



UNIVERSITÄT
DES
SAARLANDES

Course of study
Computer Science

Bachelor Thesis

– Title –

Asynchronous Federated Learning for Medical
Images: Enabling Privacy and User Interaction

submitted by

Tim Maurer

s8timaur@stud.uni-saarland.de

2566243

Saarbrücken,
January 21, 2025

Advisors

Abdulrahman Mohamed, M.Sc.

Hasan Md Tusfiquir Alam, M.Sc.

Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI)

Saarland Informatics Campus

Saarbrücken, Germany

Reviewers

Prof. Dr.-Ing. Daniel Sonntag

Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI)

Saarland Informatics Campus

Saarbrücken, Germany

Prof. Dr. Antonio Krüger

Ubiquitous Media Technology Lab, Universität des Saarlandes

Saarland Informatics Campus

Saarbrücken, Germany

Saarland University

Faculty MI – Mathematics and Computer Science

Department of Computer Science

Campus - Building E1.1

66123 Saarbrücken

Germany

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____
(Datum/Date)

(Unterschrift/Signature)

Erklärung

Ich erkläre hiermit, dass die vorliegende Arbeit mit der elektronischen Version übereinstimmt.

Statement

I hereby confirm the congruence of the contents of the printed data and the electronic version of the thesis.

Saarbrücken,-----
(Datum/Date)

(Unterschrift / Signature)

Acknowledgements

I would like to thank my supervisors, Prof. Dr. Antonio Krüger and Prof. Dr.-Ing. Daniel Sonntag, for providing me the opportunity to write this thesis within their groups in an exciting field of study. I could not have undertaken this journey without my great advisors, Abdulrahman Mohamed and Hasan Md Tusfiquir Alam. Many thanks to Abdulrahman for guiding me through the process from start to finish and always giving me valuable advice and constructive criticism along the way. Thank you, Hasan, for sharing your expertise and providing insightful feedback to enhance the quality of my thesis. Special thanks should also go to Matthias, who not only introduced me to Abdulrahman at the perfect time but also helped me develop the initial concept for my thesis. Lastly, I would also like to thank Marta, for her constant motivation and support. Her understanding and patience, especially during the long hours I dedicated to this thesis, have been invaluable.

Abstract

Centralized machine learning often faces challenges related to data availability and the limited involvement of expert users. In this study, we aim to explore approaches that enable users to contribute data and train a shared model collaboratively across multiple devices while maintaining data privacy. Federated Learning (FL) offers a decentralized solution for such training processes, but synchronous FL algorithms are not ideal for interactive settings, as they require simultaneous participation from all users. Instead, we explore asynchronous federated learning using the state-of-the-art FedBuff algorithm, which allows each user to train independently. Our work focuses on applying FedBuff to the classification of Optical Coherence Tomography (OCT) retina images within the medical domain. We compare its performance against traditional centralized models and the widely used synchronous algorithm, FedAvg, which has demonstrated effectiveness with this data type. Furthermore, we develop a browser-based proof-of-concept application using modern web technologies to examine the challenges and limitations associated with implementing a collaborative and interactive machine learning approach.

Our findings indicate that FedBuff is a viable approach for handling OCT image data in relatively simple classification tasks; however, its performance declines in more complex scenarios, such as multi-class classification problems, necessitating further investigation. Consistent with previous studies, we confirm that FedAvg achieves performance comparable to centralized models for OCT data. While FedBuff demonstrates lower performance compared to both FedAvg and centralized models, it still achieves acceptable results. Additionally, during the development of our proof-of-concept application, we identified several limitations related to web browsers and the technologies used, as many of these technologies are still in their developmental stages.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 2 | Related Work | 4 |
| 2.1 | Federated Learning | 4 |
| 2.2 | FL for Medical Images | 5 |
| 2.3 | Asynchronous FL | 5 |
| 2.4 | FL for OCT Retina Images | 8 |
| 2.5 | Browser-based DL / FL Systems | 9 |
| 3 | Methods | 10 |
| 3.1 | Training Procedure | 10 |
| 3.2 | Evaluation Metrics | 11 |
| 3.3 | Models | 11 |
| 3.4 | Centralized Learning | 12 |
| 3.5 | Federated Learning Simulations | 12 |
| 3.6 | Proof-of-Concept Application | 13 |
| 3.6.1 | Technologies | 13 |
| 3.6.2 | Application Design | 13 |
| 4 | Experiments | 17 |
| 4.1 | Data | 17 |
| 4.2 | Models | 18 |
| 4.3 | Centralized Learning | 18 |
| 4.4 | Federated Learning Simulations | 19 |
| 5 | Results | 21 |
| 5.1 | Centralized Training | 21 |
| 5.2 | Federated Learning Simulations | 22 |
| 6 | Discussion | 28 |
| 7 | Conclusion | 32 |
| 7.1 | Future Work | 33 |
| 7.2 | Code Availability | 34 |

1 | Introduction

In machine learning (ML), traditional, centralized training approaches can encounter difficulties when data availability is limited. Additionally, domain experts without technical ML knowledge are not directly involved in the training process, therefore delaying the availability of new training data. These challenges are identified by Fails and Olsen (2003), who introduce interactive machine learning (IML) to deal with these issues. Their approach allows non-technical users to train an ML model with their own data, either by manual classification or by correcting the output of the model. We aim to combine IML with a collaborative approach, i.e. multiple users training a shared model on their devices, each with their own dataset. Tseng et al. (2023) implement this concept with *Co-ML*, providing a tablet-based application where users can collaboratively and interactively build ML image classification models across multiple devices. While the goal of their approach is teaching dataset design practices, we focus on enabling this collaborative and interactive approach for the real-world use case of medical images, while preserving the privacy of the participating institutions and individuals that provide the training data.

Usually, training an ML model on a centralized dataset is the most common approach. However, the data for an ML task may be distributed on several devices without the possibility of sharing it, for example, in the medical domain due to privacy concerns. Federated learning (FL) (McMahan et al., 2017) enables decentralized training of a shared ML model while the data stays on the participating clients' devices. This is achieved by aggregating locally computed updates to the model on a central server. FL has already been proven to be viable for many use cases in medical imaging (Lee et al., 2021; Haggemüller et al., 2024; Islam et al., 2022). Several review and survey papers also confirm that FL is a promising approach for this use case (Moshawrab et al., 2023; Guan et al., 2024).

In FL, there are two main kinds of algorithms: synchronous and asynchronous. In synchronous FL (McMahan et al., 2017), all participating clients train simultaneously after receiving the latest global model, orchestrated by the central server. The server then waits for all client updates before incorporating them into the global model, which can lead to delays, caused by slow clients. In asynchronous FL (Xie et al., 2019), the clients train and send updates independently, without orchestration by the central server. Each update is applied to the global model at the time of arrival. This leads to higher efficiency in settings with varying update frequency. However, the asynchronicity may lead to stale updates that are based on an older version of the global model. Asynchronous FL can have several benefits, including flexibility in participation for the clients, as well as better scalability in scenarios with a large number of collaborating devices. When choosing an algorithm for a collaborative and interactive approach for medical images, several requirements need to be considered in terms of

client flexibility, additional privacy features, and suitability for image classification tasks.

FedBuff (Nguyen et al., 2022) is a promising asynchronous FL algorithm that satisfies these requirements. Instead of adding each client update to the global model immediately, they are aggregated into a buffer on arrival. The buffer is applied to the global model after several updates have been collected. FedBuff is compatible with secure aggregation for hiding individual client updates, as well as differential privacy clipping and noise addition for protection against inference attacks. It outperforms synchronous FL in terms of efficiency with low privacy settings and matches it with higher privacy settings. The algorithm was tested on various use cases, including two image classification tasks with various privacy budgets.

This work focuses on the diagnosis of diseases based on retina images obtained through optical coherence tomography (OCT) (Kadir et al., 2023, 2024). OCT is a non-invasive imaging technique using light waves to capture images from within optical scattering media (Huang et al., 1991). It is widely used in medical imaging, especially ophthalmology. FL has previously been successfully applied and tested on OCT image data (Lo et al., 2021; Ran et al., 2023; Gholami et al., 2023). The most common algorithm used is FedAvg (McMahan et al., 2017), while the best-performing model architectures are VGG19 (Simonyan and Zisserman, 2015) and ResNet-18 (He et al., 2015). The methods employed include both transfer learning and training the model from scratch. The experiments across these publications show that the FL model performance is similar to the models trained from centralized data. The application of asynchronous FL for medical images, especially OCT scans, is an under-explored area in the research. This work empirically investigates the performance of asynchronous FL on a recent OCT image dataset by running simulations. Results are compared between different FL algorithms and centralized training, as well as different model architectures and use cases. For the experiments, the recently published *Optical Coherence Tomography Dataset for Image-Based Deep Learning Methods* (OCTDL) (Kulyabin et al., 2024) is used. It contains over 2000 images in seven different classes, including 1231 images for age related macular degeneration (AMD), the same use case Gholami et al. (2023) studied in a FL context.

In their survey about FL for medical images, Guan et al. (2024) stated that "*FL methods, while promising, can encounter difficulties during real-world implementation, such as compatibility with existing hospital systems, integration challenges, and user adoption hurdles*". These challenges can be approached by a browser-based, interactive FL system. Compatibility and integration are simplified with a web application running in the browser, since it is the most ubiquitous platform on computer systems. An interactive front-end application can improve user adoption by allowing non-ML experts to provide and label new training data.

Deep learning (DL) in the browser has been made feasible by various JavaScript-based deep learning frameworks (Ma et al., 2019), such as *tensorflow.js* or *keras.js*. For browser-based deep learning, Google developed *Teachable Machine*, a no-code ML web application (Carney et al., 2020). The user provides images from their device via a user interface and then starts the training process. Once the training is finished, the model can be tested and exported. However, this approach focuses only on local training without further interaction or FL integration. For browser-based FL, frameworks based on both synchronous and asynchronous FL are proposed (Lian et al., 2022; Ángel Morell and Alba, 2022). In these approaches, the user would simply provide the training data in a folder on their device, also without additional interaction. This work proposes a browser-based, interactive FL system. It highlights the

challenges and limitations of previous works and presents a proof-of-concept (POC) implementation based on modern web technologies.

Our first contribution is applying asynchronous FL to OCT retina images using FedBuff. To the best of our knowledge, asynchronous FL has not been tested on this kind of data. The next contribution lies in comparing FedBuff to centralized learning and FedAvg, two methods that are established on OCT data in the literature. Notably, FedAvg has not been applied yet to the dataset we are using for the experiments. Furthermore, we developed a proof-of-concept application that shows how a browser-based FL application can be implemented using modern web technologies, enabling a secure way to train a model collaboratively. The application makes use of asynchronous FL to enable users to conduct the training process on their own behalf. During the development process, we gain insights into the challenges and limitations of such an application. These contributions aim to answer the following research questions.

- RQ1:** How viable is an asynchronous FL approach to optimize machine learning-based medical diagnosis using OCT retina images?
- RQ2:** How does asynchronous FL perform in terms of diagnostic accuracy for OCT retina images compared to established centralized and synchronous FL methods?
- RQ3:** What are the challenges and limitations of implementing FL in a browser-based environment, and how can they be addressed?

2 | Related Work

This section provides a detailed overview of the related work. First, federated learning and its applications on various medical imaging tasks are presented. Then, asynchronous federated learning and its state-of-the-art algorithm are introduced. Finally, the current advancements in FL for OCT images, as well as browser-based deep learning and FL systems, are discussed.

2.1 Federated Learning

The term *Federated Learning* was first introduced by McMahan et al. (2017). It describes a decentralized learning paradigm, which enables training a shared ML model with multiple participants (called *clients*), while the data remains on the participants' devices. In their work, the authors present *FederatedAveraging* (Algorithm 1). On the server side, a random subset of clients for the current communication round is selected. Each selected client receives the current global model and computes multiple stochastic gradient descent (Robbins and Monro, 1951; Kiefer and Wolfowitz, 1952) steps on it for a fixed number of epochs on batches of fixed size, both defined by the server. The server then aggregates all updates, once they arrive, by averaging them based on the number of data points on each client. Afterwards, the next communication round starts. This iterative process continues until the model has trained for the prescribed number of rounds.

Algorithm 1 FederatedAveraging

- 1: **Input:** Number of clients K , number of communication rounds T , learning rate η
 - 2: Initialize global model weights w_0
 - 3: **for** each round $t = 1, 2, \dots, T$ **do**
 - 4: Server sends global model weights w_t to all clients
 - 5: **for** each client $k = 1, 2, \dots, K$ **in parallel do**
 - 6: Client k downloads global model w_t
 - 7: Client k trains locally and updates weights to w_{t+1}^k
 - 8: Let n_k be the number of samples on client k
 - 9: **end for**
 - 10: Server aggregates updates from all clients: $w_{t+1} = \frac{1}{N} \sum_{k=1}^K n_k \cdot w_{t+1}^k$
 where $N = \sum_{k=1}^K n_k$ is the total number of samples across all clients
 - 11: **end for**
 - 12: **Output:** Final global model weights w_T
-

The authors extensively tested their algorithm on image classification and lan-

guage models. They found that their approach is robust to unbalanced and non-ideally distributed data. Furthermore, they highlight three properties of problems that are ideal for applying federated learning.

1. Real-world data on edge devices is better suited for training than data available in a centralized location.
2. The data is privacy sensitive or large in size.
3. The data can be labelled naturally through user interaction

2.2 FL for Medical Images

Image classification problems in the medical field satisfy all of the properties mentioned in the previous section. Therefore, it is natural that FL has been used in a variety of medical cases. These include classifying thyroid ultrasound (Lee et al., 2021) and melanoma images (Haggenmüller et al., 2024) for diagnosis, as well as identifying brain tumors in MRI scans (Islam et al., 2022). To gain an overview of this field of research, Guan et al. (2024) provide a comprehensive survey about FL for medical images, suggesting that FL appears to be useful for a large variety of tasks. These include the classification and diagnosis of diseases from images obtained by different modalities, such as X-rays or MRI scans, as well as segmentation tasks for identifying the location of a tumor. In their own experiments on a dataset for Alzheimer’s Disease, they found FL to achieve satisfactory performance compared to centralized training. Similarly, Moshawrab et al. (2023) conducted an extensive review on the usage of FL in disease prediction. The authors also identified that FL was successfully used on a variety of tasks.

Both previously mentioned works also make it clear that FL in the medical domain faces several challenges in real-world applications. Two of them are especially relevant for this work.

- Privacy and scalability need to be considered when looking at aggregation algorithms (Moshawrab et al., 2023).
- Difficulties in real-world implementations, like compatibility with existing hospital systems, integration challenges, and user adoption hurdles, need to be addressed (Guan et al., 2024).

Both of these challenges can be approached by asynchronous FL, especially the state-of-the-art *FedBuff* algorithm, which is discussed in the next section.

2.3 Asynchronous FL

Asynchronous FL, as the name suggests, does not require synchronous training among the clients. As the first algorithm of its kind, Xie et al. (2019) propose Asynchronous Federated Optimization (*FedAsync*). It functions in a fully asynchronous manner, meaning that each arriving model update from a client results in a new global model. That way, slow clients, called *stragglers*, can no longer slow down the aggregation process, a common problem in synchronous FL. However, with this approach, client updates are potentially based on an older version of the global model, which is not a concern in the synchronous approach. Therefore, a *staleness function* needs to be

used in order to assign a lower weight to *stale* updates when incorporating them into the global model. Then again, such fully asynchronous algorithms have issues with privacy, since they are not compatible with secure aggregation (SecAgg) (Bonawitz et al., 2016), as individual updates are not hidden. Furthermore, there are increased communication costs caused by the higher update frequency, as each update needs to be broadcast to all clients. Asynchronous FL algorithms are used in various applications, particularly in smart transportation and smart industry, as highlighted by a recent survey paper (Xu et al., 2023).

Nguyen et al. (2022) introduce a semi-asynchronous algorithm called *FedBuff* (Algorithm 2), exploring the design space between synchronous and fully asynchronous FL. It aims to tackle the aforementioned challenges on privacy and scalability by not updating the global model with each incoming update. The server holds a buffer where client updates are stored. The global model is only updated once the buffer holds K client updates, where K is a tunable hyperparameter. This leads to less communication between the server and the clients, since the global model is updated less frequently this way. Privacy improvements are achieved in two ways: *FedBuff* can be extended with differential privacy (DP), implemented on the server side, and it is compatible with SecAgg (Bonawitz et al., 2016), as by choosing $K > 1$, individual updates are hidden in the aggregate. For DP, clipping can be applied to each client update once received, before adding it to the buffer. Then, before the buffered updates are applied to the global model, noise can be added to the aggregate.

Numerous other works use *FedBuff* as the asynchronous FL algorithm for comparison, or use it as a base for their own novel asynchronous FL algorithms (Islamov et al., 2024; Su and Li, 2022; Zhang et al., 2023; Leconte et al., 2024). Thus, *FedBuff* appears to be the state-of-the-art asynchronous FL algorithm and provides a good starting point for exploring asynchronous FL for medical images.

Apart from the presented algorithms, there are other more recent approaches to asynchronous FL. *FedASMU* (Liu et al., 2024) addresses the common issues by implementing dynamic staleness-aware updates. During local training, clients periodically request the latest global model from the server to reduce staleness. Then, the server dynamically adjusts the weight of each client’s update using a function that considers the staleness and the client’s local loss. However, the algorithm requires the server to trigger local training on the client devices. *FedFix* (Vidal and Kamani, 2024) uses a semi-asynchronous strategy similar to *FedBuff*. Instead of aggregating a fixed number of client updates, it aggregates client contributions based on a fixed update interval and dynamically adjusts weights for staleness. Still, *FedBuff* is the algorithm of choice for the use case presented in this work. Table 2.1 provides an overview of the described algorithms in this section and evaluates the following requirements for each of them:

- **Client flexibility:** Each participating client should be able to train and send updates independently of the central server and other clients.
- **Tested with image data:** The algorithm should have been tested and validated on one or more image datasets.
- **Tested with DP:** Differential privacy measures should be supported and be part of the testing process.
- **Easy implementation:** For the purposes of this work’s proof-of-concept application, the algorithm should be comparably easy to implement.

Algorithm 2 FedBuff

Server

```

1: Input: Server learning rate  $\eta_g$ , buffer size  $K$ , client SGD steps  $Q$ 
2: Output: Federated trained global model
3: Initialize buffer  $k \leftarrow 0$ ,  $\Delta_t \leftarrow 0$ 
4: repeat
5:   Sample available clients  $c$  ▷ async
6:   Run FedBuff-client( $w^t, \eta_\ell, Q$ ) on client  $c$  ▷ async
7:   if client update received then
8:      $\Delta_i \leftarrow$  received client update
9:      $\Delta_t \leftarrow \Delta_t + \Delta_i$ 
10:     $k \leftarrow k + 1$ 
11:   end if
12:   if  $k == K$  then
13:     Aggregate updates:  $\Delta_t \leftarrow \Delta_t / K$ 
14:     Update global model:  $w^{t+1} \leftarrow w^t - \eta_g \Delta_t$ 
15:     Reset buffer:  $\Delta_t \leftarrow 0$ ,  $k \leftarrow 0$ 
16:   end if
17:    $t \leftarrow t + 1$ 
18: until Convergence

```

Client

```

1: Input: Server model  $w$ , client learning rate  $\eta_\ell$ , client SGD steps  $Q$ 
2: Output: Client update  $\Delta$ 
3:  $y_0 \leftarrow w$ 
4: for  $q = 1$  to  $Q$  do
5:    $y_q \leftarrow y_{q-1} - \eta_\ell \nabla g_q(y_{q-1})$ 
6: end for
7:  $\Delta \leftarrow y_0 - y_Q$ 
8: Send  $\Delta$  to server

```

| Algorithm | Client flexibility | Tested with image data | Tested with DP | Easy implementation |
|---------------------------------|--------------------|------------------------|----------------|---------------------|
| FedAsync (Xie et al., 2019) | ✓ | ✓ | x | ✓ |
| FedBuff (Nguyen et al., 2022) | ✓ | ✓ | ✓ | ✓ |
| FedASMU (Liu et al., 2024) | x | ✓ | x | x |
| FedFix (Vidal and Kameni, 2024) | ✓ | ✓ | x | ✓ |

Table 2.1: Comparison between different asynchronous FL algorithms.

The key factor that contributed to choosing FedBuff is the way it can be extended with differential privacy. In theory, local DP (Truex et al., 2020) can be applied to any FL algorithm, where clients apply the necessary operations to their update before sending it to the server. However, FedBuff applies DP on the server side, reducing the workload of clients. More importantly, the authors explicitly verify in their experiments that FedBuff performs well across different privacy budgets. For this reason, FedAsync and FedFix were not chosen, even though they satisfy the other requirements. FedASMU provides a new approach to deal with stale client updates, which is one of the biggest problems in asynchronous FL. Unfortunately, the training process in FedASMU is triggered by the central server, making it unusable for an interactive approach. In addition, the implementation is more difficult due to the higher number of calculations and input parameters in the algorithm.

2.4 FL for OCT Retina Images

Several works explore FL on OCT retina images for various use cases. Lo et al. (2021) apply FL to referable diabetic retinopathy (RDR) classification. Diabetic retinopathy (DR) is a diabetes complication that can lead to vision loss; referable DR involves only moderate or more severe cases of DR. They use OCT data from two universities and 700 eye scans in total. The dataset is split into two classes, RDR and non-RDR. A balanced distribution is achieved via random upsampling, and augmentations are used to artificially increase the dataset size. The authors use transfer learning with a VGG19 (Simonyan and Zisserman, 2015) architecture, pre-trained on ImageNet (Deng et al., 2009). For evaluation, each institution trained an ML model on their internal dataset. This model was then evaluated on internal validation data and on the external data of the other institution. The experiments found that FL slightly outperforms the internal and external models in terms of diagnostic accuracy.

A study on a larger scale was conducted by Ran et al. (2023), who focus on FL for glaucoma detection. Glaucoma refers to a group of diseases that can lead to vision loss and blindness by damaging the optic nerve. Their work involves a multi-centre study across seven institutions, totaling 9326 OCT scans from 2785 individuals. The authors use DenseNet121 (Huang et al., 2017) and ResNet10/18 (He et al., 2015) models, without pre-training. Results from their experiments show that FL performs similarly to the traditional, centralized model while significantly outperforming the individual models trained on each institution’s data.

Gholami et al. (2023) test FL on age-related macular degeneration (AMD), using three public OCT research datasets. AMD occurs when the macula, a part of the retina, is damaged. It causes vision loss, mostly among people of age 50 or older. The data is split into two categories, normal and AMD, to create a binary classification task. The authors do not use transfer learning and train the models from scratch. As they

use publicly available research datasets, they could also train a centralized model on the entirety of the data. They found that synchronous FL, using a ResNet18 (He et al., 2015) model architecture, achieves similar performance to the centralized model. Data augmentation is also used here to artificially increase the sample size.

2.5 Browser-based DL / FL Systems

Deep learning in the web browser has already been made feasible by various JavaScript frameworks. Ma et al. (2019) compared their performance against TensorFlow (Abadi et al., 2015), the state-of-the-art native ML framework, and found that for some tasks, the JavaScript frameworks can reach comparable performance.

A no-code, browser-based ML application, called *Teachable Machine* (Carney et al., 2020), is proposed and implemented by Google, helping people with no technical ML knowledge to train a model for their use case. The tool supports building image and sound classifiers using user-provided data that can be uploaded from the user’s device through an intuitive interface. It uses TensorFlow’s JavaScript implementation for on-device inference and training. The base model for image classification is a pre-trained mobilenet (Howard et al., 2017), which is used for transfer learning. Fifteen percent of the provided data is used for validation. Teachable Machine is dedicated to a single user experience, without FL integration.

For browser-based synchronous FL, Lian et al. (2022) present their framework *WebFed*. It implements synchronous FL on the server-side and communicates with the clients via WebSockets. The clients use local DP to add noise to their update before sending it to the server. Experiments showed that their approach could reach an accuracy close to conventional FL, while choosing an appropriate privacy budget only has a subtle impact on the model’s performance. With the synchronous FL approach, however, a fixed number of clients need to participate in the training process without the possibility of joining or leaving the training at any time. Also, clients need to stay online until all communication rounds are finished. The authors mention that there is a user interface for providing data, but no concrete details are given on what such an interface could look like. Ángel Morell and Alba (2022) present an asynchronous algorithm for FL in this context and implement it in their own platform. Their approach allows clients to join and leave the training process at any time, with their experiments showing that the platform can adapt well to these changes in client availability. Their work, however, does not mention support for DP and does not contain a user interface for providing the data.

3 | Methods

In this work, models trained on centralized data are compared against models from federated learning simulations in order to compare their performance. The model architectures, as well as the methods to obtain these models, are detailed in this chapter. Additionally, the implementation details for the POC application are covered.

3.1 Training Procedure

The training process for both centralized and federated learning is implemented using PyTorch (Paszke et al., 2019) and executed on a NVIDIA GeForce RTX 3080 GPU. The PyTorch base models for the architectures described in the next section are altered in the classification layer, according to the number of classes for the use case. For transfer learning, ImageNet (Deng et al., 2009) pre-trained weights are loaded into the models. Then, all layers of the model are frozen. Two additional trainable dense layers are added to learn more complex features, similar to Lo et al. (2021), before adding the final classification layer with an output equal to the number of classes in the respective use case.

The models are trained using the Adam optimizer (Kingma and Ba, 2017), as it is used by Kulyabin et al. (2024) in their publication associated with the dataset used in this work. Also, the authors apply a cross-entropy loss function in the training process. However, we utilize weighted cross-entropy loss (Aurelio et al., 2019) in order to deal with class imbalance, as during initial experiments, it has been shown to perform better than an unweighted cross-entropy loss function. For each class j , the weight w_j is calculated inversely proportional to its frequency in the training dataset:

$$w_j = \frac{T}{N \cdot C_j} \quad \text{for } C_j > 0$$

where N is the total number of classes, C_j is the number of samples in class j , and T is the total number of samples. If a class has no samples, then $w_j = 0$. This can be the case in federated learning simulations. The weights are further normalized such that $w_j \in [0, 1]$. Data augmentations are also applied similar to Kulyabin et al. (2024), including random crop, vertical and horizontal flip, rotation, translation and gaussian blur.

Finding the best hyperparameter values for all training processes is done using *Optuna* (Akiba et al., 2019). Optuna is an open-source hyperparameter optimization framework which allows for efficient exploration of the hyperparameter space by defining an objective function and dynamically selecting optimal values based on past trials. Also, pruning strategies can be used to stop unpromising trials early, which can significantly decrease the time it takes to complete the optimization process.

3.2 Evaluation Metrics

To evaluate the performance of the trained models, we used metrics that account for class imbalance. **Balanced accuracy** is used as it adjusts for class imbalance by averaging the recall of each class. **F1 score** is the harmonic mean of precision and recall and provides a good measure of the model’s accuracy, even with imbalanced datasets. **Macro-averaged F1 score** calculates the F1 score for each class individually and then averages them, such that each class contributes equally to the final score. Let N be the number of classes, TP_i , FP_i , FN_i be the true positives, false positives and false negatives for class i , respectively. Let $Precision_i = \frac{TP_i}{TP_i + FP_i}$ and $Recall_i = \frac{TP_i}{TP_i + FN_i}$, then:

$$\begin{aligned} \text{Balanced Accuracy} &= \frac{1}{N} \sum_{i=1}^N Recall_i \\ \text{F1 Score}_i &= \frac{2 \cdot Precision_i \cdot Recall_i}{Precision_i + Recall_i} \\ \text{Macro F1 Score} &= \frac{1}{N} \sum_{i=1}^N F1Score_i \end{aligned}$$

3.3 Models

ResNet (residual network) is a type of deep learning model used primarily for image recognition tasks (He et al., 2015). It can be scaled in terms of the number of layers (depth), which is denoted as a number in the model’s name. This work uses a ResNet18 model, as in the literature on FL for OCT data, it is the most commonly used one (Ran et al., 2023; Gholami et al., 2023). Through the experiments, we can determine if FedAvg performs similarly well as observed by Gholami et al. (2023) on a different dataset. Moreover, Kulyabin et al. (2024) trained a ResNet50 model on their dataset to help validate its suitability for deep learning. Therefore, it is also used in this work’s experiments in order to reproduce their results.

MobileNet architectures are a class of efficient models for vision tasks (Howard et al., 2017). As the name suggests, these are intended for resource-constrained environments, such as mobile devices. Therefore, it is the model of choice for Google’s *Teachable Machine* application (Carney et al., 2020). By this time, the newest generation is MobileNetV3 (Howard et al., 2019), which is tuned to mobile phone CPUs. As the POC application is intended for use on a desktop computer, this work uses the V2 architecture (Sandler et al., 2018) for its increased performance and efficiency compared to version 1.

Following Lo et al. (2021), transfer learning with ImageNet (Deng et al., 2009) pre-trained weights is used for both the ResNet18 and the MobileNetV2 model, with two fully connected, trainable layers added before the final classification layer. Although medical data is different from ImageNet images, it is a popular approach in the medical domain (Morid et al., 2021). Also, in a federated learning setting, it is preferable to reduce the amount of data sent between clients and the central server.

3.4 Centralized Learning

The centralized models are trained for a fixed maximum number of epochs. The training loop employs an early stopping strategy after training for a minimum number of epochs. After that, the training process is stopped if the model does not improve on the target metric for a set number of consecutive epochs. In addition, the target metric is passed to Optuna’s pruning strategy after each epoch to terminate the current trial early if the model’s performance is worse than the majority of previous trials at the same stage. After each trial, the weights of the best model among epochs are stored on the disk. Finally, the best performing model in terms of validation metrics is tested on previously unseen test data. The hyperparameters used for optimization are batch size, learning rate, whether or not to apply data augmentation, and dropout value.

3.5 Federated Learning Simulations

All federated learning simulations are implemented using *flower* (Beutel et al., 2020). The framework enables federating any machine learning workflow, allowing the use of the same training loop implemented for the centralized learning. For FedAvg, the framework’s implementation is used. FedBuff has a custom implementation in order to simulate stale clients, which is detailed in section 4.4. Additionally, two practical improvements suggested by Nguyen et al. (2022) in their publication on FedBuff are implemented. *Learning rate normalization* is used to linearly scale the learning rate, in case a client needs to train on a batch with a size smaller than the prescribed batch size. The new learning rate is then set to $lr_{new} = lr * b/B$, where lr is the current learning rate, b is the size of the current batch and B is the prescribed batch size. *Staleness scaling* deals with updates of clients based on an older version of the model. Their updates are scaled by $1/(1 + \sqrt{s})$, where s is the staleness of the update, i.e. how many versions the update is behind the current global model.

The goal is to compare the two different FL algorithms under fair conditions, which is achieved as follows. The simulations for both algorithms are run with the same number of clients active in the training process. The training duration is determined by the number of client updates arriving at the central server, similar to the experiments conducted by Nguyen et al. (2022) in their publication for FedBuff. The above mentioned metrics are compared with different update frequencies, i.e. the number of clients selected for training each round in FedAvg, and the choice of buffer size for FedBuff. For example, the number of total updates could be set to 100 and the update frequency to 10. Then FedAvg will run for

$$\frac{\#updates}{update\ frequency} = 100/10 = 10$$

communication rounds. For FedBuff, this would mean choosing buffer size 10 and running the training process until the buffer is filled 10 times.

The hyperparameters used in the optimization process are similar to the centralized learning. On the client side, each node receives the batch size, learning rate, dropout value, number of local epochs and a boolean value whether or not to apply augmentation. In addition, for FedBuff, there is also a server learning rate. For assessing the performance after each global model update, the newly aggregated weights are sent to each client for evaluation. The server collects the evaluation results and calculates their weighted average based on the number of samples of each client. During

the optimization process, the aggregated target metric is passed to Optuna after each update to the global model in order to use pruning. The resulting best model based on evaluation results is then tested on the same unseen test data used for centralized learning.

3.6 Proof-of-Concept Application

The proof-of-concept application aims to showcase how modern web technologies can be combined to enable collaborative, interactive machine learning. This section introduces these technologies and describes the application design in terms of architecture, user interface (UI) and workflow. Also, relevant implementation details are being presented.

3.6.1 Technologies

ONNX Runtime (ONNX Runtime developers, 2021) is a cross-platform machine learning library that enables fast on-device inference and training. Most importantly, the library is available for web browsers. It utilizes two modern browser API's to ensure an efficient machine learning workflow: *Web Assembly* (Rossberg, 2022) and *WebGPU* (Ninomiya et al., 2024). WebAssembly is a low-level, assembly-like language for the browser, enabling web applications to execute code with near-native performance. It can be a compilation target for many popular programming languages. With WebGPU, browsers can use the system's GPU for high-performance computations. It is the successor to WebGL ¹, offering better compatibility with modern GPUs. The implementation as a web application makes it easy to integrate with existing systems, since a web browser is present on nearly all computer systems. One big advantage of ONNX Runtime over other browser-based ML libraries, such as TensorFlow.js (Smilkov et al., 2019), is the compatibility with the ONNX (Open Neural Network Exchange) format. Since models from the most popular ML frameworks can be exported to ONNX, the application can easily work with existing models.

The POC application itself is implemented in Python using *Flask* (Grinberg, 2018), a lightweight framework for creating web applications. The interactive elements of the UI, including the ONNX Runtime integration, are implemented with JavaScript and encapsulated in *Web Components* ², a set of standardized APIs that allow for the creation of reusable custom elements. The Web Components standard works across modern browsers, making the application independent of any UI framework. One more important browser API used is *Indexed DB* ³. It offers persistent storage within the browser for large amounts of data, including files. This way, user data such as uploaded images, inference results and other metadata is available between sessions, even when the browser is closed.

3.6.2 Application Design

The architecture, which is shown in Figure 3.1, consists of two main components. The **FL Server** is responsible for receiving and aggregating updates from the individual clients. It holds the current global model and stores any metadata related to it. Upon

¹https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API (accessed 23.09.2024)

²https://developer.mozilla.org/en-US/docs/Web/API/Web_components (accessed 23.09.2024)

³https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API (accessed 23.09.2024)

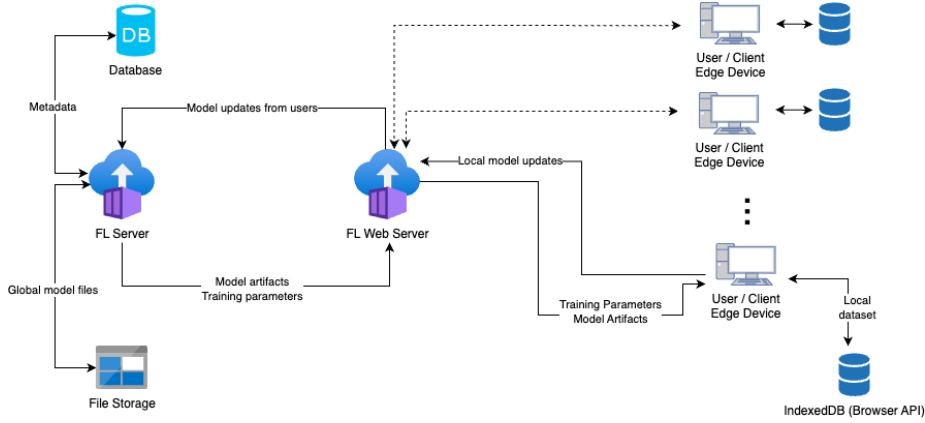


Figure 3.1: POC application: architecture overview.

request, clients are sent the current version of the model, together with the necessary files and instructions for training. The **FL Web Server** serves the web application and the necessary files to each client's edge device. While the POC application focuses on image classification, other computer vision tasks, such as object detection or image segmentation, can be implemented through the abstract classes provided by the implementation. In the context of this work, the following simple workflow is implemented.

- A user chooses one or more images. (Figure 3.2)
- The app suggests a label for each image based on the current model. (Figure 3.3)
- The user can then either accept the label or change it to a different one. (Figures 3.4 and 3.5)
- The training session can be started using the current local dataset, after reviewing all images. (Figure 3.6)
- As soon as the training is finished, the update is sent automatically to the central server for aggregation.



Figure 3.2: POC application: The user can upload images by clicking the “Select Images” button.

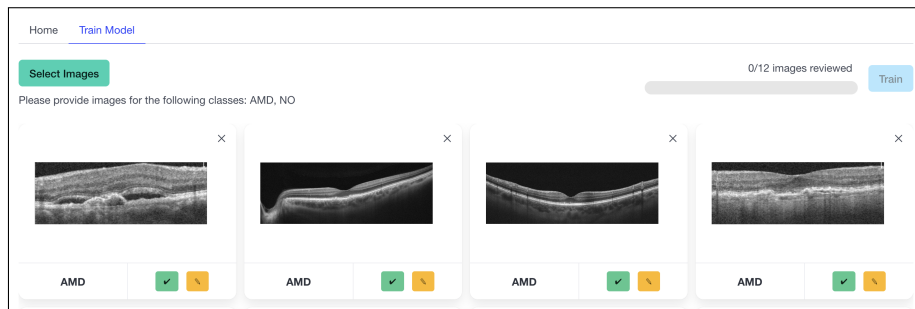


Figure 3.3: POC application: Labels are suggested after the user provides images by running inference with the current state of the model.

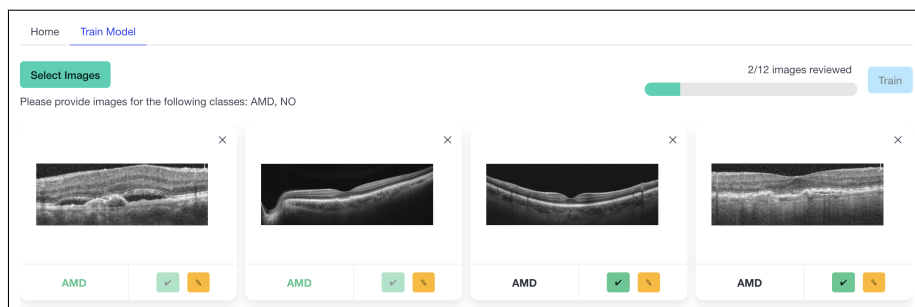


Figure 3.4: POC application: Labels can be confirmed by clicking the green button next to it. Here, the first two are confirmed and marked in green.

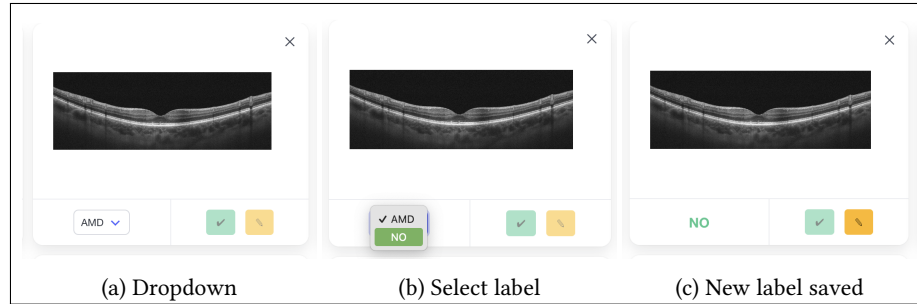


Figure 3.5: POC application: The user can change a label by clicking the yellow button. A dropdown appears where the new label can be chosen. Upon selection, the new choice is stored.

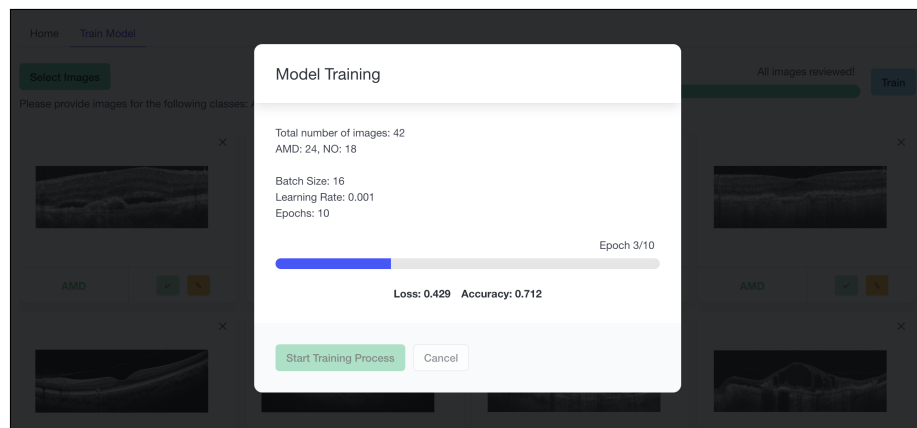


Figure 3.6: POC application: Interface during the training process.

4 | Experiments

In this chapter, the experimental setup is described in detail, showing the exact steps taken to evaluate and compare the performance of the centralized and federated learning models. First, the dataset is presented together with the strategy to distribute the data among clients in the federated learning setting, followed by the setup for the models. Afterwards, the process for obtaining the centralized base models is shown. Finally, the concrete conditions under which the federated learning setting for both algorithms is simulated are explained. For all optimizations performed, the hyperparameter ranges are specified.

During the hyperparameter optimization process for each experiment, the macro F1 score for the validation set is used to determine the best performing model, as F1 score is one of the most widely used metrics in deep learning applications for medical imaging (Anaya-Isaza et al., 2021).

4.1 Data

For the experiments conducted in this work, the *OCTDL* (Kulyabin et al., 2024) (Optical Coherence Tomography Dataset for Image-Based Deep Learning Methods) is utilized. The dataset contains over 2000 OCT images, each labeled according to specific retinal diseases: Age-related Macular Degeneration (AMD), Diabetic Macular Edema (DME), Epiretinal Membrane (ERM), Retinal Artery Occlusion (RAO), Retinal Vein Occlusion (RVO), and Vitreomacular Interface Disease (VID). Scans from normal eyes (NO) that do not have one of the diseases are also included. A summary of the dataset is provided in table 4.1. Furthermore, each record contains a patient id, so scans belonging to the same patient can be identified. The authors trained a ResNet50 Model on the entirety of their data and on combinations with other, previously published datasets.

Table 4.1: OCTDL dataset summary. The first two classes (AMD, NO) are used for the binary AMD diagnosis use case.

| Disease | Label | Number of Scans | Number of Patients |
|----------------------------------|-------|-----------------|--------------------|
| Age-related Macular Degeneration | AMD | 1231 | 421 |
| Normal | NO | 332 | 110 |
| Diabetic Macular Edema | DME | 147 | 107 |
| Epiretinal Membrane | ERM | 155 | 71 |
| Retinal Artery Occlusion | RAO | 22 | 11 |
| Retinal Vein Occlusion | RVO | 101 | 50 |
| Vitreomacular Interface Disease | VID | 76 | 51 |
| Total | | 2064 | 821 |

The experiments deal with two use cases. The first one is AMD diagnosis, i.e. classifying an OCT scan as either AMD or NO, since Gholami et al. (2023) also investigated this use case on previously published datasets. Secondly, the simulations are run on a multi-class problem, including all classes of the dataset, following the experiments conducted by Kulyabin et al. (2024) on OCTDL.

In order to test the resulting models on unseen data and to ensure a meaningful comparison, 15% of the dataset is set aside as test data for all experiments. The centralized experiments use another 15% for validation. Hereby, we ensure that each patient’s data is exclusive to each of the subsets, in the same way the authors of the dataset did. We also split the data on a patient’s level while distributing it among clients in the federated learning setting. This helps create realistic conditions for the experiments. Each client participating in the federated learning simulation reserves 20% of their data for validation. Because of the imbalanced nature of the dataset, a stratified split is used here in order to ensure that each class is present in both the training and validation set of each client.

4.2 Models

As mentioned in chapter 3, this work focuses on the ResNet and MobileNet architectures. More precisely, PyTorch’s implementations for ResNet18, ResNet50 and MobileNetV2 are used. Transfer learning is applied to the ResNet18 and the MobileNetV2 model. The ResNet18 and ResNet50 models are also trained from scratch, following the presented literature on FL for OCT retina images and the publication for the dataset. For the multi-class use case, we train MobileNetV2 with transfer learning and the ResNet50 model from scratch. An overview of the model architectures and use cases is provided in table 4.2.

Table 4.2: Overview of model architectures and use cases. TL denotes transfer learning.

| Model | Use Cases |
|----------------|----------------------------------|
| ResNet18 | AMD Diagnosis |
| ResNet18 TL | AMD Diagnosis |
| ResNet50 | AMD Diagnosis, All OCTDL Classes |
| MobileNetV2 TL | AMD Diagnosis, All OCTDL Classes |

4.3 Centralized Learning

In the centralized learning experiments, the models train for a maximum of 100 epochs. After each epoch, the model is evaluated on the validation set. Thereby, the metric by which we want to optimize the model is calculated based on the evaluation results. The implemented early stopping strategy starts after epoch 20. Then, if the model performance, i.e. the macro F1 score, does not improve for five epochs, the training is stopped. During this process, the parameters of the best performing model are continuously stored for testing purposes. The hyperparameter ranges used for optimization are specified in table 4.3.

4.4 Federated Learning Simulations

All federated learning simulations are run using 20 clients that participate in the training process. Our goal was to choose a number that is large enough so we can experiment with different numbers of clients per aggregation round but also small enough so each client receives a sufficiently large local dataset. Each FL algorithm is tested with 10, 5 and 3 client updates per global model update. The simulations run until the global model is updated 260 times, giving the models enough time to converge. For the FL optimization, the number of local epochs for each client is defined as an additional hyperparameter with a range from 1 to 10, following the experiments of Gholami et al. (2023). The server learning rate for FedBuff uses the same range as the client learning rate. These hyperparameters are also included in table 4.3. An overview of how the FL experiments are run can be found in algorithm 3.

For FedAvg, the implementation provided by *Flower* is used. FedBuff, on the other hand, uses a custom implementation within the framework. *Flower* does not provide support for asynchronous FL algorithms. However, with a few modifications to FedAvg we can simulate FedBuff’s aggregation process, since only performance of the model is tested rather than wall-clock time. First, the simulation selects clients equal to the buffer size. Then, staleness is simulated by giving each client a random staleness value. Nguyen et al. (2022) discovered during their experiments that client delays and staleness in FedBuff follow a half-normal distribution. The maximum staleness of a client is $s_{max} = R - P_r - 1$, where R is the current round and P_r is the last used parameter version. Then, each client receives parameters of the model version according to their staleness. Finally, the aggregation logic on the server side and the calculation of the update on the client side are implemented. Algorithm 4 illustrates the described FedBuff implementation.

Table 4.3: Hyperparameters used in different learning strategies.

| Hyperparameter | Used for | Values |
|----------------------|------------------------------|--------------------------------|
| Learning Rate | Centralized, FedAvg, FedBuff | [0.1, 0.0001] |
| Batch Size | Centralized, FedAvg, FedBuff | {8, 16, 32, 64, 128} |
| Apply Augmentation | Centralized, FedAvg, FedBuff | {True, False} |
| Dropout | Centralized, FedAvg, FedBuff | {0.0, 0.1, 0.2, 0.3, 0.4, 0.5} |
| Client Epochs | FedAvg, FedBuff | [1..10] |
| Server Learning Rate | FedBuff | [0.1, 0.0001] |

Algorithm 3 Federated Learning Experiments

Run for each model configuration / use case combination in table 4.2 for both algorithms: FedAvg, FedBuff

- 1: **Input:** Number of clients $N = 20$, Number of global updates $U = 260$, Model configuration M , Classes for use case CL , FL algorithm F
 - 2: $D \leftarrow$ Load dataset with classes CL
 - 3: $D_{\text{train}}, D_{\text{test}} \leftarrow$ Randomly split dataset D into training set and test set (15%) on a patient's level
 - 4: **for** each client $C_i, i \in \{1, 2, \dots, N\}$ **do**
 - 5: Assign a subset D_i of D_{train} to client C_i
 - 6: Reserve 20% of D_i as local validation set for client C_i using stratified split
 - 7: **end for**
 - 8: **for** each update frequency $f \in \{10, 5, 3\}$ **do**
 - 9: /* FedAvg: $f := \text{clients per round}$, FedBuff: $f := \text{buffer size}$ */
 - 10: $r \leftarrow \lfloor U/f \rfloor$
 - 11: Set number of rounds in F to r
 - 12: Run hyperparameter optimization using F with $C_1 \dots C_N$ on model M
 - 13: **end for**
 - 14: Test the final model on D_{test}
-

Algorithm 4 FedBuff Implementation

-
- 1: **Input:** server learning rate η_g , client learning rate η_l , client SGD steps Q , buffer size K , number of rounds R
 - 2: **Output:** Trained model
 - 3: $P \leftarrow \emptyset$ ▷ map round number to parameters of that round
 - 4: $CPV \leftarrow \{i = 0 \text{ for each client}\}$ ▷ map client to its last used parameter version
 - 5: $w_1 \leftarrow$ get weights from random client
 - 6: **for** $r = 1 : R$ **do**
 - 7: $P[r] \leftarrow w_r$ ▷ save weights of current round
 - 8: $clients \leftarrow$ select K clients at random
 - 9: **for** each client c **do**
 - 10: $s_{\text{max}} \leftarrow r - CPV[c] - 1$ ▷ calculate max staleness
 - 11: $s \leftarrow \text{RandomStaleness}(s_{\text{max}})$ ▷ sample from halfnormal distribution
 - 12: $paramVersion \leftarrow r - s$
 - 13: $CPV[paramVersion] \leftarrow r$ ▷ update last used paramter version
 - 14: $params \leftarrow P[paramVersion]$ ▷ get parameters based on staleness
 - 15: Run **FedBuff-client**($params, \eta_l, Q$) on c
 - 16: **end for**
 - 17: $U \leftarrow$ collect client updates
 - 18: $\bar{\Delta}^r \leftarrow \frac{\sum_{u \in U} \bar{\Delta}^u}{K}$
 - 19: $w_{r+1} \leftarrow w_r - \eta_g \bar{\Delta}^r$
 - 20: **end for**
-

5 | Results

In the following chapter, the results of the previously described experiments are presented. All metrics collected during the testing of the final models are shown in tabular form, where the most important values and rows are highlighted in order to emphasize the key findings. Furthermore, several charts are used to showcase the performance difference between the centralized and federated learning approaches, as well as different model configurations. Finally, for the overall best performing models, we show the learning curves for FedAvg and FedBuff. All of the code and the best performing hyperparameters for each of the experiments can be found in the GitHub repository provided in section 7.2. For the purposes of evaluating the differences in performance among approaches and models, Cohen’s h is used to measure the effect size (Cohen, 1988), which is calculated as follows, given two probabilities p_1 and p_2 :

$$\text{effectSize}(p_1, p_2) = |2 * \arcsin(\sqrt{p_1}) - 2 * \arcsin(\sqrt{p_2})|$$

Per Cohen’s rule of thumb, we consider effect sizes around $h = 0.20$ as small, $h = 0.5$ as medium and $h = 0.8$ as large differences.

5.1 Centralized Training

The centralized test results for AMD diagnosis are displayed in table 5.1. We utilize the metrics presented in section 3.2 in order to evaluate a model’s performance. For each model configuration, the balanced accuracy (Bal Acc), F1 score macro (F1 Macro) and binary F1 score for AMD (F1 Score AMD) are shown. The best performing model is ResNet18 without transfer learning, reaching an F1 macro score of 0.942. The other models tested achieved a similar performance, with a small difference ($h = 0.092$) between the best and worst performing setup.

Results for the multi-class use case on the centralized models can be seen in table 5.2. In the experiments, the MobileNetV2 model with transfer learning performed

Table 5.1: Centralized performance metrics for AMD diagnosis. The best performing model in terms of F1 macro score is marked with bold text.

| Model | Transfer Learning | Bal Acc | F1 Macro | F1 Score AMD |
|-----------------|-------------------|---------|--------------|--------------|
| ResNet18 | x | 0.953 | 0.942 | 0.972 |
| ResNet18 | ✓ | 0.963 | 0.938 | 0.969 |
| ResNet50 | x | 0.928 | 0.939 | 0.973 |
| MobileNetV2 | ✓ | 0.930 | 0.919 | 0.961 |

better than the ResNet50 model, with an F1 macro score of 0.600. The difference in performance among models is also small here but more noticeable ($h = 0.182$).

Table 5.2: Centralized performance metrics for multi-class use case. The best performing model in terms of F1 macro score is marked with bold text.

| Model | Transfer Learning | Bal Acc | F1 Macro |
|--------------------|-------------------|---------|--------------|
| ResNet50 | x | 0.663 | 0.509 |
| MobileNetV2 | ✓ | 0.669 | 0.600 |

5.2 Federated Learning Simulations

Test metrics for AMD diagnosis using FedAvg can be seen in table 5.3. The best configuration with an F1 macro score of 0.958 is the ResNet18 model without transfer learning and three clients per round. The difference between the best and worst configurations is moderate to small ($h = 0.332$). Figure 5.1 visualizes the best metrics achieved among centralized learning and FedAvg with ten (FedAvg 10), five (FedAvg 5) and three clients (FedAvg 3) per round. FedAvg outperforms the centralized models in terms of F1 macro score by a small amount ($h = 0.074$).

Table 5.3: Performance metrics for AMD diagnosis with FedAvg. For each number of clients per round, the best F1 macro score is displayed with bold text. The overall best performing model is also marked in bold.

| Model | Transfer Learning | Bal Acc | F1 Macro | F1 Score AMD | Clients per Round |
|-----------------|-------------------|---------|--------------|--------------|-------------------|
| ResNet18 | x | 0.923 | 0.928 | 0.967 | 10 |
| ResNet18 | ✓ | 0.947 | 0.941 | 0.972 | 10 |
| ResNet50 | x | 0.895 | 0.876 | 0.938 | 10 |
| MobileNetV2 | ✓ | 0.918 | 0.907 | 0.955 | 10 |
| ResNet18 | x | 0.937 | 0.946 | 0.975 | 5 |
| ResNet18 | ✓ | 0.951 | 0.937 | 0.969 | 5 |
| ResNet50 | x | 0.863 | 0.868 | 0.939 | 5 |
| MobileNetV2 | ✓ | 0.939 | 0.925 | 0.963 | 5 |
| ResNet18 | x | 0.955 | 0.958 | 0.981 | 3 |
| ResNet18 | ✓ | 0.947 | 0.941 | 0.972 | 3 |
| ResNet50 | x | 0.913 | 0.932 | 0.970 | 3 |
| MobileNetV2 | ✓ | 0.930 | 0.919 | 0.961 | 3 |

For FedBuff, table 5.4 shows the metrics for the test set, where MobileNetV2 with transfer learning shows the best performance with an F1 macro score of 0.792 and a buffer size of three. The difference in performance between the best and worst configurations is moderate to high ($h = 0.646$). Metrics of the best models for FedBuff can be seen in figure 5.2. FedBuff performed worse compared to the centralized training, showing a medium difference in performance ($h = 0.456$).

A comparison between all model architectures is presented in figure 5.3. For each model, the F1 macro score of the centralized learning, FedAvg and FedBuff is displayed. ResNet18 and MobileNetV2 with transfer learning perform similarly well on all three approaches. The ResNet18 and ResNet50 models trained from scratch performed well in the centralized and FedAvg scenario, but worse on FedBuff. Table 5.5 provides an overview of the FL performances in terms of effect size. Each entry refers

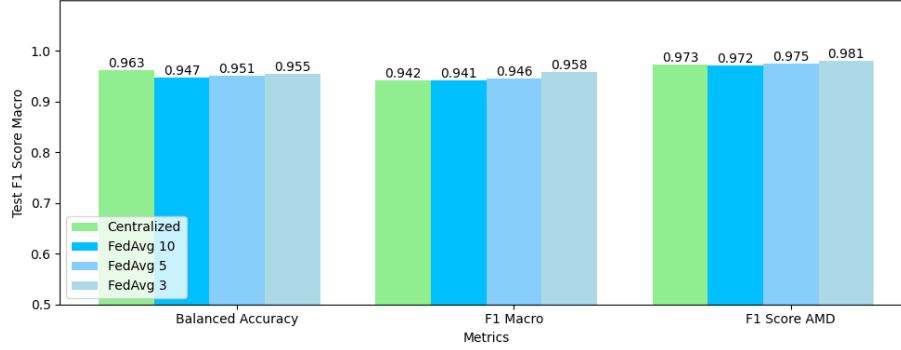


Figure 5.1: Comparison of test metrics for AMD diagnosis using FedAvg.

to the difference in performance between centralized learning and the respective FL approach.

For the multi-class use case, all results for FedAvg and FedBuff are presented in table 5.6 and table 5.7, respectively. Here, FedAvg performed better than the centralized model by a small amount ($h = 0.164$) with F1 macro score of 0.678. The metric comparisons can be found in table 5.4. The two used models differ in performance by a medium amount ($h = 0.517$), with the ResNet50 model yielding the highest score. For FedBuff, the FL approach yielded worse results, with the MobileNetV2 model reaching an F1 macro score of 0.295, which is a medium to large difference ($h = 0.623$) to the best centralized model. Both metrics are compared in figure 5.5. The overall performance of the two models is shown in figure 5.6.

Figure 5.7 compares the learning curves of ResNet18 (5.7a) and MobileNetV2 (5.7b) with transfer learning using FedAvg and FedBuff for AMD diagnosis. Also, the corresponding performance of the centralized model is shown in the graphs. In ResNet18 with FedAvg, the F1 score rises quickly across all client counts, with more clients per global update resulting in fewer updates needed to reach the maximum. ResNet18 with FedBuff shows slower, more gradual learning, with fewer clients per update (i.e. lower buffer size) also needing fewer global updates to reach the maximum. The training process appears to be more stable with a lower buffer size. For MobileNetV2 with FedAvg, learning is generally faster. The model stabilizes earlier for five clients, while three clients showed a more unstable learning curve. In MobileNetV2 with FedBuff, three clients show the best early performance, while 5 clients improve more slowly and show a more unstable learning curve. Overall, FedAvg leads to faster initial learning, while FedBuff offers more gradual, steady improvements. MobileNetV2 generally outperforms ResNet18 in terms of faster convergence, while ResNet18 shows a more stable training process across both algorithms.

Table 5.4: Performance metrics for AMD diagnosis with FedBuff. For each number of clients per round, the best F1 macro score is displayed with bold text. The overall best performing model is also marked in bold.

| Model | Transfer Learning | Bal Acc | F1 Macro | F1 Score Binary | Clients per Round |
|--------------------|-------------------|---------|--------------|-----------------|-------------------|
| ResNet18 | x | 0.755 | 0.651 | 0.737 | 10 |
| ResNet18 | ✓ | 0.618 | 0.603 | 0.784 | 10 |
| ResNet50 | x | 0.517 | 0.489 | 0.856 | 10 |
| MobileNetV2 | ✓ | 0.811 | 0.773 | 0.873 | 10 |
| ResNet18 | x | 0.697 | 0.662 | 0.801 | 5 |
| ResNet18 | ✓ | 0.806 | 0.787 | 0.891 | 5 |
| ResNet50 | x | 0.561 | 0.565 | 0.820 | 5 |
| MobileNetV2 | ✓ | 0.686 | 0.663 | 0.815 | 5 |
| ResNet18 | x | 0.563 | 0.567 | 0.838 | 3 |
| ResNet18 | ✓ | 0.685 | 0.668 | 0.823 | 3 |
| ResNet50 | x | 0.564 | 0.568 | 0.824 | 3 |
| MobileNetV2 | ✓ | 0.819 | 0.792 | 0.890 | 3 |

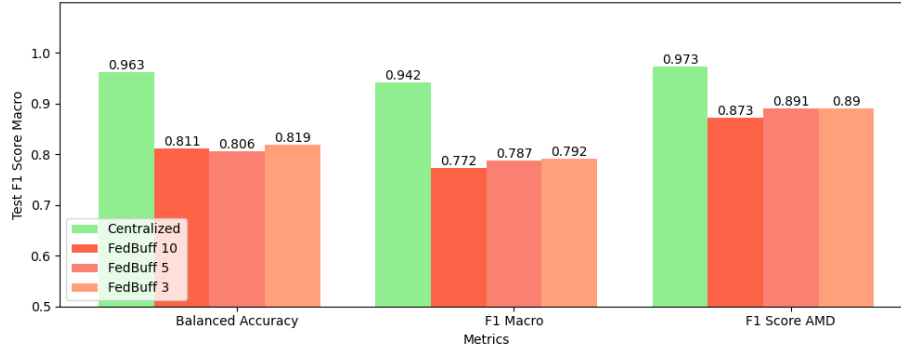


Figure 5.2: Comparison of test metrics for AMD diagnosis using FedBuff.

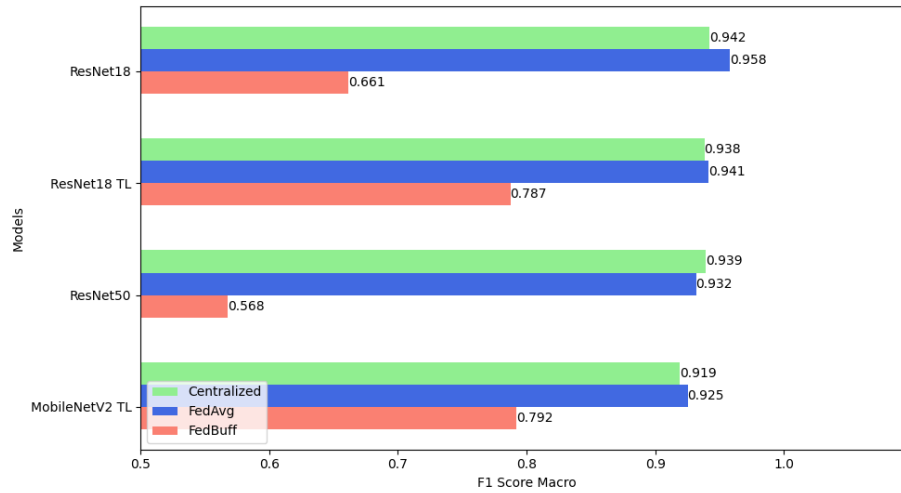


Figure 5.3: Test performance of different model configurations on AMD diagnosis. TL denotes transfer learning.

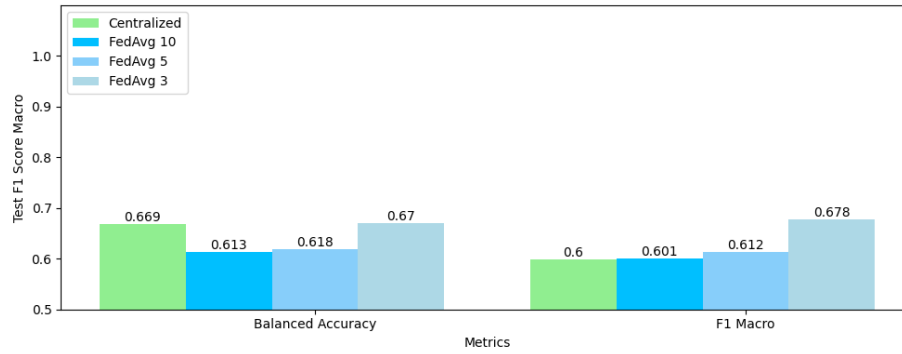


Figure 5.4: Comparison of test metrics for the multi-class use case using FedAvg.

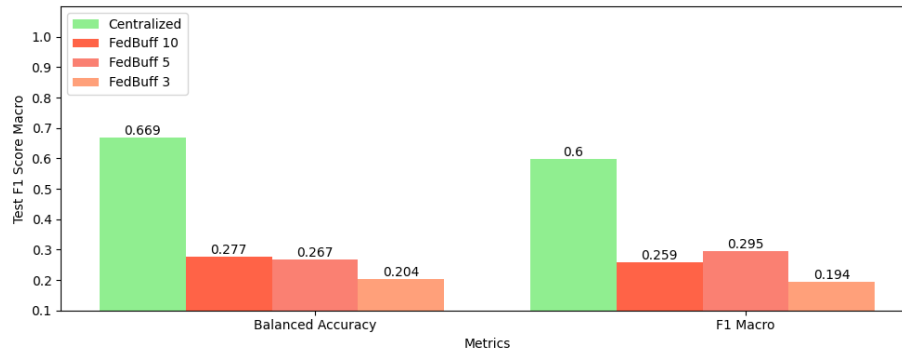


Figure 5.5: Comparison of test metrics for the multi-class use case using FedBuff.

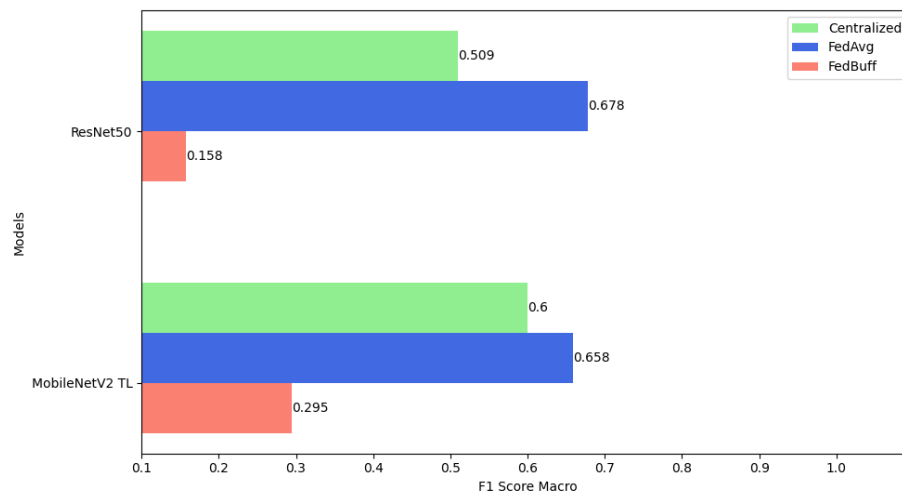
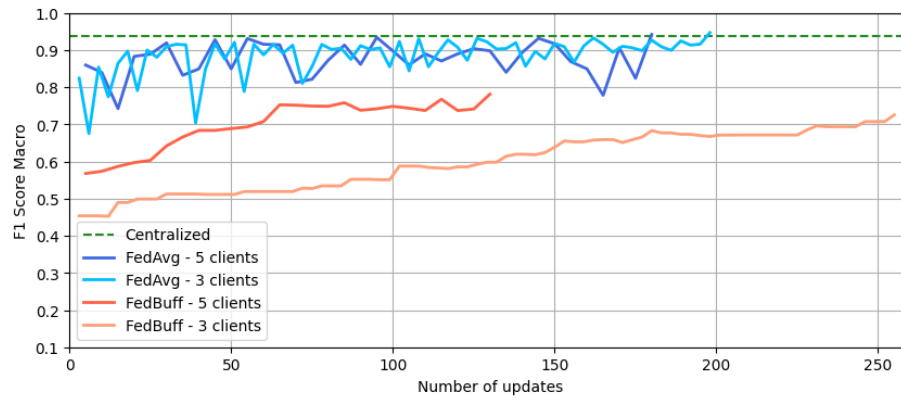
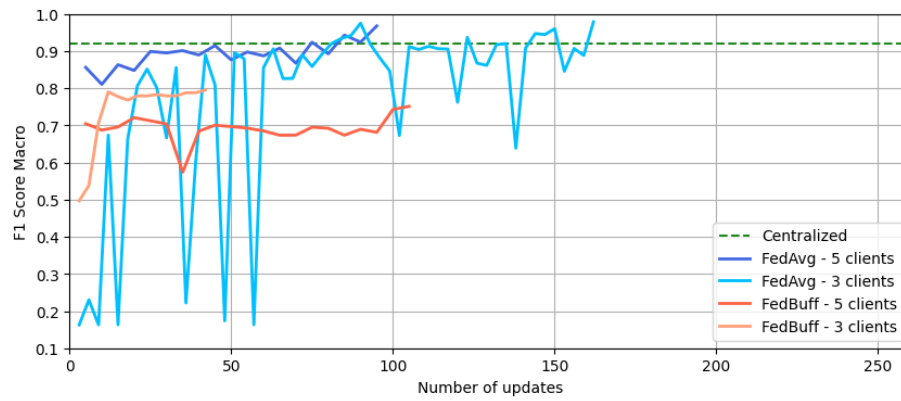


Figure 5.6: Test performance of different model configurations on the multi-class use case. TL denotes transfer learning.



(a) ResNet18 (transfer learning)



(b) MobileNetV2 (transfer learning)

Figure 5.7: Validation learning curves of best models in AMD diagnosis, FedAvg / FedBuff.

Table 5.5: Effect sizes between centralized and FL model performance for AMD diagnosis. TL indicates that the model uses transfer learning.

| Model | FedAvg | FedBuff |
|--------------|---------------|----------------|
| ResNet18 | 0.074 (small) | 0.756 (large) |
| ResNet18 TL | 0.014 (small) | 0.456 (medium) |
| ResNet50 | 0.030 (small) | 0.936 (large) |
| MobileNet TL | 0.023 (small) | 0.461 (medium) |

Table 5.6: Performance metrics for the multi-class use case with FedAvg. For each number of clients per round, the best F1 macro score is displayed with bold text. The overall best performing model is also marked in bold.

| Model | Transfer Learning | Bal Acc | F1 Macro | Clients per Round |
|-----------------|-------------------|---------|--------------|-------------------|
| ResNet50 | x | 0.394 | 0.424 | 10 |
| MobileNetV2 | ✓ | 0.613 | 0.601 | 10 |
| ResNet50 | x | 0.527 | 0.546 | 5 |
| MobileNetV2 | ✓ | 0.618 | 0.613 | 5 |
| ResNet50 | x | 0.617 | 0.678 | 3 |
| MobileNetV2 | ✓ | 0.670 | 0.658 | 3 |

Table 5.7: Performance metrics for the multi-class use case with FedBuff. For each number of clients per round, the best F1 macro score is displayed with bold text. The overall best performing model is also marked in bold.

| Model | Transfer Learning | Bal Acc | F1 Macro | Buffer Size |
|--------------------|-------------------|---------|--------------|-------------|
| ResNet50 | x | 0.126 | 0.107 | 10 |
| MobileNetV2 | ✓ | 0.277 | 0.259 | 10 |
| ResNet50 | x | 0.185 | 0.158 | 5 |
| MobileNetV2 | ✓ | 0.267 | 0.295 | 5 |
| ResNet50 | x | 0.142 | 0.114 | 3 |
| MobileNetV2 | ✓ | 0.204 | 0.194 | 3 |

Table 5.8: Effect sizes between centralized and FL model performance on the multi-class use case.

| Model | FedAvg | FedBuff |
|--------------|----------------------|----------------------|
| ResNet50 | 0.345 (small-medium) | 0.772 (large) |
| MobileNet-TL | 0.122 (small) | 0.623 (medium-large) |

6 | Discussion

In this chapter, we discuss and interpret the experimental results from the previous chapter and review challenges and limitations regarding the POC application. Here, we aim to give a thorough answer to the research questions presented in chapter 1.

RQ1: How viable is an asynchronous FL approach to optimize machine learning-based medical diagnosis using OCT retina images?

Our results regarding asynchronous FL using FedBuff on the tested model architectures and configurations are provided in table 5.4 for the binary AMD classification. In the binary classification problem, FedBuff reached an F1 macro score of 0.792. The best performing configuration uses the MobileNetV2 model with transfer learning and three clients per round (\equiv buffer size), closely followed by the pre-trained ResNet18 with five clients per round.

Table 5.7 shows the results for the multi-class use case. Here, FedBuff performed significantly worse, with an F1 macro score of 0.295, using a MobileNetV2 model with transfer learning and five clients per round. This is likely caused by the considerably different sample size in each class of the dataset. In table 4.1, we can see that the most represented class (AMD) holds 1231 samples, while the class with the least number of scans (RAO) only provides 22 data points from 11 patients. Splitting the data among 20 clients on a patient's level causes some clients to not have any samples of the RAO class.

Using these results and insights, we can see that asynchronous FL using FedBuff is a viable approach for the binary classification use case, with room for improvement, but it requires more investigation to optimize its performance further. However, for the multi-class use case, FedBuff cannot provide a well-performing model for our setup. Additional experiments are needed to investigate the behaviour of FedBuff further in this use case. Overall, we think that asynchronous FL is a valid approach for OCT medical diagnosis, and warrants further investigation across different medical use cases using different algorithms.

RQ2: How does asynchronous FL perform in terms of diagnostic accuracy for OCT retina images compared to established centralized and synchronous FL methods?

For binary AMD classification, the centralized models all showed similar performance. In the overview provided by table 5.1 we can see that the best performing model reaches an F1 macro score of 0.942. FedAvg yields similar results across models, as depicted in figure 5.1. FedBuff’s performance is noticeably lower, but only by a medium effect amount, reaching an F1 macro score of 0.792, as seen in figure 5.2. This behaviour can be caused by multiple factors. Asynchronous FL can introduce staleness, i.e. client updates which are based on an older version of the global model. Also, FedBuff requires the server learning rate as a new hyperparameter. This increases the searchable parameter space during the optimization process, which as a result, may require more trials to find good values. Fine-tuning of the additional layers may also be necessary for models using transfer learning in FedBuff.

Table 5.2 shows that the centralized models for the multi-class use case did not perform well with our setup, with the best model reaching an F1 macro score of 0.600 on the test set. The cause may be the distribution of data between the training, validation and test datasets. As described by Kulyabin et al. (2024), we split the dataset on a patient’s level, meaning that one person’s data is only present in one of the subsets. It is possible that a different random distribution of data caused the drop in performance. In the federated setting, FedAvg and FedBuff show the same behaviour as those observed in the AMD diagnosis. FedAvg performed slightly better than the centralized models, with an F1 macro score of 0.678. FedBuff performed worse, reaching a score of 0.295. The respective comparisons can be seen in figures 5.4 and 5.5.

After the described experiments, we ran an additional trial to investigate if the data distribution in the multi-class use case can be the cause for the poor performance of our centralized models. We trained the ResNet50 model for 100 epochs with fixed hyperparameters under ideal conditions, i.e. ensuring an equal distribution of classes among training, validation and test data. We achieved a balanced accuracy of 0.820 and an F1 macro score of 0.810, which is significantly closer to the metrics reported by the dataset authors. Based on this result, we can see that our assumption is plausible.

A comparison of different model architectures and configurations in figures 5.3 and 5.6 reveals that the pre-trained ResNet18 and MobileNetV2 models perform the best in AMD diagnosis when taking all three approaches into account. The models trained from scratch performed slightly better in some cases, but significantly worse in FedBuff. For the multi-class use case, MobileNetV2 using transfer learning performs better than the ResNet50 model trained from scratch, except for FedAvg.

In accordance with the reproducibility terms outlined by Gundersen and Kjensmo (2018) we compare our results with the related work. The results exhibit method reproducibility for the work by Gholami et al. (2023). On two out of three datasets they tested, ResNet18 performed similarly to the centralized models using FedAvg. As shown in table 6.1, the differences between the centralized and FedAvg models are similar to the results of our experiments. To measure the effect size between results, we use Cohen’s h , as described in section 5. We did not achieve experiment reproducibility regarding the results reported by Kulyabin et al. (2024). They trained a ResNet50 model on the entirety of the OCTDL dataset and reached an F1 score of

0.866, whereas our experiments yielded a score of 0.600. However, for a subset of the dataset used for AMD diagnosis (AMD, NO), we showed method reproducibility by training a ResNet50 model with a final F1 score of 0.939.

Table 6.1: Effect sizes between centralized and FL model performance for AMD diagnosis between our results and the results of Gholami et al. (2023) on two of their datasets (DS1, DS2).

| Results | Difference Centralized / FedAvg |
|---------------------------|---------------------------------|
| Our results | 0.074 (small) |
| Gholami et al. (2023) DS1 | 0.012 (small) |
| Gholami et al. (2023) DS2 | 0.087 (small) |

In figure 5.7, we show the validation learning curves for FedAvg and FedBuff for the models using transfer learning, as they yielded the best results on FedBuff. The graphs suggest that the ResNet18 model is the better choice for a more stable training process, whereas MobileNetV2 achieves faster convergence in our experiments. These models are potentially viable for the interactive browser-based FL application presented in this work, as they performed similarly well with asynchronous FL. Furthermore, both have relatively small file sizes, around 13 mb for MobileNetV2 and 45 mb for ResNet18. The number of trainable parameters is also much smaller with transfer learning, which leads to fewer computational resources needed on the client side.

Overall, in order to answer RQ2, we chose FedBuff as the asynchronous FL algorithm and compared it to centralized learning and FedAvg as the established FL approach. As far as we know, this is the first work applying FL to the OCTDL dataset used in our experiments. We found that FedAvg performs similarly well to the centralized learning, whereas FedBuff performs noticeably worse than FedAvg and centralized learning, but only by a medium amount. However, with the additional benefits of asynchronous FL in mind, we believe that our results show the potential of using such algorithms, especially with larger or more balanced datasets, to better understand suitable use cases.

RQ3: What are the challenges and limitations of implementing FL in a browser-based environment, and how can they be addressed?

In research question 3, we ask which challenges and limitations exist for the implementation of FL in a browser-based environment and how we can address them. The very first challenge that needs to be solved to implement a browser-based FL application is the choice of algorithm. In order to enable users to conduct the training process on their behalf, an asynchronous FL algorithm is needed that allows each client to send their local update at any time. As discussed in the previous section, asynchronous FL is a viable approach for one of the possible use cases of such an application in medical imaging, i.e. diagnosing OCT retina images. Here, FedBuff provides a good starting point for further investigation. Limited client resources also need to be kept in mind while choosing a suitable model. The ResNet18 and MobileNetV2 models, which performed best in our experiments, are relatively small in terms of file size and have only

a few trainable parameters when using transfer learning.

Another challenge, as mentioned by Guan et al. (2024), is the compatibility and integration with existing hospital systems and user adoption. By providing a browser-based application, we eliminate the need for installation of third-party software by leveraging the capabilities of modern browsers, which are already included in most operating systems. In order to ensure user adoption, the application provides an interactive user interface which guides the user through the data collection and training process.

On the technical side, the training process on the client's device needs to be implemented efficiently in order to not require a large amount of resources in terms of time and computational power. Machine learning frameworks for the browser offer solutions for this challenge. ONNX Runtime (ONNX Runtime developers, 2021) is the library of choice for this task. It is chosen over tensorflow.js (Smilkov et al., 2019) used by Google in their browser-based ML application (Carney et al., 2020). One reason is that ONNX Runtime is ML framework agnostic, supporting any model that can be exported to the ONNX file format. Another important difference is that ONNX Runtime supports WebGPU, the newest standard for executing computations directly on the user's graphics card, whereas tensorflow.js supports the older WebGL standard.

During the development of the POC application, several limitations started to show. As the ONNX Runtime integration for the web is relatively new, not all its functionalities for model training have been implemented yet for the web. One of the shortcomings is that the learning rate of the optimizer cannot be changed, so clients need to train with a default learning rate of 0.001. Also, only a small set of loss functions have been implemented so far. Weighted cross-entropy loss, as used during our FL experiments, can also not be used, as the loss function needs to be fully defined on the server side while generating the necessary files for on-device training. Furthermore, WebGPU is supported only for inference at the time, whereas the training process is limited to the WebAssembly backend. Regarding the web browser itself, it is important to note that the size of the local dataset stored in the browser's IndexedDB cannot be arbitrarily large. Each browser determines the storage limit differently based on available disk space. Lastly, the WebGPU API is currently only available in developer builds of modern browsers, which may not be allowed on systems with high-security standards.

7 | Conclusion

This work aimed to gain insights into the performance differences of asynchronous FL in comparison to synchronous FL and centralized learning. Asynchronous FL algorithms help enable a browser-based interactive FL approach without requiring a synchronous training process. To demonstrate how such an application could be implemented using modern web technologies, a proof-of-concept application was developed in the context of this work. It helped us understand the current challenges and limitations of realizing such an approach.

Our main contribution was applying FL to the OCTDL dataset, further confirming that FL is a viable strategy for OCT data. In addition to FedAvg, an established synchronous algorithm, we also tested asynchronous FL by using FedBuff, which had not been applied to OCT data yet. Several model architectures were compared to evaluate their differences in performance across the mentioned learning approaches. We tested a ResNet18 model, with and without transfer learning, and a ResNet50 model to reproduce results from the related works. Additionally, a MobileNetV2 model with transfer learning was included in the experiments, as it is optimized for resource-constrained environments in terms of file and parameter size, making it a potentially good fit for our browser-based application.

Our experiments demonstrated that federated learning with FedAvg yielded similar, slightly better results compared to centralized learning for binary AMD diagnosis and the multi-class classification of retinal diseases using OCT images. FedBuff performed worse than both synchronous FL and centralized learning, with a medium difference for the binary classification and a medium to high difference in the multi-class use case. When comparing model architectures, all models showed similar performance in centralized learning and FedAvg, with the pre-trained ResNet18 and MobileNetV2 models performing significantly better on FedBuff. Our results confirmed the reproducibility of prior work on FL for AMD diagnosis using the ResNet18 model. For centralized learning on the multi-class use case, however, we did not achieve experiment reproducibility, as our ResNet50 model performed significantly worse.

The development of the POC application demonstrated how an interactive browser-based FL approach could be implemented, from the interactive front-end to the FL aggregation process on the server, using state-of-the-art technologies. Our FL application can work as an additional service module for building trust and encouraging users to engage with interactive deep learning (IDL) systems (Sonntag et al., 2024), particularly in domains where privacy is essential.

We used ONNX Runtime for client-side training for its compatibility with popular ML frameworks and support for modern browser features, such as WebAssembly and WebGPU. The browser-based approach tackled various challenges in terms of compatibility, integration, and user adoption. During development, a number of

limitations arose from ONNX Runtime’s early-stage support for the web. These included fixed learning rates and limited loss functions. Furthermore, browsers imposed storage limits for local datasets and only supported the WebGPU standard in special developer builds. Despite these limitations, the utilized technologies could already work together to enable efficient training on the client side with support for federated learning to collaboratively train a model across devices.

7.1 Future Work

As discussed in Chapter 6, FedBuff achieved fair performance in the binary classification of OCT data, though there is room for improvement. Future work could explore a more extensive optimization process. Since FedBuff introduces a server learning rate as an additional hyperparameter, further experimentation may be required to identify optimal conditions. In our experiments with transfer learning, we used a fixed number of fully connected layers with a predetermined size before the final classification layer. Both the number and size of these layers could be included in the hyperparameter optimization for fine-tuning. Another direction for improvement is adjusting the number of clients in the training process. Testing different values could reveal whether performance improves with fewer clients, providing insights into how finely the dataset can be partitioned before performance degrades. Additionally, FedBuff could be applied to other class combinations within the dataset to see how complex of a task it can handle. Testing it on larger or combined OCT datasets may also show whether a bigger sample size enhances performance. Beyond FedBuff, experimenting with other asynchronous FL algorithms would be an interesting extension of this research. Instead of image classification, other computer vision tasks, such as object detection, can be tested with FedBuff or other asynchronous FL algorithms.

In addition, applying active learning techniques of OCT datasets, as discussed in (Kadir et al., 2024, 2023), for sample selection in FedBuff’s training can be a promising direction. Integrating active learning strategies will enable the model to selectively query the most informative and uncertain data points from clients, ensuring that only the most valuable samples are labeled and used for training. This approach can significantly reduce labeling costs, which is particularly advantageous in a medical imaging context, where obtaining labeled data is often challenging, scarce, and expensive. By focusing on the most impactful data points, this integration could enhance the efficiency and performance of FedBuff, leading to better model accuracy while minimizing the need for extensive manual labeling efforts.

A reasonable next step for the POC application would be to conduct user studies to gain insights into user adoption and performance in a real-world scenario. Another direction would be to implement other computer vision tasks in the application, such as image segmentation or object detection. An interesting addition could also be integrating from the field of explainable artificial intelligence, such as class activation maps (Selvaraju et al., 2017), for example, helping users understand and trust the machine learning process. Moreover, enabling administrators within the application to deploy new training tasks could enhance functionality. Administrators could upload ONNX models, define classes, select the type of task (e.g., classification or segmentation), select the algorithm, provide training instructions and oversee the training progress. Finally, the application could allow clients to participate in multiple different tasks with multiple local datasets for each use case.

7.2 Code Availability

All of our results and the code to generate them can be found in this GitHub repository: <https://github.com/tmaurer42/octdl-training>. The source code for the proof-of-concept application is available here: <https://github.com/tmaurer42/interactive-fl-poc>.

Bibliography

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
- Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA. Association for Computing Machinery.
- Anaya-Isaza, A., Mera-Jiménez, L., and Zequera-Diaz, M. (2021). An overview of deep learning in medical imaging. *Informatics in Medicine Unlocked*, 26:100723.
- Aurelio, Y. S., De Almeida, G. M., de Castro, C. L., and Braga, A. P. (2019). Learning from imbalanced data sets with weighted cross-entropy function. *Neural processing letters*, 50:1937–1949.
- Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Kwing, H. L., Parcollet, T., Gusmão, P. P. d., and Lane, N. D. (2020). Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*.
- Bonawitz, K. A., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. (2016). Practical secure aggregation for federated learning on user-held data. In *NIPS Workshop on Private Multi-Party Machine Learning*.
- Carney, M., Webster, B., Alvarado, I., Phillips, K., Howell, N., Griffith, J., Jongejan, J., Pitaru, A., and Chen, A. (2020). Teachable machine: Approachable web-based tool for exploring machine learning classification. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI EA '20, page 1–8, New York, NY, USA. Association for Computing Machinery.
- Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, Hillsdale, NJ, 2nd edition.

- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.
- Fails, J. A. and Olsen, D. R. (2003). Interactive machine learning. In *Proceedings of the 8th International Conference on Intelligent User Interfaces, IUI '03*, page 39–45, New York, NY, USA. Association for Computing Machinery.
- Gholami, S., Lim, J., Leng, T., Ong, S., Thompson, A., and Alam, M. (2023). Federated learning for diagnosis of age-related macular degeneration. *Frontiers in Medicine*, 10.
- Grinberg, M. (2018). *Flask web development: developing web applications with python*. "O'Reilly Media, Inc."
- Guan, H., Yap, P.-T., Bozoki, A., and Liu, M. (2024). Federated learning for medical image analysis: A survey. *Pattern Recognition*, 151:110424.
- Gundersen, O. E. and Kjensmo, S. (2018). State of the art: Reproducibility in artificial intelligence. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).
- Haggenmüller, S., Schmitt, M., Krieghoff-Henning, E., Hekler, A., Maron, R. C., Wies, C., Utikal, J. S., Meier, F., Hobelsberger, S., Gellrich, F. F., Sergon, M., Hauschild, A., French, L. E., Heinzerling, L., Schlager, J. G., Ghoreschi, K., Schlaak, M., Hilke, F. J., Poch, G., Korsing, S., Berking, C., Heppt, M. V., Erdmann, M., Haferkamp, S., Drexler, K., Schadendorf, D., Sondermann, W., Goebeler, M., Schilling, B., Kather, J. N., Fröhling, S., and Brinker, T. J. (2024). Federated Learning for Decentralized Artificial Intelligence in Melanoma Diagnostics. *JAMA Dermatology*, 160(3):303–311.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.
- Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L.-C., Tan, M., Chu, G., Vasudevan, V., Zhu, Y., Pang, R., Adam, H., and Le, Q. (2019). Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324.
- Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861.
- Huang, D., Swanson, E. A., Lin, C. P., Schuman, J. S., Stinson, W. G., Chang, W., Hee, M. R., Flotte, T., Gregory, K., Puliafito, C. A., and Fujimoto, J. G. (1991). Optical coherence tomography. *Science*, 254(5035):1178–1181.
- Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269.
- Islam, M., Reza, M. T., Kaosar, M., and Parvez, M. Z. (2022). Effectiveness of federated learning and cnn ensemble architectures for identifying brain tumors using mri images. *Neural Processing Letters*, 55.

- Islamov, R., Safaryan, M., and Alistarh, D. (2024). AsGrad: A sharp unified analysis of asynchronous-SGD algorithms. In Dasgupta, S., Mandt, S., and Li, Y., editors, *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 649–657. PMLR.
- Kadir, M. A., Alam, H. M. T., and Sonntag, D. (2023). Edgeal: an edge estimation based active learning approach for oct segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 79–89. Springer.
- Kadir, M. A., Alam, H. M. T., Srivastav, D., Profitlich, H.-J., and Sonntag, D. (2024). Partial image active annotation (piaa): An efficient active learning technique using edge information in limited data scenarios. *KI-Künstliche Intelligenz*, pages 1–12.
- Kiefer, J. and Wolfowitz, J. (1952). Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 23(3):462 – 466.
- Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.
- Kulyabin, M., Zhdanov, A., Nikiforova, A., Stepichev, A., Kuznetsova, A., Ronkin, M., Borisov, V., Bogachev, A., Korotkich, S., Constable, P. A., and Maier, A. (2024). OCTDL: Optical Coherence Tomography Dataset for Image-Based Deep Learning Methods. *Scientific Data*, 11(1):365.
- Leconte, L., Nguyen, V. M., and Moulines, E. (2024). Favano: Federated averaging with asynchronous nodes. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5665–5669.
- 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 Med Inform*, 9(5):e25869.
- Lian, Z., Yang, Q., Zeng, Q., and Su, C. (2022). Webfed: Cross-platform federated learning framework based on web browser with local differential privacy. In *ICC 2022 - IEEE International Conference on Communications*, pages 2071–2076.
- Liu, J., Jia, J., Che, T., Huo, C., Ren, J., Zhou, Y., Dai, H., and Dou, D. (2024). Fedasmu: Efficient asynchronous federated learning with dynamic staleness-aware model update. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(12):13900–13908.
- Lo, J., Yu, T. T., Ma, D., Zang, P., Owen, J. P., Zhang, Q., Wang, R. K., Beg, M. F., Lee, A. Y., Jia, Y., and Sarunic, M. V. (2021). Federated learning for microvasculature segmentation and diabetic retinopathy classification of oct data. *Ophthalmology Science*, 1(4):100069.
- Ma, Y., Xiang, D., Zheng, S., Tian, D., and Liu, X. (2019). Moving deep learning into web browser: How far can we go? In *The World Wide Web Conference, WWW '19*, page 1234–1244, New York, NY, USA. Association for Computing Machinery.
- McMahan, B., Moore, E., Ramage, D., Hampson, S., and Arcas, B. A. y. (2017). Communication-Efficient Learning of Deep Networks from Decentralized Data. In Singh, A. and Zhu, J., editors, *Proceedings of the 20th International Conference on*

- Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR.
- Morid, M. A., Borjali, A., and Del Fiol, G. (2021). A scoping review of transfer learning research on medical image analysis using imagenet. *Computers in Biology and Medicine*, 128:104115.
- Moshawrab, M., Adda, M., Bouzouane, A., Ibrahim, H., and Raad, A. (2023). Reviewing federated machine learning and its use in diseases prediction. *Sensors*, 23(4).
- Nguyen, J., Malik, K., Zhan, H., Yousefpour, A., Rabbat, M., Malek, M., and Huba, D. (2022). Federated learning with buffered asynchronous aggregation. In Camps-Valls, G., Ruiz, F. J. R., and Valera, I., editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 3581–3607. PMLR.
- Ninomiya, K., Blandy, J., and Jones, B. (2024). WebGPU. W3C Working Draft, W3C.
- ONNX Runtime developers (2021). ONNX Runtime. Version: 1.19.0.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.
- Ran, A. R., Wang, X., Chan, P. P., Wong, M. O. M., Yuen, H., Lam, N. M., Chan, N. C. Y., Yip, W. W. K., Young, A. L., Yung, H.-W., Chang, R. T., Mannil, S. S., Tham, Y.-C., Cheng, C.-Y., Wong, T. Y., Pang, C. P., Heng, P.-A., Tham, C. C., and Cheung, C. Y. (2023). Developing a privacy-preserving deep learning model for glaucoma detection: a multicentre study with federated learning. *British Journal of Ophthalmology*.
- Robbins, H. and Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407.
- Rossberg, A. (2022). WebAssembly Core Specification. Technical report, W3C. Version Number: 2.0.
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520.
- Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626.
- Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.
- Smilov, D., Thorat, N., Assogba, Y., Yuan, A., Kreeger, N., Yu, P., Zhang, K., Cai, S., Nielsen, E., Soergel, D., Bileschi, S., Terry, M., Nicholson, C., Gupta, S. N., Sirajuddin, S., Sculley, D., Monga, R., Corrado, G., Viégas, F. B., and Wattenberg, M. (2019). Tensorflow.js: Machine learning for the web and beyond. *CoRR*, abs/1901.05350.

- Sonntag, D., Barz, M., and Gouvêa, T. (2024). A look under the hood of the interactive deep learning enterprise (no-idle). *arXiv preprint arXiv:2406.19054*.
- Su, N. and Li, B. (2022). How asynchronous can federated learning be? In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*, pages 1–11.
- Truex, S., Liu, L., Chow, K.-H., Gursoy, M. E., and Wei, W. (2020). *LDP-Fed: federated learning with local differential privacy*, page 61–66. Association for Computing Machinery, New York, NY, USA.
- Tseng, T., King Chen, J., Abdelrahman, M., Kery, M. B., Hohman, F., Hilliard, A., and Shapiro, R. B. (2023). Collaborative machine learning model building with families using co-ml. In *Proceedings of the 22nd Annual ACM Interaction Design and Children Conference, IDC '23*, page 40–51, New York, NY, USA. Association for Computing Machinery.
- Vidal, R. and Kameni, L. (2024). A general theory for federated optimization with asynchronous and heterogeneous clients updates. *J. Mach. Learn. Res.*, 24(1).
- Xie, C., Koyejo, S., and Gupta, I. (2019). Asynchronous Federated Optimization. *arXiv e-prints*.
- Xu, C., Qu, Y., Xiang, Y., and Gao, L. (2023). Asynchronous federated learning on heterogeneous devices: A survey. *Computer Science Review*, 50:100595.
- Zhang, T., Gao, L., Lee, S., Zhang, M., and Avestimehr, S. (2023). Timelyfl: Heterogeneity-aware asynchronous federated learning with adaptive partial training. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 5064–5073.
- Ángel Morell, J. and Alba, E. (2022). Dynamic and adaptive fault-tolerant asynchronous federated learning using volunteer edge devices. *Future Generation Computer Systems*, 133:53–67.