono, D., Giacomello, E., Gozzi, N., Amigoni, F., Mainardi, L., Lanzi, P. L., and Chiti, A. (2021). Distributed learning: a reliable privacy-preserving strategy to change multicenter collaborations using AI. *European Journal of Nuclear Medicine and Molecular Imaging*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25.

Lalitha, A., Kilinc, O. C., Javidi, T., and Koushanfar, F. (2019). Peer-to-peer Federated Learning on Graphs. *arxiv preprint arXiv:1901.11173*.

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2323.

LeCun, Y., Cortes, C., and Burges, C. (2010). MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

Lee, H., Chai, Y. J., Joo, H., Lee, K., Hwang, J. Y., Kim, S. M., Kim, K., Nam, I. C., Choi, J. Y., Yu, H. W., Lee, M. C., Masuoka, H., Miyauchi, A., Lee, K. E., Kim, S., and Kong, H. J. (2021). Federated learning for thyroid ultrasound image analysis to protect personal information: Validation study in a real health care environment. *JMIR Medical Informatics*, 9(5).

Li, Q., Wen, Z., Wu, Z., Hu, S., Wang, N., Li, Y., Liu, X., and He, B. (2019a). A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection. *arXiv preprint arXiv:1907.09693*.

Li, W., Milletarì, F., Xu, D., Rieke, N., Hancox, J., Zhu, W., Baust, M., Cheng, Y., Ourselin, S., Cardoso, M. J., and Feng, A. (2019b). Privacy-preserving Federated Brain Tumour Segmentation. In *Lecture Notes in Computer Science - Machine Learning in Medical Imaging*, volume 11861, pages 133–141.

Menze, B. H., Jakab, A., Bauer, S., Kalpathy-Cramer, J., Farahani, K., Kirby, J., Burren, Y., Porz, N., Slotboom, J., Wiest, R., Lanczi, L., Gerstner, E., Weber,

M. A., Arbel, T., Avants, B. B., Ayache, N., Buendia, P., Collins, D. L., Cordier, N., Corso, J. J., Criminisi, A., Das, T., Delingette, H., Demiralp, ., Durst, C. R., Dojat, M., Doyle, S., Festa, J., Forbes, F., Geremia, E., Glocker, B., Golland, P., Guo, X., Hamamci, A., Iftekharuddin, K. M., Jena, R., John, N. M., Konukoglu, E., Lashkari, D., Mariz, J. A., Meier, R., Pereira, S., Precup, D., Price, S. J., Raviv, T. R., Reza, S. M., Ryan, M., Sarikaya, D., Schwartz, L., Shin, H. C., Shotton, J., Silva, C. A., Sousa, N., Subbanna, N. K., Szekely, G., Taylor, T. J., Thomas, O. M., Tustison, N. J., Unal, G., Vasseur, F., Wintermark, M., Ye, D. H., Zhao, L., Zhao, B., Zikic, D., Prastawa, M., Reyes, M., and Van Leemput, K. (2015). The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS). *IEEE Transactions on Medical Imaging*, 34(10):1993–2024.

Remedios, S., Roy, S., Blaber, J., Bermudez, C., Nath, V., Patel, M. B., Butman, J. A., Landman, B. A., and Pham, D. L. (2019). Distributed deep learning for robust multi-site segmentation of CT imaging after traumatic brain injury. In *Medical Imaging 2019: Image Processing*, volume 10949.

Remedios, S. W., Roy, S., Bermudez, C., Patel, M. B., Butman, J. A., Landman, B. A., and Pham, D. L. (2020). Distributed deep learning across multisite datasets for generalized CT hemorrhage segmentation. *Medical Physics*, 47(1):89–98.

Rieke, N., Hancox, J., Li, W., Milletarì, F., Roth, H. R., Albarqouni, S., Bakas, S., Galtier, M. N., Landman, B. A., Maier-Hein, K., Ourselin, S., Sheller, M., Summers, R. M., Trask, A., Xu, D., Baust, M., and Cardoso, M. J. (2020). The future of digital health with federated learning. *npj Digital Medicine*, 3(1).

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Lecture Notes in Computer Science - Medical Image Computing and Computer-Assisted Intervention*, volume 9351, pages 234–241.

Roth, H. R., Chang, K., Singh, P., Neumark, N., Li, W., Gupta, V., Gupta, S., Qu, L., Ihsani, A., Bizzo, B. C., Wen, Y., Buch, V., Shah, M., Kitamura, F., Mendonça, M., Lavor, V., Harouni, A., Compas, C., Tetreault, J., Dogra, P., Cheng, Y., Erdal, S., White, R., Hashemian, B., Schultz, T., Zhang, M., McCarthy, A., Yun, B. M., Sharaf, E., Hoebel, K. V., Patel, J. B., Chen, B., Ko, S., Leibovitz, E., Pisano, E. D., Coombs, L., Xu, D., Dreyer, K. J., Dayan, I., Naidu, R. C., Flores, M., Rubin, D., and Kalpathy-Cramer, J. (2020). Federated Learning for Breast Density Classification: A Real-World Implementation. In *Lecture Notes in Computer Science - Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*, volume 12444, pages 181–191.

Roy, A. G., Siddiqui, S., Pölsterl, S., Navab, N., and Wachinger, C. (2019). BrainTorrent: A Peer-to-Peer Environment for Decentralized Federated Learning. *arxiv preprint arXiv:1905.06731*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Ryffel, T., Trask, A., Dahl, M., Wagner, B., Mancuso, J., Rueckert, D., and Passerat-Palmbach, J. (2018). A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*.

Sarma, K. V., Harmon, S., Sanford, T., Roth, H. R., Xu, Z., Tetreault, J., Xu, D., Flores, M. G., Raman, A. G., Kulkarni, R., Wood, B. J., Choyke, P. L., Priester, A. M., Marks, L. S., Raman, S. S., Enzmann, D., Turkbey, B., Speier, W., and Arnold, C. W. (2021). Federated learning improves site performance in multicenter deep learning without data sharing. *Journal of the American Medical Informatics Association*, 28(6):1259–1264.

Scherer, J., Nolden, M., Kleesiek, J., Metzger, J., Kades, K., Schneider, V., Bach, M., Sedlaczek, O., Bucher, A. M., Vogl, T. J., Grünwald, F., Kühn, J.-P., Hoffmann, R.-T., Kotzerke, J., Bethge, O., Schimmöller, L., Antoch, G., Müller, H.-W., Daul, A., Nikolaou, K., la Fougère, C., Kunz, W. G., Ingrisch, M., Schachtner, B., Ricke, J., Bartenstein, P., Nensa, F., Radbruch, A., Umutlu, L., Forsting, M., Seifert, R., Herrmann, K., Mayer, P., Kauczor, H.-U., Penzkofer, T., Hamm, B., Brenner, W., Kloeckner, R., Düber, C., Schreckenberger, M., Braren, R., Kaissis, G., Makowski, M., Eiber, M., Gafita, A., Trager, R., Weber, W. A., Neubauer, J., Reisert, M., Bock, M., Bamberg, F., Hennig, J., Meyer, P. T., Ruf, J., Haberkorn, U., Schoenberg, S. O., Kuder, T., Neher, P., Floca, R., Schlemmer, H.-P., and Maier-Hein, K. (2020). Joint Imaging Platform for Federated Clinical Data Analytics. *JCO Clinical Cancer Informatics*, 4:1027–1038.

Sharma, P., Shamout, F. E., and Clifton, D. A. (2019). Preserving Patient Privacy while Training a Predictive Model of In-hospital Mortality. In *Proceedings of the Conference on Health, Inference, and Learning*, pages 25–35.

Sheller, M. J., Edwards, B., Reina, G. A., Martin, J., Pati, S., Kotrotsou, A., Milchenko, M., Xu, W., Marcus, D., Colen, R. R., and Bakas, S. (2020). Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. *Scientific Reports*, 10.

Sheller, M. J., Reina, G. A., Edwards, B., Martin, J., and Bakas, S. (2019). Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation. In *Lecture Notes in Computer Science* -

*Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, volume 11383, pages 92–104. Springer Verlag.

US Department of Health and Human Services (2020). Health Insurance Portability and Accountability Act (HIPPA).

Van Panhuis, W. G., Paul, P., Emerson, C., Grefenstette, J., Wilder, R., Herbst, A. J., Heymann, D., and Burke, D. S. (2014). A systematic review of barriers to data sharing in public health. *BMC Public Health*, 14(1):1144.

Vapnik, V. N. (1995). *The nature of statistical learning theory*. Springer-Verlag, Berlin.

Wang, P., Shen, C., Roth, H. R., Yang, D., Xu, D., Oda, M., Misawa, K., Chen, P.-T., Liu, K.-L., Liao, W.-C., Wang, W., and Mori, K. (2020). Automated Pancreas Segmentation Using Multi-institutional Collaborative Deep Learning. In *Lecture Notes in Computer Science - Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*, pages 192–200.

Xu, J., Glicksberg, B. S., Su, C., Walker, P., Bian, J., and Wang, F. (2021). Federated Learning for Healthcare Informatics. *Journal of Healthcare Informatics Research*, 5(1):1–19.

Xu, Y., Ma, L., Yang, F., Chen, Y., Ma, K., Yang, J., Yang, X., Chen, Y., Shu, C., Fan, Z., Gan, J., Zou, X., Huang, R., Zhang, C., Liu, X., Tu, D., Xu, C., Zhang, W., Yang, D., Wang, M. W., Wang, X., Xie, X., Leng, H., Holalkere, N., Halin, N. J., Kamel, I. R., Wu, J., Peng, X., Wang, X., Shao, J., Mongkolwat, P., Zhang, J., Rubin, D. L., Wang, G., Zheng, C., Li, Z., Bai, X., and Xia, T. (2020). A collaborative online AI engine for CT-based COVID-19 diagnosis. *medRxiv preprint PMID: 32511484*.

Yan, Z., Wicaksana, J., Wang, Z., Yang, X., and Cheng, K. T. (2020). Variation-Aware Federated Learning with Multi-Source Decentralized Medical Image Data. *IEEE Journal of Biomedical and Health Informatics*, 25:2615 – 2628.

Yang, D., Xu, Z., Li, W., Myronenko, A., Roth, H. R., Harmon, S., Xu, S., Turkbey, B., Turkbey, E., Wang, X., Zhu, W., Carrafiello, G., Patella, F., Cariati, M., Obinata, H., Mori, H., Tamura, K., An, P., Wood, B. J., and Xu, D. (2021). Federated semi-supervised learning for COVID region segmentation in chest CT using multi-national data from China, Italy, Japan. *Medical Image Analysis*, 70:101992.

Zhao, C., Zhao, S., Zhao, M., Chen, Z., Gao, C. Z., Li, H., and Tan, Y. a. (2019). Secure Multi-Party Computation: Theory, practice and applications. *Information Sciences*, 476:357–372.